No. 18-956

IN THE
Supreme Court of the United States

GOOGLE LLC,
Petitioner,
v.
ORACLE AMERICA, INC.,
Respondent.

On Writ of Certiorari to the United States Court of Appeals for the Federal Circuit

BRIEF OF MICROSOFT CORPORATION AS *AMICUS CURIAE* IN SUPPORT OF PETITIONER

LISA W. BOHL
MOLOLAMKEN LLP
300 N. LaSalle St.
Chicago, IL 60654
(312) 450-6700

LEONID GRINBERG
MOLOLAMKEN LLP
430 Park Ave.
New York, NY 10022
(212) 607-8160

JEFFREY A. LAMKEN
Counsel of Record
MICHAEL G. PATTILLO, JR.
MOLOLAMKEN LLP
The Watergate, Suite 660
600 New Hampshire Ave., NW
Washington, DC 20037
(202) 556-2000
jlamken@mololamken.com

Counsel for Amicus Curiae

WILSON-EPES PRINTING CO., INC. – (202) 789-0096 – WASHINGTON, D.C. 20002

TABLE OF CONTENTS

TABLE OF CONTENTS—Continued

TABLE OF AUTHORITIES

TABLE OF AUTHORITIES—Continued

TABLE OF AUTHORITIES—Continued

TABLE OF AUTHORITIES—Continued

TABLE OF AUTHORITIES—Continued

IN THE

Supreme Court of the United States

NO. 18-956

GOOGLE LLC,

Petitioner,

v.

ORACLE AMERICA, INC.,

Respondent.

On Writ of Certiorari to the United States Court of Appeals for the Federal Circuit

---

BRIEF OF MICROSOFT CORPORATION AS *AMICUS CURIAE* IN SUPPORT OF PETITIONER

INTEREST OF *AMICUS CURIAE*[1]

Microsoft Corporation ("Microsoft") is a leading innovator in computer software; it has been creating software platforms and APIs for application developers for over

---

[1] Pursuant to Supreme Court Rule 37.6, counsel for *amicus curiae* states that no counsel for a party authored this brief in whole or in part. No counsel or party made a monetary contribution intended to fund the preparation or submission of this brief, and no person other than *amicus* or its counsel made such a contribution. The parties have provided written consent to the filing of this *amicus* brief.

forty years. Microsoft's mission is to enable individuals and businesses throughout the world to realize their full potential by creating technology that transforms the ways people work, play, and communicate. Microsoft develops, manufactures, licenses, sells, and supports a wide range of software programs, devices, and services, including Windows, Microsoft Azure, Microsoft Office 365, Surface, Xbox and Xbox Live, and Bing. And it invests billions of dollars on research, development, and promotion of new technologies, products, and services to compete in dynamic technology markets.

This case concerns the Copyright Act's authorization of "the fair use of a copyrighted work" under 17 U.S.C. § 107—in particular, how it applies to functional computer code. That issue has profound consequences for innovation in today's computer industry, which depends upon reuse of functional code for purposes of collaborative development and ensuring interoperability and compatibility across a wide array of software platforms and hardware devices. Courts have long adopted a flexible application of the fair use doctrine to accommodate those realities. But in overturning the jury's verdict here, the decision below upends that settled law. It takes an unduly narrow view of fair use that elevates functional code to the same level of copyright protection as the creative expression in a novel. That ruling threatens modern paradigms of software development.

Microsoft has a unique—and balanced—perspective on the technological, legal, and economic issues this case implicates. On the one hand, Microsoft relies on copyright protection, which enables it to license its own products and services and earn a fair return for its creations. On the other hand, Microsoft recognizes that limits on copyright—such as the fair use doctrine—are essential to

the operation of the computer industry. For example, Microsoft also uses and licenses copyrighted works, and has a longstanding interest in preserving room for legitimate reverse-engineering, competitive analysis, and innovative follow-on development of existing software. Microsoft, its customers, and its developers also need their own products to interoperate with systems, platforms, and solutions provided by multiple vendors. Further, Microsoft uses, contributes to, and sponsors open-source projects, which also rely on settled copyright law—both its protections and exceptions. For example, Microsoft has for several years been the most active corporate maintainer of open-source projects on GitHub, the leading collaborative software development platform, which Microsoft acquired in 2018. Microsoft also implements APIs from the open-source community in its Windows and Azure products. Similarly, third parties implement Microsoft APIs in their products to enable compatibility and interoperability. Microsoft thus has a profound interest in ensuring the Court appreciates how innovation in today's computer industry is impacted by the copyright issues presented in this case.

SUMMARY OF ARGUMENT

I. The nature of software production has shifted dramatically in the twenty-first century. Innovation today depends on collaborative development; it is no longer the case that software is predominantly created by a single entity or individual who develops a proprietary product in isolation. Instead, developers rely on sharing, modifying, and enhancing previously developed code to create new products and develop new functionality. Both a cause and effect of this collaborative development is the increased demand for seamless interoperability and compatibility—*i.e.*, the ability of different products, de-

vices, and applications to communicate and work together without effort from the consumer. That can be as basic as enabling a document created in Microsoft Word to be opened, edited, and saved in another application, or as complex as enabling an ecosystem of smart devices, applications, and platforms to connect and interoperate seamlessly in the cloud. This collaborative development and interoperability depend upon the reuse of computer code, like APIs, that serves largely functional purposes—like calling on a program to perform a specific task.

II. It is no surprise that a broad swath of the software industry—from individual developers and computer scientists to start-ups to large companies—have filed *amicus* briefs supporting petitioner in this case. The modern software industry's development paradigm, which accepts and expects that much functional code can be reused by follow-on developers, depends upon a flexible application of copyright law. For decades, courts have adapted the fair use doctrine to address the unique issues software presents. Fair use promotes two fundamental but competing interests of copyright law: assuring authors the right to their original expression, and encouraging innovation built upon the ideas and information conveyed by a prior work. In the software context, courts have recognized that copyright should provide protections for aspects of software that reflect truly creative expression from piracy and other forms of identical copying. But courts have balanced that with the recognition that copyright must allow some reuse of software's functional aspects to enable the collaborative development and interoperability that are critical to the modern computer industry. Copyright holders and follow-on developers alike have flourished under that approach.

III. The balance of interests in that flexible approach favors upholding the jury's finding of fair use here. In overturning that verdict, the Federal Circuit disregarded the critical significance of the nature of the copyrighted material, affording functional software elements the same level of protection as creative aspects of a work of fiction. The Federal Circuit's opinion disregards three controlling Ninth Circuit opinions holding that fair use is essential to allow follow-on developers to access existing copyrighted programs to develop compatible new software—even software that might compete with the programs being used. The court also imposed a problematically narrow standard for evaluating "transformative use" of functional code. While Google used the software interfaces at issue for the same purpose as in Oracle's Java platform—allowing a program to invoke computer functionalities—it incorporated them into a completely different platform that opened new possibilities for programmers and consumers. Such follow-on innovation promotes the purposes of copyright law, and fair-use analysis should give it due weight.

IV. The Federal Circuit's approach here threatens innovation in the software industry. Its rigid analysis undermines the viability of modern collaborative software development, in which a developer's software product is not an end point, but a launching pad for further innovation. If companies can no longer reuse software's functional elements without explicit authorization from the software's creator, such innovative follow-on development will be compromised. The Federal Circuit's decision also undermines the reuse of functional code to achieve compatibility and interoperability, endangering another linchpin of today's interconnected world.

ARGUMENT

The nature of innovation in the computer industry has changed dramatically over recent decades. Gone are the days when computing products operate in isolation. It is less common that a single company develops entirely proprietary products. Rather, more software products are developed through collaboration among many different parties. And consumers now demand that products be able to interoperate across myriad software platforms and hardware devices. Such collaborative development and interoperability are facilitated by an industry paradigm that expects, and accepts, that much existing functional software code may be reused by follow-on developers.

That paradigm depends in large part upon an application of copyright law that reflects the realities of software development today. Industry participants expect that copyright will provide robust protection for aspects of their software that reflect creative expression. At the same time, the law must allow for the reuse of software's functional aspects to facilitate follow-on innovation. For decades, courts have addressed those competing interests through a flexible application of the fair use doctrine. Copyright holders and follow-on developers alike have flourished under that approach.

After hearing the evidence, the jury concluded here that Google's fair-use defense was valid. The Federal Circuit's reversal of that verdict as a matter of law—setting down rigid rules in the process—threatens disastrous consequences for innovation. It extinguishes the necessary "breathing room" for the ecosystem of innovation fair use protects. This Court should reverse the Federal Circuit's ruling on fair use to ensure that copy-

right "promote[s]," rather than impedes, "the Progress of Science and useful Arts." U.S. Const. art. I, § 8, cl. 8.

I. Innovation In Today's Computer Industry Depends On Collaborative Development And Seamless Interoperability—Both Of Which Require Reuse Of Functional Code

Two driving forces of innovation in the modern computer industry are collaborative development of products, and seamless interoperability of those products across platforms and devices. Both depend on the industry's expectation that follow-on developers can reuse existing functional software code, like the declaring code of APIs at issue in this case.

A. Innovation in the Modern Computer Industry Relies on Collaborative Development

Software production "has undergone a radical transformation" in recent decades. Clark D. Asay, *Software's Copyright Anticommons*, 66 Emory L.J. 265, 280 (2017). Under the previously dominant "producer" model, a single company created a "proprietary software offering." *Id.* at 284. Programmers "wrote software much like authors wrote manuscripts: they would come up with an idea and write down the program necessary to make the idea come to fruition." Sonia K. Katyal, *The Paradox of Source Code Secrecy*, 104 Cornell L. Rev. 1183, 1199 (2019).

The proprietary model still serves an important role in the industry. But software today typically is no longer produced by a single author; instead, "more and more software is collaboratively built." Asay, *Anticommons*, *supra*, at 279. In the modern software industry, "open collaborative innovation projects" increasingly serve as important "sources of innovative products, processes, and

services." Carliss Baldwin & Eric von Hippel, *Modeling a Paradigm Shift: From Producer Innovation to User and Open Collaborative Innovation*, 22 Org. Sci. 1399, 1411 (2011). That collaboration takes forms that were rare, if not unheard of, twenty years ago. For example, commercial and non-commercial software producers now collaborate to "bundle cooperatively-developed software with proprietary code." Katyal, *supra*, at 1207. And firms that are otherwise competitors now organize in networks to collectively develop products that benefit the entire industry.

That collaboration, fueled by advances in software engineering techniques, has allowed developers to treat portions of code as "building blocks." Asay, *Anticommons*, *supra*, at 281. Those "building blocks" are both "self-contained, meaning they can function independently," and "'modular,' meaning that other [building blocks] can be created and used with [preexisting blocks] without having to completely rewrite the preexisting" code. *Ibid.* These "increasingly * * * modularized design" practices have simplified and reduced the costs of collaborative development. Baldwin & von Hippel, *supra*, at 1399. A developer can now create a new program by "simply select[ing] a group of preexisting" code "and combin[ing] them in a new way," or by "add[ing] some new [building blocks] for interacting with" previously developed ones. Asay, *Anticommons*, *supra*, at 281. The building-block approach has "dramatically increase[d] the pace of software innovation." *Ibid.*

New collaborative programming methods have increased not only the speed, but also the type of innovation. These methods have facilitated the "capacity to produce unanticipated change through unfiltered contributions from broad and varied audiences." Jonathan

Zittrain, *The Future of the Internet and How To Stop It* 70 (Yale Univ. Press, 1st ed. 2008). Software improvements can now be made not just by corporations or professional developers, but also by amateurs or end-users of software products who modify existing software to better serve their needs. Such user-based innovation has resulted in "commercially significant product and process development and modification in many fields." Baldwin & von Hippel, *supra*, at 1400. In the cutting-edge realm of artificial intelligence, for example, companies are developing highly sophisticated, deep-learning systems while recognizing that third parties may have ingenious new ideas for services that utilize such systems' capabilities.[2]

Under the modern "paradigm," parties have "collaboratively built some of the most popular and important software technologies in the world, including Linux, Android, Apache Web Server, Firefox, * * * and many others that power much of the Internet and computing world." Asay, *Anticommons*, *supra*, at 283. The open-source industry, which is one example of this collaborative paradigm, was estimated to be worth $17 billion in 2019 and is predicted to reach $33 billion by 2022.[3] That trend will only continue—especially as the world shifts to cloud computing, where 90% of workloads use the Linux

---

[2] See Mark Kaelin, *Microsoft Cognitive Services: Leading the AI Charge*, TechRepublic (May 22, 2017), https://www.techrepublic.com/article/build-2017-ai-will-change-everything-and-microsoft-looks-to-lead-the-way/.

[3] See CB Insights, *Open-Source Software Has Changed the Way Software Is Developed. Here's Where The $33B Industry Is Headed* 3 (2019), https://www.cbinsights.com/reports/CB-Insights_Future-Of-Open-Source.pdf.

open-source operating system and other open-source components.[4] This collaborative development approach has become the "default innovation paradigm." Asay, *Anticommons*, *supra*, at 283.

B. Interoperability Is a Key Component of Technological Innovation Today

This transformation in today's computer ecosystem has been driven in large part by the demand for interoperability. Interoperability is the ability of "heterogeneous products and services to exchange software interfaces * * * and share data." Asay, *Anticommons*, *supra*, at 279. In the early era of software development, "programs were largely written for and confined to specific hardware products," so "[l]ittle to no interoperability * * * existed." *Id.* at 286. Later, however, a "greater separation between hardware and software" developed, Katyal, *supra*, at 1192, as companies began mass-marketing products "designed to operate on any number of machines from one or more manufacturers," Nat'l Comm'n on New Tech. Uses of Copyrighted Works (CONTU), *Final Report on the National Commission on New Technological Uses of Copyrighted Works*, 3 Computer L.J. 53, 58 (1981). That created an increased need for interoperability across computing platforms.

Indeed, consumers today now expect and rely on their different products, devices, and applications to communicate and work together without any effort on their part. Interoperability makes that possible: It is "the reason

---

[4] See Jonathan Corbet & Greg Kroah-Hartman, Linux Found., *2017 Linux Kernel Development Report* 1 (Oct. 25, 2017), https://www.linuxfoundation.org/publications/2017/10/2017-state-of-linux-kernel-development/.

[users] can read a web site regardless of what Internet browser [they] use" or "read documents on a PC even though someone wrote them on a Mac," or why "messages can pass from phone to computer to tablet." Joseph Gratz & Mark A. Lemley, *Platforms and Interoperability in* Oracle v. Google, 31 Harv. J.L. & Tech. 603, 610 (2018).

Interoperability is also critical to today's "cloud" model, in which files are stored not on local devices, but on remote, third-party servers that can be accessed from different devices and locations. A user's files—whether images, documents, spreadsheets, or music—now pass through multiple applications and servers operating in multiple software environments, all in service of immediate availability on any device. Interoperability helps achieve the promise of the cloud: the availability of user data no matter what platform is accessing it. Without such interoperability, consumers would be "compelled to retain one platform * * * because their data is trapped there." Zittrain, *supra*, at 177.

Today's growing "Internet of Things" is exponentially increasing the demands of interoperability across "a wide array of devices beyond computers." Gratz & Lemley, *supra*, at 612. Software "has made its way into more and more everyday goods, including cars, household appliances, televisions, watches, treadmills, phones, security systems, cooling and heating systems, and more." Asay, *Anticommons*, *supra*, at 287. Those "smart" products often use Internet connectivity to offer advanced features, and require different computing devices and applications to work together to provide users with an integrated experience.

For example, in an interconnected home system, a consumer's phone will automatically instruct the ther-

mostat[5] to turn on and the oven to heat up[6] when she is thirty minutes away from home. Meanwhile, the home alarm system—which sends the homeowner a phone alert whenever it is tripped[7]—will automatically disable the moment she reaches the door. Each device must communicate with and share standards used by other third-party products to ensure compatibility. If, as in computing's early days, every device had its own proprietary interface, one could never add a product outside of a particular vendor's offerings to the system. But in today's interoperable ecosystem, consumers generally can choose smart products based on their merits and functionality, without worrying about compatibility with their existing system.

C. Reuse of Functional Software Code, Including APIs, Is Critical To Promoting Collaborative Development and Interoperability

The collaborative development and interoperability that drives innovation in today's computer industry is made possible in no small part by the reuse of functional software code. Software has a dual character: "[A]lthough code can reflect expressive choices," it can also be "primarily functional and constrained * * * by the specific purposes it is designed to achieve." Simon J. Frankel & Ethan Forrest, *What Remains of Fair Use for Software After* Oracle v. Google*?,* 8 N.Y.U. J. Intell. Prop. & Ent.

---

[5] See, *e.g.*, Google, *Compare Nest Thermostats*, https://store.google.com/us/magazine/compare_thermostats.

[6] See, *e.g.*, GE Appliances, *Connected Ranges & Wall Ovens*, https://www.geappliances.com/ge/connected-appliances/ranges-ovens-cooking.htm.

[7] See, *e.g.*, Ring, *Home Security System*, https://shop.ring.com/pages/security-system.

L. 310, 311 (2019). Functional object code, for instance, "tell[s] [a computer] to perform a given function—by feeding it a set of instructions regarding which circuits to turn on, and which to turn off, and when." James Gibson, *Once and Future Copyright*, 81 Notre Dame L. Rev. 167, 174 (2005). Such code forms the basic "plumbing" of software applications.

APIs, which are interfaces that allow applications to communicate with one another, are one type of functional code. In particular, an API's "declaring code command[s] the computer to execute the associated implementing code, which gives the computer the step-by-step instructions for carrying out the declared function." Pet. App. 126a. In other words, the declaring code simply identifies a function to be performed, while the implementing code actually tells the computer how to perform that function.

Among the code that Google used in this case, for example, was declaring code that facilitates the operation of opening a file.[8] Programs that need to open files are as numerous as they are varied, including music players, image editors, word processors, email clients, and video games. Engineers working on these diverse applications can reuse functional API declaring code to create an interoperable system in which a file created in a word processor can be sent as an attachment in an email client and opened on a mobile phone. And if, for example, the "open file" method is extended to work with a new kind of device, none of the user-level applications would have to

[8] See Pet. App. 126a n.2 (listing "java.io," a package containing classes that read files).

be updated because the declaring code would remain the same.

The ability of developers to repurpose such functionality without obtaining explicit authorization from any specific software creator frees software developers to focus on adding new, innovative features, rather than constantly rewriting new declarations for already-known functions. And reusing declaring code not only conserves engineering resources, but also enables engineers to have a lingua franca for functional operations more generally, which facilitates collaboration among developers.

The reuse of functional elements of APIs has long promoted competition, innovation, and consumer choice. In the late 1980s, for example, IBM dominated the market for PC-compatible computers "through its control of the IBM PC BIOS," which "provide[d] an API for software, including the operating system, to communicate with the computer's processor." Gratz & Lemley, *supra*, at 610. To create PC-compatible computers, other companies "reimplemented" the functional aspects of the IBM API—including "a hierarchy of command" and a software "call" that "would write a particular letter to the screen." *Id.* at 611. That "led to a proliferation of IBM PC-compatible 'clone' computers from Compaq, Dell, and others." *Ibid.*

In another example from the 1990s, an open-source developer created a program called WINE, which allowed developers to enable Windows applications to run on computers that used the Linux open-source system, without explicit authorization from Microsoft. Gratz & Lemley, *supra*, at 611. To create WINE, the developer "use[d] the same hierarchy of function names" of various Windows APIs. *Id.* at 612. Years later, Microsoft created "the inverse of WINE," reimplementing the

structure of certain Linux APIs to create the Windows Subsystem for Linux, a program that allowed Linux programs to run on Windows. *Ibid.* The Windows-Linux experience shows that reuse of functional code is a "two-way street" that benefits both the original creator and the follow-on developer—and ultimately the consumer. See *ibid.*

II. COURTS HAVE LONG APPLIED A FLEXIBLE FAIR USE DOCTRINE TO ADDRESS SOFTWARE'S UNIQUE NATURE

"From its beginning, the law of copyright has developed in response to significant changes in technology." *Sony Corp. of Am.* v. *Universal City Studios, Inc.*, 464 U.S. 417, 430 (1984). Software reflected a leap forward in technology. But it also presented new issues for copyright law not posed by traditional literary works. Unlike a novel, modern software is built collaboratively. See pp. 7-10, *supra*. And it is a "hybrid" of both extremely creative and highly functional elements. *Comput. Assocs. Int'l, Inc.* v. *Altai, Inc.*, 982 F.2d 693, 712 (2d Cir. 1992). For decades, courts have adapted copyright law to address that reality: They have afforded strong protections to creative aspects of software to prevent wholesale piracy, while allowing broad reuse of functional software code under a robust fair use doctrine for developing new technologies. The computer industry has flourished under that approach, benefiting copyright holders and third parties alike. The Federal Circuit's decision upends that approach and threatens the technological innovation it fostered.

A. Software's Collaborative and Functional Elements Distinguish It from Traditional Creative Works Subject to Copyright Protection

Although "[m]ost of the law of copyright * * * developed in the context of literary works such as novels, plays, and films," "[t]he problem presented by computer programs is fundamentally different." *Lotus Dev. Corp.* v. *Borland Int'l, Inc.*, 49 F.3d 807, 819 (1st Cir. 1995) (Boudin, J., concurring). Software is made differently from, and serves different purposes than, traditional literary works.

Unlike a novel, software today is often built collaboratively, not by an individual, siloed author. See pp. 7-10, *supra*. Appreciation of that collaborative paradigm is critical to fair-use analysis. Under the traditional single-producer model, it was once assumed that strong intellectual-property protection for software was "the only feasible way to cover the costs of innovation." Baldwin & von Hippel, *supra*, at 1411. But today's collaborative ecosystem shows the traditional calculus "that software creators will not incur the costs necessary to develop software without exclusive rights in that software" no longer holds. Asay, *Anticommons*, *supra*, at 271. Instead, "any given piece of software may include dozens, hundreds, or even thousands of copyright holders." *Id.* at 279. With new revenue streams that do not depend on a proprietary model, companies are incentivized to innovate even if they do not capture monopoly profits. See Baldwin & von Hippel, *supra*, at 1399-1400.

Much of this collaborative process is facilitated by copyright-based licensing agreements that have developed over time. Asay, *Anticommons*, *supra*, at 279. Nonetheless, collaborative innovation frequently occurs without the express permission of the copyright holder—

yet within the boundaries of copyright law. As explained below (at 21-22), the computer industry has adapted in particular to the breathing room that courts have construed the fair use doctrine to provide. See Gratz & Lemley, *supra*, at 610. An impractically rigid approach to copyright could "make * * * collaboratively built resource[s] more difficult" to produce, stymieing the software-development models that are now ascendant. Asay, *Anticommons*, *supra*, at 268.

Software differs from traditional literary works in other critical respects. Unlike a novel, software is not purely a work of creative expression—it is a "hybrid" of creative and functional elements. *Altai*, 982 F.2d at 712. A programmer's "program structure and design may be highly creative and idiosyncratic." *Sega Enters. Ltd.* v. *Accolade, Inc.*, 977 F.2d 1510, 1524 (9th Cir. 1992), as amended (Jan. 6, 1993). But other aspects of software are utilitarian and serve functional purposes, such as "facilitat[ing] communication between the user and the computer." *Apple Comput., Inc.* v. *Microsoft Corp.*, 35 F.3d 1435, 1444 (9th Cir. 1994). Software thus "'hover[s] * * * closely to the elusive boundary'" between idea and expression that marks copyright's bounds. *Lexmark Int'l, Inc.* v. *Static Control Components, Inc.*, 387 F.3d 522, 535 (6th Cir. 2004). The software interfaces at issue in this case are one example. While they reflect certain minimal creativity, these interfaces are largely functional, allowing a programmer to invoke a function on a device. See Pet. App. 179a-180a; pp. 12-15, *supra*. As explained above (at 7-15), today's industry expects that follow-on developers can reuse such code for the purpose of creating compatible products.

Software thus presents unique practical challenges for copyright law. Compared to traditional works, there are

likely to be both more copyright holders in any given piece of software, and a greater practical need to reuse aspects of software to foster follow-on innovation. Those "changes" from the literary context require a reasoned "response" from the courts when applying fair use. *Sony*, 464 U.S. at 430.

B. A Flexible Fair Use Doctrine Is Essential To Promoting Collaboration and Interoperability in Modern Software Development—As Courts Have Long Recognized

Copyright promotes two fundamental but competing interests: On the one hand, it seeks to "assure[ ] authors the right to their original expression"; on the other, it "encourages others to build freely upon the ideas and information conveyed by a work." *Feist Publ'ns, Inc.* v. *Rural Tel. Serv. Co.*, 499 U.S. 340, 349-350 (1991). To that end, copyright provides broad protections for an author's "creative expression," which "falls within the core" of the work copyright law is intended to foster. *Campbell* v. *Acuff-Rose Music, Inc.*, 510 U.S. 569, 586 (1994). But an array of doctrines also makes clear that copyright law allows great latitude for the reuse of facts, ideas, and other functional elements underlying an author's work. Those include the "idea/expression" dichotomy, *Feist*, 499 U.S. at 350, *scènes à faire*, *Swirsky* v. *Carey*, 376 F.3d 841, 850 (9th Cir. 2004), and most relevant here, fair use, *Campbell*, 510 U.S. at 576; see *Stewart* v. *Abend*, 495 U.S. 207, 237 (1990) ("[F]air use is more likely to be found in factual works than in fictional works.").

Fair use is a "flexible" and adaptable doctrine. *Perfect 10, Inc.* v. *Amazon.com, Inc.*, 508 F.3d 1146, 1163 (9th Cir. 2007). That extends to accounting for the technological realities the copyrighted work presents. See

Sony, 464 U.S. at 430. For decades, courts have tailored fair-use analysis to account for software's hybrid nature—affording protection against piracy for the creative aspects of software, while offering little to no protection to its utilitarian aspects. Courts have been particularly willing to find fair use where functional aspects of software are used to achieve interoperability or compatibility with other software and devices.

In *Sega*, for example, the Ninth Circuit upheld Accolade's copying of object code to develop video games that could be played on Sega's Genesis console. 977 F.2d at 1514-1515, 1525. The code was "essentially utilitarian"—covering the "subroutines" that allowed "the user to interact with the video game" and "the game cartridge to interact with the console"—and thus warranted only "'thin'" copyright protection. *Id.* at 1524-1525. Accolade copied it, moreover, not to appropriate expressive content, but to access "functional" elements needed for "compatibility." *Id.* at 1522. And even though Accolade created "a competing product," that commercial purpose did not "preclude[] a finding of fair use" because it was "rebutted by" the resulting "public benefit": Accolade's use led to an "increase in the number of independently designed video game programs offered for use with the Genesis console." *Id.* at 1522-1523.

The Ninth Circuit applied similar reasoning in *Sony Computer Entertainment, Inc.* v. *Connectix Corp.*, 203 F.3d 596 (9th Cir. 2000). Connectix had copied Sony's "BIOS"—software that controlled the basic functions of Sony's PlayStation game console. *Id.* at 603. Connectix did so in connection with creating new software that enabled users to play video games that had been developed for Sony's console on PCs. *Id.* at 601. The court noted that Sony's BIOS "lies at a distance from the core

[of copyright protection] because it contains unprotected [functional] aspects." *Id.* at 603. The BIOS code thus was entitled to a "'lower degree of protection than more traditional literary works.'" *Ibid.* The court also found that Connectix's program was a transformative use of the BIOS, because it "afford[ed] opportunities for game play in new environments." *Id.* at 606. The court held that Connectix's copying was fair use, despite the fact that it was done to create a product that competed with—and did not otherwise expand the capabilities of or market for—Sony's own product. *Id.* at 608.

Similar examples abound. Time and again, courts have held that copying software to access its functional elements—to develop follow-on, compatible or interoperable technologies—is fair use that furthers copyright law's purposes. See, *e.g.*, *Lewis Galoob Toys, Inc.* v. *Nintendo of Am., Inc.*, 964 F.2d 965, 971 (9th Cir. 1992) (fair use for consumers to use a product that was compatible with Nintendo's games); *Atari Games Corp.* v. *Nintendo of Am. Inc.*, 975 F.2d 832, 843-844 (Fed. Cir. 1992) (reverse-engineering a game console's software to make compatible games was fair use).

Until the Federal Circuit's decision below, the broad application of fair use in such circumstances was considered "settled law." Gratz & Lemley, *supra*, at 610. That "approach to software—grounded in the primarily functional, rather than expressive, nature of most programming—has * * * permitted developers to build upon their predecessors' advances." Frankel & Forrest, *supra*, at 311. The computer industry has structured its conduct in reliance on the breathing room for reuse of functional code that such decisions provided.

C. Experience Has Shown That a Flexible Fair Use Doctrine Fosters Innovation in Computer Software

Experience has shown that the pragmatic approach courts have taken to fair use of software code has fostered the "growth in creative expression * * * that the Copyright Act was intended to promote." *Sega*, 977 F.2d at 1523. Indeed, some have urged that copyright's "solicitousness to copying for the purpose of interoperability is the reason we have a vibrant and competitive [computer] industry" today. Gratz & Lemley, *supra*, at 610.

The video-game industry, for instance, has flourished since the seminal decisions finding fair use in the 1990s. Previously, game development was tied to the maker of the game console and its licensees. Allowing third parties to develop compatible games by reverse-engineering game-console software "facilitat[ed] the entry" of "new competitor[s]." *Sega*, 977 F.2d at 1523. Since then, the video-game industry has grown to generate over $43 billion in annual U.S. revenues.[9]

Nor did the video-game industry's growth come at the expense of the original copyright holders. In *Connectix*, Sony argued that it would lose sales and profits if Connectix were permitted to create a competing platform that could run games created for Sony's PlayStation. 203 F.3d at 607. But the latest PlayStation has sold more than 91.6 million units.[10] In one recent quarter, Sony's

---

[9] Entm't Software Ass'n, *U.S. Video Game Sales Reach Record-Breaking $43.4 Billion in 2018*, (Jan. 22, 2019), https://www.theesa.com/press-releases/u-s-video-game-sales-reach-record-breaking-43-4-billion-in-2018/.

[10] Erik Kain, *Putting the PlayStation 4's 91.6 Million Sales into Context*, Forbes (Jan. 11, 2019), https://www.forbes.com/sites/erik

PlayStation division brought in $670 million in profit, outperforming "other parts of [Sony's] business."[11]

Thus, a central premise of the Federal Circuit's approach—that a fair-use finding here would undermine incentives to produce computer programs—is belied by history. A flexible application of fair use has not harmed the ability of software producers to enter into productive copyright-licensing arrangements and otherwise recoup their investments in innovation. Copyright holders have continued to thrive because allowing reasonable fair use of functional code enables innovation that creates new opportunities for the whole market to grow.

III. THE FEDERAL CIRCUIT'S DECISION DEFIES SETTLED FAIR-USE PRINCIPLES AND MISAPPREHENDS THE COLLABORATIVE NATURE OF THE COMPUTER INDUSTRY

After hearing the evidence in this case, the jury reasonably concluded that Google's reuse of functional Java code was fair use. Pet. App. 9a. In reversing that finding as a matter of law, the Federal Circuit's decision upends the computer industry's settled expectations about fair use of software code. Instead of treating fair use as a "flexible" doctrine that can adapt to address software's dual nature, *Perfect 10*, 508 F.3d at 1163, the Federal Circuit took a rigid view that treats even the functional aspects of software as if they were entitled to the same protection as creative literary works. It also

---

kain/2019/01/11/putting-the-playstation-4s-91-6-million-sales-into-context/#543b0c856c50.

[11] Jamie Rigg, *PlayStation Keeps Making Money, Sony Phones Keep Losing It*, Engadget (Feb. 1, 2019), https://www.engadget.com/2019/02/01/sony-playstation-4-profit/.

took a straitjacketed view of the "transformative use" factor of fair use, failing to acknowledge that Google's reuse of the Java software interfaces in its Android operating system has made a world of new features possible for Java programmers and consumers alike.

The purpose of copyright law is to "promote" "the Progress of Science and useful Arts." U.S. Const. art. I, §8, cl. 8. The Federal Circuit's cramped fair-use analysis defies that purpose, threatening to disrupt collaborative software development and restrict creativity in the most vital and inventive sector of our economy. Reversal is warranted.

A. The Federal Circuit's Disregard of the Functional Nature of Oracle's Declaring Code and SSOs in the Fair-Use Analysis Defies Precedent and Industry Reality

The Copyright Act requires consideration of the "nature of the copyrighted work" in any fair-use analysis. 17 U.S.C. §107(2). That factor "calls for recognition that some works are closer to the core of intended copyright protection than others, with the consequence that fair use is more difficult to establish when the former works are copied." *Campbell*, 510 U.S. at 586. Conversely, "[w]orks that are merely compilations of fact" or of "functional concepts" receive "'thin'" protection, so fair use is easier to establish. *Sega*, 977 F.2d at 1524 (quoting *Feist*, 499 U.S. at 349). Given that software may contain both highly creative elements and essentially functional elements, courts have long recognized that a focus on "the nature of the copyrighted work" taken is "particularly significant" in software cases. *Micro Star* v. *Formgen Inc.*, 154 F.3d 1107, 1113 (9th Cir. 1998).

For example, in *Connectix*, the copied work consisted of software code that controlled the basic functions of the

PlayStation game console. 203 F.3d at 603. Because of its functional nature, the court explained, the code "lies at a distance from the core" of copyright protection, *ibid.*—a factor that "strongly favor[ed]" finding fair use, *id.* at 605. Similarly, in *Sega*, the Ninth Circuit emphasized that the functional nature of the object code that was copied was "important to the resolution" of the fair-use question. 977 F.2d at 1522.

In the decision below, the Federal Circuit took the opposite view. It declared that the nature of the copyrighted software at issue is "'not * * * terribly significant in the overall fair use balancing.'" Pet. App. 42a. The court thus took no real account of the fact that, while the Java declaring code and the SSOs meet the minimum creativity requirements for copyrightability, they are essentially functional—they are the means by which a programmer triggers a function on a device when writing software in the Java language. See, *e.g.*, Pet. App. 126a, 226a, 228a. Under longstanding software copyright principles, the functional nature of the code should have been an analytical pivot point favoring fair use. But the Federal Circuit discarded its significance altogether, and instead treated the relevant software code like a highly creative work within the core of copyright's protection.

In holding that the nature of the copyrighted work is not significant, the Federal Circuit cited only cases involving traditional creative works, such as fictional books, artistic dolls and images, and television programming. Pet. App. 42a-43a (citing *Dr. Seuss Enters., L.P.* v. *Penguin Books USA, Inc.*, 109 F.3d 1394 (9th Cir. 1997) (Dr. Seuss's *The Cat in the Hat*); *Mattel Inc.* v. *Walking Mountain Prods.*, 353 F.3d 792 (9th Cir. 2003) (Barbie doll); *Fox News Network, LLC* v. *TVEyes, Inc.*, 883 F.3d 169 (2d Cir. 2018) (news broadcasts), cert. denied, 139 S.

Ct. 595 (2018)). It is unsurprising that courts "give little attention to the nature-of-the-work factor in run-of-the-mill fair use analyses," Pamela Samuelson & Clark D. Asay, *Saving Software's Fair Use Future*, 31 Harv. J.L. & Tech. 535, 560 (2018), where the copied works contain the "'creativity, imagination and originality'" at the heart of copyright protection, Pet. App. 42a (quoting *Dr. Seuss*, 109 F.3d at 1402).

But software requires a different approach. As this Court has explained, copyright law must "respon[d] to significant changes in technology." *Sony*, 464 U.S. at 430. For decades, courts understood that the nature-of-the-work "factor carries greater weight" in this context "because of software's functional nature." Samuelson & Asay, *supra*, at 560. The Federal Circuit summarily dismissed the Ninth Circuit's decisions in *Connectix* and *Sega* as involving "materially" different "facts," Pet. App. 54a, without addressing their broader reasoning that functional code is entitled to less protection—particularly where it is reused for the purpose of achieving interoperability or compatibility with the copyrighted product, see pp. 18-20, *supra*.

The Federal Circuit justified its contrary conclusion on the grounds that "allowing this one factor"—the functional nature of the code—"to dictate a conclusion of fair use in all cases involving copying of software" would "negate" Congress's declaration "that software is copyrightable." Pet. App. 43a. But recognizing that the functional code here is entitled to thinner protection would not dictate the outcome in "all cases" involving software. Different aspects of software lie on a spectrum, with more creative elements lying closer to "the core of intended copyright protection" than the code here. *Campbell*, 510 U.S. at 586. Courts are more than capable of

drawing that distinction and tailoring the degree of fair-use protection to the nature of the code in the cases before them. See, *e.g.*, *Wall Data Inc.* v. *L.A. Cty. Sheriff's Dep't*, 447 F.3d 769, 780 (9th Cir. 2006) (concluding that the "nature of the copyrighted work weigh[ed] against a finding of fair use" for computer terminal emulation software); *Sega*, 977 F.2d at 1525 (distinguishing between functional and expressive aspects of video-game code). The Federal Circuit's failure to do so here—and its indication that such distinctions are "not * * * significant" in future cases—upsets the computer industry's long-settled expectations, with potentially disastrous consequences for innovation. See pp. 30-32, *infra*.[12]

B. The Federal Circuit Fundamentally Misunderstood What Constitutes a "Transformative Use" of Software

Another critical factor in the fair-use analysis concerns "whether and to what extent the new work is 'transformative.'" *Campbell*, 510 U.S. at 579. The "central purpose of [that] investigation" is to determine "whether the new work merely 'supersedes the objects' of the original creation, or instead adds something new, with a further purpose or different character, altering the first with new expression, meaning, or message." *Ibid.* (citations and brackets omitted). The latter such works "lie at

---

[12] The Federal Circuit was required to apply Ninth Circuit copyright law to this case. See *Atari Games Corp.* v. *Nintendo of Am., Inc.*, 897 F.2d 1572, 1575 (Fed. Cir. 1990) (recognizing that Federal Circuit must apply regional circuit law on subjects not within its exclusive jurisdiction). But the Federal Circuit instead purported to create national law to "guide resolution of [the fair-use] question in all future cases" involving software. Pet. App. 18a.

the heart of the fair use doctrine's guarantee of breathing space within the confines of copyright, and the more transformative the new work, the less will be the significance of other factors * * * that may weigh against a finding of fair use." *Ibid.* (citation omitted).

The Federal Circuit concluded that Google's use of the Java software-interface code was not transformative because "the purpose of the API packages in Android is the same as the purpose of the packages in the Java platform"; "Google made no alteration to the expressive content or message of the copyrighted material"; and "smartphones were not a new context." Pet. App. 31a-32a. That analysis misapprehends the purposes of copyright law and the nature of the code at issue—with critical consequences for future software cases.

1. The Federal Circuit took a rigid view of the "purpose" of Google's reuse that ignores the realities of the computer industry, which relies on reuse of functional code for innovation. The court found that Google's use was not transformative because the Java declaring code and SSOs "'serve the same function in both'" Java and Google's Android. Pet. App. 33a. But while that code served the same broad "purpose" in both works—calling on a device to perform a function—the same could be said of any software code that is reused. Unlike literary works, software code serves not to enlighten or entertain, but "to carry out specific, preassigned computing functions." Clark D. Asay, *Transformative Use in Software*, 70 Stan. L. Rev. Online 9, 14 (2017). As a consequence, "reuses of software will typically implicate the very same functions." *Ibid.* The Federal Circuit's analysis thus leads to an absurd result: It makes it *more* difficult to establish fair use for reusing functional software than for repurposing aspects of a creative fictional work. See *id.*

at 10. That does not merely turn copyright law on its head. It "imperil[s]" the "productive balance that fair use helps strike between copyright holders and follow-on software innovators." *Ibid.*

2. The Federal Circuit also missed the point in focusing on the fact that Google did not alter the "expressive content or message of the copyrighted material" *itself*, Pet. App. 31a-32a—as opposed to acknowledging what Google did with that code in its Android operating system.

The transformative-use factor properly asks "whether the *new work* * * * adds something new, with a further purpose." *Campbell*, 510 U.S. at 579 (emphasis added). Here, the jury reasonably could have found that Google's reuse of the Java software interfaces was transformative because Google utilized the Java code in the context of a totally different software program, Android, that implemented the functions that the Java code invokes using totally different code. See Pet. App. 218a-219a. And unlike the Java platform, which "was developed to run on desktop computers and enterprise servers," Pet. App. 216a-217a, Android "was designed specifically for mobile devices," Pet. App. 196a, and thus "ha[d] to accommodate" factors like "limited memory and battery life, that did not apply to [the Java platform]," Pet. 25.

The Federal Circuit summarily dismissed the notion that Google's use of the Java software interfaces in the Android mobile-phone platform was transformative simply because Java "was already being used in smartphones." Pet. App. 35a. But that broad statement has little bearing on whether Google's use was, in reality, a transformative use of the code. Whether or not "other smartphone manufacturers" had already licensed Java for use in mobile phones, *ibid.*, the fact is that Android

"completely transformed the mobile computing industry and powered innovation in the smartphone market," Asay, *Anticommons*, *supra*, at 315.[13] Indeed, without new platforms like Android, a single mobile operating system—Apple's iOS—likely would have dominated the smartphone market. Cf. Pet. App. 219a ("Android-based mobile devices * * * now comprise a large share of the United States market.").

Ultimately, Google was "not seeking to appropriate the advances" in the Java software interfaces, but "to give [Java programmers] an option to exploit their own prior investment in learning" the Java language. *Lotus*, 49 F.3d at 821 (Boudin, J., concurring). And just as WINE had enabled Windows applications to run in a Linux environment, see pp. 14-15, *supra*, Android also opened up new possibilities to Java programmers, fostering the development of additional, compatible programs. Because Google "facilitate[d] greater compatibility and collaboration" among Java programmers "outside of strictly Sun/Oracle products," its use "represents a different purpose than that of the original creation, and arguably one with greater societal potential." Asay, *Anticommons*, *supra*, at 314-315. It cannot be that Google's use was not transformative *as a matter of law*.

3. The Federal Circuit's emphasis on Google not having "alter[ed] * * * the expressive content or message of the copyrighted material" itself, Pet. App. 31a-32a, is misplaced for another reason. The expressive content in, for example, the Java declaring code, lies in the names

---

[13] While Oracle has a copyright in Java, the law "does not confer" copyright holders in software with "control over the market for devices" that run that software. *Connectix*, 203 F.3d at 607.

chosen to invoke various functions. See Pet. App. 150a. The Federal Circuit could identify no way in which Google altering the names of functions in the declaring code would serve copyright law's purpose of "promot[ing] the Progress of Science and useful Arts." U.S. Const. art. I, §8, cl. 8. Having more names for the same software functions does not enrich society. Quite the opposite—that is akin to having "every typewriter maker * * * scramble the [QWERTY] keyboard." Pet. App. 104a. In short, the Federal Circuit's analysis represents the type of thinking this Court has warned against: It seeks to "simplif[y]" the fair-use analysis with "bright-line rules," rather than performing "case-by-case analysis" and application "in light of the purposes of copyright." *Campbell*, 510 U.S. at 577-578. For that reason, too, reversal is warranted.

IV. The Federal Circuit's Rigid Approach To Fair Use Threatens The Viability Of The Interconnected Software Ecosystem

If allowed to stand, the Federal Circuit's decision would have ramifications far beyond the dispute between Oracle and Google over the Java code in this case. While fair use is supposed to involve a "case-by-case analysis," *Campbell*, 510 U.S. at 577-578, the Federal Circuit made clear that it intended the analytical framework it adopted to "guide resolution of [the fair-use] question in all future cases" involving software, Pet. App. 18a. The Federal Circuit's failure to take a view of fair use that accounts for the real-world uses of functional software code thus threatens profoundly negative consequences for innovation in the computer industry as a whole.

The Federal Circuit's decision threatens the model of open collaboration that is critical to innovation in today's computer industry. See pp. 7-15, *supra*. The existing

"[c]ommunity of practice," which "refers to the social learning that occurs when individuals who have a common interest in some topic or field collaborate over an extended period of time to share knowledge and experience," has been integral to "develop[ing] solutions[ ] and build[ing] prototypes" in technology. Charles C. Snow et al., *Organizing Continuous Product Development and Commercialization: The Collaborative Community of Firms Model*, 28 J. Prod. Innovation Mgmt. 3, 8 (2010). But if companies and individuals can no longer assume that reuse of functional elements of an original software product for such purposes will be protected as fair use, that threatens to impede such follow-on, collaborative innovation at the most basic level.

The Federal Circuit's decision also threatens another pillar of today's computer ecosystem—seamless interoperability and compatibility across software platforms and hardware devices made possible through the reuse of common functional code. See Gratz & Lemley, *supra*, at 609-613; pp. 10-15, *supra*. Under prior law like *Sega* and *Connectix*, companies could take comfort that reusing such code for the purpose of achieving interoperability or compatibility would be fair use. But the Federal Circuit's decision upends those assumptions, creating uncertainty and disincentives to innovation.

Ultimately, the Federal Circuit's decision means less collaboration, less interoperability, and less innovation for consumers—the opposite of the progress copyright law is intended to foster. By contrast, the jury's finding of fair use has no detrimental effect on the ability of software producers to recoup their investment in software creation. Technological changes have reduced the costs of innovation, and it is no longer the case that producers always require decades of exclusive rights to profit from

their software creations. See pp. 7-10, *supra*. In addition, industry experience in the wake of decisions like *Sega* and *Connectix* demonstrates that robust application of fair use tends to expand the overall market for the technology at issue, to the benefit of the original copyright holders. See pp. 21-22, *supra*.

This Court should restore the flexible approach to fair use that is essential to striking the correct balance between copyright protection and follow-on innovation.

CONCLUSION

The judgment of the court of appeals should be reversed.

Respectfully submitted.

LISA W. BOHL
MOLOLAMKEN LLP
300 N. LaSalle St.
Chicago, IL 60654
(312) 450-6700

LEONID GRINBERG
MOLOLAMKEN LLP
430 Park Ave.
New York, NY 10022
(212) 607-8160

JEFFREY A. LAMKEN
Counsel of Record
MICHAEL G. PATTILLO, JR.
MOLOLAMKEN LLP
The Watergate, Suite 660
600 New Hampshire Ave., NW
Washington, D.C. 20037
(202) 556-2000
jlamken@mololamken.com

Counsel for Amicus Curiae

JANUARY 2020