Notes

Protecting Computer Programs as Compilations Under *Computer Associates v. Altai*

Jon S. Wilkins

Writing software requires both technical expertise and artistic creativity. A competent programmer must have a firm grasp of the intricacies of hardware organization, programming languages, and other basic principles of computer science. This base of knowledge supplies the constituent elements of software. Designing a program, however, involves more than competent engineering. Selecting, arranging, and coordinating program elements to construct a complete software work is a subjective process that requires imagination and creativity. Just as a composer creates a symphony by selecting and arranging underlying elements—from the specific notes played to broad choices of theme and orchestral composition—a programmer designs software by selecting and integrating programming techniques, problem-solving algorithms, data structures, and other software elements. Although programming also entails the more mechanical process of translating design into functioning code, this necessary step does not detract from the significant creative expression embodied in the design and overall structure of software.

Unfortunately, the leading test of software copyright infringement employed by courts fails to consider selection and arrangement expression in program structure. In *Computer Associates International v. Altai, Inc.*,¹ the Second Circuit established a three-part test for deciding software infringement

1. 982 F.2d 693 (2d Cir. 1992).

cases.² Known as “abstraction-filtration-comparison” or “successive filtration,” this test requires courts to dissect programs and then evaluate the copyright protection afforded each part. While this approach rests on a sound doctrinal foundation and provides a useful analytical framework for evaluating complex software, it fails to account for the creative authorship required to design program structure.

Most traditional programs contain such design-level expression as a significant creative element. Underprotection of these programs alone offers sufficient reason to reevaluate abstraction-filtration-comparison. The software industry’s increasing reliance on a new technology known as object-oriented programming suggests even more strongly, however, that *Computer Associates* should be modified. Object-oriented programmers create large programs out of preexisting software building blocks, an approach that offers many advantages but tends to limit possibilities for creative expression to the selection and arrangement of software “objects.” Although selection and arrangement expression is but one kind of expression present in traditional software, in many cases it will be the only protectable element present in an object-oriented program. Courts must consider this kind of expression in order to address the next generation of software litigation.

Copyright law’s protection of works of “compilation” focuses on identifying precisely the selection and arrangement expression ignored by *Computer Associates*. The Supreme Court most recently addressed works of compilation in *Feist Publications, Inc. v. Rural Telephone Service Co.*,³ and various lower courts have developed the principles elaborated in *Feist*. This Note argues that courts should apply these principles to protect design-level expression in software. Part I provides a brief overview of relevant copyright and software issues. Part II explores the appropriateness of compilation protection for computer program design. Part III develops in more detail how compilation doctrine applies at each phase of abstraction-filtration-comparison. Part IV demonstrates the urgency of this doctrinal modification by discussing the importance of compilation expression in object-oriented software.

2. *Computer Associates* addresses “nonliteral” infringement of computer programs. Early software cases dealt with “literal” copying: line-for-line reproduction of the instructions that make up a program, similar to photoduplicating the text of a book or play. Copyright protects an author against more than just verbatim copying, however. “[J]ust as literary works such as books and plays enjoy copyright protection vis-a-vis not only their precise words, but their development and plot as well, so the nonliteral elements of computer programs have been held subject to copyright protection.” 1 MELVILLE B. NIMMER & DAVID NIMMER, NIMMER ON COPYRIGHT § 2.04[C], at 2-46.7 (1993) [hereinafter NIMMER] (footnotes omitted). For an overview of computer programs and the early history of software copyright, see generally Anthony L. Clapes et al., *Silicon Epics and Binary Bards: Determining the Proper Scope of Copyright Protection for Computer Programs*, 34 UCLA L. REV. 1493, 1502-05, 1510-36 (1987).

3. 499 U.S. 340 (1991).

I. OVERVIEW: PUTTING ABSTRACTION-FILTRATION-COMPARISON IN COPYRIGHT CONTEXT

A. Basic Copyright Principles

Cases of nonliteral⁴ infringement of software focus on one issue: whether the copying alleged is legally sufficient to constitute infringement.⁵ Not all copying of a copyrighted work is illegal. For example, copying public domain elements, or copying only a small portion of a copyrighted work, does not infringe a copyright.⁶ Determining whether copying constitutes infringement turns instead on a comparison of the copyrighted and accused works that requires a court to determine "the extent of similarity which will constitute a *substantial* and hence infringing similarity."⁷

The issue underlying substantial similarity analysis is copyright law's distinction between "ideas" and "expression": Original expression is protectable, ideas are not.⁸ The example of *Romeo and Juliet* demonstrates this principle's basic operation. A work that copies the play's dialogue and stage directions line-for-line clearly would infringe Shakespeare's copyright because it would replicate his original expression.⁹ A play that merely employs similar general ideas—for example, another play with a plot involving young lovers whose families bitterly oppose their relationship—would not infringe, because the only similarity between the two works would be at the level of ideas. As

4. For an explanation of the difference between literal and nonliteral infringement, see *supra* note 2.

5. See NIMMER, *supra* note 2, § 13.03[F], at 13-117 to 13-118; see, e.g., *Gates Rubber Co. v. Bando Chem. Indus.*, 9 F.3d 823, 833 (10th Cir. 1993) ("[E]ven if generalized copying is established . . . , it will ultimately still be necessary to establish copying of precisely identified protected elements of a program before copyright infringement can be established."). To show infringement, a copyright plaintiff must satisfy two requirements: "(1) ownership of a valid copyright, and (2) copying of constituent elements of the work that are original." *Feist*, 499 U.S. at 361. Litigants typically do not dispute copyright validity because the statutory definition of "literary works" clearly encompasses computer programs, see 17 U.S.C. § 101 (1988) (defining "literary works" as "works . . . expressed in . . . verbal or numerical symbols . . . , regardless of the nature of the material objects, such as . . . tapes, disks, or cards, in which they are embodied"), and courts agree that protection extends both to literal and nonliteral aspects of software. See generally NIMMER, *supra*, § 2.04[C] (citing statutes, legislative history, and cases). Also, the question of "actual" copying often goes undisputed because cases typically involve scenarios in which a company hires a competitor's programmer to write a "new" version of the competitor's successful program. See, e.g., *Gates Rubber*, 9 F.3d at 830-31; *Computer Assocs.*, 982 F.2d at 699-700; *Mitek Holdings, Inc. v. Arce Eng'g Co.*, No. 91-2629-CIV-MOORE, 1994 WL 515922, at *2-*4 (S.D. Fla. Sept. 9, 1994); see also *Walter A. Effross, Assaying Computer Associates v. Altai: How Will the "Golden Nugget" Test Pan Out?*, 19 RUTGERS COMPUTER & TECH. L.J. 1, 38-39 (1993). Alternatively, the company may reverse-engineer a competitor's source code from commercially distributed object code, resulting in infringement if the reverse-engineered program copies protectable elements of the original. See, e.g., *Atari Games Corp. v. Nintendo of Am. Inc.*, 975 F.2d 832, 839-40, 842-45 (Fed. Cir. 1992) (applying Ninth Circuit law).

6. See generally 3 NIMMER, *supra* note 2, § 13.03[A].

7. *Id.* § 13.03[A], at 13-29.

8. See 17 U.S.C. § 102(b) (1988); *Computer Assocs.*, 982 F.2d at 703 (citing *Mazer v. Stein*, 347 U.S. 201 (1954); *Baker v. Selden*, 101 U.S. 99 (1879)).

9. Of course, modern American copyright law provides no protection to *Romeo and Juliet*.

a matter of law, similarity only as to ideas does not reach the "substantial" level of similarity required for infringement.¹⁰

Although the distinction between specific dialogue and general themes in this example seems clear, protected expression and unprotected ideas in fact exist along a continuum. Consider a comparison of *Romeo and Juliet* and *West Side Story*. The literal expression of the Broadway production differs from Shakespeare's play. Yet the similarity in plot, characters, and sequence of events extends far beyond merely sharing a common theme. At some point along this continuum, a court must draw the line between protected expression and unprotected ideas.¹¹

In a famous passage from *Nichols v. Universal Pictures Corp.*,¹² Learned Hand articulates the "abstractions" test used by courts as the general approach to this problem:

Upon any work . . . a great number of patterns of increasing generality will fit equally well, as more and more of the incident is left out. The last may perhaps be no more than the most general statement of what the [work] is about, and at times might consist only of its title; but there is a point in this series of abstractions where they are no longer protected, since otherwise the [author] could prevent the use of his "ideas," to which, apart from their expression, his property is never extended.¹³

For example, a highly general statement of *Romeo and Juliet* is "a play about star-crossed lovers." More specific levels of abstraction include plot sequence or character development. The most specific level would be the words of a particular dialogue. After dissecting a work into general levels of abstraction, courts distinguish between unprotectable idea and protectable expression on a case-by-case basis.¹⁴ Similarity of expression constitutes infringement; similarity of ideas does not.

However, even case-by-case analysis cannot always cleanly dissociate protectable expression from unprotectable ideas. One copyright axiom is that a copyright holder should not obtain an effective monopoly over an idea by virtue of receiving protection for her particular expression. Courts accordingly

10. Nimmer notes that the idea/expression dichotomy should be understood "not so much [as] a limitation on the copyrightability of works as it is a measure of the degree of similarity which must exist as between a copyrightable work and an unauthorized copy, in order to constitute the latter an infringement." 1 NIMMER, *supra* note 2, § 2.03[D], at 2-34; *see also* *Worth v. Selchow & Righter Co.*, 827 F.2d 569, 572 (9th Cir. 1987), *cert. denied*, 485 U.S. 977 (1988).

11. The example is Nimmer's. *See* 3 NIMMER, *supra* note 2, § 13.03[A], at 13-35 to 13-39; *see also* *Kepner-Tregoe, Inc. v. Leadership Software, Inc.*, 12 F.3d 527, 535-36 (5th Cir. 1994) (using *Romeo and Juliet* to demonstrate idea/expression dichotomy), *cert. denied*, 63 U.S.L.W. 3223 (U.S. Oct. 3, 1994).

12. 45 F.2d 119 (2d Cir. 1930), *cert. denied*, 282 U.S. 902 (1931).

13. *Id.* at 121.

14. *See* *Peter Pan Fabrics, Inc. v. Martin Weiner Corp.*, 274 F.2d 487, 489 (2d Cir. 1960) (noting that "[d]ecisions must . . . inevitably be *ad hoc*").

deny protection if only a few ways to express an idea exist, under the theory that the idea and expression have "merged." For example, in *Baker v. Selden*, the Court denied copyright protection for the printed forms associated with a new kind of accounting system, reasoning that copyright protection for the forms would effectively give the author a patentlike monopoly over the system itself.¹⁵

Similarly, courts deny copyright protection to expression portraying standard elements of a work when allowing protection would have the effect of precluding others from creating works of the same genre. For example, in *Hoehling v. Universal City Studios, Inc.*¹⁶ the Second Circuit refused to extend protection to an author's descriptions of certain aspects of German life around the time of World War II, noting that "it is virtually impossible to write about a particular historical era or fictional theme without employing certain 'stock' or standard literary devices."¹⁷ This *scènes à faire* doctrine, like merger, supplements the basic idea/expression dichotomy by ensuring that protecting a work's expression does not have the unintended effect of also protecting ideas.

Once a court has used these doctrines to identify a work's protectable elements, substantial similarity has a second, more intuitive meaning: *how much* protectable material must be copied to make the accused work substantially similar. The degree of similarity required to become "substantial" significantly varies by type of work.¹⁸ For example, the content of a factual compilation such as a phone directory must be virtually identical to that of the copyrighted work in order to constitute infringement.¹⁹ In contrast, one Second Circuit decision says that copying even a single line of a poem may constitute infringement.²⁰ Determining substantial similarity ultimately turns on a court's ad hoc value judgment of whether the copying is *qualitatively* significant to the copyrighted work,²¹ a standard that obviously offers only general guidance to courts and litigants.²²

15. *Baker v. Selden*, 101 U.S. 99 (1879); see also *Herbert Rosenthal Jewelry Corp. v. Kalpakian*, 446 F.2d 738, 742 (9th Cir. 1971).

16. 618 F.2d 972 (2d Cir.), cert. denied, 449 U.S. 841 (1980).

17. *Id.* at 979 (denying protection to descriptions of beer halls, common greetings, and period songs).

18. See *Computer Assocs. Int'l v. Altai, Inc.*, 982 F.2d 693, 704 (2d Cir. 1992); *Sega Enters. v. Accolade, Inc.*, 977 F.2d 1510, 1524 (9th Cir. 1992); 3 NIMMER, *supra* note 2, § 13.03[A], at 13-49 to 13-55.

19. See *Key Publications, Inc. v. Chinatown Today Publishing Enters.*, 945 F.2d 509, 516-17 (2d Cir. 1991); see also *Kregos v. Associated Press*, 937 F.2d 700, 709 (2d Cir. 1991) (discussing high degree of similarity required to infringe compilation of racing statistics).

20. *Heim v. Universal Pictures Co.*, 154 F.2d 480, 487 n.8 (2d Cir. 1946) ("There may be wrongful copying, though small quantitatively; so if someone were to copy the words, 'Euclid alone has looked on Beauty bare,' or 'Twas brillig and the slithy toves.'"); cf. *Worth v. Selchow & Righter Co.*, 827 F.2d 569, 572 (9th Cir. 1987) (noting that "a verbatim copy or close paraphrase is not a necessary element to establish infringement" of fictional work), cert. denied, 485 U.S. 977 (1988).

21. See *infra* note 122 and accompanying text.

22. See *infra* notes 32-33 and accompanying text.

Although these copyright principles apply to computer programs just as to musical works or plays, most judges and juries have virtually no familiarity with software. Thus, while even a copyright neophyte can apply the idea/expression dichotomy to *Romeo and Juliet*, a court faced with the task of separating ideas from expression in a computer program requires a conceptual tool that relates traditional infringement analysis to software works. The Second Circuit's abstraction-filtration-comparison test provides such a framework and therefore represents a basically sound approach to deciding software infringement cases.

B. *Abstraction-Filtration-Comparison: Applying Traditional Principles to Computer Programs*

In *Computer Associates*, the Second Circuit drew upon fundamental copyright principles to develop a three-part test of substantial similarity for nonliteral aspects of computer software. The first prong adopts the *Nichols* abstractions test, a test traditionally applied to novels and plays.²³ One justification for applying this test to software is that copyright law classifies programs as literary works.²⁴ A more powerful argument relates abstraction to the very nature of programming. Generalizing broadly, a programmer starts with an abstract idea for a program; designs broad program structure; creates the more specific structure of functional subprograms; and finally writes the literal program code. Retracing these steps satisfies the abstraction test's conceptual approach of moving from the general to the specific.²⁵

In practice, the abstraction process has the effect of separating software into its constituent elements. For example, a court starts with the program's "general idea," such as running a dental office or managing an electronic mail system. Next, the court may identify the constituent subprograms that constitute the software whole. A dental office management program probably would have specific procedures to call up a patient's record, to schedule appointments, or to place an order for supplies. Other program parts may include data structures, interfaces, and libraries. Although these program parts roughly correspond to theoretical levels of traditional *Nichols*-type abstraction, in practice abstraction means dissecting programs.²⁶

After abstraction reveals a program's constituent elements, filtration requires a court to apply doctrines such as merger and *scènes à faire* to decide

23. *Computer Assocs. Int'l v. Altai, Inc.*, 982 F.2d 693, 706-07 (2d Cir. 1992).

24. See sources cited *supra* note 5.

25. *Computer Associates*, noting that the *Nichols* formulation "is adaptable to computer programs," recommends that courts "retrace and map each of the designer's steps—in the opposite order in which they were taken during the program's creation." 982 F.2d at 706-07; see also *Gates Rubber Co. v. Bando Chem. Indus.*, 9 F.3d 823, 834 (10th Cir. 1993).

26. See, e.g., *Gates Rubber*, 9 F.3d at 834 (identifying first step as "dissecting the allegedly infringed program according to the abstractions test").

which program parts contain protectable expression and which parts do not. Thus, elements that are so general as to constitute only ideas; have merged with their underlying ideas; are stock-in-trade or externally dictated elements;²⁷ or are taken from the public domain²⁸ are unprotectable.²⁹ The application of each of these doctrines during filtration prevents a software owner from turning copyright protection into a patentlike monopoly.³⁰

At the final step, comparison, "the court's substantial similarity inquiry focuses on whether the defendant copied any aspect of this protected expression, and assesses the copied portion's importance to the plaintiff's overall program."³¹ Whether the observed degree of similarity is substantial "poses essentially a value judgment" for the court.³² In other words, after reducing the copyrighted work to a core of protectable expression through abstraction and filtration, the decisionmaker must make a qualitative judgment as to whether the accused work copies "too much" protected expression. As in other copyright contexts, this determination is essentially ad hoc, and the complexity of software makes such case-by-case resolution very difficult.³³

Although courts remain far from unanimous on how to protect software,³⁴ the Second Circuit's approach in *Computer Associates* has become the leading test of substantial similarity for computer programs.³⁵ Most courts and

27. In the software context, one purpose of *scènes à faire* analysis is to prevent a copyright holder from monopolizing an entire category of software or system of program development by receiving protection for general formats or specifications. See 3 NIMMER, *supra* note 2, § 13.03[F], at 13-133 to 13-141 (listing five examples of elements that should be filtered by *scènes à faire*).

28. Public domain elements are not protectable because they are not original to the author. See *Gates Rubber*, 9 F.3d at 837-38.

29. *Computer Assocs.*, 982 F.2d at 707-10.

30. See *id.* at 710-11; *Atari Games Corp. v. Nintendo of Am. Inc.*, 975 F.2d 832, 842 (Fed. Cir. 1992); cf. Bruce G. Joseph, *Computers and Compilations*, in *ADVANCED SEMINAR ON COPYRIGHT LAW 1993*, at 140-43 (PLI Patents, Copyrights, Trademarks, & Literary Property Course Handbook Series No. 362, 1993) (criticizing *Computer Associates* court for being more concerned with monopolization than with traditional copyright principles).

31. *Computer Assocs.*, 982 F.2d at 710.

32. 3 NIMMER, *supra* note 2, § 13.03[F], at 13-146.

33. Comparison is "ad hoc" mainly in the sense that it is impossible to state any more precise standard than that "substantial" similarity means "qualitatively significant" similarity. By definition, "qualitative" implies a work-specific kind of analysis that defies determination on anything but a case-by-case basis.

34. See, e.g., *Engineering Dynamics, Inc. v. Structural Software, Inc.*, 26 F.3d 1335, 1341-42 (5th Cir. 1994) (adopting abstraction-filtration-comparison but noting that "[c]ourt decisions are in a state of creative ferment concerning the methods by which nonliteral elements of computer programs may be identified and analyzed for copyrightability").

35. See 3 NIMMER, *supra* note 2, § 13.03[A], at 13-49; see, e.g., *Engineering Dynamics*, 26 F.3d at 1343; *Kepner-Tregoe, Inc. v. Leadership Software, Inc.*, 12 F.3d 527, 534 (5th Cir. 1994) (noting abstraction-filtration-comparison is approved method of analysis), *cert. denied*, 63 U.S.L.W. 3223 (U.S. Oct. 3, 1994); *Gates Rubber Co. v. Bando Chem. Indus.*, 9 F.3d 823, 834 (10th Cir. 1993) (adopting abstraction-filtration-comparison), *vacating in part and aff'g in part* *Gates Rubber Co. v. Bando Am. Inc.*, 798 F. Supp. 1499, 1511 (D. Colo. 1992) (criticizing *Computer Associates*); *Autoskill, Inc. v. National Educ. Support Sys., Inc.*, 994 F.2d 1476, 1490 (10th Cir.) (identifying *Computer Associates* test as permissible approach), *cert. denied*, 114 S. Ct. 307 (1993); *Sega Enters. v. Accolade, Inc.*, 977 F.2d 1510, 1525 (9th Cir. 1992) ("In our view . . . the Second Circuit's approach is an appropriate one"); *Atari Games Corp. v. Nintendo of Am. Inc.*, 975 F.2d 832, 839 (Fed. Cir. 1992) (using *Computer Associates* approach to identify protectable expression); *Mitek Holdings, Inc. v. Arce Eng'g Co.*, No. 91-2629-CIV-MOORE, 1994

commentators agree that *Computer Associates* enjoys several advantages, especially as compared with prior judicial formulations.³⁶ First, abstraction-filtration-comparison provides an analytical mechanism for evaluating the copyright status of complex software works.³⁷ Second, the *Computer Associates* test rests firmly on established copyright principles and policy considerations.³⁸ Under *Feist*, an infringing work is one that contains elements substantially similar to *original* (i.e., protectable) parts of the copyrighted work.³⁹ Reducing a work to protectable expression before comparison therefore is an appropriate method of substantial similarity analysis.⁴⁰

C. Criticisms of Computer Associates

Commentators offer two primary criticisms of the Second Circuit's approach. First, *Computer Associates* provides little explanation of exactly how courts should proceed when conducting abstraction-filtration-comparison analysis. The opinion spends several pages elaborating on the kinds of analysis a court should apply at the filtration step, discussing the purposes of various

WL 515922, at *7, *9-*16 (S.D. Fla. Sept. 9, 1994) (noting that Second Circuit approach has "gained the widest acceptance" and applying abstraction-filtration-comparison); *Data Gen. Corp. v. Grumman Sys. Support Corp.*, 825 F. Supp. 340, 355 (D. Mass. 1993) (describing *Computer Associates* as "erudite opinion"); *Cmax/Cleveland, Inc. v. UCR, Inc.*, 804 F. Supp. 337, 352-53 (M.D. Ga. 1992) (applying *Computer Associates* analysis).

36. See *Whelan Assocs. v. Jaslow Dental Lab., Inc.*, 797 F.2d 1222 (3d Cir. 1986), *cert. denied*, 479 U.S. 1031 (1987). *Whelan* says that any program has one idea—for example, "to aid in the business operations of a dental laboratory"—and therefore "the detailed structure of [the program] is part of the expression, not the idea, of that program." *Id.* at 1238-39. Courts and commentators for the most part disagree with *Whelan*, finding that its "sweeping rule and broad language extend copyright protection too far." 3 NIMMER, *supra* note 2, § 13.03[A], at 13-46; see also *Computer Assocs. Int'l v. Altai, Inc.*, 982 F.2d 693, 705-06 (2d Cir. 1992); Julian Velasco, Note, *The Copyrightability of Nonliteral Elements of Computer Programs*, 94 COLUM. L. REV. 242, 261-62 (1994). But see Clapes et al., *supra* note 2, at 1579-83 (defending *Whelan*).

37. See *Computer Assocs.*, 982 F.2d at 706; *Atari Games*, 975 F.2d at 839 ("By separating the program into manageable components, [abstraction] eases the court's task of discerning the boundaries of protectable expression."); Effross, *supra* note 5, at 88; Velasco, *supra* note 36, at 282-84.

38. See *Gates Rubber*, 9 F.3d at 834, 839, 841-42; Velasco, *supra* note 36, at 276, 283. For example, using filtering doctrines such as merger and *scènes à faire*, see *supra* notes 13-14 and accompanying text, to exclude certain elements of programs from protection is an entirely appropriate, and indeed vital, step for courts to employ. In certain respects, the range of expression available to programmers is limited, see Peter S. Menell, *An Analysis of the Scope of Copyright Protection for Application Programs*, 41 STAN. L. REV. 1045, 1058-70 (1989); Todd Shuster, *Originality in Computer Programs and Expert Systems: Discerning the Limits of Protection Under Copyright Laws of France and the United States*, 5 TRANSNAT'L LAW. 1, 13-17 (1992); Timothy S. Teter, Note, *Merger and the Machines: An Analysis of the Pro-Compatibility Trend in Computer Software Copyright Cases*, 45 STAN. L. REV. 1061, 1066-69, 1097 (1993), and allowing monopolization in these areas contradicts copyright's aim of encouraging the free flow of ideas, see generally *Sony Corp. of Am. v. Universal City Studios, Inc.*, 464 U.S. 417, 428-33 (1984).

39. See *Feist Publications, Inc. v. Rural Tel. Serv. Co.*, 499 U.S. 340, 348 (1991) ("[C]opyright protection may extend only to those components of a work that are original to the author.").

40. See *Apple Computer, Inc. v. Microsoft Corp.*, Nos. 93-16867, 93-16869, 93-16883, 1994 WL 506999, at *8-*9 (9th Cir. Sept. 19, 1994); *Gates Rubber*, 9 F.3d at 832 n.7; cf. 3 NIMMER, *supra* note 2, § 13.03[E], at 13-93 to 13-94, 13-120 to 13-123 (suggesting that "successive filtering" test should be used not only for computer software, but also as general method of infringement analysis).

filters and the elements each one targets. In contrast, the decision provides only minimal discussion of abstraction and comparison.⁴¹ As several commentators have observed, this lack of guidance leaves significant ambiguity regarding the precise scope of protection available for software.⁴²

A second, more strident opposition to *Computer Associates* argues that dissecting software into constituent parts before comparison condemns the copyright to "death by a thousand cuts."⁴³ These critics argue that a copyrighted program must be considered in its entirety,⁴⁴ and they draw upon the Ninth Circuit's line of "concept and feel" cases⁴⁵ to support the claim that analytic dissection cannot completely replace "as a whole" consideration.⁴⁶ Courts have protected the "concept and feel" of works easily understood in their collective entireties, such as greeting cards and children's television shows. A copier arguably should be held liable for blatantly appropriating the creative essence of such a work even if the defendant can point to specific differences, and a layperson jury may be able to perform this evaluation.⁴⁷ In contrast, only experienced computer scientists could even plausibly conduct a holistic comparison of two complex computer programs.⁴⁸ Although general concern for protecting overall program structure is not misplaced, the "work

41. See *Computer Assocs.*, 982 F.2d at 706-11.

42. See Richard A. Beutel, *Software Engineering Practices and the Idea/Expression Dichotomy. Can Structured Design Methodologies Define the Scope of Software Copyright?*, 32 JURIMETRICS J. 1, 29 (1991) (noting that "[t]he problem with [successive filtration] is that it does not effectively address the vast corpus of software 'stuff' that inhabits the realm between" general ideas and literal code); David A. Lowe, Comment, *A Square Peg in a Round Hole: The Proper Substantial Similarity Test for Nonliteral Aspects of Computer Programs*, 68 WASH. L. REV. 351, 358 (1993); John W.L. Ogilvie, Note, *Defining Computer Program Parts Under Learned Hand's Abstractions Test in Software Copyright Infringement Cases*, 91 MICH. L. REV. 526, 547-48 (1992) (criticizing Second Circuit for "creat[ing] an unclear and incomplete set of abstraction parts").

43. Anthony L. Clapes & Jennifer M. Daniels, *Revenge of the Luddites: A Closer Look at Computer Associates v. Altai*, COMPUTER LAW., Nov. 1992, at 11, 12-13; see also Joseph, *supra* note 30, at 227-28; Lowe, *supra* note 42, at 366-71.

44. See, e.g., *Gates Rubber Co. v. Bando Am., Inc.*, 798 F. Supp. 1499, 1511-13 (D. Colo. 1992), *vacated in part and aff'd in part sub nom. Gates Rubber Co. v. Bando Chem. Indus.*, 9 F.3d 823 (10th Cir. 1993); Lowe, *supra* note 42, at 361; Clapes & Daniels, *supra* note 43, at 13.

45. See, e.g., *Shaw v. Lindheim*, 919 F.2d 1353, 1356-57 (9th Cir. 1990), *Sid & Mary Krofft Television Prods., Inc. v. McDonald's Corp.*, 562 F.2d 1157, 1164 (9th Cir. 1977); *Roth Greeting Cards v. United Card Co.*, 429 F.2d 1106 (9th Cir. 1970).

46. See Clapes & Daniels, *supra* note 43, at 13.

47. Although "concept and feel" cases appear to use a very different approach to determining substantial similarity, in important respects the substantive analysis in these cases is quite similar to that in *Computer Associates*. The main difference is that laypersons may be able to identify and consider unprotectable elements when comparing the "concept and feel" of commonly understood works such as books, plays, or television shows. In other words, a jury is capable of deciding on its own that mere similarity at the level of basic themes or stock-in-trade mechanisms does not mean that two television programs are "substantially" similar. But see 3 NIMMER, *supra* note 2, § 13.03[E], at 13-101 to 13-114 (questioning *Shaw* and *Krofft* as possibly invalid after *Feist*).

48. Cf. *Gates Rubber*, 798 F. Supp. at 1513-14 (identifying "expert audience" test as alternate substantial similarity test).

as a whole" position remains a distinctly minority view that seems misapplied to complex software works.⁴⁹

Thus, a brief overview of the substance of and reaction to the *Computer Associates* test shows that it provides focus to established copyright doctrine by creating an analytical framework that is necessary because of the complex nature of software. The Second Circuit's terse treatment of certain issues, however, leaves continuing ambiguity as to how courts should conduct abstraction-filtration-comparison analysis. This uncertainty raises a number of concerns. Potential litigants experience difficulty assessing claims. Faced with the daunting task of "abstracting" a complex program into its constituent parts without any specific guidelines, courts may rely on expert witnesses rather than exercise independent judgment.⁵⁰ Moreover, despite its advantages, the test's emphasis on dissection narrows a court's range of view and diverts judicial attention away from the higher-level expression embodied in program structure and design.⁵¹

D. Object-Oriented Software: The Future of Software Development

The lack of protection for program design under *Computer Associates* assumes special importance because of a technological change occurring in the software industry: the widespread adoption of object-oriented design methodologies. Object-oriented design is a new software paradigm that emphasizes the use of self-contained, interchangeable software parts for constructing large systems. Under abstraction-filtration-comparison, most elements of an object-oriented program will be unprotectable, and appropriately so. Most of the creative authorship in an object-oriented program exists at the level of designing the interaction of its constituent parts: selecting which software objects to use, arranging them into a functioning software whole, and coordinating their interaction.⁵² This "forest-level" authorship is the expression most easily overlooked by courts intent on dissecting a program.

49. Indeed, protecting the "concept and feel" of software would conflict with the requirement that courts distinguish between a work's protectable and unprotectable elements. See *Feist Publications, Inc. v. Rural Tel. Serv. Co.*, 499 U.S. 340, 361 (1991); sources cited *supra* note 40; cf. *Bellsouth Advertising & Publishing Corp. v. Donnelley Info. Publishing, Inc.*, 999 F.2d 1436, 1445 (11th Cir. 1993) ("By comparing the overall appearance of the two directories . . . , the district court effectively failed to consider whether Donnelley copied the 'constituent elements of the work that are original.'" (quoting *Feist*, 499 U.S. at 361)), *cert. denied*, 114 S. Ct. 943 (1994). Note that the Ninth Circuit has moved away from using "concept and feel" analysis for functional works, see *Apple Computer, Inc. v. Microsoft Corp.*, Nos. 93-16867, 93-16869, 93-16883, 1994 WL 506999, at *9-*11 (9th Cir. Sept. 19, 1994); *Cooling Sys. & Flexibles, Inc. v. Stuart Radiator, Inc.*, 777 F.2d 485, 493 (9th Cir. 1985), retaining it only for traditional artistic works, see *McCulloch v. Albert E. Price, Inc.*, 823 F.2d 316, 321 (9th Cir. 1987).

50. See *Clapes & Daniels*, *supra* note 43, at 15.

51. See *Velasco*, *supra* note 36, at 278-79, 285; cf. *Atari Games Corp. v. Oman*, 979 F.2d 242, 245 (D.C. Cir. 1992) (noting that focusing on details of work may have caused Register of Copyrights to neglect higher-level aspects).

52. See *infra* part IV.B.

Virtually all courts and commentators base their understandings of software design on the traditional "top-down" approach to writing programs.⁵³ Top-down design requires the programmer to break a programming problem into more manageable subproblems, each of which in turn can be solved by further decomposition. Ultimately the top-down programmer writes every program instruction. Rather than attempting to compose a single block of computer instructions to balance a checkbook, for example, the top-down programmer writes a number of subprograms—such as one to ask the user for input, one to display the ledger, one to calculate an updated balance—then integrates them into a single software work.⁵⁴

Object-oriented programs are quite different from top-down programs. One journalist provides a useful analogy for understanding the difference:

Imagine that you are building a car. You could start from scratch by designing and building every component such as body parts, tires, bolts and mirrors. That's the way programs traditionally have been written.

Object-oriented programming, however, is more like the way cars are actually built. Auto makers focus design efforts on what the new car's unique features will be . . . but most of the car is still assembled from existing parts. In fact, some of the parts, such as tires, spark plugs and rear-view mirrors, are purchased from other companies. In programming terms, these parts are objects.⁵⁵

Rather than writing entirely new programs each time, object-oriented designers focus on how they can bring together existing software components in creative ways that add capabilities to existing systems or solve new problems without duplicating prior work.⁵⁶

The model underlying object-oriented design is based on reusable, independent software components. This focus produces a range of benefits for programmers: greater economy of programming expression, more code reuse, easier incremental modification, and greater resiliency.⁵⁷ For these reasons,

53. See, e.g., *Computer Assocs. Int'l. v. Altai, Inc.*, 982 F.2d 693, 697–98 (2d Cir. 1992), 3 NIMMER, *supra* note 2, § 13.03[F], at 13-124 to 13-125; Clapes et al., *supra* note 2, at 1543–44; Lowe, *supra* note 42, at 353–54; Ogilvie, *supra* note 42, at 532–33; Velasco, *supra* note 36, at 245–46. See generally David M. Barkan, *Software Litigation in the Year 2000: The Effect of Object-Oriented Design Methodologies on Traditional Software Jurisprudence*, 7 HIGH TECH. L.J. 315 (1992) (contrasting traditional and object-oriented approaches and considering implications for intellectual property protection of software).

54. In programming terminology, subprograms that perform a single specific task are called "subroutines," "functions," or "procedures." See STEPHEN J. GARLAND, *INTRODUCTION TO COMPUTER SCIENCE* 136–54 (1986). Larger subprograms consisting of groups of functions or procedures organized into separate modules are "libraries." *Id.* at 155–65.

55. Lawrence J. Magid, *An Easter Way to Create Software*, L.A. TIMES, June 1, 1989, § 4, at 3.

56. See Barkan, *supra* note 53, at 324.

57. *Id.* at 320–21. See generally SAMUEL N. KAMIN, *PROGRAMMING LANGUAGES* 273–348 (1990) (providing detailed discussion of object-oriented language "Smalltalk"). For just one of many other discussions of the advantages of object-oriented design, see Jane Poss, *Rules of the Game: Local Group Seeks To Influence How the Next Generation of Software Works*, BOSTON GLOBE, Nov. 25, 1990, at A33.

most software developers soon will have object-oriented products on the market.⁵⁸ One of the only legal commentators to write on the subject predicts that, as a consequence, "[t]he next generation of software cases is likely to involve alleged infringement of . . . programs designed according to the object-oriented model."⁵⁹

The problem of protecting object-oriented software highlights the concern that abstraction-filtration-comparison fails to account for design-level expression in software. Indeed, whereas top-down programs may contain a range of protectable expression, object-oriented programs often contain only the selection and arrangement expression reflected in program design. Unless modified to consider selection and arrangement expression, the Second Circuit's test will continue to underprotect software and will have great difficulty addressing the next generation of software cases. Parts II and III address this problem by developing how courts could incorporate compilation principles into abstraction-filtration-comparison.

II. THE NATURE OF COMPILATION EXPRESSION IN SOFTWARE

Copyright law provides a specific category of protection for the creative authorship embodied in the selection, arrangement, and coordination of a work's constituent parts: compilation expression. Although many cases involve fact-cataloguing directories such as telephone listings, courts have identified compilation expression in a wide range of works. An analysis of the legal characteristics of compilations and the nature of software design confirms that computer programs should be protected as works of compilations.

A. *Characteristics of Works of Compilation*

The statutory definition of a compilation is "a work formed by the collection and assembling of preexisting materials or of data that are selected, coordinated, or arranged in such a way that the resulting work as a whole

58. One software designer notes that "virtually any mid- to large-scale application project begun today would be object-oriented." Telephone Interview with Christopher R. Bingham, Software Engineer, Apple Computer, Inc. (Apr. 5, 1994); see also John W. Verity & Evan I. Schwartz, *Software Made Simple: Will Object-Oriented Programming Transform the Computer Industry?*, BUS. WK., Sept. 1991, at 92. To pick a particularly visible example, Apple and IBM embarked on a joint venture in 1991 that led to the creation of "Taligent," a company attempting to develop an object-oriented operating system. See generally Charles T.C. Compton, *Cooperation, Collaboration, and Coalition: A Perspective on the Types and Purposes of Technology Joint Ventures*, 61 ANTITRUST L.J. 861, 873 n.53 (1993) (discussing Taligent); Andrew C. Hruska, Note, *A Broad Market Approach to Antitrust Product Market Definition in Innovative Industries*, 102 YALE L.J. 305, 316-17 (1992) (same); Evelyn Richards, *IBM, Apple Plan Broad Cooperation Venture To Develop New Technology That Could Alter Industry*, WASH. POST, July 4, 1991, at A1; Mark Trumbull, *New Software Building Blocks Could Transform US Computing*, CHRISTIAN SCI. MONITOR, Jan. 7, 1994, at A11 (noting that Hewlett-Packard recently joined Apple-IBM alliance).

59. Barkan, *supra* note 53, at 316.

constitutes an original work of authorship."⁶⁰ A yellow pages illustrates these concepts. In a copyrighted yellow pages, "selection" would refer to the author's choice of which businesses to include in the directory. "Arrangement" would reflect the author's grouping of these businesses into categories (e.g., "SPORTING GOODS"). "Coordination" would refer to the ordering of those categories within the directory, or to actual placement of specific businesses into categories.⁶¹ Although conceptually different, these three kinds of authorship overlap considerably in practice depending on the nature of the underlying work.

Legal analysis of compilations begins with the apparent conflict between two copyright axioms: Individual facts are not copyrightable, but compilations of facts are. The Supreme Court in *Feist*⁶² reconciles this "undeniable tension" by ruling that the Constitution requires "originality" for an element of a work to receive copyright protection.⁶³ Facts are not protectable because they do not owe their origin to any author. In contrast, a compilation may contain original expression in the selection, arrangement, and coordination of its constituent facts.⁶⁴

Courts have considered compilation expression in many works that are not traditional factual directories.⁶⁵ A protectable "compilation" results from the assembly of constituent material such that "the resulting work as a whole constitutes an original work of authorship."⁶⁶ When deciding what kinds of works satisfy this requirement, courts have looked to whether the final work is "greater than the sum of its parts."⁶⁷ For example, in *Sem-Torq, Inc. v. K Mart Corp.*,⁶⁸ a sign manufacturer argued that an in-store display consisting of several otherwise unprotectable signs should be protected as a work of

60. 17 U.S.C. § 101 (1988).

61. See, e.g., *Key Publications, Inc. v. Chinatown Today Publishing Enters.*, 945 F.2d 509, 513-14 (2d Cir. 1991).

62. *Feist Publications, Inc. v. Rural Tel. Serv. Co.*, 499 U.S. 340 (1991).

63. *Id.* at 345-46.

64. *Id.* at 347-48.

65. See, e.g., *Apple Computer, Inc. v. Microsoft Corp.*, Nos. 93-16867, 93-16869, 93-16883, 1994 WL 506999, at *10 (9th Cir. Sept. 19, 1994) (graphical user interfaces for computer programs); *Engineering Dynamics, Inc. v. Structural Software, Inc.*, 26 F.3d 1335, 1346 (5th Cir. 1994) (input formats for computer programs); *Atari Games Corp. v. Oman*, 979 F.2d 242, 246 (D.C. Cir. 1992) (video games); *Harper House, Inc. v. Thomas Nelson, Inc.*, 889 F.2d 197, 204 (9th Cir. 1989) (organizer notebooks); *Educational Testing Servs. v. Katzman*, 793 F.2d 533, 538 (3d Cir. 1986) (college admissions tests); *Apple Barrel Prods., Inc. v. Beard*, 730 F.2d 384, 387-88 (5th Cir. 1984) (country music shows); see also 1 NIMMER, *supra* note 2, § 2.11[D], at 2-172.26 ("There is some indication that the courts will regard a biography, history or other factual account as a judicious selection and arrangement of facts . . ."). Nimmer cites a number of cases in which courts acknowledge separate protection for selection and arrangement in works far removed from traditional factual directories. See, e.g., *Pacific & S. Co. v. Duncan*, 744 F.2d 1490, 1494 (11th Cir. 1984) (stating that "editorial judgment" makes news broadcast original work of authorship), *cert. denied*, 471 U.S. 1004 (1985).

66. 17 U.S.C. § 101 (1988); *Feist*, 499 U.S. at 356.

67. *Sem-Torq, Inc. v. K Mart Corp.*, 936 F.2d 851, 855 (6th Cir. 1991) (quoting *Financial Info., Inc. v. Moody's Investors Serv., Inc.*, 751 F.2d 501, 505 (2d Cir. 1984), *cert. denied*, 484 U.S. 820 (1987)).

68. 936 F.2d at 851.

compilation. Finding that the arrangement offered nothing of independent value that would suggest the existence of original expressive content distinct from the signs themselves, the Sixth Circuit denied compilation protection.⁶⁹

This definition of a protectable "compilation" indicates that courts should approach selection and arrangement as a separate, independent category of authorship. A work containing compilation expression may or may not also contain other expression. Compilation-level authorship is the only protectable expression in a telephone book because none of the constituent elements—the names, addresses, and telephone numbers in the directory—can be protected independently by copyright. Compilation expression also exists, however, even where some constituent material does receive protection. Literary anthologies, legal casebooks, and similar works involving the integration of separately protectable elements into a new work also receive protection for original compilation authorship.⁷⁰ Because compilation expression is the only protectable element in fact-based works, such works enjoy "thin" copyright protection;⁷¹ works that combine elements themselves protectable enjoy a "thin" measure of protection for selection and arrangement in addition to any protection afforded the constituent material.

B. *Identifying Protectable Compilation Expression*

While courts agree that compilation expression exists as an independently significant, separately identifiable part of a work, this general definition provides little guidance for actually identifying which elements of which works merit protection under compilation principles. Several post-*Feist* decisions from federal appellate courts address the question of how to identify protectable compilation expression. An analysis of these cases supports the argument that program design should receive protection as a form of compilation expression.

1. *The Importance of Available Options*

Courts stress that a factor distinguishing protectable compilation expression is the existence of a range of options for the compiling author. For example, "selection" authorship in factual listings "implies the exercise of judgment in choosing which facts from a given body of data to include in a compilation."⁷² In *Key Publications, Inc. v. Chinatown Today Publishing*

69. *Id.* at 855–56.

70. Compilations consisting of materials that are themselves protectable by copyright are "collective works." See 17 U.S.C. § 101 (1988); 1 NIMMER, *supra* note 2, § 3.02.

71. *Feist Publications, Inc. v. Rural Tel. Serv. Co.*, 499 U.S. 340, 349 (1991); cf. Robert C. Denicola, *Copyright in Collections of Facts: A Theory for the Protection of Nonfiction Literary Works*, 81 COLUM. L. REV. 516, 538–42 (1981) (arguing that arrangement expression provides basis for protecting nonfiction works such as biographies and histories).

72. *Key Publications, Inc. v. Chinatown Today Publishing Enters.*, 945 F.2d 509, 513 (2d Cir. 1991).

Enterprises, the Second Circuit extended protection to an author's selection of a list of businesses because she excluded those that, in her judgment, would not remain open very long.⁷³ Considering an author's range of options helps identify protectable expression because the existence of choices increases the likelihood that the inclusion of particular elements results from authorial judgment.⁷⁴

Courts also look to available options when considering compilations that serve a specific purpose instead of simply cataloguing facts. In *Kregos v. Associated Press*,⁷⁵ the Second Circuit considered a "functional" compilation: a table of baseball statistics printed in newspapers to help fans predict the outcomes of games. This work expresses a specific idea—that a certain system can predict winners of baseball games—and the *Kregos* court therefore focused its inquiry on whether the work's particular selection and arrangement had merged with this underlying purpose.

Finding that the work's expression and idea had not merged, the Second Circuit emphasized the *subjective* nature of the author's claim that the statistical listing had predictive qualities:

As long as selections . . . involve matters of taste and personal opinion, there is no serious risk that [protecting the particular selection] will extend protection to an idea. . . . However, where a selection . . . is the first step in an analysis that yields a precise result . . . , protecting the "expression" of the selection would clearly risk protecting the idea of the analysis.⁷⁶

The court's reasoning indicates that the selection and arrangement embodied in a functional compilation should be denied protection when objective factors dictate a particular arrangement, whereas selections reflecting an author's subjective judgment should be protected. Subjective preferences influence choices of selection and arrangement only when there are many ways to achieve the work's basic purpose. *Kregos'* distinction between objectively and

73. *Id.*

74. See *Atari Games Corp. v. Nintendo of Am. Inc.*, 975 F.2d 832, 840 (Fed. Cir. 1992) (emphasizing arbitrariness of selection of programming instructions); *Eckes v. Card Prices Update*, 736 F.2d 859, 862–63 (2d Cir. 1984) (finding selection in choice of 5000 "premium" baseball cards from among 18,000 possible cards); *Coates-Freeman Assocs., Inc. v. Polaroid Corp.*, 792 F. Supp. 879, 883–85 & n 1 (D. Mass. 1992) (denying protection to selection and arrangement of management training chart because author chose "one of a handful . . . of highly similar models"); cf. 1 NIMMER, *supra* note 2, § 3.02, at 3-7 to 3-8 (giving example that collecting *all* of author's writings into anthology would not receive separate compilation protection because it would require no independent judgment by compiler).

75. 937 F.2d 700 (2d Cir. 1991).

76. *Id.* at 707; see also *id.* at 711 (Sweet, J., dissenting) (arguing that idea of work should be defined more narrowly to support holding that merger had occurred).

subjectively designed works therefore underscores the importance of the range of options for compilation works.⁷⁷

Further developing the Second Circuit's analysis in *Key Publications* and *Kregos*, the Eleventh Circuit directly discussed the importance of options in *Bellsouth Advertising & Publishing Corp. v. Donnelley Information Publishing, Inc.*⁷⁸ To satisfy copyright's minimum originality threshold, an author's selection and arrangement must be drawn from a range of *viable* options; the court reasoned that the existence of useful alternative approaches provides evidence of the creativity and originality of an author's choice.⁷⁹ In a recent example from the software context, the Fifth Circuit extended protection to the input formats used to provide data to a computer program, reasoning that "[t]he creativity inherent in [the formats] is proved by the existence [of] other, dissimilar structural engineering programs available in the market."⁸⁰ Thus, "[t]he relevant inquiry is not whether there is some imaginable, although manifestly less useful, method of arranging [a compilation's constituent elements]," but whether authors face a range of viable alternative forms of expression.⁸¹

The general principle underlying these decisions is that a protectable compilation results when an author's choice of a selection and arrangement is driven not by any external factor—whether some objective functional requirement or simply a lack of other alternatives—but rather results from her own subjective judgment. Because programmers employ their experience and creativity to pick particular architectural solutions from a wide range of viable engineering alternatives, software design epitomizes the compilation authorship dynamic described by leading cases. Indeed, examining software through the lens of traditional compilation principles reveals the prominence of selection and arrangement expression in computer programs.

2. *The Range of Options During Software Design*

Computer programs fit copyright's definition of a work of compilation: A functioning program undoubtedly has independent value compared with a disaggregated grouping of instructions, subprograms, and other program elements. A crucial question for the claim that courts should protect the selection and arrangement of computer program therefore is the range of options available to programmers. Courts and commentators provide two

77. See *Kregos*, 937 F.2d at 704, 704 n.3, 706; see also John A. Odozynski, *Infringement of Compilation Copyright After Feist*, 17 U. DAYTON L. REV. 457, 481–83 (1992).

78. 999 F.2d 1436 (11th Cir. 1993), *cert. denied*, 114 S. Ct. 943 (1994).

79. For example, an alphabetically arranged telephone directory should not receive protection simply because entries conceivably could be arranged alphabetically by middle initial, chronologically by date of birth, or according to some other useless principle of arrangement.

80. *Engineering Dynamics, Inc. v. Structural Software, Inc.*, 26 F.3d 1335, 1346 (5th Cir. 1994).

81. *Bellsouth*, 999 F.2d at 1443.

sharply conflicting views on the range of options in programming. Neither position fully captures the multidimensional nature of writing software, a process that involves both the mechanical task of coding and the unconstrained art of creating overall design.

Some commentators argue that the many external constraints on a programmer—such as compatibility requirements, efficiency concerns, and programming conventions—severely limit the range of expressive choices available. “As machine parts, programs often contain technologies developers find necessary to meet technical requirements or standard industry practice. . . . [S]uch technologies rarely constitute a developer’s original expression”⁸² These writers favor narrow protection for software, emphasizing the utilitarian nature of programs and characterizing programming as a mechanical exercise that presents only a narrow range of alternatives.⁸³

Others contest this view. Copyright’s leading treatise cautions that “[c]omputer programming is a highly creative and individualistic endeavor. A court should not be led . . . to believe that complex programs consist only of commonly known techniques and materials strung together without significant originality or skill.”⁸⁴ One court writes that the software writer “is faced with a virtually endless series of decisions as to how to carry out the assigned task. . . . At every level, the process is characterized by choice, often made arbitrarily, and only occasionally dictated by necessity.”⁸⁵ This view supports broader protection for software, and some of these writers characterize programming as an authorial act comparable to writing traditional literary works such as novels and plays.⁸⁶

The contrast between these two characterizations of programming verges on the extreme, yet each reflects some truth. While any software engineer or computer scientist would confirm that building software has a creative

82. Shuster, *supra* note 38, at 13.

83. See, e.g., *Computer Assocs. Int'l v. Altai, Inc.*, 982 F.2d 693, 704 (2d Cir. 1992); *Sega Enters. v. Accolade, Inc.*, 977 F.2d 1510, 1524 (9th Cir. 1992); *Brief Amicus Curiae of Eleven Copyright Law Professors in Sega Enterprises Ltd. v. Accolade, Inc.*, 33 JURIMETRICS J. 147, 158 (1992); Shuster, *supra* note 33, at 9 (characterizing programs as mechanical); see also *Key Publications, Inc. v. Chinatown Today Publishing Enters.*, 945 F.2d 509, 513–15 (2d Cir. 1991) (indicating that mechanically arranged elements of compilation lack minimum creativity required for copyright); Leo J. Raskind, *Assessing the Impact of Feist*, 17 U. DAYTON L. REV. 331, 338 (1992) (explaining *Feist*’s originality requirement as applying to authorship that is “in no sense mechanical”); John F. Hayden, Note, *Copyright Protection of Computer Databases After Feist*, 5 HARV. J.L. & TECH. 215, 229–34 (1991) (questioning whether any utilitarian or functional work can contain truly “subjective” selection in same sense as purely aesthetic works).

84. 3 NIMMER, *supra* note 2, § 13.03[F], at 13–140; see also Clapes et al., *supra* note 2, at 1528–38; Joseph, *supra* note 30, at 140–45; Recent Case, 106 HARV. L. REV. 510, 512–13 (1992) (criticizing Second Circuit’s treatment of design choices as essentially functional); Clapes & Daniels, *supra* note 43, at 14 (“[S]oftware authorship is a creative, unconstrained form of writing, not a mechanistic task.”)

85. *SAS Inst., Inc. v. S & H Computer Sys.*, 605 F. Supp. 816, 825 (M.D. Tenn. 1985); see also *Sega Enters. v. Accolade, Inc.*, 977 F.2d 1510, 1524 (9th Cir. 1992) (“To the extent that there are many possible ways of accomplishing a given task . . . the programmer’s choice of program structure and design may be highly creative and idiosyncratic.”); *M. Kramer Mfg. Co. v. Andrews*, 783 F.2d 421, 436 (4th Cir. 1986) (noting existence of virtually innumerable ways to program even basic computations).

86. See, e.g., Beutel, *supra* note 42, at 4; Clapes et al., *supra* note 2, at 1507, 1583.

component far removed from the mechanistic portrayal,⁸⁷ comparisons to the purely literary seem forced as well. Compared to a novelist, the software author undeniably faces constraints such as functional requirements and the relatively small number of instructions in computer languages.⁸⁸

A more accurate description of software development falls between these two extremes. Software creation involves two distinct acts: designing and coding. Designing is the creation of a program's structure at a broadly conceptual level without reference to specific programming instructions. Coding is the more mechanical process of implementing this structure.⁸⁹ Unfortunately, courts generally fail to distinguish between these two activities that, though related, reflect very different degrees of authorship.⁹⁰

3. *Coding v. Designing*

In many respects coding is a mechanical process. Most parts of a program's literal code exhibit only minimal creativity because few choices are available at this highly specific level.⁹¹ For example, most computer languages provide only a few ways to implement a "loop," a programming construct that directs the computer to repeat a group of instructions a specified number of times.⁹² All programs contain standard elements of this type that leave little room for programmer creativity.

In contrast, the range of options in designing a program is almost unlimited; indeed, the malleability of the software medium is the characteristic that most distinguishes software from other engineering disciplines. Using the building blocks provided by a set of instructions, an operating system's features, and the capabilities of the underlying hardware, a programmer can create as grand, complicated, artful, or elegant a program as she can imagine, free of the constraints of physical laws that limit architects or mechanical engineers. One of computer science's leading experts on software engineering writes that "[t]he 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

87. See, e.g., JON BENTLEY, PROGRAMMING PEARLS at v-vi (1986) (describing "programming pearls" as software elements "whose origins lie beyond solid engineering, in the realm of insight and creativity"); Beutel, *supra* note 42, at 5 (quoting programmer's description of the "imagining" involved in designing structure).

88. See Velasco, *supra* note 36, at 273.

89. See Beutel, *supra* note 42, at 5-6; Velasco, *supra* note 36, at 246-47.

90. See, e.g., Computer Assocs. Int'l v. Altai, Inc., 982 F.2d 693, 708 (2d Cir. 1992) (treating implementing subroutines and designing structure as similar activities for purposes of merger analysis).

91. See Clapes et al., *supra* note 2, at 1533 ("Taken individually, of course, no one . . . logic element[] can be said to be highly expressive in itself.").

92. For example, in programming languages such as "Pascal" or "C," equivalent loops could be designed using one of only a few constructs, such as "FOR—NEXT," "WHILE—DO," or "DO—UNTIL." See GARLAND, *supra* note 54, at 85-89 (Pascal); BRIAN W. KERNIGHAN & DENNIS M. RITCHIE, THE C PROGRAMMING LANGUAGE 60-64 (2d ed. 1988).

polish and rework, so readily capable of realizing grand conceptual structures.”⁹³

For example, software writers enjoy broad discretion in some of the basic elements of design: shaping modular structure,⁹⁴ directing control and data flow,⁹⁵ and choosing particular permutations of programming logic.⁹⁶ In copyright terms, at the design phase programmers have wide discretion to “select” particular algorithms, techniques, and structures; to “arrange” these elements by creating a flow of program and data control; and to “coordinate” these program parts into a working software whole. Creating object-oriented programs epitomizes this design dynamic. Because object-oriented programming emphasizes reuse of preexisting software “objects,” the creator of an object-oriented program may not write any new code herself. Object-oriented programming mainly involves selecting objects and arranging them in a way that fits the requirements of the particular programming task. In fact, given a sufficiently user-friendly way to select and arrange objects, someone with little formal software engineering background could create moderately sophisticated object-oriented programs.⁹⁷

Thus, “[d]esign is the qualitative result of combining structural, flow, and logic elements in the particular fashion chosen by the author. . . . The resulting combination is a tapestry of decisions and actions that is the essence of the author’s expression.”⁹⁸ The functional and linguistic constraints on programmers dictate that two programs that accomplish the same general task may use many of the same underlying algorithms and techniques, and even

93. FREDERICK P. BROOKS, JR., *THE MYTHICAL MAN-MONTH* 7–8 (1975).

94. To simplify development and maintenance, every large program is broken up into components referred to as libraries or modules. See *supra* note 54. A crucial aspect of design is establishing the interfaces that these subprograms use to communicate. GARLAND, *supra* note 54, at 155–58, 180–82. However, nothing demands the use of a particular modular structure or interface design, and typically many different viable structures exist.

95. A program’s flow of control is, quite simply, the order in which the program’s instructions execute. For example, a programmer may design the “main loop” of a program to begin with initialization routines, move to user input routines, and conclude with “clean-up” routines that save changed data and reset the system. Although a general pattern of control flow applies to most software (e.g., initialization routines execute first), nothing dictates the particular control flow of a program. Similarly, data flow describes the process by which data enters and is acted upon by the program. Designing data flow requires the programmer to answer questions such as: Where does data come from? (from a file, from user input?), Where does data reside while the program executes? (in cache memory, in main memory, or on disk?). See Ogilvie, *supra* note 42, at 534–35 (defining “system architecture” level of abstraction as “how the program operates,” including flow of control and data) (emphasis omitted).

96. While choosing a particular element of program logic—such as selecting a “FOR—NEXT” construct to perform multiple iterations over a range of values—typically involves little creativity, programmers have much more discretion when creating larger groupings of logical elements to perform complicated, multistep tasks. See Clapes et al., *supra* note 2, at 1531–33.

97. For example, “visual programming languages” allow users to create object-oriented programs by simply connecting on-screen representations of objects using a mouse or other graphical user interface device. Apple Computer’s “HyperCard” system provides a rudimentary example. See Magid, *supra* note 55, at 3 (discussing HyperCard and noting that “[i]n the future, it will probably be possible for non-programmers to use [object-oriented] techniques to create their own software without having to learn a line of programming code”); Telephone Interview with Christopher R. Bingham, *supra* note 58.

98. Clapes et al., *supra* note 2, at 1533–34.

contain many similar segments of code. Compilation doctrine suggests, however, that a copyrighted program should receive protection for the expression embodied in overall program design.

C. Summary: The Appropriateness of Compilation Protection for Software

The preceding overview reveals several features of copyright's protection of compilations. A protectable compilation possesses independent value derived from bringing together constituent parts that may or may not be separately protectable. A court determining substantial similarity does not consider unprotectable material such as constituent facts, but rather distills a protectable core of selection, arrangement, and coordination expression. The decisionmaker compares only this distinct part of the general work to elements of the accused work to determine whether observed similarity constitutes infringement.⁹⁹

Comparing software to traditional works of compilation suggests two justifications for extending copyright's scheme of protection for compilations to software. First, designing a program entails the same selection and arrangement authorship that receives protection when present in standard works of compilation such as factual directories. Recent cases affirm the principle that compilation expression exists when the resulting work as a whole is greater than the sum of its parts, regardless of whether those parts are protectable or not. Computer programs clearly obtain independent value by virtue of programmers' assembly of particular design elements into a functioning whole.

Second, copyright's doctrinal approach to protecting traditional compilations shares several common elements with *Computer Associates'* approach to protecting software. Compilation analysis carefully distinguishes between unprotectable constituent elements and the integrated whole of selection and arrangement expression. Similarly, abstraction-filtration-comparison identifies unprotectable parts of computer programs by testing for elements dictated by efficiency, demanded by external factors, or taken from the public domain.¹⁰⁰ Like the names and addresses in a telephone directory, such "filtered" elements do not receive protection because they are not original to the work's author.¹⁰¹ These doctrinal similarities underscore the degree of

99. See *Feist Publications, Inc. v. Rural Tel. Serv. Co.*, 499 U.S. 340, 349-50 (1991); see also 3 NIMMER, *supra* note 2, § 13.03[B], at 13-69 to 13-73; cf. 17 U.S.C. § 103(b) (1988) ("The copyright in a compilation . . . work extends only to the material contributed by the author . . . and does not imply any exclusive right in the preexisting material.").

100. See *supra* notes 23-33 and accompanying text. In fact, compilations and software are among the only kinds of works currently subject to this sort of explicit dissection. Cf. 3 NIMMER, *supra* note 2, § 13.03[E], at 13-93 to 13-94, 13-120 to 13-123 (noting that "successive filtering" test has been used mainly for software).

101. For example, public domain elements or elements dictated by external factors such as manufacturer compatibility standards clearly fail an originality test: They owe their origin to someone other than the program's author. Copyright similarly assumes that "ideas" such as efficient programming

similarity between copyright's conceptual understandings of traditional compilations and computer programs, and they suggest the ease with which compilation principles can be applied to software.

Considering software in light of compilation principles emphasizes the puzzling shortcoming of *Computer Associates*. Compilation doctrine says that a work's selection and arrangement must be considered separately from other elements; the Second Circuit's focus on breaking programs into abstracted parts without also considering their selection and arrangement, however, raises the concern that such original expression will not be protected. Indeed, as one commentator has pointed out, "[it] is not clear how *Computer Associates*, after 'filtering out' all that is unprotectable, would deal with the 'selection and arrangement' of those unprotected elements. Is compilation-type authorship ignored, or is it simply a different 'level of abstraction?'"¹⁰²

Computer Associates' ambiguous treatment of compilation expression belies its importance as a creative element of software. One author writes that "[a] program's creativity results primarily from the coordination between and interrelationship among its components. Unprotectable elements may interact with each other in a creative way, just as protectable and unprotectable elements may interact with each other in a creative way."¹⁰³ Although care must be taken to identify areas where programmers have wider or narrower ranges of options, compilation doctrine can play an important role in protecting software. The Second Circuit's test therefore should incorporate compilation doctrine at each stage of abstraction-filtration-comparison.

III. IDENTIFYING AND PROTECTING COMPILATION EXPRESSION DURING ABSTRACTION-FILTRATION-COMPARISON

General support for protecting selection and arrangement expression in software comes from a variety of sources. No court or commentator, however, has discussed how to apply a thorough analysis based on the principles set out in leading compilation cases. This Part considers existing support for protecting expression in software and then directly discusses how to bring principles of compilation protection to bear during abstraction-filtration-comparison analysis.

algorithms or coding conventions are in the public domain for copyright purposes. See *Gates Rubber Co. v. Bando Chem. Indus.*, 9 F.3d 823, 833, 837 (10th Cir. 1993); *Apple Computer, Inc. v. Microsoft Corp.*, 779 F. Supp. 133, 134 (N.D. Cal. 1991) (noting that "there is apparently no principled means of distinguishing the traditional doctrines of merger, functionality, and scenes a faire" from the doctrine that an element is unprotectable because unoriginal). *aff'd*, Nos. 93-16867, 93-16869, 93-16883, 1994 WL 506999 (9th Cir. Sept. 19, 1994).

102. Joseph, *supra* note 30, at 143.

103. Lowe, *supra* note 42, at 353, 366-67.

A. Existing Support for Introducing Compilation Analysis

Courts and at least one leading commentator offer some support for protecting compilation expression in software. Professor David Nimmer, a proponent of the filtering approach adopted by *Computer Associates*,¹⁰⁴ cautions that

[i]n performing the filtering, the court should be sensitive to the myriad ways in which copyrightable creativity can manifest itself; the analysis should not proceed mechanically simply by isolating physical elements out of the copyrightable work. . . . [T]he structure and arrangement of [otherwise unprotectable elements] may evidence plaintiff's originality, in which case such elements may not be eliminated from the analysis.¹⁰⁵

A few courts also have suggested that compilationlike expression should be protected in the software context. In *Engineering Dynamics, Inc. v. Structural Software, Inc.*, the Fifth Circuit, conducting an abstraction-filtration-comparison analysis, protected selection and arrangement expression in the input formats used to provide data to a computer program.¹⁰⁶ In *Atari Games Corp. v. Nintendo of America Inc.*, the Federal Circuit identified a program's "creative organization and sequencing" as a protectable element under abstraction-filtration-comparison. "At a minimum, [the copyright holder] may protect under copyright the unique and creative arrangement of instructions in [the copyrighted program]."¹⁰⁷

As a final and especially descriptive example, the district court presiding over Apple Computer's infringement suit against Microsoft noted the potential pitfalls of filtration-type analysis:

Suppose defendant copied plaintiff's abstract painting composed entirely of geometric forms arranged in an original pattern. The alleged infringer could argue that each expressive element (i.e., the geometric forms) is unprotectible under the functionality, merger, scenes a faire, and unoriginality theories and, thus, all elements should be excluded prior to the substantial similarity of expression analysis. Then, there would be nothing left for purposes of determining substantial similarity of expression. In this example, elimination of

104. The Second Circuit takes the test directly from Nimmer's treatise. See *Computer Assocs. Int'l v. Altai, Inc.*, 982 F.2d 693, 707 (2d Cir. 1992). Nimmer refers to the test as the "successive filtering method." See generally 3 NIMMER, *supra* note 2, § 13.03[F].

105. 3 NIMMER, *supra* note 2, § 13.03[F], at 13-145 to 13-146. Nimmer does not elaborate further as to either the form such selection and arrangement expression may take or the appropriate scope of protection it should receive.

106. *Engineering Dynamics, Inc. v. Structural Software, Inc.*, 26 F.3d 1335, 1346 (5th Cir. 1994) (citing *Lotus Dev. Corp. v. Borland Int'l*, 831 F. Supp. 223, 231 (D. Mass. 1993)).

107. *Atari Games Corp. v. Nintendo of Am. Inc.*, 975 F.2d 832, 840 (Fed. Cir. 1992).

“unprotectible” elements would result in a finding of no copyright infringement, which would be clearly inconsistent with the copyright law’s purpose of providing incentives to authors of original works.¹⁰⁸

This analysis captures the element missing from the Second Circuit’s formulation.¹⁰⁹

Despite these examples of courts recognizing design-level expression, on the whole judicial efforts remain tentative and incomplete. *Atari v. Nintendo* remains the only decision to identify and protect compilation-type expression in program design as part of an abstraction-filtration-comparison analysis, and that opinion offers only an attenuated identification of “creative organization and sequencing” instead of an application of compilation principles. The Fifth Circuit’s express recognition of selection and arrangement expression in *Engineering Dynamics* does offer a positive sign for the application of compilation doctrine in the software context, but that case dealt with input formats, not program design. On the whole, courts adopting the *Computer Associates* test have applied ad hoc abstraction analyses that fail to identify compilation expression.¹¹⁰ The continuing doctrinal development of abstraction-filtration-comparison under established copyright principles¹¹¹ requires the identification and protection of selection and arrangement expression.

B. *Incorporating Compilation Principles During Abstraction, Filtration, and Comparison*

The Second Circuit’s test sets out a standardized process for bringing a range of legal principles to bear on complex software works. Although compilation principles could be introduced into software infringement analysis

108. *Apple Computer, Inc. v. Microsoft Corp.*, 779 F. Supp. 133, 136 (N.D. Cal. 1991), *aff’d*, Nos. 93-16867, 93-16869, 93-16883, 1994 WL 506999 (9th Cir. Sept. 19, 1994).

109. The court in *Apple v. Microsoft* was not applying the Second Circuit’s substantial similarity test, but rather ruling on an early Microsoft motion to exclude unprotectable elements from later substantial similarity analysis. *Id.* at 134. In the *Computer Associates* litigation, the Second Circuit accepted the district court’s abstraction analysis, in which the identified element closest to “selection and arrangement” was the program’s general “organizational chart.” The court dismissed this level of abstraction as unprotectable without reference to any particular justification or points of analysis. See *Computer Assocs. Int’l v. Altai, Inc.*, 982 F.2d 693, 715 (2d Cir. 1992) (citing *Computer Assocs. Int’l v. Altai, Inc.*, 775 F. Supp. 544, 562 (E.D.N.Y. 1991)); see also Joseph, *supra* note 30, at 137 (criticizing *Computer Associates’* abstraction analysis as inadequate and overly simplistic).

110. See, e.g., cases cited *supra* note 35; see also Lowe, *supra* note 42, at 369; Velasco, *supra* note 36, at 285; Clapes & Daniels, *supra* note 43, at 13 (arguing that *Computer Associates’* “court completely discarded . . . the principle that a selection, arrangement and organization of unprotected elements may itself be protected”).

111. See *Gates Rubber Co. v. Bando Chem. Indus.*, 9 F.3d 823, 839–42 (10th Cir. 1993) (discussing test’s role in “natural evolution” of copyright laws).

in a variety of ways,¹¹² this Note advocates an approach that expressly addresses compilation expression at each phase of abstraction-filtration-comparison. At each stage, principles of compilation protection can help courts more effectively identify and protect the creative expression contained in software.

1. *Identifying Compilation Expression as a Standard Program Element During Abstraction*

Protecting selection and arrangement expression should be part of infringement analysis from the outset. Identifying compilation expression as a separate program element during abstraction begins this process. The idea of a compilation level of abstraction satisfies both underlying abstractions theory as well as the practical goal of identifying program elements. Compilation expression exists when a work's constituent elements are selected, arranged, or coordinated in a creative fashion. After identifying levels of abstraction that correspond to various program parts, a court can quite naturally also assess the more general level of abstraction that comprises their particular selection, arrangement, and coordination.¹¹³ At a practical level, the design-level description of a program is just as much a part of the program as are data structures, subprograms, algorithms, and other elements. Identifying compilation expression as a distinct level of abstraction makes theoretical sense and focuses judicial attention on an important program element.

As a second benefit, identifying compilation expression offers courts at least one standard abstraction element found in all computer programs. A number of authors note that courts lack guidance when conducting abstraction analyses.¹¹⁴ Although abstraction must be ad hoc to an extent because of the inherent complexity of computer programs, grounding program dissection on the identification of at least a few common elements provides a much-needed degree of standardization.

Finally, identifying compilation expression at the first stage of abstraction-filtration-comparison facilitates the application of compilation principles at later

112. For example, one commentator suggests that courts employ compilation principles in safety-net fashion, recommending the addition of a "reincorporating compilations" step between filtration and comparison as a way to catch otherwise overlooked expression. *See Velasco, supra* note 36, at 285–86. Also, although abstraction-filtration-comparison offers many benefits—both as a general method of analysis and as a mechanism for applying compilation doctrine—and remains the leading test, circuits not adopting the Second Circuit's filtration method can and should protect selection and arrangement expression in software under principles of compilation protection.

113. For the sake of simplicity, this discussion refers to one general compilation level of abstraction for a given program. In practice, however, a large program could contain many separately identifiable levels of compilation expression. For example, the selection, arrangement, and coordination of a given module or subprogram could be found to be protectable while that of others may be found not to be. *See Gates Rubber*, 9 F.3d at 835 (noting that "[s]tructure exists at nearly every level of a program").

114. *See supra* note 42 and accompanying text.

stages. Filtering doctrines such as merger and *scènes à faire* should be applied to compilation elements, and any remaining expression should be compared to elements of the accused work based on an appropriate comparison standard. Making the identification of selection, arrangement, and coordination expression a standard part of abstraction provides a starting point for incorporating compilation doctrine throughout the *Computer Associates* test.

2. Filtering Compilation Elements

At the filtration stage, a court decides whether each of the program elements identified during abstraction qualifies for protection by applying doctrines developed to prevent copyright protection from extending to ideas.¹¹⁵ After identifying selection and arrangement elements, courts should evaluate the range of design options available to the programmer and then apply principles of merger and *scènes à faire*. Applying these filtering principles is a crucial step required to ensure that protecting selection and arrangement expression does not have the effect of inappropriately expanding the scope of protection for software.¹¹⁶

In *Kregos v. Associated Press*, the Second Circuit considered how the merger doctrine applies to functional compilations.¹¹⁷ The court held that a particular selection and arrangement does not merge with its underlying function as long as the author's subjective judgment, rather than some objective requirement, influences a particular selection and arrangement.¹¹⁸ *Kregos* therefore instructs courts to focus on the degree of subjectivity involved in selecting, arranging, and coordinating the elements of a program.

As discussed in Part II, many aspects of software design are highly subjective. Program designers face a wide range of viable options in structuring complex software, and only the simplest programs could be said objectively to require one particular selection and arrangement of elements. The process of coding a particular part of a program, during which a programmer may well have few options, must be distinguished from the much more creative and subjective act of designing a program at a conceptual level. Consequently, merger should deny protection to compilation expression only in unusual cases involving small, simple programs.

Similarly, in *Bellsouth Advertising*, the Eleventh Circuit employed an implicit *scènes à faire* analysis as one justification for denying protection to the particular arrangement and coordination of a yellow pages, noting that the

115. See *supra* notes 8–16 and accompanying text.

116. See *Gates Rubber*, 9 F.3d at 836–37; *Atari Games Corp. v. Nintendo of Am. Inc.*, 975 F.2d 832, 839 (Fed. Cir. 1992).

117. 937 F.2d 700 (2d Cir. 1991); see *supra* notes 75–77 and accompanying text.

118. 937 F.2d at 706–07.

work's structure reflected "standard industry practices."¹¹⁹ Program design typically follows certain standard conventions, such as using a "main" subprogram or putting all input-output routines in one library. To the extent a software designer selects, arranges, and coordinates program elements according to standard conventions or industry norms, a *scènes à faire* filter correctly denies protection.

3. *Compilation-Doctrine Guidance for Conducting Comparison*

After abstraction and filtration focus a court's analysis, comparison addresses the ultimate question of infringement liability: whether the accused work has copied enough protected expression to become "substantially" similar. Surprisingly, courts have failed to address the degree of copying required to constitute substantial similarity between structural elements of two computer programs.¹²⁰ A compilation-based analysis offers guidance for developing a standard of comparison for software infringement that follows established copyright principles.

The degree of similarity required for infringement directly determines the scope of protection for a given type of work: The higher the degree of similarity required, the easier it is for an admittedly copying work to avoid infringement, and the weaker the protection for the copyrighted work. For example, poems receive a broader scope of protection than phone books because a poem may be infringed by a work copying only a line or two, whereas a phone book will only be infringed by a work copying entire categories of listings.¹²¹ As this example suggests, the degree of similarity required for two works to be substantially similar varies widely according to the type of work and the nature of the protected interest.

The principle dictating different standards for different kinds of works is that infringement requires *qualitative* similarity.¹²² A set of yellow pages cannot be said to be similar in any qualitative sense to another listing that copies only isolated information; in contrast, a poem that copies a line of stylized verse may well be qualitatively similar to the original. No court has yet explored qualitative similarity in the context of computer software, and

119. *Bellsouth Advertising & Publishing Corp. v. Donnelley Info. Publishing, Inc.*, 999 F.2d 1436, 1444 (11th Cir. 1993), *cert. denied*, 114 S. Ct. 943 (1994).

120. *But cf.* *Apple Computer, Inc. v. Microsoft Corp.*, Nos. 93-16867, 93-16869, 93-16883, 1994 WL 506999, at *10-*11 (9th Cir. Sept. 19, 1994) (requiring that selection and arrangement of infringing graphical user interface must be "virtually identical"); *Mitek Holdings, Inc. v. Arce Eng'g Co.*, No. 91-2629-CIV-MOORE, 1994 WL 515922, at *16 (S.D. Fla. Sept. 9, 1994) (using "bodily appropriation" standard) (citing *Apple Computer, Inc. v. Microsoft Corp.*, 821 F. Supp. 616, 623 (N.D. Cal. 1993), *aff'd in part and rev'd in part on other grounds*, 1994 WL 506999).

121. See cases cited *supra* note 20.

122. See *Worth v. Selchow & Righter Co.*, 827 F.2d 569, 570 n.1 (9th Cir. 1987), *cert. denied*, 485 U.S. 977 (1988); *Whelan Assocs. v. Jaslow Dental Lab., Inc.*, 797 F.2d 1222, 1245 (3d Cir. 1986), *cert. denied*, 479 U.S. 1031 (1987); 3 NIMMER, *supra* note 2, § 13.03[A], at 13-51 to 13-52.

software's many unique characteristics prevent basing a standard on direct analogy to some other kind of work. Copyright's traditional "personality-based" conception of copyright, however, offers a helpful starting point for giving some content to qualitative similarity for computer programs.

Professor Jane Ginsburg identifies Justice Holmes' opinion in *Bleistein v. Donaldson Lithographing Co.*¹²³ as the wellspring of what she terms the "personality-based" understanding of copyright:

Holmes found the source of authors' claims to protection in each creator's unique individuality: "[The work] is the personal reaction of an individual upon nature. Personality always contains something unique. It expresses its singularity even in handwriting, and a very modest grade of art has in it something irreducible, which is one man's alone. That something he may copyright unless there is a restriction in the words of the act."¹²⁴

Under this understanding of copyright, authorial presence, not the aesthetic merit or intended purpose of a work, forms the basis of protection.¹²⁵ Thus, a useful way to gauge what constitutes qualitatively significant copying for a given kind of work is to distinguish between works of "high authorship . . . such as novels and narrative histories" and works of "low authorship . . . such as telephone directories and compilations of stock quotations."¹²⁶

This distinction reflects the principle that copyright protects works to the degree that they "reflect the personalities of their authors or, at the very least, embody their creator's subjective choices in the selection or arrangement of material."¹²⁷ High-authorship works such as poems, plays, or novels receive broad protection in the form of a substantial similarity standard making only modest similarity qualitatively significant. Authorial presence dominates these works and they receive an accordingly high level of protection.

In contrast, low-authorship works receive less protection because authorial personality exists, if at all, only in the integrated whole of selection and arrangement. A work must contain a highly similar selection and arrangement

123. 188 U.S. 239 (1903).

124. Jane C. Ginsburg, *Creation and Commercial Value: Copyright Protection of Works of Information*, 90 COLUM. L. REV. 1865, 1882 (1990) (quoting *Bleistein*, 188 U.S. at 250).

125. See *id.* at 1866. Indeed, Holmes' *Bleistein* opinion "upheld the copyrightability of a functional work, commercial art, which at the time suffered the kind of opprobrium sometimes cast today on a modern functional work—computer programs." *Id.* at 1888.

126. See *id.* at 1870. Ginsburg's main argument—that copyright law should acknowledge industrious collection as the basis for copyright in low-authorship works, protecting this low-authorship interest with a system of compulsory licensing—has apparently been foreclosed by the Supreme Court's rejection of "sweat of the brow" authorship. See *Feist Publications, Inc. v. Rural Tel. Serv. Co.*, 499 U.S. 340, 354, 359–60 (1991). However, her distinction between the scope of protection afforded high-authorship and low-authorship works is useful.

127. Ginsburg, *supra* note 124, at 1867; see also Raskind, *supra* note 83, at 335.

of elements to infringe a low-authorship work such as a yellow pages,¹²⁸ a standard requiring that an infringing work not only copy elements from the original but also arrange them in the same way. For example, in *Key Publications* the court found no infringement because the accused directory did not duplicate any one entire category of businesses; instead, no category in the accused directory contained more than a few copied entries.¹²⁹

Gauging the degree of authorial presence in a work requires careful consideration of the way that work is created. Indeed, while a phone book provides a paradigmatic example of a low-authorship work afforded little protection by copyright, all works containing compilation expression should not be characterized as low-authorship works.¹³⁰ Highly imaginative selection and arrangement expression may exhibit significant authorial presence that should receive more protection than the minimally creative selection and arrangement of a factual directory.

The structure and design of computer programs reflect significant authorial presence. The wider the range of options available to an author, the greater her ability to impart her personality and style to the work;¹³¹ as discussed, program design offers a broad range of structural options, especially as compared with the more limited choices available during coding.¹³² Program design therefore is the primary conduit through which programmers impart authorial presence to their programs.¹³³ One court agrees that

“[t]here 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

128. For example, the court in *Key Publications* held that a directory of Chinese businesses was not infringed by another work that copied a total of 17% of its listings. *Key Publications, Inc. v. Chinatown Today Publishing Enters.*, 945 F.2d 509, 515–16 (2d Cir. 1991); see also *Worth v. Selchow & Righter Co.*, 827 F.2d 569, 570, 573 (9th Cir. 1987) (holding that copying 27.9% of facts in copyrighted trivia encyclopedia by Trivial Pursuit game cards does not infringe because arrangement of facts was different), *cert. denied*, 485 U.S. 977 (1988).

129. See *Key Publications*, 945 F.2d at 517; see also *Worth*, 827 F.2d at 570, 573; Odozynski, *supra* note 77, at 497.

130. Note that this point goes only to the degree of copying of a work's selection and arrangement expression required for infringement; while an infringing yellow pages may have to present all listed dentists in exactly the same order to infringe, a work that infringes a high-authorship compilation may only generally have to follow the arrangement of the original. Selection and arrangement still define the nature of the protected interest; the crucial issue is how similar the selection and arrangement of two works must be for one to infringe the other.

131. See *Worth*, 827 F.2d at 572 (noting that “[f]ictional works . . . , which may be expressed with ‘infinite variations,’ enjoy a broader protection; a verbatim copy or close paraphrase is not a necessary element to establish infringement”); *Landsberg v. Scrabble Crossword Game Players, Inc.*, 736 F.2d 485, 488 (9th Cir.) (“Factual works are different. Subsequent authors wishing to express the ideas contained in a factual work often can choose from only a narrow range of expression Therefore, similarity of expression may have to amount to verbatim reproduction or very close paraphrasing before a factual work will be deemed infringed.”), *cert. denied*, 469 U.S. 1037 (1984); Ginsburg, *supra* note 124, at 1867, 1901 (emphasizing subjectivity of author's selection and arrangement as key to high-authorship status).

132. See *supra* notes 92–98 and accompanying text.

133. See Clapes et al., *supra* note 2, at 1535–36.

program in exactly the same way (except perhaps in the case of the most simple program). . . . The possibility of two programmers creating identical programs [was compared by the testifying expert] to the likelihood of a monkey sitting at a typewriter producing Shakespeare."¹³⁴

The compilation expression embodied in program structure therefore should be considered high-authorship in nature for purposes of defining a standard of substantial similarity.¹³⁵

This analysis indicates that the standard used to evaluate whether an accused work has "substantially" copied protected selection and arrangement expression—that is, infringed the program's copyright¹³⁶—should be satisfied by a moderate degree of similarity, perhaps more than that required of purely literary works, but less than that required of low-authorship works such as phone books. Courts typically provide broader protection to high personality-content compilations, such as textbooks or anthologies, than to low-authorship directories.¹³⁷ Software should be protected similarly. As Nimmer writes,

[E]ven a quantitatively small amount of copied material may be sufficiently important to the operation of plaintiff's program to justify a finding of substantial similarity. For instance, a small portion of the structure or code of a program may nonetheless give it distinctive features or may make the program especially creative or desirable. In

134. *Id.* at 1539–40 (quoting *Apple Computer, Inc. v. Mackintosh Computers, Ltd.*, No. T-1232-84, No. T-1235-84, slip op. at 3 (Can. Fed. Ct. Apr. 29, 1986)).

135. *But cf.* *Apple Computer, Inc. v. Microsoft Corp.*, Nos. 93-16867, 93-16869, 93-16883, 1994 WL 506999, at *10–*11 (9th Cir. Sept. 19, 1994) (holding graphical user interfaces more analogous to fact-based works than to artistic works for purposes of defining comparison standard).

136. Recall that abstraction-filtration-comparison—as well as this Note—deals only with the question of nonliteral infringement of programs. Literal infringement, such as line-for-line copying of parts of a program's code, is a different question with different applicable standards. In terms of ultimate legal impact, however, infringement is infringement, whether literal or nonliteral. Thus, this Subsection argues that copying relatively significant portions of the selection and arrangement expression embodied in program design should constitute infringement of a program.

137. See 1 NIMMER, *supra* note 2, § 2.11, at 2-172.26 to 2-172.27 (listing cases). *But see* Ginsburg, *supra* note 124, at 1905–07 (noting that courts applying "sweat of the brow" rule—since repudiated by Supreme Court in *Feist*—sometimes afforded more protection to low-authorship directories than to high-authorship factual works such as histories). Also, note that courts restricting the scope of copyright in historical or other factual works tend to focus on the general idea/expression dichotomy for the entire work, allowing broad copying of ideas such as historical theories or strategies, but expressly distinguishing this inquiry from the question of infringement of the work's selection and arrangement. See, e.g., *Landsberg v. Scrabble Crossword Game Players, Inc.*, 736 F.2d 485, 488–89 (9th Cir.), *cert. denied*, 469 U.S. 1037 (1984). *But see* *Hoehling v. Universal City Studios, Inc.*, 618 F.2d 972, 978 (2d Cir.) ("[B]road latitude must be granted to subsequent authors who make use of historical subject matter, including theories or plots '[T]here cannot be any such thing as copyright in the order of the presentation of the facts, nor, indeed, in their selection.'" (quoting *Myers v. Mail & Express Co.*, 35 C.O.Bull. 478, 479 (S.D.N.Y. 1919))), *cert. denied*, 449 U.S. 841 (1980). Nimmer contends that *Hoehling* is only correct to the degree that it says that a particular interpretation of facts—as manifested by a particular selection and arrangement of facts—is not protectable expression. See 1 NIMMER, *supra* note 2, § 2.11[D], at 2-172.27

such a case, a finding of substantial similarity would be appropriate.¹³⁸

In other words, an infringing work need not copy virtually all of a copyrighted program's structure, nor precisely follow the selection and arrangement. Instead, infringement liability should attach for copying any significant integrated portion of program structure in a way that suggests clear parallels to the original's selection and arrangement.¹³⁹

The similarity required of an infringing work's selection and arrangement remains, in the end, a question of line drawing on a case-by-case basis.¹⁴⁰ The "moderately high" standard advocated in this Subsection obviously can serve as only a general guide. Nonetheless, although it is not possible to set out bright-line rules of protection for complicated software works, compilation principles offer much-needed grounding for courts faced with cases of alleged software infringement. As one court has suggested, *Computer Associates'* failure to elaborate a principle of comparison leaves the test with "the real potential to eviscerate the application of the prevailing substantial similarity test" for literary works generally¹⁴¹—namely, that "quantity plays a minor role in relation to quality."¹⁴² Identifying compilation elements as a standard part of abstraction, then bringing relevant principles of compilation protection to bear during filtration and comparison, can help ensure that the Second Circuit's formulation provides appropriate protection for software.

138. 3 NIMMER, *supra* note 2, § 13.03[F], at 13-147 to 13-148 (footnotes omitted).

139. In fact, dictum from the Second Circuit's recent *Key Publications* decision supports this kind of protection even for low-authorship works. See *Key Publications, Inc. v. Chinatown Today Publishing Enters.*, 945 F.2d 509, 517 (2d Cir. 1991) ("If the [accused work] had exactly duplicated a substantial designated portion of [the copyrighted work]—for example, all its listings of professionals such as medical doctors, lawyers, accountants, engineers, and architects, an infringement action would succeed.").

140. Consider, for example, a hypothetical case involving an infringement suit by the owner of a small program that performs financial calculations against a large software package that performs a range of functions, from word processing to data tracking to financial management. On one hand, a fundamental principle of copyright is that the noninfringing portion of an accused work is irrelevant to substantial similarity analysis; a defendant cannot avoid infringement simply by adding volumes of original material to an otherwise infringing copy. See *Worth v. Selchow & Righter Co.*, 827 F.2d 569, 570 n.1 (9th Cir. 1987), *cert. denied*, 485 U.S. 977 (1988); *Sheldon v. Metro-Goldwyn Pictures Corp.*, 81 F.2d 49, 56 (2d Cir.) ("No plagiarist can excuse the wrong by showing how much of his work he did not pirate."), *cert. denied*, 298 U.S. 669 (1936); 1 NIMMER, *supra* note 2, § 13.03[A], at 13-51. However, courts have hinted in recent compilation cases that the addition of new material by an accused work may indicate a different, and hence nonsubstantially similar, principle of selection and arrangement. See *Key Publications*, 945 F.2d at 516 (noting that accused directory contained 500 listings not found in copyrighted directory in explaining finding of no infringement); *Kregos v. Associated Press*, 937 F.2d 700, 710 (2d Cir. 1991) (suggesting that addition of extra statistics by accused work "may well insulate the competitor from a claim of infringement"). Distinguishing between seeking to disguise an infringing work by adding new material and creating a noninfringing work based on an original principle of selection and arrangement merely inspired by another work can be resolved only on a case-by-case basis.

141. *Gates Rubber Co. v. Bando Am., Inc.*, 798 F. Supp. 1499, 1516 (D. Colo. 1992), *vacated in part and aff'd in part sub nom. Gates Rubber Co. v. Bando Chem. Indus.*, 9 F.3d 823 (10th Cir. 1993).

142. *Id.* at 1518.

IV. PROTECTING OBJECT-ORIENTED SOFTWARE UNDER ABSTRACTION-FILTRATION-COMPARISON

Parts II and III propose that courts apply compilation principles during abstraction-filtration-comparison in order to protect the compilation expression found in all programs. The emergence of a new kind of software makes this modification particularly important. Whereas most traditionally designed programs contain other protectable elements in addition to selection and arrangement expression, compilation authorship is the source of much or all of the protectable expression in object-oriented programs. Incorporating compilation principles therefore will prepare courts for the coming wave of technological change in software cases.

A. *Introduction to Object-Oriented Design*

1. *Traditional Design Principles*

Under the traditional top-down approach, a programmer divides the general programming task into more manageable subprograms, then creates interfaces through which the subprograms communicate. For example, a programmer may write a "function"¹⁴³ that calculates the value of a mathematical formula with two variables. This subprogram's interface, or "parameter list," consists of the two input values. After writing the function, the programmer simply "calls" it to solve the formula for two particular inputs anywhere else in the program. The most important part of calling a subprogram is correctly communicating input data to it through its "parameter list" interface. As one writer explains,

[c]ommunication 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.¹⁴⁴

Although listing parameters in proper order seems simple enough, for large programs the separation of procedures from data caused by this procedure-centered conception of software becomes a serious constraint. A project may

143. A "function" is one kind of subprogram. See *supra* note 54.

144. Clapes et al., *supra* note 2, at 1526. Computers are simply machines that process digital logic. They therefore are incapable of "knowing" that the person's address came second, not first. For example, consider a procedure that draws shapes on the screen and has a parameter list that expects input in the form (location, size). If the caller intends to draw a square at location 5 that is 3 units wide on a side but mistakenly provides inputs in the order (3, 5), the procedure will draw the square at location 3 with sides 5 units long. See GARLAND, *supra* note 54, at 141.

require fifty engineers, each writing hundreds of subprograms, with each subprogram requiring many parameters. Defining parameters is an important and complicated task, and engineers repeatedly revise interface decisions as a project evolves. This revision process can be extremely complex because any subprogram can be called by any other part of the program, and subprograms performing basic functions could be called hundreds of times. A single change to the parameter list of a single subprogram therefore "ripples" throughout the rest of the program, requiring changes to every invocation of that procedure. The schedule-wrecking consequences of this effect mean that developing a complete, stable set of interfaces is a major challenge during traditional software development.¹⁴⁵

Top-down design's emphasis on procedures rather than data also creates other problems. Interface-dependent programs force software developers to reinvent the wheel often, rewriting the same code to mesh with slightly different data sets. Development speed and quality diminish proportionally to the complexity of interfaces. Programmers cannot exploit natural relationships among data types.¹⁴⁶ Top-down design provides many important benefits,¹⁴⁷ but its procedure-oriented outlook also imposes serious limitations.

2. *The Advantages of Object-Oriented Design*

Object-oriented design realizes the advantages of moving toward a *data-oriented* model of programming. Object-oriented programs minimize the problem of interface interdependency by "encapsulating" data within the procedures that operate on them. Instead of allowing individual data elements to exist within the general program, data exists only within independent software objects. The result is that large programs become less intricately interrelated, facilitating easier construction and incremental modification.¹⁴⁸ One author observes that "[t]he beauty of the object-oriented model is that . . . internal changes can be made with absolutely no effect on any other part of the

145. Telephone Interview with Jeffrey R. Cobb, Software Engineer, Apple Computer, Inc. (Aug. 5, 1994).

146. For example, the steps required to draw squares, rectangles, and other parallelograms are very similar. The inflexibility of interfaces in top-down programs, however, would make it very difficult to design a general procedure to draw all of these shapes using the same code.

147. Top-down design is an example of "structured programming." Structured programming's practice of breaking a program into subprograms enables multiple programmers to coordinate work on large projects more effectively; once the program's general structure of subprograms is established, different programmers can work on subprograms independently. Also, top-down design's use of generalized code improves reliability. The goal of functions, procedures, and modules is to provide functional code that can be used repeatedly over a range of inputs; reuse of code improves reliability because, once tested, a subprogram can be counted on to function properly. Finally, structured design makes programs more maintainable and adaptable. See generally GARLAND, *supra* note 54, at 130-36; Beutel, *supra* note 42, at 6-8.

148. See Barkan, *supra* note 53, at 320-21.

program.”¹⁴⁹ This eliminates the “ripple effect” generated by changing small parts of highly interdependent interfaces in traditional top-down programs.

Understanding object-oriented programming as a model based on independent software building blocks suffices for the purposes of a general legal analysis. Object-oriented languages do much more, however, than facilitate the parceling of software into reusable parts. For example, object-oriented languages’ implementation of “classes” and “inheritance”—properties that allow programmers to design hierarchies of objects that share common properties and functions—facilitates reusable, incrementally improvable code. Other object-oriented techniques allow objects to change classes as a program executes, allowing even greater generalization of code and further reducing complexity.¹⁵⁰ As object-oriented languages become more widely adopted, academic and commercial research efforts promise to further develop the object-oriented programming model.

B. *Copyright Protection for Object-Oriented Software*

Traditional design methodologies such as top-down programming require a programmer to rewrite each new program from scratch, engaging in extensive problem decomposition to create modules and subprograms. Many observers note that this design process lends itself to copyright’s traditional abstractions analysis.¹⁵¹ Indeed, courts’ exclusive reliance on the top-down model for understanding programming helps explain *Computer Associates’* focus on dissecting programs. Because the top-down programmer starts from the ground up with each new program, her software product may contain a wide range of protectable expression at each level of abstraction. Any sense of overall design quickly fades before the need to identify and analyze these various elements.

Yet this focus on dissection and filtration suggests that object-oriented software would receive virtually no protection. Although one benefit of object-oriented programming is that software objects can be widely distributed and reused, through either the public domain or licensing of software object libraries,¹⁵² any software objects in a program taken from public sources would be unprotectable because they would not be original to the work’s author. Second, object-oriented programming encourages programmers to refine and standardize software objects. Courts should, however, deny protection to

149. *Id.* at 324.

150. “Polymorphism” and “dynamic binding,” two distinguishing features of object-oriented languages, allow such run-time changes in data types. See Barkan, *supra* note 53, at 325–34 and sources cited therein for further explanation of object-oriented design. See also KAMIN, *supra* note 57, at 273, 343; Magid, *supra* note 55, at 3.

151. See *supra* note 25 and accompanying text.

152. See Barkan, *supra* note 53, at 324.

objects that accomplish standard tasks, model real-world behavior, or embody optimally efficient methods.¹⁵³ The conflict between the characteristics of object-oriented programming and basic principles of copyright lead one author to conclude that abstraction-filtration-comparison, at least as currently applied, provides almost no protection for object-oriented software.¹⁵⁴

The doctrinal modification advocated in Parts II and III addresses this shortcoming. Although both top-down and object-oriented designers engage in compilation authorship, the process of selecting, arranging, and coordinating preexisting material could not be more apparent than in the object-oriented context. Object-oriented designers select objects from available libraries and public domain materials, and arrange and coordinate those objects to accomplish a programming goal. In fact, like a low-authorship factual directory, many object-oriented programs—for example, a program constructed entirely of public domain objects—may exhibit creative expression *only* at the level of selection and arrangement. On the whole, object-oriented software may receive less protection than traditional software, but courts must protect the “thin” copyright interest in the selection and arrangement of objects. Allowing an appropriately narrow scope of protection for object-oriented software, one limited mainly to the protection of selection, arrangement, and coordination expression, is consistent with copyright’s mandate,¹⁵⁵ and with the purposes of object-oriented programming as well: encouraging creative expression that facilitates the efforts of future software authors.

V. CONCLUSION

*[W]e are cognizant that computer technology is a dynamic field which can quickly outpace judicial decision making. Thus, in cases where the technology in question does not allow for a literal application of the procedure we outline below, our opinion should not be read to foreclose the district courts of our circuit from utilizing a modified version.*¹⁵⁶

*The race to develop successful “object oriented” software is heating up.*¹⁵⁷

Much of the creative authorship in software involves selecting and arranging program elements into a functioning software whole. Although abstraction-filtration-comparison provides a useful framework for software

153. See *id.* at 346–51.

154. *Id.* at 354–55. Barkan’s proposed solution is to protect object-oriented software through the patent system. *Id.* at 358–64.

155. See *Sony Corp. of Am. v. Universal City Studios, Inc.*, 464 U.S. 417, 432 (1984) (“The immediate effect of our copyright law is to secure a fair return for an ‘author’s’ creative labor. But the ultimate aim is, by this incentive, to stimulate artistic creativity for the general public good.” (quoting *Twentieth Century Music Corp. v. Aiken*, 422 U.S. 151, 156 (1975))).

156. *Computer Assocs. Int’l v. Altai, Inc.*, 982 F.2d 693, 706 (2d Cir. 1992).

157. Trumbull, *supra* note 58, at A11.

infringement analysis, courts should address the test's failure to consider compilation expression. Of course, developing appropriate schemes of protection for rapidly changing, high-technology works is perhaps modern copyright law's most difficult task. Courts must address this challenge by drawing analogies to conceptually similar works, by applying fundamental copyright principles, and above all by maintaining doctrinal flexibility. Comparing software to traditional works of compilation and considering the principles used to protect compilations strongly suggest that courts should protect the selection and arrangement of computer programs. Moreover, the increasing importance of object-oriented software is a technological change with significant implications for software copyright law. Applying principles of compilation protection to computer program structure offers a response that will appropriately protect an important form of software-based expression.