1 The Interoperability Debate

1.1 Introduction

We live in an interoperable world. Computer hardware and software products manufactured by different vendors can exchange data within local networks and around the globe via the Internet. Competition enabled by interoperability has led to innovation and lower prices, and this has placed extraordinary computing capacity in the hands of ordinary users.

This interoperable world represents a dramatic change from the computing environment of the 1970s. In those days, once a company purchased a computer system, the company was essentially "locked in" to that system: the system was not compatible with the products manufactured by other companies, and the conversion costs were high. Although "locking in" was extremely profitable for dominant vendors, such as IBM, competitors and users suffered from high prices, indifferent service, limited choice, and slow innovation.

Many factors have contributed to the transition from the locked-in environment of the 1970s to today's interoperable world, including consumer demand, business strategy, government policy, and the ideology of technologists. One factor that is often overlooked is the evolution of copyright law over the past 30 years. Because computer programs are copyrightable, copyright law determines the rules for competition in the information-technology industry. For this reason, there has been a 30-year debate concerning the application of copyright to software.

The parties to the debate are the dominant vendors (who want to lock in users and lock out competitors) and the developers of interoperable software products (who want to compete with the dominant vendors). The debate has occurred in courts in North America, Europe, and the Pacific Rim; in the U.S. Congress and the European Parliament; and in law schools, think tanks, and legal publications. It has centered on two related matters:

the scope of copyright protection for program elements necessary for interoperability and the permissibility of the reverse engineering necessary to uncover those elements in a competitor's program. Underlying these two matters is the central competitive issue confronting the software industry: Could one firm prevent other firms from developing software products that interoperated with the products developed by the first firm?

In 1995 we published *Interfaces on Trial: Intellectual Property and Interoperability in the Global Software Industry*. That 370-page book closely examined the interoperability debate in the United States, the European Union, and Japan. Its first chapter provided a general overview of computer technology, the structure of the computer industry, and the significance of intellectual-property protection to innovation and competition in the industry. Its second chapter reviewed the fundamentals of intellectual-property law, focusing on copyright and on the application of copyright to software. Its third chapter tackled the first controversy in the interoperability debate: copyright protection for interface specifications. It explored the early missteps in the 1980s by the U.S. Court of Appeals for the Third Circuit, and the Second Circuit's[1] 1992 landmark decision (rejecting the earlier rulings) in *Computer Associates v. Altai*. Its fourth chapter treated the second controversy in the interoperability debate: the permissibility of software reverse engineering. It reviewed the resolution of this controversy by the Ninth Circuit in *Sega v. Accolade*. The book then addressed the development of the EU Software Directive (in chapter 5) and the interoperability debate in Japan (in chapter 6).

To our pleasant surprise, *Interfaces on Trial* ran through three printings; to our great relief, it received very favorable reviews.[2]

At the time we published *Interfaces on Trial*, we thought that the interoperability debate was largely over. In the United States, several appellate courts had followed *Computer Associates* and *Sega*, so those decisions'

1. The U.S. federal court system has three levels: the federal district courts (which conduct trials), the intermediate U.S. Courts of Appeals (which hear appeals from the district courts), and the U.S. Supreme Court (which hears appeals from the U.S. Courts of Appeals and from state supreme courts). Most of the judicial decisions discussed in this book were issued by the U.S. Courts of Appeals. These courts are organized in eleven regional circuits. In addition, the U.S. Court of Appeals for the Federal Circuit has exclusive jurisdiction over patent appeals. In this book, "a decision by the Ninth Circuit," for example, means a decision by the U.S. Court of Appeals for the Ninth Circuit.

2. Robbie Downing, book review, 3 *International Journal of Law and Information Technology* 198 (1995); book review, 15 *Northwestern Journal of International Law and Busi-*

holdings seemed well entrenched. In the European Union, the member states had implemented the EU Software Directive's reverse-engineering exceptions with little difficulty.

Although the Software Directive ended the interoperability debate in the European Union, the debate continued in the United States and elsewhere. In the U.S., litigation proceeded on both the protectability of interface specifications and the permissibility of reverse engineering. Outside the Third Circuit, courts have issued decisions consistent with *Computer Associates* and *Sega*.

However, two new threats to interoperability emerged in the United States. First, several courts enforced contractual restrictions on reverse engineering, even when the vendors placed the restrictions in "shrinkwrap" or "click-on" licenses for widely distributed consumer software. Second, the World Intellectual Property Organization Copyright Treaty, adopted in December 1996, required signatories to take adequate measures to prevent the circumvention of copy-protection technologies for purposes of infringement. As Congress was implementing this requirement, developers of interoperable software recognized that the broad prohibition Congress was considering would allow dominant firms to frustrate interoperability by placing "locks" on their software. Accordingly, the developers lobbied for and secured an interoperability exception in the Digital Millennium Copyright Act (DMCA).

Significantly, the European Union anticipated both of these issues in its Software Directive, which contains provisions that expressly invalidate contractual restrictions on reverse engineering and that permit the circumvention of technological protection measures for the purpose of performing lawful reverse engineering.

The interoperability debate also continued in the Pacific Rim after 1995. Dominant U.S. companies, with the assistance of the U.S. Trade Representative, vigorously opposed the adoption of reverse-engineering exceptions based on the EU Software Directive in Australia, Hong Kong, Korea, and the Philippines.

This book picks up the story where *Interfaces on Trial* left off. Sections 1.2 and 1.3 of this chapter provide a quick review of the interoperability debate in the European Union and the United States before 1995. Chapter

---

ness 707 (1995); Book review, 20 *New Matter* 35 (1995); Zack Higgens, book review, 9 *Harvard Journal of Law and Technology* 585 (1996); Robert Brookshire, book review, 7 *Law and Policy Book Reviews* 206 (1997).

2 discusses the U.S. copyright cases since 1995 addressing the protectability of interface specifications and the permissibility of reverse engineering, and closes by noting that the executive and legislative branches have finally endorsed this pro-interoperability case law. Chapter 3 looks at the legislative history of the interoperability exception in the DMCA, as well as the interoperability cases decided under the DMCA. Chapter 4 examines the enforceability of contractual restrictions on reverse engineering, including the treatment of this issue in the context of the Uniform Computer Information Transactions Act (UCITA). Chapter 5 reviews the interoperability debate in the Pacific Rim, with stops in Australia, Singapore, Hong Kong, South Korea, and the Philippines. Chapter 6 briefly considers issues that may have more impact on interoperability in the future.

In this book, certain terms have the same meanings as in *Interfaces on Trial*:

- "Interoperability" is synonymous with "compatibility" and has two dimensions: interchangeability and connectability. "Interchangeability" refers to the degree to which one product can substitute for or compete with another product. "Connectability" refers to the degree to which a product can participate in a joint activity with another product.
- "Interface" means a functional characteristic of an element's interaction with other elements of the computer system, i.e., a permissible input, output, or control. This book focuses on interfaces between software and hardware, or between two software elements. This book does not examine user interfaces—that is, the interfaces between users and computers.
- "Interface specifications" are the rules of interconnection between two program elements. An interface specification can have different implementations—e.g., it can be encoded in different ways. A programming language or particular commands can be a form of interface specification.
- "Disassembly" and "decompilation" refer to the translation of machine-readable object code into a higher-level, human-readable format. "Disassembly" is the term usually used in the U.S. legal context; "decompilation" typically is used outside the United States. Accordingly, we will use "disassembly" when discussing the activity in the context of U.S. legal developments, and "decompilation" when referring to the activity in the international policy context.
- "Black-box reverse engineering" means observing the externally visible characteristics of a program as it operates, without looking into the program itself.

These terms, and computer technology generally, are discussed in much greater detail in *Interfaces on Trial*.

The present volume is intended to connect to, and not substitute for, *Interfaces on Trial*. Thus, it does not repeat the earlier volume's background information on computer technology, the structure of the computer industry, intellectual-property law, and the economics of standardization. Additionally, since the publication of *Interfaces of Trial* there has been a profusion of scholarly writings concerning the complex interaction between copyright and digital technology.[3] This book does not attempt to address this vast academic literature. Rather, it provides the second volume of the history of an ongoing legal debate.

Although we attempt to present contentious issues in a balanced manner, the reader should be forewarned that we are hardly objective observers in this debate. Rather, we have devoted significant time and energy over the past 20 years to advocating the views of developers of interoperable software. We believe that the triumph of interoperability will benefit both the information-technology industry and computer users around the world.

1.2 The Interoperability Debate in the European Union before 1995

In 1991, after a vigorous debate (described in detail in *Interfaces on Trial*), the European Union adopted its Software Directive.[4] During the three-year process that led up to the promulgation of the directive, dominant firms, developers of interoperable software, and computer users battled over the protectability of interface specifications and the permissibility of reverse engineering. The directive that emerged from this political process reflects a policy judgment that copyright should not interfere with interoperability. The Software

3. See, e.g., Pamela Samuelson and Suzanne Scotchmer, "The Law and Economics of Reverse Engineering," 111 *Yale Law Journal* 1575 (2002); Peter Menell, "Envisioning Copyright Law's Digital Future," 46 *New York Law School Law Review* 63 (2002–03); Douglas Lichtman, "Property Rights in Emerging Platform Technologies," 29 *Journal of Legal Studies* 615 (2000); Peter Menell, "An Epitaph for Traditional Copyright Protection of Network Features of Computer Software," 43 *Antitrust Bulletin* 651 (fall-winter 1998); Dennis Karjala and Peter Menell, "Applying Fundamental Copyright Principles in *Lotus Development Corp. v. Borland International Inc.*," 10 *High Technology Law Journal* 177 (1995); Pamela Samuelson, Randall Davis, Mitchell Kapor, and Gerald Reichman, "A Manifesto Concerning the Legal Protection of Computer Programs," 94 *Columbia Law Review* 2308 (1994); Andrew Johnson-Laird, "Software Reverse Engineering in the Real World," 19 *University of Dayton Law Review* 843 (1994); Dennis Karjala, "Copyright Protection of Computer Software, Reverse Engineering, and Professor Miller," 19 *University of Dayton Law Review* 975 (1994).

4. Council Directive 91/250/EEC, 1991 O.J. (L 122).

Directive has been implemented by all 27 member states of the European Union, and also by Croatia, Norway, Russia, Switzerland, and Turkey.

Article 5(3) of the Software Directive provides a broad exception from liability for "black-box reverse engineering"—activities such as observing the behavior of a program as it runs, input/output tests, and line traces. Article 6 provides a narrower exception for decompilation. Decompilation or disassembly involves translating machine-readable object code into a higher-level, human-readable form. Article 6 permits decompilation for purposes of achieving interoperability when the information has not previously been made available, when the decompilation is limited to those parts of the program necessary for interoperability, and when the final product created by the reverse engineer does not infringe on the copyright of the original product. There has been extensive debate on exactly what these provisions mean,[5] but to date there has been no copyright litigation concerning article 6.[6]

One particularly enigmatic provision is article 6(1)(b), which requires that "the information necessary to achieve interoperability has not previously been readily available" to the reverse engineer. One commentator has stated that "since the information must be 'readily' available, third parties would have no duty to ask for information if it is not contained in generally available documentation. Nor can it be said that interface information is 'readily' available if the rightholder is only willing to disclose it upon payment of a license fee, since this would undermine the very purpose of limited, but reliable access to interface information."[7] Others have interpreted this provision as requiring the reverse engineer to request the interface information from the developer of the target software before decompilation. The reverse engineer obviously would prefer not to have to make such a request, because the request would alert the first developer to the reverse engineer's business plans and would delay the decompilation.

5. See Jonathan Band and Masanobu Katoh, *Interfaces on Trial: Intellectual Property and Interoperability in the Global Software Market* (Westview, 1995), at 246–255. The governmental bodies of the European Union were lobbied heavily concerning the Software Directive. The Business Software Alliance attempted to limit the article 5 and 6 exceptions as much as possible. The European Committee for Interoperable Systems, led by Olivetti, Fujitsu Espana, and Bull, lobbied for broad exceptions. See id. at 230–241.

6. As will be discussed below, the European Court of First Instance interpreted the word "interoperability" in the directive during the course of the European Commission's competition case against Microsoft.

7. Thomas Drier, "The Council Directive of 14 May 1991, on the Legal Protection of Computer Programs," 9 *European Intellectual Property Review* 319, 324 (1991).

Article 9(1) of the Software Directive provides that any contractual restriction on the reverse-engineering exceptions in articles 5 and 6 is "null and void." Similarly, article 7 contains a reverse-engineering exception to the directive's prohibition on the circumvention of technological protection measures.

Thus, since 1991 there has been a high degree of certainty and predictability in Europe concerning the lawfulness of reverse engineering. The reverse engineer incurs no copyright liability for black-box reverse engineering for any purpose, nor for decompilation for purposes of achieving interoperability. The reverse engineer can ignore with impunity a contractual term prohibiting reverse engineering, presumably even in a negotiated contract. Further, the reverse engineer can circumvent a technological protection measure for purposes of engaging in other lawful reverse engineering.

The Software Directive does not address with any specificity the question of the scope of copyright protection: To what extent could the reverse engineer use what he learned through his reverse engineering? Rather, article 1(2) provides that "[i]deas and principles which underlie any element of a computer program, including those which underlie its interfaces, are not protected by copyright."[8] Commentators have interpreted this to mean that interface information necessary to achieve interoperability must fall on the idea side of the idea/expression dichotomy; otherwise the detailed decompilation provision in article 6 would be of little utility. Once again, there has been no copyright litigation in Europe concerning this.

In sum, the Software Directive settled the copyright issues relating to interoperability within the European Union in 1991. Indeed, in 2000 the European Commission issued a report on the implementation and effects of the Software Directive which concluded that "the objectives of the Directive have been achieved and the effects on the software industry are satisfactory (demonstrated for example by industry growth and decrease in software piracy)."[9] Accordingly, "there appears to be no need to amend the Directive."

Since 1991, the legal battle in the European Union concerning interoperability has centered on a competition-law (antitrust, in U.S. terminology) complaint brought by the European Commission against Microsoft.

8. The directive's eleventh "Whereas" clause defines interfaces as "the parts of the program which provide for . . . interconnection and interaction between elements of software and hardware."

9. Report from the Commission to the Council, the European Parliament and the Economic and Social Committee on the implementation and effects of Directive 91/250/EEC on the legal protection of computer programs, COM(2000) 199 final, at 2.

Though significant, this litigation is beyond the scope of this book because of its basis in competition law rather than copyright law.

However, the European Court of First Instance (CFI) did interpret the meaning of the word "interoperability" in the directive during the course of the litigation. This interpretation ratified the European Commission's long-standing view of the scope of the article 6 decompilation exception.

The case concerned Microsoft's alleged abuse of its dominant position by withholding interface information necessary for Sun Microsystems to make its Solaris operating system fully compatible with technologies based on Microsoft Windows.[10] In 2004, after an investigation, the European Commission found that Microsoft had abused its dominant position and ordered it to provide the necessary specifications to Sun and other companies on reasonable and nondiscriminatory terms. Microsoft appealed the Commission's decision to the CFI, arguing *inter alia* that the Commission's order was inconsistent with the legislative policy of the Software Directive. Specifically, Microsoft asserted that "interoperability" in the directive meant only the ability of one computer program to connect to another program. Because Microsoft licensed interface information to developers of application programs designed to run on Windows, Microsoft claimed that it satisfied the directive's objectives and thus did not abuse its dominant position.

The Commission, on the other hand, interpreted "interoperability" in the directive more broadly to mean the ability to connect to *or* substitute for another program. Because Microsoft refused to license interface information to Sun, whose Solaris operating system competed with Windows, the Commission argued that Microsoft frustrated the directive's intent and thereby abused its dominant position.

In 2007, the CFI ruled as follows:

> [W]hat is at issue in the present case is a decision adopted in application of Article 82 [of the European Community Treaty], a provision of higher rank than [the Software Directive]. The question in the present case is not so much whether the concept of interoperability in the contested decision is consistent with the concept envisaged in that directive as whether the Commission correctly determined the degree of interoperability that should be attainable in the light of the objectives of Article 82 EC.[11]

Nonetheless, the CFI held that the Commission's "two-way" interpretation of "interoperability" as including the ability to connect to and substitute

10. For a more detailed discussion of the case, see Pamela Samuelson, "Are Patents on Interfaces Impeding Interoperability?" 93 *Minnesota Law Review* 1943, 1989–1996 (2009).

11. Case T-201/04, *Microsoft Corp. v. Comm'n*, 2007 E.C.R. II-3601 ¶ 227.

for computer programs "is consistent with that envisaged in" the Software Directive.[12]

By interpreting the word "interoperability" in the directive as it did, the CFI eliminated any possible ambiguity concerning the scope of article 6's permitting decompilation "to obtain the information necessary to achieve the interoperability of an independently created computer program with other programs." Without question, article 6 allows the reverse engineer to decompile an existing computer program for the purpose of developing his own connecting *or* competing computer program. The Commission has consistently understood article 6 in this manner since 1991.[13] The CFI decision thus finally laid to rest the argument that article 6 permits decompilation only for the purpose of developing connecting products.[14]

The CFI decision also strongly implied that article 1(2) of the directive excludes copyright protection for interface specifications. As was noted above, article 1(2) provides that "[i]deas and principles which underlie any element of a computer program, including those which under lie its interfaces, are not protected by copyright." The CFI stated:

> In requiring, by way of remedy, that an undertaking in a dominant position disclose the interoperability information, the Commission refers to a detailed technical description of certain rules of interconnection and interaction that can be used within the work group networks to deliver work group services. That description does not extend to the way in which the undertaking implements those rules, in particular, to the internal structure or to the source code of its products.
>
> The degree of interoperability thus required by the Commission enables competing operating systems to interoperate with the dominant undertaking's domain architecture on an equal footing in order to be able to compete viably with the latter's operating systems. It does not entail making competitors' products work in exactly the same way as its own and does not enable its competitors to clone or reproduce its products or certain features of those products.[15]

12. Id. at ¶ 225.

13. Commission of the European Communities, Twentieth Report on Competition Policy (1991); Michael Sucker, "The Software Directive—Between the Combat Against Piracy and the Preservation of Undistorted Competition," in *A Handbook of European Software Law* (Oxford University Press, 1993).

14. During the drafting of the directive, BSA attempted to limit the decompilation exception to the development of connecting products. See Band and Katoh, *Interfaces on Trial* at 237–240. Similarly, as other countries have considered reverse-engineering exceptions, BSA has argued that article 6 applies only to the development of connecting products. See chapter 5 below.

15. Case T-201/04, *Microsoft Corp. v. Comm'n*, 2007 E.C.R. II-3601 Summary of Judgment ¶4.

The distinction the CFI drew between the "detailed technical description of certain rules of interconnection and interaction" and the way in which a company "implements those rules" in "the internal structure" or "the source code of its products" parallels the idea/expression dichotomy embodied by article 1(2).

Although the Software Directive resolved the copyright issues relating to interoperability within Europe, since 1995 fierce legislative wars have been waged in several Pacific Rim countries over the adoption of the Software Directive's exceptions for reverse engineering. These wars are described in chapter 5.

1.3 The Interoperability Debate in the United States before 1995

In the United States, the story before and after 1995 is much more complex for both of the central questions of the interoperability debate. This is because both questions were resolved in the United States in a common-law, case-by-case manner by the federal courts, rather than by the legislative process of the Software Directive.

1.3.1 The Unprotectability of Interface Specifications

Between 1983 and 1995, U.S. courts became increasingly sophisticated in their understanding of the unique characteristics of computer programs. The courts became more aware that, although the copyright law classifies programs as literary works, they in fact are functional works operating in highly constrained environments. Accordingly, by 1995, courts understood that many program elements should not receive copyright protection, particularly the information necessary for achieving interoperability.

When courts first looked at the issue of interoperability, they favored protection of interface information. In 1983, for example, the U.S. Court of Appeals for the Third Circuit suggested that compatibility was a "commercial and competitive objective which does not enter into the somewhat metaphysical issue of whether particular ideas and expression have merged."[16] Under this reasoning, copyright could protect interface specifications. Three years later, the Third Circuit reinforced this protectionist trend in *Whelan v. Jaslow*.[17]

16. *Apple Computer, Inc. v. Franklin Computer Corp.*, 714 F.2d 1240, 1253 (3d Cir. 1983), *cert. dismissed*, 464 U.S. 1033 (1984).

17. *Whelan Associates, Inc. v. Jaslow Dental Laboratory, Inc.*, 797 F.2d 1222 (3d Cir. 1986), *cert. denied*, 479 U.S. 1031 (1987).

1.3.1.1 *Whelan v. Jaslow* (1987)

Jaslow, the owner of a dental laboratory, hired Whelan, a computer programmer, to develop a computer program to run his business. They agreed that Whelan would retain the copyright in the program and that Jaslow would try to market the program to other dental laboratories. Jaslow soon realized that the Whelan program, written in Event Driven Language (EDL) for an IBM Series One computer, was not compatible with the computers many dental laboratories already possessed. Jaslow then developed a dental lab program in BASIC, which could run on these computers. Whelan sued for copyright infringement.

At trial, Jaslow's expert testified that he compared the source and object code of the two programs, and found "substantive differences in programming style, in programming structure, in algorithms and data structures."[18] Whelan's expert agreed that the Jaslow program was not a simple translation of the Whelan program, but stated that the programs were similar in several respects. The file structures and screen outputs, for example, were virtually identical. Further, five important subroutines "performed almost identically within both programs."[19] Even Jaslow's expert confirmed that the programs had "overall structural similarities."[20]

The district court ruled for Whelan. Jaslow appealed. Jaslow's primary argument on appeal was that copyright protected only the literal elements of a computer program—the actual lines of source or object code—and not the non-literal elements such as program structure. In a lengthy opinion, the U.S. Court of Appeals for the Third Circuit rejected Jaslow's argument and held that copyright could program the non-literal elements of a computer program, including its "structure, sequence, and organization." The reasoning and language used by the Third Circuit, however, went much farther than necessary to reach this conclusion.

Upon completing a background discussion on the basic principles of copyright law applicable to the case, the Third Circuit turned to "whether a program's copyright protection covers the structure of the program or only the program's literal elements, *i.e.*, its source and object codes."[21] The court observed that "computer programs are classified as literary works for the purposes of copyright," and that "[o]ne can violate the copyright of a play or book by copying its plot or plot devices." Accordingly, the court reasoned that copyright protection should extend to a computer program's structure.

18. Id. at 1228.
19. Id.
20. Id.
21. Id. at 1234.

The court then formulated the following rule for separating idea from expression in utilitarian works:

> [T]he purpose or function of a utilitarian work would be the work's idea, and everything that is not necessary to the purpose or function would be part of the expression of the idea. Where there are various means of achieving the desired purpose, then the particular means chosen is not necessary to the purpose; hence, there is expression not idea.[22]

The court defined the idea in the case before it as "the efficient management of a dental laboratory."[23] It then went on to say that "[b]ecause that idea could be accomplished in a number of different ways with a number of different structures, the structure of the [Whelan] program is part of the program's expression, not its idea."[24]

The *Whelan* court's reasoning contained two related flaws. First, the *Whelan* court identified a single, highly abstract idea in the entire computer program. Second, the court incorrectly reduced the idea/expression dichotomy to the merger doctrine. In the court's view, if several means existed for performing the program's basic function (its idea), then the means did not merge with the function and thus were protected expression. The court failed to understand that each means of performing the function could in its own right be unprotected under section 102(b) as a procedure, process, system, method, or operation. Patents, not copyrights, protect "the means for carrying the idea out."[25] By protecting the means for performing a function, the *Whelan* court in effect used copyright to protect patentable subject matter.[26]

The *Whelan* decision contained two justifications for this extreme result. First, because Congress classified computer programs as literary works, the court treated them as traditional literary works, comparable to novels and plays, without recognizing their utilitarian nature.[27] Second, the *Whelan* court noted that "the coding process is a comparatively small part of programming,"[28] whereas "among the more significant costs in computer

22. Id.
23. Id. at n. 28.
24. Id.
25. *Kruger v. Whitehead*, 153 F.2d 238, 239 (9th Cir. 1946), *cert. denied*, 332 U.S. 774 (1947).
26. See Arthur J. Levine, "Comment on Bonito Boats Follow-Up: The Supreme Court's Likely Rejection of Nonliteral Software Copyright Protection," *The Computer Lawyer* 29, 30 (July 1989).
27. *Whelan*, 797 F.2d at 1237 (citations omitted).
28. Id. at 1231.

programming are those attributable to developing the structure and logic of the program."[29] It observed that "[t]he rule proposed here . . . would provide the proper incentive for programmers by protecting their most valuable efforts, while not giving them a stranglehold over the development of new computer devices that accomplish the same end."[30] The court evidently believed that a programmer's method for solving a program deserved protection—so long as other methods for solving the program existed—because the method was the most valuable part of the program.

In other words, *Whelan* suggested that, in the computer context, the court need only assess whether alternative methods of accomplishing the basic ideas exist to determine whether the elements the defendant copied constitute protected expression. This truncated protected expression analysis invariably affords programs "thick" copyright protection—indeed, thicker protection than is accorded traditional literary works such as novels and plays, which undergo a complete protected expression analysis. Although *Whelan* did not specifically concern interoperability, its reasoning inevitably led to the conclusion that detailed program elements such as interface specifications received copyright protection.

The *Whelan* decision was controversial from the moment it was issued. Just five months later, the Fifth Circuit rejected its reasoning in a case involving programs with similar design specifications that assisted cotton farmers in growing and marketing their product (*Plains Cotton Co-Op Ass'n v. Goodpasture Computer Serv., Inc.*).[31] Nonetheless, lower courts followed *Whelan* until 1992, when the Second Circuit revealed its serious flaws in a case that did involve interoperability: *Computer Associates Int'l, Inc. v. Altai, Inc.*[32]

1.3.1.2 *Computer Associates v. Altai* (1992)

Computer Associates developed an application program with a component, ADAPTER, that permitted the application to run on different IBM mainframe operating systems. Altai developed a similar application designed to run on a single IBM operating system. Altai then decided to develop a component that allowed its program to run on other IBM operating systems. Computer Associates filed suit, alleging that Altai's component, OSCAR 3.4, infringed the copyright in ADAPTER. Altai determined that 30 percent of the OSCAR 3.4 code was copied from ADAPTER, and it conceded

29. Id.
30. Id. (footnotes omitted).
31. 807 F.2d 1256 (5th Cir.), *cert. denied*, 484 U.S. 821 (1987).
32. 982 F.2d 693 (2d Cir. 1992).

liability with respect to OSCAR 3.4. Altai then rewrote its component in a clean room, without access to ADAPTER. Computer Associates amended its complaint to allege that the new version, OSCAR 3.5, also infringed its copyright.

The district court rejected *Whelan* as simplistic and as leading to excessively broad protection for computer programs. The court then compared the two programs. Because of the use of the clean room, the code was completely different. The parameter lists and macros of the programs were similar, but the court determined that these similarities were dictated by the IBM operating systems with which the programs were designed to interoperate. There was overlap in the list of services, but this too was dictated by function. Finally, the court found similarity in the programs' organization charts, but the charts were "simple and obvious" and of *de minimus* importance. Accordingly, the district court concluded that the programs were not similar in protected expression.

Computer Associates appealed. After reviewing the principles of computer program design and the facts of the case, the U.S. Court of Appeals for the Second Circuit acknowledged the "essentially utilitarian nature of a computer program."[33] Identifying the seminal U.S. Supreme Court decision in *Baker v. Selden* as the "doctrinal starting point in analyses"[34] of the scope of protection for computer programs, the Second Circuit emphasized that "compared to aesthetic works, computer programs hover even more closely to the elusive boundary line described in Section 102(b)."[35]

The Second Circuit rejected the principles for analyzing computer programs offered in *Whelan*, holding that "[t]he crucial flaw in [*Whelan*'s] reasoning is that it assumes that only one 'idea' in copyright law terms, underlies any computer program."[36] It also agreed with the district court that a computer program's "ultimate function or purpose is the composite result of interacting sub-routines."[37] The Second Circuit wrote: "[S]ince each sub-routine is itself a program, and thus, may be said to have its own 'idea,' *Whelan*'s general formulation that a program's overall purpose equates with the program's idea is descriptively inadequate."[38] The Second Circuit further agreed with the district court's rejection of *Whelan*'s

33. *Computer Associates,* 982 F.2d at 704.
34. Id.
35. Id.
36. Id. at 705 (citations omitted).
37. Id.
38. Id.

terms "structure, sequence and organization," observing that they were based on a "somewhat outdated appreciation of computer science."[39]

Noting that "*Whelan*'s approach to separating idea from expression in computer programs relies too heavily on metaphysical distinctions and does not place enough emphasis on practical considerations,"[40] the Second Circuit proposed a three-part procedure for determining whether an allegedly copied program is "substantially similar" to another copyrighted program:

> In ascertaining substantial similarity under this approach, a court would first break down the allegedly infringed program into its constituent structural parts. Then, by examining each of these parts for such things as incorporated ideas, expression that is necessarily incidental to those ideas, and elements that are taken from the public domain, a court would then be able to sift out all non-protectable material. Left with a kernel, or perhaps kernels, of creative expression after following this process of elimination, the court's last step would be to compare this material with the structure of an allegedly infringing program.[41]

The Second Circuit based its first step—abstraction—on Judge Learned Hand's famous test in *Nichols v. Universal Pictures Corp*.[42] The court explained:

> In a manner that resembles reverse engineering on a theoretical plane, a court should dissect the allegedly copied program's structure and isolate each level of abstraction contained within it. This process begins with the code and ends with an articulation of the program's ultimate function.[43]

The discussion of the second step—filtration—is perhaps the most significant part of the opinion. The court adopted the "successive filtering method" proposed by the well-respected treatise *Nimmer on Copyright*, which "entails examining the structural components at each level of abstraction to determine whether their inclusion at that level was 'idea' or was dictated by considerations of efficiency, so as to be necessarily incidental to that idea; required by factors external to the program itself; or taken from the public domain and hence is nonprotectable expression."[44]

The "successive filtering method" is a refinement of Judge Hand's abstractions test. In its classical formulation, the abstractions test calls for a court to analyze a work's levels of abstraction and to then draw a line above

39. Id. at 706.
40. Id.
41. Id.
42. 45 F.2d 119 (2d Cir. 1930), *cert. denied*, 282 U.S. 902 (1931).
43. Id. at 707.
44. Id.

which everything is idea and below which everything is expression. Here, the Second Circuit suggested that idea and expression may be present at each level of abstraction.

The court then provided additional detail on the non-protectability of elements dictated by efficiency and by external factors. It observed that "[e]fficiency is an industry-wide goal,"[45] and that "[w]hile, hypothetically, there might be a myriad of ways in which a programmer may effectuate certain functions within a program—i.e., express the idea embodied in a given subroutine—efficiency concerns may so narrow the practical range of choice as to make only one or two forms of expression workable options."[46] Under these circumstances, the expression would merge with the idea and would not receive copyright protection.

Discussing external factors, the Second Circuit stated that "in many instances it is virtually impossible to write a program to perform particular functions in a specific computing environment without employing standard techniques."[47] The Second Circuit went on to hold that under the doctrine of *scènes à faire* copyright protection should not extend to those program elements in which a programmer's "freedom of design choice"[48] is "circumscribed by extrinsic considerations such as (1) mechanical specifications of the computer on which a particular program is intended to run; (2) compatibility requirements of other programs with which a program is designed to operate in conjunction; (3) computer manufacturers' design standards; (4) demands of the industry being serviced; and (5) widely accepted programming practices within the computer industry."[49] Applying these principles to the facts before it, the Second Circuit affirmed the district court and held that Altai had not copied protected expression.

Thus, relying on the *scènes à faire* doctrine, the Second Circuit held that similarities resulting from the need to interoperate with other components of a computer system did not constitute copyright infringement.[50]

45. Id. at 708.

46. Id.

47. Id. at 709 (quotation omitted).

48. Id.

49. Id. at 709–710.

50. Under the *scènes à faire* doctrine, courts "deny protection to those expressions that are standard, stock or common to a particular topic or that necessarily follow from a common theme or setting. Granting copyright protection to the necessary incidents of an idea would effectively afford a monopoly to the first programmer to express those ideas." *Gates Rubber Co. v. Bando Chem. Indus., Inc.*, 9 F.3d 823, 838 (10th Cir. 1993) (citations omitted).

In essence, the Second Circuit ruled that interface specifications were not protected expression, and that a competitor could conform to the rules of intercommunications developed by another vendor without infringing that vendor's copyright.

The reasoning of the *Computer Associates* decision was so powerful that many courts throughout the United States and abroad adopted it rapidly.[51] *Whelan* was thoroughly repudiated, and courts began applying the abstraction-filtration-comparison methodology in a wide range of copyright cases, including cases that did not involve computer programs. Additionally, other courts soon followed *Computer Associates'* specific rulings concerning interoperability.

1.3.1.3 *Atari v. Nintendo* and *Sega v. Accolade* (1992)

Just three months after the Second Circuit issued *Computer Associates*, the U.S. Court of Appeals for the Federal Circuit relied upon it in *Atari Games Corp. v. Nintendo of America, Inc.*, stating that "the court must filter out as unprotectable . . . expression dictated by external factors (like the computer's mechanical specifications, compatibility with other programs, and demands of the industry served by the program)."[52] In *Atari*, both the district court and the Federal Circuit extended protection to Nintendo program elements that currently had no purpose but that Atari argued would be necessary for Atari to achieve compatibility in the future with Nintendo products not yet on the market. The Federal Circuit stated that "[t]he district court did not abuse its discretion by refusing to allow Atari to rely on speculative future events to justify inclusion of unnecessary [Nintendo] program elements in the [Atari] program."[53] The Federal Circuit made it clear, however, that it would not protect program elements needed to achieve compatibility at the time of the writing of the compatible program.

A month later, the U.S. Court of Appeals for the Ninth Circuit, also relying on *Computer Associates*, expressly recognized, in *Sega Enters. Ltd. v. Accolade, Inc.*, that computer programs "contain many logical, structural, and visual display elements that are dictated by . . . external factors such as compatibility requirements and industry demands," and that "[i]n some

51. To get the full flavor of the *Computer Associates* tidal wave, see Band and Katoh, *Interfaces on Trial* at 131–150.
52. 975 F.2d 832, 839 (Fed. Cir. 1992).
53. Id. at 845.

circumstances, even the exact set of commands used by the programmer is deemed functional rather than creative for purposes of copyright."[54]

1.3.2 The Permissibility of Reverse Engineering

Following *Computer Associates*, the *Sega* court had little trouble concluding that copyright did not protect program elements necessary for interoperability. A trickier issue for the *Sega* court was the permissibility of the copying that occurred while examining a competitor's product to uncover these program elements. To be sure, the U.S. Supreme Court has long recognized that there is nothing inherently wrong with studying a competitor's product to understand how it works and to figure out how to make a better product. For example, in *Kewanee Oil Co. v. Bicron Corp*.[55] the Supreme Court stated that "trade secret law . . . does not offer protection against discovery by fair and honest means, such as . . . by so-called reverse engineering, that is by starting with a known product and working backward to divine the process which aided in its development or manufacture."

The Supreme Court has also recognized the benefits of reverse engineering: "Reverse engineering . . . often leads to significant advances in technology."[56] Further, the Supreme Court has noted that "the competitive reality of reverse engineering may act as a spur to the inventor, creating an incentive to develop inventions that meet the rigorous requirements of patentability."[57]

Copyright law, however, has the potential of raising obstacles to reverse engineering of software. Because of the nature of computer technology, reverse engineering of software almost always requires the making of a reproduction or derivative work. For example, the reverse-engineering method of disassembly or decompilation involves "translating" the publicly distributed, computer-readable program into a higher-level, human-readable form. This act of translation could be considered the preparation of a derivative work.[58] Black-box reverse engineering is less intrusive than disassembly because an engineer observes the program's behavior and interaction with its environment without looking at the program itself. Although less intrusive than disassembly, black-box reverse engineering requires that the program be copied into the computer's random-access memory (RAM)

54. 977 F.2d 1510, 1524 (9th Cir. 1992) (citations omitted).
55. 416 U.S. 470, 476 (1974).
56. *Bonito Boats, Inc., v. Thunder Craft Boats, Inc.*, 489 U.S. 141, 160 (1989).
57. Id.
58. See 17 U.S.C. §106(2).

as the computer runs the program. Such copying arguably infringes the reproduction right.[59] As was noted above, the European Union, in its 1991 Software Directive, established a statutory copyright exception excusing the copying that occurs during reverse engineering. In contrast, in the early 1990s U.S. courts employed the doctrine of fair use, codified at 17 U.S.C. §107, to permit reverse engineering.

The first thorough judicial consideration of software reverse engineering occurred in 1992 in *Sega Enters. v. Accolade Inc*.[60] Accolade, a developer of computer games, decompiled software in the Sega video console and in Sega-compatible games in order to learn the interface specifications that would enable it to port its games to the Sega console. Sega sued for copyright infringement, and the district court issued a preliminary injunction against Accolade. The U.S. Court of Appeals for the Ninth Circuit reversed, finding that "where disassembly is the only way to gain access to the ideas and functional elements embodied in a copyrighted computer program and where there is a legitimate reason for seeking such access, disassembly is a fair use of the copyrighted work, as a matter of law."[61] In the *Sega* case, the court concluded that achieving interoperability between the Accolade games and the Sega game console was such a legitimate reason.

Much of the *Sega* court's fair-use analysis centered on the second of the four fair-use factors: the nature of the copyrighted work. The court recognized the unique characteristics of software and understood that if reverse engineering were not permitted the developer would receive *de facto* protection over uncopyrightable ideas. In *Atari Games Corp. v. Nintendo of America, Inc*.[62] the Federal Circuit reached the same conclusion for the same reason.

In sum, by 1995 courts in the United States had ruled that copyright did not protect interface specifications and that the copying incidental to the reverse engineering necessary for interoperability did not infringe copyright. But, as we shall see, the interoperability debate was far from over.

59. The Ninth Circuit in *MAI Systems Corp. v. Peak Computer, Inc.*, 991 F.2d 511 (9th Cir.), *cert. denied*, 510 U.S. 1033 (1993), found that the loading of a program into a computer's RAM constituted a copy for purposes of the Copyright Act. However, as we discuss below in section 2.5, the Second Circuit's decision in *Cartoon Network LP v. CSC Holdings, Inc.*, 536 F.3d 121 (2d Cir. 2008), *cert. denied*, 129 S. Ct. 2890 (2009), suggests that not all temporary copies are fixed within the meaning of the Copyright Act.

60. 977 F.2d 1510 (9th Cir. 1992).

61. Id. at 1527–1528.

62. 975 F.2d 832 (Fed. Cir. 1992).