Nos. 2017-1118, 2017-1202

IN THE

United States Court of Appeals for the Federal Circuit

ORACLE AMERICA, INC.,

Plaintiff-Appellant,

v.

GOOGLE, INC.,

Defendant-Appellee,

Appeals from the United States District Court for the Northern District of California in No. 10-CV-3561, Judge William H. Alsup

BRIEF OF COMPUTER SCIENTISTS AS AMICI CURIAE IN SUPPORT OF DEFENDANT-APPELLEE

Phillip R. Malone
Jef Pearlman
Juelsgaard Intellectual Property and Innovation Clinic
Mills Legal Clinic at Stanford Law School
559 Nathan Abbott Way
Stanford, CA 94305-8610
Telephone: 650-725-6369
Fax: 650-723-4426

Attorneys for Amici Curiae

UNITED STATES COURT OF APPEALS FOR THE FEDERAL CIRCUIT

Oracle America, Inc. **v.** Google, Inc.

Case No. 2017-1118, 2017-1202

CERTIFICATE OF INTEREST

Counsel for the:

☐ (petitioner) ☐ (appellant) ☐ (respondent) ☐ (appellee) ☒ (amicus) ☐ (name of party)

Public Knowledge and Law Professors

certifies the following (use "None" if applicable; use extra sheets if necessary):

1. Full Name of Party Represented by me	2. Name of Real Party in interest (Please only include any real party in interest NOT identified in Question 3) represented by me is:	3. Parent corporations and publicly held companies that own 10 % or more of stock in the party
Computer Scientists (See Attachment on next page)	None	None

4. The names of all law firms and the partners or associates that appeared for the party or amicus now represented by me in the trial court or agency or are expected to appear in this court (**and who have not or will not enter an appearance in this case**) are:

Juelsgaard Intellectual Property & Innovation Clinic, Mills Legal Clinic, Stanford Law School: Phillip R. Malone, Jef Pearlman

May 30, 2017
Date

/s/ Phillip R. Malone
Signature of counsel

Please Note: All questions must be answered

Phillip R. Malone
Printed name of counsel

cc: Counsel for Amici Curiae

ATTACHMENT TO CERTIFICATE OF INTEREST

List of COMPUTER SCIENTIST Amici Curiae

(In alphabetical order)

1. Harold Abelson
2. Tom Ball
3. Brian Behlendorf
4. Gordon Bell
5. Jon Bentley
6. Matthew Bishop
7. Joshua Bloch
8. Dan Boneh
9. Gilad Bracha
10. Eric Brewer
11. Frederick Brooks
12. Rick Cattell
13. Vinton G. Cerf
14. William Cook
15. Mark Davis
16. Miguel de Icaza
17. Jeffrey Dean
18. L Peter Deutsch
19. Whitfield Diffie
20. David L. Dill
21. Lester Earnest
22. Brendan Eich
23. Dawson Engler
24. Martin Fowler
25. Neal Gafter
26. Robert Harper
27. John Hennessy
28. Tom Jennings
29. Alan Kay
30. Brian Kernighan
31. David Klausner
32. Ray Kurzweil

33. Kin Lane
34. Ed Lazowska
35. Doug Lea
36. Bob Lee
37. Sheng Liang
38. Barbara Liskov
39. Paul Menchini
40. Andrew W. Moore
41. James H. Morris
42. Peter Norvig
43. Martin Odersky
44. Tim Paterson
45. David Patterson
46. Alex Payne
47. Tim Peierls
48. Simon Phipps
49. Bill Pugh
50. Ronald L. Rivest
51. Curtis Schroeder
52. Robert Sedgewick
53. Mary Shaw
54. Barbara Simons
55. Dave Snigier
56. Alfred Z. Spector
57. Bjarne Stroustrup
58. Gerald Jay Sussman
59. Ivan E. Sutherland
60. Andrew Tanenbaum
61. Brad Templeton
62. Ken Thompson
63. Michael Tiemann
64. Linus Torvalds
65. Andrew Tridgell
66. Jeffrey Ullman
67. Andries van Dam
68. Guido van Rossum
69. John Villasenor
70. Jan Vitek

71. Philip Wadler
72. James H. Waldo
73. Dan Wallach
74. Peter Weinberger
75. Steve Wozniak
76. Frank Yellin

TABLE OF CONTENTS

TABLE OF AUTHORITIES

Oren Bracha & Talha Syed, *Beyond Efficiency: Consequence-Sensitive Theories of Copyright*,
29 Berkeley Tech. L.J. 229 (2014) 19

Peter S. Menell, *An Analysis of the Scope of Copyright Protection for Application Programs*,
41 Stan. L. Rev. 1045 (1989) 16

Peter S. Menell, *Rise of the API Copyright Dead?: An Updated Epitaph for Copyright Protection of Network and Functional Features of Computer Software*
(UC Berkeley Pub. Law Research, Paper No. 2893192),
https://ssrn.com/abstract=2893192 ..... 10, 17, 21

Robert Sedgewick & Kevin Wayne, *Algorithms*
(4th ed. 2011) 5, 26

Ryan Paul, *Why Google Chose the Apache Software License Over GPLv2 for Android*,
Ars Technica (Nov. 6, 2007, 7:26 AM), https://arstechnica.com/?post_type=post&p=79053 34

Stephen Cass, *The 2016 Top Programming Languages,*
IEEE Spectrum (July 26, 2016, 16:00 GMT), http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages 11

Uri Sarid, *A Non-Apocalypse: APIs, Copyright, and Fair Use*,
Wired (May 13, 2014 10:01 AM), https://www.wired.com/insights/2014/05/non-apocalypse-apis-copyright-fair-use 22

WineHQ, *About Wine*, https://www.winehq.org/about (last visited May 30, 2017) 13

STATEMENT OF INTEREST[1]

Amici are 76 individual computer scientists, engineers, and professors who are pioneering and influential figures in the computer industry.[2] Amici include the architects of iconic computers from the mainframe, minicomputer, and microcomputer eras, including the IBM S/360, DEC Vax, and Apple II; languages such as AppleScript, AWK, C, C++, Go, Haskell, JavaScript, Python, Scala, Scheme, Standard ML, and Smalltalk; and operating systems such as MS-DOS, Unix, and Linux. Amici are responsible for key advances in the field, such as computer graphics, cloud computing, public key cryptography, object-oriented programming, virtual reality, and the Internet itself. Amici wrote the standard college textbooks in areas including artificial intelligence, algorithms, computer architecture, computer graphics,

---

[1] No party or party's counsel authored any part of this brief or contributed money towards its preparation or submission. No one, other than amici and their counsel, contributed money towards the preparation or submission of this brief. Pursuant to Fed. R. App. P. 29(a), all parties have consented to the filing of this brief.

[2] Amici's biographies are attached as Appendix—List of *Amici Curiae*, and also can be found at https://law.stanford.edu/list-of-amici-curiae-in-oracle-america-inc-v-google-inc. Amici wish to thank Stanford Law School Juelsgaard Intellectual Property and Innovation Clinic Certified Law Students Daniel Chao and Robert Paris for their substantial assistance in drafting this brief.

computer security, data structures, functional programming, Java programming, operating systems, software engineering, and the theory of programming languages.

Amici have been widely recognized for their achievements. They include at least 8 Association for Computing Machinery (ACM) Turing Award winners (computer science's most prestigious award); 31 ACM Fellows; 14 Institute of Electrical and Electronics Engineers (IEEE) Fellows; 20 American Academy of Arts and Sciences (AAAS) Fellows; 7 National Academy of Sciences Members; 24 National Academy of Engineering Members; 7 National Medal of Technology recipients; and numerous professors at many of the world's leading universities.

Amici have joined this brief because they believe the jury and the District Court correctly decided that Google's reimplementation of the Java application programming interfaces (APIs) was fair use. As computer scientists, amici have relied on API reimplementations and the programs built on them to create and operate new software. Amici have an interest in seeing copyright law evolve in a way that furthers

creativity and enables continued vigorous innovation.[3] Furthermore, amici depend on APIs remaining open to sustain widespread compatibility standards used by startups and incumbents alike. Reversing the District Court would dangerously undermine the settled expectations of computer scientists and the entire computer industry that rely upon the open nature of APIs.

SUMMARY OF ARGUMENT

The software industry has long relied on and benefitted from the open nature of application programming interfaces (APIs). This openness enabled innovations in computing hardware, operating systems, programming languages, internet network protocols, and cloud computing. The long-standing and prevailing industry custom is open access to API reimplementation. Affirming the principle that API reimplementation is fair use will sustain interoperability, encourage innovation, and discourage lock-in.

Though this Court previously held that APIs are copyrightable, it remanded the case because of the open question of fair use, stressing in

[3] Many of the amici here previously filed an amicus brief in the prior appeal of this case, arguing that APIs are not copyrightable.

particular that the functional nature of software APIs and their role in interoperability may be relevant to the fair use analysis. On remand, a jury found that Android's use of Java APIs was fair use.

This result should be upheld. APIs are inherently functional, which weighs strongly in favor of fair use and should guide the remaining fair use analysis. Android is transformative because it successfully brought the Java language and its APIs into an entirely new context: smartphones and tablets. Moreover, API reimplementations enable increased interoperability, favoring fair use.

Overruling the jury's decision would undermine this long-established custom and practice that the computer software industry relies upon so routinely and thoroughly. We urge this Court to affirm the District Court's judgment.

ARGUMENT

I. The Software Industry Has Long Relied on Freely Reimplementing Existing APIs.

Since the birth of modern computing, progress and innovation in the software industry has been predicated on the free and open nature of APIs. APIs have always been freely reimplemented by third parties,

enabling interoperability and contributing to rapid innovation in computer technology.[4]

A. API Reimplementation is Fundamental to the Very Concept of APIs.

APIs facilitate interaction between two software components. They are specifications for a set of functionalities independent of how they are implemented. An API defines what the functionalities are and how they are used, whereas an API's implementation specifies how the functionalities are achieved. A core concept of computing is that APIs may have multiple implementations, which provides "the freedom to substitute new and improved implementations." Robert Sedgewick & Kevin Wayne, *Algorithms* 33 (4th ed. 2011). Code using an API can run on any platform implementing that API, irrespective of the details of that implementation or platform. In programming languages like Java, APIs allow programmers to use existing library code as building blocks for their own applications. Java code that uses Java APIs is compatible with any platform that implements the relevant APIs.

---

[4] For a more in depth history of API reuse and its importance to the industry, see Brief of Amici Curiae Computer Scientists in Support of Petitioner, *Google, Inc. v. Oracle Am., Inc.*, 135 S. Ct. 2887 (2015) (No. 14-410), 2014 WL 5868950.

A reimplementation uses an existing interface to create a new tool or system. To illustrate, a "keytar" can be considered a reimplementation of the piano keyboard, because it uses an existing interface (the piano keyboard) on a new instrument. Figure 1. The wiring beneath the keytar's keyboard interface and the sound it outputs differ from that of the original. The keyboard *interface* can be *implemented* by making sounds in different ways. For example, a piano uses hammers striking strings to produce notes, while a keytar uses electronic synthesizers to do the same. Similarly, an API reimplementation uses an existing interface to produce a new and compatible system.

Figure 1: A keytar.

Computer scientists have freely reimplemented APIs for as long as APIs have existed, allowing software to evolve and improve around accepted standards. *See* Figure 2. For example, Samba is a reimplementation of Microsoft's Server Message Block (SMB) protocols,

which allows Windows systems to share files and printers across local networks. Samba expanded SMB by enabling non-Windows systems to communicate using the SMB protocol as well. *About Samba*, https://www.samba.org (last visited May 30, 2017).

API	**Creator**	**Year**	**Reimplementer**	**Year**
FORTRAN library	IBM	1958	Univac	1961
IBM S/360 ISA	IBM	1964	Amdahl Corp.	1970
C standard library	AT&T/Bell Labs	1976	Mark Williams Co.	1980
Unix system calls	AT&T/Bell Labs	1976	Mark Williams Co.	1980
VT100 Esc Seqs	Dec	1978	Heathkit	1980
IBM PC BIOS	IBM	1981	Phoenix Technologies	1984
MS-DOS CLI	Microsoft	1981	FreeDOS Project	1998
Hayes AT cmd set	Hayes Micro	1982	Anchor Automation	1985
PostScript	Adobe	1985	Alladin Enterprises (Ghostscript)	1987
SMB	Microsoft	1992	Samba Project	1993
Win32	Microsoft	1993	Wine Project	1996
Java 2 class libs	Sun	1998	Google/Android	2008
Delicious web API	Delicious	2003	Pinboard	2009

Figure 2: Examples of APIs reimplemented by third parties.

B. The Free Reimplementation of APIs Drives Innovation by Promoting Software Interoperability.

API reimplementation enables interoperability and innovation. Interoperability is compatibility between computer systems and can take multiple forms.

Hardware-software interoperability is the ability of a program to run on a certain type of hardware. *See* David R. Owen, *Interfaces and Interoperability in Lotus v. Borland: A Market-Oriented Approach to the Fair Use Doctrine*, 64 Fordham L. Rev. 2381, 2395-96 (1996). Software-software interoperability refers to the ability of multiple software components to interact with each other. *Id.* This includes the ability for software written for one platform to run on another; for example, software written for Oracle's Java can run on Google's Android platform. To analogize, music written for the piano can be played (at least partially) on the keytar. User-software interoperability, also known as intersystem consistency,[5] allows a programmer who has learned the APIs on one platform to transfer these skills to another. In

[5] The District Court distinguished intersystem consistency from interoperability. *See Oracle Am., Inc. v. Google Inc.*, No. C 10-03561 WHA, 2016 WL 3181206, at *6 n.6 (N.D. Cal. June 8, 2016)

the keytar example, a musician familiar with the piano could adapt her existing knowledge of the keyboard interface to playing the keytar.

Crucially, a computer system is not, as Oracle and its amici suggest, either interoperable or not. Systems may achieve significant levels of interoperability without being fully compatible. For example, Android is highly interoperable with Oracle's Java. Most libraries, tools, and frameworks run equally well on Android and Oracle's Java, regardless of which platform they were originally written for.

APIs that are reimplemented and widely adopted can become *de facto* standards in their industries. The result is two-fold. First, newcomers need not expend resources to develop brand-new but redundant standards. Second, industry players compete around a single standard, allowing them to direct their efforts at improving their implementations of the standard. API reimplementation focuses innovation on improving the underlying substance of software through new or enhanced features.

The market rewards API development. Companies that invest in API development receive a first-mover advantage, enabling them to quickly develop relationships with developers and clients. Peter S.

Menell, *Rise of the API Copyright Dead?: An Updated Epitaph for Copyright Protection of Network and Functional Features of Computer Software* 161 (UC Berkeley Pub. Law Research, Paper No. 2893192), https://ssrn.com/abstract=2893192. Additionally, innovators benefit from a stronger reputation and overall recognition in the industry. *Id.*

C. Freely Reimplementable Interfaces Were Essential to the Development of Various Computer Technologies.

The long-standing industry practice of reimplementing existing APIs has allowed for rapid innovation in computer technology and given rise to important technologies that would otherwise not exist. IBM's first home computer came with the PC Basic Input/Output System (BIOS), firmware that provided an API to a system's underlying hardware. Because of the IBM PC's success, popular software like the spreadsheet program Lotus 1-2-3 was written specifically for it. Greg Williams, *Lotus Development Corporation's 1-2-3*, Byte Magazine, Dec. 1982, at 182.

While competitors like Phoenix Technologies could not directly copy IBM's BIOS implementation, *see Apple Comput., Inc. v. Franklin Comput. Corp.,* 714 F.2d 1240 (3d Cir. 1983), they developed their own compatible BIOS using clean room design—building their own

implementation from scratch. James Langdell, *Phoenix Says Its BIOS May Foil IBM's Lawsuits*, PC Magazine, July 1984, at 56. As a result, IBM's API was freely implemented by competitors to create IBM-compatible PCs able to run compatible software. This directly contributed to innovation and growth in the PC industry through cheaper, faster, and more portable computers. *See, e.g.*, Mark Dahmke, *The Compaq Portable*, Byte Magazine, Jan. 1983, at 30-36. And because these computers were interoperable, software developers could distribute their software widely and compete directly on features and price. Since APIs have historically been freely reimplemented, successful APIs often outlive platforms for which they were originally created. For example, though IBM no longer makes PCs, all mainstream desktop and laptop PCs run on an implementation of an API derived from IBM's standard.

The interoperability of the C programming language helped establish its ongoing success in the software industry.[6] Programs

[6] The Institute of Electrical and Electronics Engineers ranked C as the most popular programming language in 2016. Stephen Cass, *The 2016 Top Programming Languages,* IEEE Spectrum (July 26, 2016, 16:00 GMT), http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages.

written in C use the C standard library API to execute their functions and operate the computer on which they run. Though the C programming language was originally closely tied to the UNIX operating system, C programmers can now write software for any system that provides a reimplementation of the C standard library. Third party implementations[7] of the C standard library exist today on every main operating system, allowing C—as well as the countless applications, tools, and communities based on C—to flourish.

The Java platform itself reimplemented preexisting APIs for interoperability as well. The Java math APIs (java.lang.Math) were largely reimplementations of the C standard library, ensuring that C programmers could easily migrate to Java. Similarly, the Java "regular expression" APIs (java.util.regex) were copied from the Perl

---

[7] Microsoft reimplemented the C standard library for Windows as part of the Microsoft C Run-Time Library. *C Run-Time Libraries*, Microsoft Developer Network, http://msdn.microsoft.com/en-us/library/abx4dbyh(v=vs.80).aspx (last visited May 30, 2017). Google developed Bionic, an implementation for its Android operating system. Ed Burnette, *Patrick Brady Dissects Android*, ZDNet (June 4, 2008), http://www.zdnet.com/article/patrick-brady-dissects-android. The GNU Project developed its own library, glibc, for Unix-like operating systems. GNU, *The GNU C Library (glibc)*, https://www.gnu.org/software/libc (last visited May 30, 2017).

programming language, ensuring that existing regular expressions—essentially mini-programs that define search patterns—would continue to work, and that knowledge of Perl regular expressions would transfer directly to Java.

Interoperability benefits consumers as well. Wine, an open source project started in 1993, is a compatibility layer that allows Unix-like operating systems such as Linux and macOS to run software written for Microsoft Windows. WineHQ, *About Wine*, https://www.winehq.org/about (last visited May 30, 2017). Wine achieves this through a reimplementation of the Windows API—without a license or agreement from Microsoft. Today, millions of users use Wine to run Windows software that would otherwise be incompatible with their systems. *Id.*

The software industry does not, contrary to assertions by Oracle and its amici, rely on copyright protection for APIs. The general understanding has always been that the free reimplementation of APIs is legal and beneficial to the industry. Jonathan Schwartz, the former CEO of Sun Microsystems, initially applauded Android, Jonathan Schwartz, *Congratulations Google, Red Hat and the Java Community!*, Jonathan's Blog (Nov. 5, 2007), http://web.archive.org/web/

20101023072550/http://blogs.sun.com/jonathan/entry/congratulations_google, and later testified that the expectation was that Java APIs were free for competitors to implement. Trial Tr. at 501-20.

II. API Reimplementation Encourages Innovation, Competition, and Interoperability, Fulfilling the Public Policy Goals of Copyright and Fair Use.

Allowing third parties to freely reimplement APIs comports with the goal of copyright law because it promotes rapid innovation, encourages competition, and allows for interoperability.

A. Copyright and Fair Use are Designed to Incentivize Creation and Innovation.

The goal of copyright is to incentivize creation and "promote the Progress of Science and useful Arts." U.S. Const. art. I, § 8, cl. 8. Copyright, however, does not protect facts or ideas, or works under a sweat-of-the-brow theory. *Feist Pubs., Inc. v. Rural Tel. Serv. Co., Inc.*, 499 U.S. 340, 349-50 (1991). While copyright assures authors the right to their original expression, "[it] encourages others to build freely upon the ideas and information conveyed by a work." *Id.* at 349-50.

Similarly, fair use promotes creativity and innovation. Fair use is a flexible and adaptable remedy that permits copying for "uses that promise to boost, rather than diminish, creativity overall." Clark D.

Asay, *Software's Copyright Anticommons*, 66 Emory L.J. 265, 273 (2017). By limiting infringement liability, the fair use doctrine recognizes that some forms of copying benefit the public good, and that encouraging such use better serves copyright's intended goals.

B. Copyright Law Provides Specific Carve-Outs for Interoperability in Computer Software.

Copyright allows carve-outs for interoperability and the reuse of standards, particularly in software. Judges and legislators alike have recognized that these carve-outs are essential to promoting innovation in deciding reverse-engineering cases, interpreting the idea-expression dichotomy, and crafting the Digital Millennium Copyright Act's (DMCA) anti-circumvention provisions. This backdrop should guide this Court's fair use analysis.

Courts and legislators have interpreted fair use in reverse-engineering cases to allow for interoperability and compatibility. The Ninth Circuit has held that copying computer code to access a computer program's functional elements in order to achieve interoperability is fair use. *Sega Enters. Ltd. v. Accolade, Inc.*, 977 F.2d 1510, 1527-28 (9th Cir. 1992), *as amended* (Jan. 6, 1993). Similarly, in *Sony Computer Entertainment, Inc. v. Connectix Corp.*, the Ninth Circuit held that a

form of reverse engineering was fair use. 203 F.3d 596, 598-99 (9th Cir. 2000) (holding Connectix's reverse engineering of Sony's basic input-output system (the BIOS system) was fair use because it allowed Connectix to produce an interoperable Virtual Game Station, enabling games designed for Sony PlayStation to be compatible with other machine platforms). In addition, the DMCA includes legislative carve-outs for interoperability specifically for reverse engineering when "necessary to achieve interoperability of an independently created computer program with other programs." 17 U.S.C. § 1201(f)(1) (2015).

Courts and scholars have similarly emphasized that software copyright is afforded less protection due to the functional demands of copying and reusing software *interfaces* to achieve interoperability. *See e.g.*, Peter S. Menell, *An Analysis of the Scope of Copyright Protection for Application Programs*, 41 Stan. L. Rev. 1045, 1048 (1989) (arguing that courts should liberally apply § 102(b)'s idea-expression dichotomy to software interfaces). Though not an issue on this appeal, courts have previously held command interfaces in a software program were not copyrightable. *Lotus Dev. Corp. v. Borland Int'l, Inc.*, 49 F.3d 807, 809 (1st Cir. 1995), *aff'd*, 516 U.S. 233 (1996). The *Lotus* court emphasized

that its decision became "clearer when one considers program compatibility," especially with regards to users who "must learn how to perform the same operation in a different way for each program used." *Id.* at 817-18. Though recognizing the creativity involved in labeling the structure of the command hierarchy, "once such function names were learned by programmers, however, they took on tremendous importance to the user community." Menell, *Rise of the API Dead*, *supra*, at 148.

As applied to software, copyright law's limiting doctrines have always recognized the important role of interoperability in computer software programs. As this Court noted, the copyrightability question focuses on the "compatibility needs and programming choices of the party claiming copyright protection," *Oracle Am., Inc. v. Google Inc.*, 750 F.3d 1339, 1371 (Fed. Cir. 2014), while the interoperability concerns are "expressly noted" as relevant to the fair use analysis. *Id.* at 1377. The backdrop of copyright law's carve-outs for interoperability is instructive to this Court's fair use analysis. Fair use recognizes that some forms of express copying encourage creativity, competition, and innovation, which are all served through allowing interoperability.

Notably, Judge Boudin, in his concurring opinion in *Lotus*, highlighted that notwithstanding the copyrightability decision, copying the command hierarchy was also a "privileged use," an analogue to fair use. *Lotus*, 49 F.3d at 821 (Boudin, J. concurring). Judge Boudin emphasized that reusing the existing command hierarchy provided users familiar with the former interface the "option to exploit their own prior investment," while offering new users an "arguably more attractive" command interface of its own. *Id.* at 821. The same underlying goals of interoperability and reuse of programmer knowledge should guide this Court's fair use analysis of API reimplementation.

C. Overturning the Decision Below Would Stifle Innovation and Disrupt Well-Settled Industry Practices.

This Court should rule that API reimplementations that achieve interoperability are fair use. Not only is this conclusion dictated by the law, but it is critical to quell uncertainty and preserve long-standing foundations in the software industry. *See* Oren Bracha & Talha Syed, *Beyond Efficiency: Consequence-Sensitive Theories of Copyright*, 29 Berkeley Tech. L.J. 229, 315 n.90 (2014) ("[P]roperly understood and

applied fair use can yield a sufficient level of predictability and avoid chilling effects."). Fair use for API reimplementations is necessary for continued innovation in the computing and software industry.

API reimplementations are essential to preserving programmer mobility and investment in knowledge, preventing lock-in, and encouraging the development of new features and capabilities. Restricting reuse of APIs would stifle competition by preventing the development of interoperable programs and systems. Moreover, a definitive ruling of fair use will not reduce incentives to create, as API reimplementation has led to rapid innovation for decades.

1. API Reimplementation Enables Intersystem Consistency, Which Preserves Investment in Knowledge and Encourages Standardization.

Fair use for API reimplementations ensures that a programming language can be used in multiple contexts, and by programmers already familiar with the language. In creating the Android platform, Google retained essential Java APIs. While Google changed the underlying implementation of the APIs, the specifications—including the function names and the types of inputs and outputs—remained the same. This

means that Java programmers can write software for both Oracle's and Google's platforms.

API methods are given straightforward names describing the functions they achieve. These function names, and their corresponding labels of inputs and outputs comprise the API. Programmers learn these functions as part of the language and can apply this knowledge in any platform that has implemented those APIs. Unrestricted API reimplementation protects programmers' reliance on common naming conventions for widespread functionalities. It prevents fragmentation, where industry participants use different APIs for the same purpose, needlessly forcing programmers to learn new interfaces. Instead, programmers can focus on improving product features and functionality, which directly benefits consumers.

As this Court has already noted, reimplementing APIs to "capitalize on a preexisting community of Java programmers" is a fair use issue. *Oracle*, 750 F.3d at 1371-72. The Java platform itself reimplemented APIs from C standard libraries and Perl for ease of use by programmers familiar with those languages. API reimplementations

for the purpose of preserving a programmer's knowledge investment in a common vocabulary should be allowed under fair use.

2. The Freedom to Reimplement APIs Encourages Competition and Innovation, and Reduces the Potential for Lock-In and Fragmentation.

API reimplementation encourages competition and innovation by promoting standardized, interoperable platforms, and reduces the risks of lock-in or fragmentation. Interoperable software interfaces are especially important because developers and consumers benefit when their devices and software can communicate seamlessly. However, these same network effects can become detrimental if APIs cannot be freely reimplemented. *See, e.g.*, Menell, *Rise of the API Dead*, *supra* at 15 (discussing the network effects of computer hardware, software, and programming languages, and concerns that "companies could use API strategies to lock-in consumers and lock-out competitors").

First, restricting API reimplementation can encourage platform lock-in. Custom and practice in the software space has relied on open and freely reimplementable APIs, allowing for rapid improvements in platform functionality and design. The alternative to this regime of open APIs encourages lock-in of a single API, with limited opportunities

to create new platforms or bridge old programming languages with new hardware, platforms, or functionalities. As competition is locked out, incumbents are less incentivized to improve their products. Users also face high costs in switching to newer platforms, even if the new platform offers significant technological advantages.

Second, restricting API reimplementations can cause fragmentation. A company considering licensing a costly API may instead choose to compete by designing its own API. Though the competing API may be similar in functionality, it would not be compatible or interoperable. While there are multiple ways of creating an API, "[t]he last thing the API world needs is more strange and unique ways to connect services: that slows everything down, introduces friction that saps the energy from more useful endeavors like testing out new business models, and leads directly to more error-prone software." Uri Sarid, *A Non-Apocalypse: APIs, Copyright, and Fair Use*, Wired (May 13, 2014, 10:01 AM), https://www.wired.com/insights/2014/05/non-apocalypse-apis-copyright-fair-use. The purpose of reimplementing an API is to maintain continuity within the

programming language as a whole and to preserve compatibility of existing programs and functions on new platforms.

Third, restricting API reimplementation could lead to chilling effects and underuse of common software because of software's interconnected nature. Software creators and programmers build on top of one another. The resulting layers of rights owners could be paralyzing for future users who would need to license each software component, and might be subject to holdup. This paralysis could lead to the underuse of common platforms and otherwise socially beneficial resources (what is referred to as the "anticommons" problem). Asay, *Copyright Software Anticommons*, *supra*, at 267-68. The custom and practice of openly reimplementable APIs, and the resulting rapid innovation the industry achieved as a result, highlights the detrimental effect this paralysis could have.

III. Because APIs are Inherently Functional, the Second Fair Use Factor Weighs Strongly in Favor of Fair Use and Should Guide the Remaining Fair Use Analysis.

While all four fair use factors support a finding of fair use in this case, the second factor in particular should drive this Court's analysis of what constitutes fair use of software. The second fair use factor

considers the "nature of the copyrighted work." 17 U.S.C. § 107(2) (2015). Though this factor is often considered later in a court's fair use analysis, it is more instructive in the software context to begin with the nature of APIs, which are essential and functional components of programming languages.

Reusing APIs helps achieve interoperability among software and hardware platforms, and is necessary for programmers to make use of the Java programming language. This same functionality of APIs allows for innovative reimplementations such as Android that adapted the Java APIs to new, transformative contexts.

A. Computer Software's Functional Nature Must Guide the Rest of the Fair Use Inquiry.

Computer software's inherent functionality drives both the second fair use factor and the rest of the fair use inquiry. Computer programs and their API structures contain functional elements that are "dictated by considerations of efficiency or other external factors . . . [which] should be afforded a lower degree of protection than more traditional

literary works." *Oracle*, 750 F.3d at 1375.[8] As this Court has already noted, Google's reimplementation of the Java APIs to achieve interoperability bears heavily on the second fair use factor. *Id.* at 1377. The functional nature of APIs is essential to any reimplementation. Unlike other copyrightable materials, "software's functional characteristics make it unlike other copyrightable materials in key respects—the most important of which is that any given software component by definition has a singular computing purpose." Clark D. Asay, *Transformative Use in Software*, 70 Stan. L. Rev. Online 9, 17 (2017). The functional nature of APIs is necessarily reused as part of the language's core interface. This allows reimplementations to achieve interoperability.

B. APIs are an Integral Part of the Unprotected Java Programming Language, Serve a Predominantly Functional Purpose, and Allow for Interoperability.

APIs are a functional and essential part of the Java language. Similar to an instruction manual, APIs are documented in any book

[8] The Ninth Circuit's recent decision in *Bikram Yoga Collective of India, L.P. v. Evolation Yoga, LLC,* finding a sequence of yoga poses uncopyrightable, affirmed the principle that functional elements of a work are considered broadly—and protected narrowly—throughout the Copyright Act. 803 F.3d 1032, 1034.

that teaches a computer programming language. Sedgewick & Wayne, *supra*, at 28 ("A critical component of modular programming is documentation that explains the operation of library methods that are intended for use by others."). To a programmer, a strong identifying feature of any programming language is the set of API methods being invoked. Both technically and practically, the core APIs of a language are inseparable from the language.

To make effective use of the Java language, Android had to reimplement more than just the API packages mentioned in the Java language specification, because API packages are interdependent. The APIs at issue in this case, like all APIs, are specifications: they provide instructions for how one module in a system interacts with others in the system, providing the names of the functions, the input and output variables, and how one function interacts with related modules. Multiple API packages are required to effectively use the Java language. The Java language specification directly relies on 60 classes consisting of more than 750 public methods and fields spread across 3 packages. All of these packages have dependencies: the declarations of those methods in turn rely on other classes in other packages. Because

of these dependencies, the exact number of "core" packages required to use the Java language is higher than was stipulated. Notwithstanding this stipulation, use of the unprotected Java language necessitates the reuse of many interdependent API packages.

Reusing these API declarations is necessary to achieve interoperability. This Court emphasized that interoperability considerations bear on the second fair use factor particularly "with respect to those core packages which it seems may be necessary for anyone to copy if they are to write programs in the Java language. And, it may be that others of the packages were similarly essential components of any Java language-based program." *Oracle*, 750 F.3d at 1377. Reimplementing the Java language requires reusing a number of core and interrelated API packages.

The creativity involved in designing an API does not diminish its utilitarian goal. Oracle and its amici stress the difficulty and creativity involved in creating an API. But much of Oracle's API design—like most API designs—draws heavily on existing practice, facilitating the interoperability in new software and hardware platforms.

IV. Android's Incorporation of Java APIs Is Fair Use Because It Is Transformative and Achieves Substantial Interoperability.

Android is transformative because it brought the Java programming language and APIs to a new context and platform. Under the first fair use factor, which considers "the purpose and character of the use," 17 U.S.C. § 107(1), the degree to which a use is transformative weighs heavily on the purpose and the character of that use. *Campbell*, 510 U.S. at 586. A use is transformative if it "adds something new, with a further purpose or different character, altering the first with new expression, meaning, or message." *Id.*

Moreover, Android facilitates interoperability with the existing Java platform, which encourages software reuse and innovation. That software enables interoperability supports a finding of fair use. *See generally* Edward Lee, *Technological Fair Use*, 83 S. Cal. L. Rev. 797 (2010) (arguing that courts should broadly apply fair use doctrine to protect technological innovation).

A. Oracle's Constrained Interpretation of Transformative Use is Unworkable in Software.

This Court should reject Oracle's cramped view of transformativeness, as it would eliminate the possibility of

transformative use in software in all but non-functional uses. Oracle and its amici repeatedly argue that Google's reimplementation of Java APIs is not transformative because the APIs serve the same purpose in Android. Oracle relies extensively on non-software fair use cases to argue that adapting one work for use in another for the same purpose is not transformative. Plaintiff-Appellant Br. at 29-37. This overly constrained interpretation of fair use ignores the fact that software's functional nature dictates its reuse.

This Court should take into account the unique aspects of software when evaluating transformativeness. *See* Asay, *Transformative Use in Software*, *supra*, at 15-17. That software reuse must involve an element of functional replication should not preclude transformativeness. Software is inherently functional. Unlike in other creative works, APIs in isolation can only serve one functional purpose because they are technical specifications with defined meanings. Thus, an API reimplementation—a functional reuse—must necessarily inherit functional characteristics of the original work. The analysis of whether software is transformative should examine the extent to which the new

work, as a whole, transcends the original through new purpose and context.

B. Android Reimplements Java APIs in a New Context and Is Tailored for New Uses and Constraints.

Android's use of Java APIs is transformative because Android occupies an entirely new context: smartphones and tablets. Android does not merely implement Java APIs on a new format or medium; it is an entirely new operating system for mobile devices that selectively incorporates and augments the Java API packages.[9] Android revolutionized the mobile software landscape. *See* Claudio Giachetti, *Competitive Dynamics in the Mobile Phone Industry* 65-67 (2013). Though Android incorporated Java APIs, software development for Android is different from software development for server and desktop environments. Android reflects these differences through its implementing code, its own virtual machine,[10] and Android-specific APIs.

---

[9] Oracle's Java ME has little in common with Java SE. It runs a small subset of the Java platform and is largely incompatible.

[10] The virtual machine is a piece of software that allows Java programs to run on certain hardware.

Technical and practical concerns set mobile and desktop environments apart. First, smartphones use touchscreens, whereas some Java APIs presuppose a mouse and keyboard user interface. Instead of reimplementing these APIs, Android crafted new touchscreen-specific APIs. Second, smartphones utilize various sensors, including GPSs, accelerometers, cameras, compasses, and microphones. The Java APIs did not adequately address these peripherals, which are less central to the desktop setting. Third, smaller battery sizes mean that smartphones are nearly always starved for electric power. Relatedly, mobile phones run on a different type of computer processor (ARM chips), which are more energy efficient. These significant differences constrain API implementation and alter how mobile developers interact with the platform. Though Android used the Java language and reimplemented many essential Java APIs, Android incorporated many new APIs to form a platform that transcended the original.

Android is illustrative of how API reimplementation can encourage development of compatible software in new and innovative contexts. This is especially true as hardware and software platforms

evolve. *See* Asay, *Copyright Software Anticommons*, *supra*, at 314 ("For instance, reuse of software technologies such as software interfaces or objects in order to promote compatibility more generally will often result in the use of these software technologies in completely new contexts, such as enabling otherwise distinctive software services to exchange data in an ever-expanding Internet of Things economy."). Restricting freely reimplementable APIs will limit innovation and interoperability of future hardware and software platforms.

C. Android's Use of Java APIs Enables Interoperability.

The character and purpose of Android's use of the Java API is to facilitate interoperability. Interoperability includes compatibility, both between and within platforms. The reimplementation of Java's API in the Android platform achieves significant interoperability in two ways. First, programmers familiar with Java can adapt their knowledge of Java and its APIs to program smartphones. Second, hundreds of millions of lines of existing Java software that use Java APIs can run on both platforms. Contrary to what Oracle and its amici claim, most software written for Oracle's Java—besides certain use cases inapplicable to mobile phones—runs without modification on Android.

See Figure 3 (listing libraries that interoperate with both Java and Android platforms).

Library Name	**Description**
Guava	Core library
Apache Commons-math	Math library
Bouncycastle	Crypto library
Okio	I/O library
Guice, Dagger	Dependency injection frameworks
RxJava	Concurrency framework
Timber	Logging framework
Bugsnag	Analytics and exception tracking framework
Retrofit	REST adapter
Jackson, Gson	JSON parser
Nano Proto	Protocol buffer parser
OkHttp, Apache HttpClient	HTTP clients
Jetty	HTTP server
Gradle, Maven	Build systems
JUnit	Unit testing framework

Figure 3: List of large and complex third party libraries that run equally well on both Java and Android platforms.

Interoperability favors a finding of fair use. The reuse of APIs enables software technologies to expand into new contexts and develop new features. Android is not, as Oracle claims, simply a copy of Oracle's product. In fact, Oracle, and previously Sun Microsystems, had failed where Android succeeded. Because Android was largely interoperable from the outset, the platform was quickly adopted in the mobile development community. In addition, Android was licensed more

permissively than Oracle's Java platform. This gave third parties—including carriers and other manufacturers—the freedom to develop proprietary features on top of Android. Ryan Paul, *Why Google Chose the Apache Software License Over GPLv2 for Android*, Ars Technica (Nov. 6, 2007, 7:26 AM), https://arstechnica.com/?post_type=post&p=79053.

Android represents an enormous contribution to the Java community. The existence of the Android platform gave superpowers to every Java developer: Java programmers could now apply their Java skills to new contexts, including programming modern smartphones and creating mobile applications. Android breathed new life into an aging Java platform, and has been at the forefront of smartphone innovation since its introduction in 2008. The functional interoperability embedded in Android was integral to growth in creative expression in the mobile application space.

D. Android's Use of Java APIs Was Necessary to Achieve Its Purpose.

The reimplementation of Java APIs in Android was necessary to achieve its transformative purpose. Though the Java language and its APIs are technically distinct, they cannot be separated as a practical

matter because the essential Java APIs have become a fixture of the language itself. Android's reimplementation of Java APIs was necessary if it was to make use of the Java language in a meaningful way. It would not have been reasonable, or even conceivable for Google to provide different names and parameters for all the methods, classes, and packages in the APIs. This would have violated common sense, engineering best-practices, and decades of standard industry practice.

The Java APIs are in no way the "heart" of the Java platform. When considering the substantiality of the Java platform used in Android, the interface, together with the implementing code, should be examined as a single work. *See* Asay, *Copyright Software Anticommons*, supra, at 319-22. After all, an interface serves a functional purpose but cannot operate without an implementation. *See id.* While the API declarations are necessary to achieve interoperability, they constitute a negligible portion of the code necessary to define and implement the interfaces. Within the Android platform, the declarations comprise less than a tenth of a percent of the code. By reusing a tiny portion of Java's platform for interoperability, Google was able to develop an innovative and transformative platform.

CONCLUSION

For the aforementioned reasons, Google's reimplementation of the Java APIs was fair use, and this Court should affirm the judgment below.

/s/ Phillip R. Malone

Phillip R. Malone
Jef Pearlman
Juelsgaard Intellectual Property and Innovation Clinic
Mills Legal Clinic at Stanford Law School
559 Nathan Abbott Way
Stanford, CA 94305
Telephone: 650-725-6369
Fax: 650-723-4426

Attorneys for Amici Curiae

CERTIFICATE OF SERVICE

I certify that I served a copy on counsel of record on May 30, 2017 by Electronic Means (by CM/ECF).

/s/ Phillip R. Malone

Phillip R. Malone
Juelsgaard Intellectual Property and Innovation Clinic
Mills Legal Clinic at Stanford Law School
559 Nathan Abbott Way
Stanford, CA 94305
Telephone: 650-725-6369
Fax: 650-723-4426
E-mail: pmalone@law.stanford.edu

Attorney for Amici Curiae

CERTIFICATE OF COMPLIANCE

This brief complies with the type-volume limitation of Federal Rule of Appellate Procedure 32(a)(7)(B) or Federal Rule of Appellate Procedure 28.1(e). The brief contains 6252 words, excluding the parts of the brief exempted by Federal Rule of Appellate Procedure 32(a)(7)(B)(iii) and Federal Circuit Rule 32(b).

This brief complies with the typeface requirements of Federal Rule of Appellate Procedure 32(a)(5)(A) and the type styles requirements of Federal Rule of Appellate Procedure 32(a)(6). The brief has been prepared in a proportionally spaced typeface using Microsoft Word, in 14-point Century Schoolbook font.

/s/ Phillip R. Malone

Phillip R. Malone
Juelsgaard Intellectual Property and Innovation Clinic
Mills Legal Clinic at Stanford Law School
559 Nathan Abbott Way
Stanford, CA 94305
Telephone: 650-725-6369
Fax: 650-723-4426

Attorney for Amici Curiae

APPENDIX—LIST OF *AMICI CURIAE*

(In alphabetical order)

Amici are signing this brief on their own individual behalf and not on behalf of the companies or organizations with whom they are affiliated. Those affiliations are only for identification. This includes those amici indicated by an asterisk (*), who are presently Google employees, consultants, and/or directors. Those amici are signing this brief as individual computer scientists whose work in the field long preceded their affiliation with Google. They are not signing this brief on behalf of Google or at Google's request.

1. Harold Abelson.* Dr. Harold "Hal" Abelson is a Professor of Electrical Engineering and Computer Science at MIT, a fellow of the IEEE, and a founding director of both Creative Commons and Public Knowledge. He directed the first implementation of the Logo computing language for the Apple II, which made the language widely available on personal computers beginning in 1981, and published a popular book on Logo in 1982. Abelson co-developed MIT's introductory computer science subject, which included innovative advances in curricula designed for students pursuing different kinds of computing expertise. These curricula had a worldwide impact on university computer science education. Notable awards include the Bose Award (MIT School of Engineering, 1992), the Taylor L. Booth Education Award (IEEE-CS, 1995), and the SIGCSE 2012 Outstanding Contribution to Computer Science Education (ACM, 2012). Abelson holds an A.B. from Princeton University and a Ph.D. in mathematics from MIT.

2. Tom Ball.* Tom Ball is a Staff Engineer at Google, working on Java-based developer tools. He was previously a Distinguished Engineer at Sun Microsystems, and a member of the JDK team that first released Java publicly. He wrote the first Java debugger (jdb), was a member of the AWT and Swing teams, and developed the Jackpot automated refactoring tool designed by James Gosling. His current project is J2ObjC (http://j2objc.org), an open source tool that converts Java source to Objective-C for use by iOS applications (which cannot run Java).

3. Brian Behlendorf. Brian Behlendorf is Executive Director of Hyperledger, an open source blockchain technology collaborative based at the Linux Foundation. He also serves as Chairman of the Board of the Electronic Frontier Foundation, and a member of the boards of the Mozilla Foundation and Benetech. He also co-founded the Apache Software Foundation, has worked as CTO for the World Economic Forum, advised and served the White House on open data and open source software issues, and co-founded a string of successful startups.

4. Gordon Bell. Gordon Bell is a Microsoft researcher emeritus, and former Digital Vice President of R&D, where he led the development of the first mini- and time-sharing computers. As NSFs founding Director for Computing (CISE), he led the plan for NREN (Internet). Bell has researched and written about computer architecture, high-tech startup companies, and lifelogging. He is a member of the American Academy of Arts and Sciences, the National Academy of Engineering, National Academy of Science, and received The 1991 National Medal of Technology. He is a founding trustee of the Computer History Museum, Mountain View, CA.

5. Jon Bentley. Jon Bentley's research interests include programming techniques, algorithm design, and the design of software tools and interfaces. He has written three books on programming and over a hundred articles on a variety of topics, ranging from the theory of algorithms to software engineering. He received a B.S. from Stanford in 1974 and an M.S. and Ph.D. from the University of North Carolina in 1976, then taught Computer Science at Carnegie Mellon for six years. He joined Bell Labs Research in 1982, where he became a Distinguished Member of Technical Staff. He left Bell Labs in 2001 to join Avaya Labs research, from which he retired in 2013. He has been a visiting faculty member at West Point and Princeton, and has been a member of teams that have shipped software tools, telephone switches, telephones and web services. He holds over 40 US Patents. In March 2000 he received the Dr. Dobb's Excellence in Programming Award for advancing the craft of computer programming.

6. Matthew Bishop. Matthew Bishop received his Ph.D. in computer science from Purdue University, where he specialized in computer security, in 1984. He is on the faculty at the Department of Computer Science at the University of California at Davis. His main research area is the analysis of vulnerabilities in computer systems, including modeling, detecting, and analyzing them. Currently, he has research projects involving data sanitization, modeling election processes, and analyzing attacks. He is co-leading an education project aimed at improving the practice of programming using a "secure programming clinic" to help students improve the robustness and security of their programs. He has been active in the area of UNIX security since 1979, and has presented tutorials at SANS, USENIX, and other conferences. He also has done work on electronic voting, and was one of the two principle investigators of the California Top-to-Bottom Review, which performed a technical review of all electronic voting systems certified for use in the State of California. His textbook, *Computer Security: Art and Science* (Addison-Wesley, 2002), is used at many academic institutions throughout the world.

7. Joshua Bloch. Joshua Bloch is an expert on API design, with over a quarter century of experience. He is a Professor of Computer Science at Carnegie Mellon University. Previously, he was Chief Java Architect at Google, a Distinguished Engineer at Sun Microsystems, and a Senior Systems Designer at Transarc Corporation. He led the design and implementation of numerous Java APIs and language features, including the award-winning Java Collections Framework. He is the author of several books, including the bestselling, Jolt Award-winning *Effective Java* (Addison-Wesley, 2001; Second Edition, 2008), the de facto standard guide to Java best practices. He served on the National Academies CSTB Certifiably Dependable Software Committee. He holds a B.S. from Columbia and a Ph.D. in Computer Science from Carnegie Mellon University.

8. Dan Boneh. Dan Boneh is a Professor of Computer Science at Stanford University, where he heads the applied cryptography group. Dr. Boneh's research focuses on applications of cryptography to computer security. His work includes cryptosystems with novel

properties, security for mobile devices, web security, and cryptanalysis. He is the author of over a hundred publications in the field and is a recipient of the 2013 Gödel prize, the Packard Award, the Alfred P. Sloan Award, the RSA award in mathematics and five best paper awards. In 2011 Dr. Boneh received the Ishii award for industry education innovation. Dr. Boneh's wife is a current Google employee.

9. Gilad Bracha.* Gilad Bracha is the creator of the Newspeak programming language and a software engineer at Google. A well known researcher in the area of object-oriented programming languages, he was awarded the senior Dahl-Nygaard prize in 2017. Previously, he was a VP at SAP Labs in Palo Alto, a Distinguished Engineer at Cadence, and a Computational Theologist and Distinguished Engineer at Sun. He has authored or co-authored several books including the Java Language and Virtual Machine Specifications, and the Dart Programming Language. Prior to joining Sun, he worked on Strongtalk, the Animorphic Smalltalk System. He received his B.Sc in Mathematics and Computer Science from Ben Gurion University in Israel and a Ph.D. in Computer Science from the University of Utah..

10. Eric Brewer.* Eric Brewer pioneered early "cloud" computing starting in the 1990s with research on large-scale services implemented on clusters of commodity servers, for which he was elected to the National Academy of Engineering. In 1996, Brewer co-founded Inktomi Corporation, an early search engine that also influenced the modern Internet architecture. He formulated the CAP theorem, one the tenets of modern distributed systems. In 2000, working with President Clinton, he led the development of usa.gov, the primary federal portal. He is a tenured professor in the Computer Science department at UC Berkeley, but is currently on leave at Google as VP, Infrastructure. Brewer received a BS in EECS from UC Berkeley, and an M.S. and Ph.D. from MIT.

11. Frederick Brooks. Brooks is the Kenan Professor of Computer Science (Emeritus) at University of Northern Carolina at Chapel Hill. As Corporate Project Manager for IBM's System/360 (mainframe) computer family hardware and the Operating

System/360 software, he in 1964 switched the standard computer byte size from 6 to 8 bits. He was an architect of the Stretch and Harvest supercomputers. He founded UNC's Computer Science Department. He's researched computer architecture, software engineering, the design process, and graphics virtual environments. He wrote The Mythical Man-Month, The Design of Design, and with G.A. Blaauw, Computer Architecture. Honors include the National Medal of Technology, the ACM Turing award, the National Academies of Engineering and Science, and British and Dutch academies.

12. Rick Cattell. R. G. G. "Rick" Cattell is an independent consultant in database systems. He previously worked as a Distinguished Engineer at Sun Microsystems. Dr. Cattell served for 20 years at Sun Microsystems in management and senior technical roles, and for 10 years in research at Xerox PARC and Carnegie Mellon University. He is best known for his contributions in database systems and middleware, including database scalability, Enterprise Java, object/relational mapping, object-oriented databases, and database interfaces. At Sun he instigated Enterprise Java, JDBC, Java DB, and Java Blend, and contributed to many Java APIs and products. He previously developed Xerox PARC's Cedar DBMS, Sun's Simplify database GUI, and SunSoft's CORBA-database integration. He is a co-founder of SQL Access (predecessor to ODBC), founder and chair of the Object Data Management Group (ODMG), author of the world's first monograph on object/relational and object databases, recipient of the ACM Outstanding Ph.D. Dissertation Award, and an ACM Fellow.

13. Vinton G. Cerf.* Vinton G. "Vint" Cerf is vice president and Chief Internet Evangelist for Google, where he contributes to global policy development and the continued spread of the Internet. Widely known as one of the "Fathers of the Internet," Cerf is the co-designer of the TCP/IP protocols and the architecture of the Internet. He has served in executive positions at MCI, the Corporation for National Research Initiatives, the Defense Advanced Research Projects Agency, and on the faculty of Stanford University. Cerf served as chairman of the board of the Internet Corporation for Assigned Names and Numbers (ICANN) from 2000-

2007. Cerf is a Fellow of the IEEE, ACM, and AAAS, the American Academy of Arts and Sciences, the International Engineering Consortium, the Computer History Museum, and is a member of the National Academy of Engineering. He is a former President of the ACM and Founding President of the Internet Society. President Obama appointed him to the National Science Board in 2012. Cerf is a recipient of numerous awards and commendations in connection with his work on the Internet, including the US Presidential Medal of Freedom, US National Medal of Technology, the Queen Elizabeth Prize for Engineering, the ACM Turing Award, Officer of the Legion d'Honneur and 29 honorary degrees. In December 1994, People magazine identified Cerf as one of that year's "25 Most Intriguing People." Cerf holds a B.S. from Stanford, and an M.S. and Ph.D. from UCLA.

14. William Cook. William Cook is an Associate Professor in the Department of Computer Sciences at the University of Texas at Austin. His research is focused on object-oriented programming, programming languages, modeling languages, and the interface between programming languages and databases. Prior to joining UT in 2003, Dr. Cook was Chief Technology Officer and co-founder of Allegis Corporation. He was chief architect for several award-winning products, including the eBusiness Suite at Allegis, the Writer's Solution for Prentice Hall, and the AppleScript language at Apple Computer. At HP Labs his research focused on the foundations of object-oriented languages, including formal models of mixins, inheritance, and typed models of object-oriented languages. He completed his Ph.D. in Computer Science at Brown University in 1989. He received the Dahl-Nygaard Senior Prize in 2014 for his contributions to the theory and practice of object-oriented programming.

15. Mark Davis.* Dr. Mark Davis has been the Chief Internationalization Architect at Google since 2006, focusing on effective and secure use of Unicode, software internationalization libraries, and related areas. Dr. Davis is also the co-founder and has been president of the Unicode Consortium since its inception in 1991, and is a key technical contributor to the Unicode specifications. In 2003, he founded the Unicode Common Locale

Data Repository (CLDR) project, the standard repository for locale data worldwide. He is co-author of BCP 47 ("Tags for Identifying Languages"), used to identify human languages in all XML and HTML documents, and in all modern programming libraries. Mark provided the original architecture of ICU, the premier Unicode software internationalization library, and the Java internationalization libraries. At IBM, he was Chief Software Globalization Architect. At Taligent, he was manager and architect for the international frameworks. At Apple, he co-authored the first Macintosh system to support Japanese (KanjiTalk), and authored the first Macintosh Arabic and Hebrew systems. Mark holds a Ph.D. from Stanford University and a B.A. from the University of California, Irvine.

16. Miguel de Icaza. Miguel de Icaza is currently a Distinguished Engineer at Microsoft and was an early contributor to Linux projects. In 1997, he cofounded the GNOME project, with the goal to create a completely free desktop environment. In 2001, he co-founded and directed the Mono Project, with the goal to reimplement Microsoft's .NET development platform on Linux. He has started two companies: Ximian in 1999, which focused on the Linux desktop and was sold to Novell in 2003; and Xamarin which was founded in 2011 to build mobile development tools and was sold to Microsoft in 2016. He has received numerous awards and recognitions including: the Free Software Foundation Free Software Award, the MIT Technology Review Innovator of the Year Award, and was named one of Time Magazine's 100 innovators for the new century.

17. Jeffrey Dean.* Jeffrey Dean joined Google in 1999 and is currently one of two Senior Fellows in the company, where he leads the Google Brain team, Google's artificial intelligence research team. He has co-designed/implemented five generations of Google's crawling, indexing, and query serving systems, and co-designed/implemented major pieces of Google's initial advertising and AdSense for Content systems. He is also a co-designer and co-implementor of Google's distributed computing infrastructure, including the MapReduce, BigTable and Spanner systems, protocol buffers, LevelDB, systems infrastructure for statistical machine translation, the TensorFlow

open-source machine learning system, and a variety of internal and external libraries and developer tools. Prior to joining Google, Jeff did computer systems research at Digital Equipment Corporation's Western Research Lab. Jeff has also worked for both the Centers for Disease Control and the World Health Organization, designing computer software for epidemiology and for statistical analysis of the HIV/AIDS pandemic. He is a Fellow of the ACM and the AAAS, a member of the U.S. National Academy of Engineering, and a recipient of the Mark Weiser Award and the ACM-Infosys Foundation Award in the Computing Sciences. Jeff holds a B.S., summa cum laude, in computer science and economics from the University of Minnesota, and a M.S. and Ph.D. in computer science from the University of Washington.

18. L Peter Deutsch. Dr. L Peter Deutsch received a Ph.D. in Computer Science from U.C. Berkeley in 1973. Subsequently at Xerox PARC, he helped develop the Interlisp-D, Cedar Mesa, and Smalltalk-80 programming systems. Deutsch's work on Smalltalk implementation, among other innovations, was an important contributor to the just-in-time compilation technology now used widely to dramatically improve the performance of Java and JavaScript implementations. He is also the author of a number of RFCs and of The Eight Fallacies of Distributed Computing, and originated the Deutsch limit adage about visual programming languages. From 1986 to 1991, as Chief Scientist at ParcPlace Systems, he developed cross-platform JIT technology. From 1986 to 2003, dba Aladdin Enterprises, he was the creator of Ghostscript, an Open Source implementation of the PostScript language. In 1993, he was a co-recipient of the ACM Software System Award, and was also named a Distinguished Alumnus of the U.C. Berkeley Computer Science program; he was named an ACM Fellow in 1994. In 1994, he founded Artifex Software to license Ghostscript commercially while continuing its development and its release as Open Source; Artifex today is a multi-million-dollar business. In 1999-2000, he served on the board of the Open Source Initiative. He is a co-inventor on two patents.

19. Whitfield Diffie. Dr. Whitfield Diffie serves as advisor to a variety of startups, primarily in the field of security. He is best known for

discovering the concept of public key cryptography, which underlies the security of internet commerce and all modern secure communication systems. Diffie's two principal positions after leaving Stanford University in the late 1970s were Manager of Secure Systems Research for Bell-Northern Research, the laboratory of the Canadian telephone system, and Chief Security Officer at Sun Microsystem. Diffie received the 2015 Turing Award and in 2017 was elected to both the National Academy of Engineering and the Royal Society.

20. David L. Dill. David Dill is The Donald E. Knuth Professor in the School of Engineering at Stanford University. Professor Dill's Ph.D. thesis, "Trace Theory for Automatic Hierarchical Verification of Speed Independent Circuits" was named as a Distinguished Dissertation by the Association for Computing Machinery (ACM), and published as such by M.I.T. Press in 1988. He was named a Fellow of the Institute of Electrical and Electronics Engineers (IEEE) in 2001 for his contributions to verification of circuits and systems, and a Fellow of the ACM in 2005 for contributions to system verification and for leadership in the development of verifiable voting systems. In 2008, he received the first "Computer-Aided Verification" award for fundamental contributions to the theory of real-time systems verification. In 2013, he was elected to the National Academy of Engineering and the American Academy of Arts and Sciences. In 2016, he received the Alonzo Church Award for Outstanding Contributions to Logic and Computation.

21. Lester Earnest. Lester Earnest is a widely-recognized computer scientist, best known for his deep involvement with the Advanced Research Project Agency Network (ARPAnet) startup committee, which led to his invention of the Finger social networking protocol. He served as a US Navy Aviation Electronics Officer and Digital Computer Project Officer at the Naval Air Development Center, and later joined MIT to help design the Semi-Automatic Ground Environment air defense system. Later, he innovated numerous early features in the nascent field of word processing, including the first spell-checker, search engine, self-driving vehicle, robotic hand-eye assembler that took verbal instructions, online restaurant

reviews, online news service, and a number of other successful innovations.

22. Brendan Eich. Brendan is the former CTO of Mozilla, and is widely recognized for his enduring contributions to the Internet revolution. In 1995, Eich invented JavaScript (ECMAScript), the Internet's most widely used programming language. He co-founded the mozilla.org project in 1998, serving as chief architect, and was a board member of the Mozilla Foundation since its inception in 2003 through 2014. Brendan helped launch the award-winning Firefox Web browser in November 2004 and Thunderbird e-mail client in December 2004.

23. Dawson Engler. Dawson Engler is an Associate Professor at Stanford. He received his Ph.D. from MIT for his work on the exokernel operating system and his undergraduate degree from University of Arizona. His research focuses on devising automatic methods to find as many interesting bugs in real code as possible, including static analysis, implementation level model checking, and symbolic execution. His research group has won numerous "Best Paper" awards. Its early static tools have found millions of errors in mature open source and commercial systems and have formed the basis of a successful company, Coverity. His group's more recent tool, KLEE, is a symbolic execution system widely used in the research community. He won the 2006 Weiser award and the 2008 ACM Grace M Hopper award.

24. Martin Fowler. Martin Fowler is an author and educator on software development. He is currently chief scientist at ThoughtWorks, a global system delivery and consulting firm. Mr. Fowler concentrates on the design of enterprise software: what makes a good design and what practices are needed to enhance it. He is the author of seven books on software development, which have over a million copies in print in over a dozen languages. He is the editor of a book series with Addison-Wesley on software design. His website, http://martinfowler.com, is a wide-ranging resource of software development techniques attracting around 150,000 visitors per month.

25. Neal Gafter. Neal Gafter is a Principal Engineer at Microsoft, where he is a technical lead for the Roslyn Project (Microsoft's implementation for the C# and Visual Basic programming languages). Previously he was a software engineer and Java Evangelist at Google, where he designed and implemented the distributed storage architecture for Google Calendar, and a Senior Staff Engineer at Sun Microsystems, where he led the development of the Java compiler and implemented the Java language features in releases 1.4 through 5.0. Neal was a member of the C++ Standards Committee and led the development of C and C++ compilers at Sun Microsystems, Microtec Research, and Texas Instruments. He holds a B.S. in computer engineering from Case Western Reserve University and a Ph.D. in computer science from the University of Rochester.

26. Robert Harper. Robert Harper is a professor in the computer science department at Carnegie Mellon University. He holds a Ph.D. in computer science from Cornell University. His main research interest is in the application of type theory to the design and implementation of programming languages and to the mechanization of their meta-theory. Harper made major contributions to the design of the Standard ML programming language and the LF logical framework. Harper is a recipient of the Allen Newell Medal for Research Excellence and the Herbert A. Simon Award for Teaching Excellence, and is an Association for Computing Machinery Fellow.

27. John Hennessy.* John Hennessy is a Professor of Electrical Engineering and Computer Science and Director of the Knight-Hennessy Scholars Program at Stanford University. Professor Hennessy previously served as President of Stanford University for sixteen years until 2016. He serves on the boards of Google, Cisco Systems, and the Gordon and Betty Moore Foundation. Professor Hennessy is an IEEE Fellow, a member of the National Academy of Sciences and the National Academy of Engineering, and a Fellow of the American Academy of Arts and Sciences and the Association for Computing Machinery. He is the co-author of two internationally used undergraduate and graduate textbooks on computer architecture design.

28. Tom Jennings. Tom Jennings has specialized in computers, software, and electronics since 1977; computer networking since 1984; and the Internet since 1992. Jennings was on the team that wrote the interface specification (API in today's parlance) for Phoenix Software's IBM compatible ROM BIOS. Jennings is the creator of FidoNet, the first and most influential message and file networking system protocol for networking computer bulletin boards. Jennings built Wired magazine's first internet presence as its first webmaster and ran an early regional internet service provider, TLGnet. Currently, Jennings is on the faculty at Calarts Art+Technology program.

29. Alan Kay. Alan Kay is one of the pioneers of object-oriented programming, personal computing, and graphical user interfaces. For this work, Dr. Kay has received the Draper Prize from the National Academy of Engineering, the ACM Turing Award, and the Kyoto Prize from the Inamori Foundation. Alan has been elected a fellow of the American Academy of Arts and Sciences, the National Academy of Engineering, the Royal Society of Arts, the AAAS, and the Computer History Museum. Alan has held fellow positions at HP, Disney, Apple, and Xerox, and has served as the chief scientist at Atari. While at Xerox PARC, he was one of the key members there to develop prototypes of networked workstations using the programming language Smalltalk. He is an adjunct professor of computer science at UCLA and an advisor to One Laptop per Child. At Viewpoints Research, Alan also continues his work with "powerful ideas education" for the world's children, as well as the development of advanced personal computers and networking systems.

30. Brian Kernighan.* Brian Kernighan is a professor in the Computer Science Department of Princeton University. He worked at Bell Labs alongside Unix creators Ken Thompson and Dennis Ritchie and contributed to the development of Unix. He co-authored a number of Unix programs, including widely used document preparation tools. He is also the author or co-author of 11 books on computing, including the first book on the C programming language with Dennis Ritchie; these books have been translated into more than two dozen languages. He is also a co-creator of the AWK and

AMPL programming languages. In collaboration with Shen Lin he devised well-known heuristics for two fundamental NP-complete optimization problems: graph partitioning and the traveling salesman problem. Kernighan received a Bachelor's degree in engineering physics from the University of Toronto, and his Ph.D. in electrical engineering from Princeton University. He is a member of the National Academy of Engineering.

31. David Klausner. David Klausner has over 50 years of software/hardware development and consulting experience in the computer and software industry. He has written software for commercial products as an engineer, developer, supervisor, project manager, department manager, middle manager and company executive. He has worked in forensic investigation and in reverse engineering. He has been employed in various capacities for various companies, such as Microsoft, AT&T, Cisco, IBM, Hewlett Packard, and Intel Corporation. He holds a Bachelors of Arts degree in Mathematics, and a Master of Science degree in Electrical Engineering. He has taught programming, public speaking, has guest lectured at Stanford University, and been an invited speaker by IBM, AT&T, and others. His technical opinions have been confirmed in several cases by the United States Court of Appeals for the Federal Circuit.

32. Ray Kurzweil.* Ray Kurzweil is an inventor, author and futurist. He was the principal inventor of the first CCD flat-bed scanner, the first omni-font optical character recognition, the first print-to-speech reading machine for the blind, the first text-to-speech synthesizer, the first music synthesizer capable of recreating the grand piano and other orchestral instruments, and the first commercially marketed large-vocabulary speech recognition. Kurzweil is the recipient of the National Medal of Technology, was inducted into the National Inventors Hall of Fame, holds twenty honorary Doctorates, and has received honors from three U.S. Presidents. He is presently a Director of Engineering at Google heading up a team developing machine intelligence and natural language understanding.

33. Kin Lane. Kin is a computer scientist and API Evangelist working to understand the technology, business and politics of APIs and help

share this insight with the world. He is the author of the book, Business of APIs, and is behind the popular API Evangelist blog. He has over 20 years of experience as a programmer, database administrator, architect, product developer, manager, and executive in the API space.

34. Ed Lazowska. Ed Lazowska holds the Bill & Melinda Gates Chair in the Paul G. School of Computer Science & Engineering at the University of Washington. His research concerns the design, implementation, and analysis of high performance computing and communication systems, and, more recently, the techniques and technologies of data-intensive discovery. He co-chaired (with Marc Benioff) the President's Information Technology Advisory Committee from 2003-05, and (with David E. Shaw) the Working Group of the President's Council of Advisors on Science and Technology to review the Federal Networking and Information Technology Research and Development Program in 2010. He is a Member of the National Academy of Engineering and a Fellow of the American Academy of Arts and Sciences.

35. Doug Lea. Doug Lea is a Professor of Computer Science at the State University of New York at Oswego. He is an author of books, articles, reports, and standardization efforts on object oriented software development including those on specification, design and implementation techniques, distributed, concurrent, and parallel object systems, and software reusability; he has served as chair, organizer, or program committee member for many conferences and workshops in these areas. He is the primary author of several widely used software packages and components.

36. Bob Lee. Bob Lee is CEO of Present Company, makers of Present, an upcoming social network for women. Prior to that, as Square's CTO, Bob built Square's core products, scaled the team from 12 to 1200 people, and created Square Cash. Before Square, Bob worked at Google where he created Guice and was the core library lead for Android.

37. Sheng Liang. Sheng Liang is a software entrepreneur. He is co-founder and CEO of Rancher Labs, an enterprise software company.

He was CTO of the Cloud Platform group at Citrix Systems after their acquisition of Cloud.com, where he was co-founder and CEO. Sheng was co-founder and CTO of Teros, a provider of perimeter and network security solutions for enterprises and service providers, acquired by Citrix Systems in 2005. He also served as VP of Engineering at SEVEN Networks, and Director of Software Engineering at Openwave Systems. He was a Staff Engineer in Java Software at Sun Microsystems, where he designed the Java Native Interface (JNI) and led the Java Virtual Machine (JVM) development for the Java 2 platform. He has a B.S. from the University of Science and Technology of China and a Ph.D. from Yale University.

38. Barbara Liskov. Barbara Liskov is one of the world's leading authorities on computer language and system design. Liskov joined MIT in 1972 as a member of the Department of Electrical Engineering and computer Science. She is also a member of the MIT laboratory for Computer Science and Artificial Intelligence and heads the programming methodology group. Her research interests lie in programming methodology, programming languages and systems, and distributed computing. Major projects include: the design and implementation of CLU, the first programming language to support data abstraction; the design and implementation of Argus, the first high-level language to support implementation of distributed programs; and the Thor object-oriented database system, which provides transactional access to persistent, highly-available objects in wide-scale distributed environments. Liskov is a fellow of the American Academy of Arts and Sciences, the National Academy of Inventors, the Association for Computing Machinery, and the Massachusetts Academy of Science. She is a member of the National Academy of Science and the National Academy of Engineering. In 2009, she received the A.M. Turing Award from the ACM. Other honors include the Society of Women Engineers' Achievement Award, the IEEE von Neumann medal, the ACM SIGPLAN Programming Languages Achievement Award, the University of Pennsylvania Harold Pender Award, the ACM SIGOPS Hall of Fame Award, the CMU and Tokyo University of Technology Katayanagi Award for Research Excellence, the ACM

SIGOPS Lifetime Achievement Award, and five honorary doctorates. She holds a B.A. from UC Berkeley and a Ph.D. from Stanford.

39. Paul Menchini. Paul Menchini is the Chief Information Security Officer at the North Carolina School of Science and Mathematics. Previously, he held technical positions at HP, Intel, GE Microelectronics, CLSI and OrCAD. As a member of the "Woods Hole Summer Study on Hardware Description Languages," he contributed to the specifications for VHDL; subsequently, he edited two revisions of IEEE Std 1076 VHDL and developed the first commercially successful VHDL compiler. As part of the compiler project, he developed an API for a VHDL intermediate form, which was subsequently standardized by the IEEE. He holds a Masters in Computer Engineering from Stanford University and is the recipient of numerous technical awards, including charter membership in the "IEEE Golden Core."

40. Andrew W. Moore. Andrew Moore is the Dean of the School of Computer Science at Carnegie Mellon University. His research interests are data mining, machine learning, artificial intelligence, algorithms, and theory. Previously, he was a VP of Engineering at Google, co-director of the Biomedical Security Center at the University of Pittsburgh, and co-founded a consultancy for statistical data mining in manufacturing. Moore is a Fellow of the Association for the Advancement of Artificial Intelligence (AAAI). He holds a Ph.D. in Computer Science and a B.S. in Mathematics and Computer Science from Cambridge University.

41. James H. Morris. Dr. James H. Morris is a Professor of Computer Science at Carnegie Mellon University, where he served as Dean of the Silicon Valley Campus, Dean of the School of Computer Science, Head of the Computer Science Department, and Director of the Information Technology Center, a joint project with IBM that developed a prototype university computing system. He founded Carnegie Mellon's Human Computer Interaction Institute, Robot Hall of Fame, and Silicon Valley Campus. He was an Associate Professor at UC Berkeley, where he developed two fundamental principles of programming languages: inter-module protection and

lazy evaluation. He was co-discoverer of the Knuth-Morris-Pratt string-searching algorithm. He was Principal Scientist and Research Fellow at Xerox PARC, where he was part of the team that developed the Alto, a precursor to today's personal computers. He is a founder of MAYA Design Group and an ACM Fellow. He holds a B.S. from CMU and an M.S. and Ph.D. from MIT.

42. Peter Norvig.* Peter Norvig is a Director of Research at Google; previously he directed Google's core search algorithms group. He is co-author of *Artificial Intelligence: A Modern Approach*, the leading textbook in the field, and co-teacher of an Artificial Intelligence class that signed up 160,000 students, helping to kick off the current round of massive open online classes (MOOCs). He is a fellow of the AAAI, ACM, California Academy of Science and American Academy of Arts and Sciences.

43. Martin Odersky. Martin is a professor at EPFL in Lausanne, Switzerland. He is best known as the creator and principal designer of the Scala programming language. Prior to that, he made several contributions to the development of Java. He created the Pizza and GJ languages, designed the original version of generics for Java, and wrote the javac reference compiler for Java. He is a fellow of the ACM.

44. Tim Paterson. Tim began his career designing one of the first 16-bit microcomputer systems at Seattle Computer Products. He then wrote an operating system for that computer, which was later sold to Microsoft and became widely used as MS-DOS. He went on to found his own company, Falcon Technology, whose primary products were hard disk systems for personal computers. He moved on to Microsoft where he was a software engineer for many years, working on such products as QuickBASIC, Visual Basic, VBScript, and Visual J++ (Java). After his retirement in the late '90s he has continued developing software and microcontroller-based hardware projects as a hobby and part-time small business. He has been granted three U.S. patents on software methods.

45. David Patterson.* David Patterson joined UC Berkeley in 1977. He has been Director of the Par Lab, Chair of UC Berkeley's CS

Division, Chair of the Computing Research Association, and President of the Association for Computing Machinery. His most successful projects have been Reduced Instruction Set Computers (RISC), Redundant Arrays of Inexpensive Disks (RAID), and Network of Workstations. All helped lead to multibillion-dollar industries. This research led to many papers, six books, and about 35 honors, including election to the National Academy of Engineering, the National Academy of Sciences, the Silicon Valley Engineering Hall of Fame, and Fellow of the Computer History Museum. He shared the IEEE von Neumann Medal and the NEC C&C Prize with John Hennessy, former President of Stanford University and co-author of two of his books.

46. Alex Payne. Alex Payne consults, advises, and invests in early-stage technology startups. As Platform Lead at Twitter he managed one of the web's most popular APIs. He was subsequently co-founder and Chief Technology Officer of online banking service Simple, acquired by BBVA in 2014. Alex organizes an annual conference showcasing advances in programming languages and has co-authored a book on the Scala programming language (O'Reilly, 2009). He is a regular speaker at technology and business conferences worldwide and has lectured on API design at Stanford.

47. Tim Peierls. Since receiving a BS in Computer Science from Yale in 1983 and an MS in CS from Cornell in 1986, Tim has continuously worked in the software industry, first at Bell Labs (airline crew scheduling), then co-founding the Lightstone Group in 1990 (aircraft scheduling, delivery vehicle routing and scheduling, sold to Descartes Systems Group in 1998) and Seat Yourself in 2002 (online ticketing for school performing arts groups). For the last fifteen years, almost all of his programming work has been in Java. He has served on the Expert Groups of several Java Specification Requests (166, 201, 330, 334) and on the SE/EE Executive Committee of the Java Community Process; he co-authored a book, *Java Concurrency in Practice*; and he contributes code, support, and advice to various open source projects, including Restlet, Hazelcast, and JClouds.

48. Simon Phipps. Simon is a director and past-president of the Open Source Initiative, the global steward of the Open Source Definition.

OSI serves to advocate for, educate about, and build bridges within the open source community. His career has included early engagement in establishing Java, XML and weblogs as computer industry technologies as well as contributions to open standards in a variety of fields. As chief open source officer at Sun Microsystems he supervised the open source relicensing of Solaris Unix, Java and many other software systems. He is currently founder and CEO of Meshed Insights Ltd, a UK firm offering management services related to open source and digital rights.

49. Bill Pugh. Bill Pugh invented Skip Lists, a randomized data structure that is widely taught in undergraduate data structure courses. He has also made research contributions in techniques for analyzing and transforming scientific codes for execution on supercomputers, and in a number of issues related to the Java programming language, including the development of JSR 133—Java Memory Model and Thread Specification Revision. Current research projects include FindBugs, a widely used static analysis tool for Java, and Marmoset, an innovative framework for improving the learning and feedback cycle for student programming projects. He is currently a professor emeritus of computer science at the University of Maryland.

50. Ronald L. Rivest. Ronald L. Rivest is an MIT Institute Professor in the Electrical Engineering and Computer Science Department. He is well-known as a co-inventor of the RSA public-key cryptosystem, for which he received the ACM Turing Award. He is a co-author of the widely-used textbook *Introduction to Algorithms*. His current research interest is voting systems and election integrity.

51. Curtis Schroeder. Curtis is a Hardware-in-the-Loop Simulation Engineer at Draper. He served as the Drafting Group Editor for the Simulation Interoperability Standards Organization (SISO) Common Image Generator Interface (CIGI) 4.0 international standard. The success of SISO international standards depends upon implementation of said copyrighted standards by numerous simulation vendors and end-users, including NATO. Previously, Curtis has worked for Antycip Simulation in the UK and the Lockheed Martin Aeronautics Company, where he utilized a number

of open standards in projects he was involved in. He earned B.S. & M.S. Computer Science degrees at the Missouri University of Science & Technology.

52. Robert Sedgewick. Robert Sedgewick is the founding chair and the William O. Baker Professor in the Department of Computer Science at Princeton and served for many years as a member of the board of directors of Adobe Systems. He has over 50 years of experience working with software systems. He has held visiting research positions at Xerox PARC, Palo Alto, CA; Institute for Defense Analyses, Princeton, NJ; and INRIA, Rocquencourt, France. He regularly serves on journal editorial boards and organizing program committees of conferences and workshops on data structures and the analysis of algorithms held throughout the world. Professor Sedgewick's research interests include analytic combinatorics, algorithm design, the scientific analysis of algorithms, curriculum development, and innovations in the dissemination of knowledge. He has published widely in these areas and is the author of twenty books, including a series of books on algorithms that have been bestsellers for four decades and have sold nearly one million copies. He has also published extensive online content (including studio-produced video lectures) on analysis of algorithms and analytic combinatorics and (with Kevin Wayne) algorithms and computer science. Their MOOC on algorithms has been named one of the "top 10 MOOCs of all time."

53. Mary Shaw. Mary Shaw is the Alan J. Perlis University Professor of Computer Science in the Institute for Software Research at Carnegie Mellon University. Her research focuses on software engineering and software design, particularly software architecture and design of systems used by real people. She has made fundamental and significant contributions to an engineering discipline for software through developing data abstraction with verification, establishing software architecture as a major branch of software engineering, designing influential and innovative curricula in software engineering and computer science supported by two influential textbooks, and helping to found the Software Engineering Institute at Carnegie Mellon. She has received the United States' National Medal of Technology and Innovation, the

George R. Stibitz Computer & Communications Pioneer Award, the ACM SIGSOFT Outstanding Research Award, the IEEE Computer Society TCSE's Distinguished Educator and Distinguished Women in Software Engineering Awards, and CSEE&T's Nancy Mead Award for Excellence in Software Engineering Education. She is an elected Life Fellow of the ACM and the IEEE and an elected Fellow of the AAAS. She holds a BA *cum laude* from Rice and a Ph.D. from Carnegie Mellon.

54. Barbara Simons. Barbara Simons is a former President of the Association for Computing Machinery (ACM), the nation's oldest and largest educational and scientific society for computing professionals. She is the only woman to have received the Distinguished Engineering Alumni Award from the College of Engineering of U.C. Berkeley, where she earned her Ph.D. in computer science. A fellow of ACM and the American Association for the Advancement of Science, she also received the Computing Research Association Distinguished Service Award and the Electronic Frontier Foundation Pioneer Award. She has published *Broken Ballots: Will Your Vote Count?*, a book on voting machines co-authored with Douglas Jones. She has been on the Board of Advisors of the U.S. Election Assistance Commission since 2008, and she co-authored the report that led to the cancellation of Department of Defense's Internet voting project (SERVE) in 2004 because of security concerns. She was a member of the National Workshop on Internet Voting, convened by President Clinton, which conducted one of the first studies of Internet Voting and produced a report in 2001. She is Board Chair of Verified Voting and is retired from IBM Research.

55. Dave Snigier. Dave Snigier is an enterprise architect at the University of Massachusetts President's Office, designing systems to help keep public higher education cost effective. He has led several successful projects as part of the Emerging Technologies group at UMass including a system-wide paperless initiative.

56. Alfred Z. Spector. Alfred Spector is Chief Technology Officer and Head of Engineering at Two Sigma, a firm dedicated to using information to optimize diverse economic challenges. Prior to joining

Two Sigma, Dr. Spector spent nearly eight years as Vice President of Research and Special Initiatives, at Google, where his teams delivered a range of successful technologies including machine learning, speech recognition, and translation. Prior to Google, Dr. Spector held various senior-level positions at IBM, including Vice President of Strategy and Technology (or CTO) for IBM Software and Vice President of Services and Software research across the company. He previously founded and served as CEO of Transarc Corporation, a pioneer in distributed transaction processing and wide-area file systems, and he was a professor of computer science at Carnegie Mellon University. Dr. Spector received a bachelor's degree in Applied Mathematics from Harvard University and a Ph.D. in computer science from Stanford University. He is a Fellow of both the Association for Computing Machinery and the IEEE. He is an active member of the National Academy of Engineering and the American Academy of Arts and Sciences, where he serves on the Council. Dr. Spector won the IEEE Kanai Award for Distributed Computing in 2001 and the ACM Software Systems Award for his work on the Andrew File System (AFS) in 2016.

57. Bjarne Stroustrup. Bjarne Stroustrup is the inventor of the C++ programming language. He wrote the standard textbook on the language and its implementation, *The C++ Programming Language*, and many other academic and popular books and articles. He has served on the ISO Standards committee since its creation in 1989. He is a fellow of the ACM, the IEEE and the CHM, and an elected member of the National Academy of Engineering. He holds a masters degree in mathematics and computer science from Aarhus University, in Denmark, and a Ph.D. in computer science from the University of Cambridge, where he is an honorary fellow of Churchill College.

58. Gerald Jay Sussman. Gerald Jay Sussman is the Panasonic (formerly Matsushita) Professor of Electrical Engineering at the Massachusetts Institute of Technology. He has been involved in artificial intelligence research at M.I.T. since 1964. His research has centered on understanding the problem-solving strategies used by scientists and engineers, with the goals of automating parts of the process and formalizing it to provide more effective methods of

science and engineering education. Sussman has also worked in computer languages, in computer architecture, and in VLSI design. Sussman is a coauthor of the introductory computer science textbook that included innovative advances in curricula designed for students pursuing different kinds of computing expertise, which has had a worldwide impact on university computer-science education. Sussman has received numerous awards and recognitions including: the ACM's Karl Karlstrom Outstanding Educator Award, the Amar G. Bose award for teaching, a fellow of the Institute of Electrical and Electronics Engineers, a fellow of the American Academy of Arts and Sciences, a member of the National Academy of Engineering, and a fellow of the American Association for the Advancement of Science. He received the S.B. and the Ph.D. degrees in mathematics from the Massachusetts Institute of Technology in 1968 and 1973.

59. Ivan E. Sutherland. Ivan E. Sutherland received his B.S. degree from the Carnegie Institute of Technology, his M.S. degree from the California Institute of Technology, and his Ph.D. degree from the Massachusetts Institute of Technology, all in electrical engineering. He holds honorary degrees from Harvard University, the University of North Carolina, and the University of Utah. He joined Sun in 1990 as a Sun Fellow, Sun's highest technical rank. He joined Portland State University in 2009 to found the Asynchronous Research Center. He leads a small group working on self-timed VLSI systems; his group develops self-timed circuit methodologies and design techniques for fast CMOS circuits and applies them to new hardware architectures. His book, *Logical Effort* (1999) with joint authors Sproull and Harris, describes the mathematics of designing fast circuits. His 1963 MIT Ph.D., Sketchpad, is widely known, and he has been called the "father of computer graphics." He is author of more than 70 patents, as well as numerous publications and lectures. Dr. Sutherland holds the 1988 ACM Turing Award, the 2012 Kyoto Prize and the IEEE Von Neumann Award. He is a Fellow of the ACM and a member of both the National Academy of Engineering and the National Academy of Sciences.

60. Andrew Tanenbaum. Andrew S. Tanenbaum has an S.B. degree from M.I.T. and a Ph.D. from the University of California. He is a professor emeritus at the Vrije Universiteit in Amsterdam.

Tanenbaum is the principal designer of three operating systems: TSS-11, Amoeba, and MINIX, as well as a considerable amount of other open-source software. In addition, Tanenbaum is the author or coauthor of five books, which together have been translated in more than 20 languages and over 175 editions. Tanenbaum has lectured on a variety of topics all over the world. He has been keynote speaker at 40 conferences and has given talks at over 100 universities and companies in 15 countries all over North America, Europe, Asia, and Australia. In 2004, Tanenbaum became an Academy Professor of the Royal Netherlands Academy of Arts and Sciences. In 2008, he received a prestigious European Research Council Advanced Grant. Tanenbaum is a Fellow of the ACM, a Fellow of the IEEE, and a member of the Royal Netherlands Academy of Arts and Sciences. In 1994 he was the recipient of the ACM Karl V. Karlstrom Outstanding Educator Award. In 1997 he won the ACM SIGCSE Award for Outstanding Contributions to Computer Science. In 2007 he won the IEEE James H. Mulligan, Jr., Education Medal. In 2008 he won the USENIX Lifetime Achievement Award and in 2015 he won the inaugural Eurosys Lifetime Achievement Award. He has also won numerous other awards, some of which are on his Wikipedia page. He has two honorary doctorates.

61. Brad Templeton. Brad Templeton, active in the computer network community since 1979, was founder and publisher at ClariNet Communications Corp., the electronic newspaper that was perhaps the earliest dot-com company. He participated in the building and growth of USENET from its earliest days, and in 1987 founded and edited rec.humor.funny, for many years the world's most widely read electronic publication. He was the first employee of Personal Software/Visicorp, the first major microcomputer applications software company. He later founded Looking Glass Software and over the years was author of a dozen packaged microcomputer software products, including VisiPlot for the IBM-PC, various games, popular tools and utilities for Commodore computers, special Pascal and Basic programming environments designed for education (ALICE), an add-in spreadsheet compiler for Lotus 1-2-3 (3-2-1 Blastoff), and various network related software tools. He currently

is track chair for computing and networks at Singularity University, a consultant and speaker on self-driving cars, and is on the board of the Electronic Frontier Foundation and the Foresight Nanotech Institute. He is Chairman Emeritus of the Electronic Frontier Foundation.

62. Ken Thompson.* Ken Thompson spent much of his career at Bell Laboratories where he co-designed and implemented the original Unix operating system, invented the B programming language that was a precursor to the C programming language, invented the Bon programming language, co-developed the Plan 9 operating systems, developed the CTSS version of the editor QED, developed ed, which is the standard text editor on Unix, and the definition of the UTF-8 encoding, which is used for more than half of all Web pages. Thompson also co-developed the software and hardware for Belle, which was the first computer built for the sole purpose of chess playing, and it officially became the first master-level machine in 1983. He is currently a Google Advisor and was formerly a Distinguished Engineer at Google, where he invented new programming languages (including the Go programming language as a co-inventor), among other projects. Thompson is a recipient of numerous awards and commendations in connection with his work on Unix, including the IEEE Emanuel R. Piore Award (1982), the Turing Award (1983), the IEEE Richard W. Hamming Medal (1990), the National Medal of Technology (1999), and the Japan Prize (2011). He is a member of the National Academy of Sciences and the National Academy of Engineering. Thompson holds a B.S. and an M.S., both in Electrical Engineering and Computer Science, from the University of California, Berkeley. He has been awarded two honorary Ph.D degrees.

63. Michael Tiemann. Michael Tiemann is a true open source software pioneer. He made his first major open source contribution more than three decades ago by adapting the GNU C compiler to support the C++ language and numerous RISC microprocessors. Tiemann co-founded Cygnus Solutions, the first open source software company. Tiemann was the first winner of the Usenix STUG (Software Tools and User Group) Award in 1996. When Cygnus was acquired by Red Hat in 2000, Tiemann became Red Hat's Chief Technical Officer

(CTO), later becoming its first Vice President of Open Source Affairs. Tiemann graduated from the Moore School at the University of Pennsylvania (Class of 1986) with a BS CSE degree, and later did research at INRIA (1988) and Stanford University (1988-1989). Tiemann retired as President of the Board at the Open Source Initiative from 2005-2012.

64. Linus Torvalds. Linus Torvalds is the principal developer of the Linux kernel, which lies at the heart of the Linux operating system. Linux runs on billions of devices from cellphones to supercomputers. Torvalds is a fellow of the Computer History Museum and the Linux Foundation. He was awarded the Millennium Technology Prize, IEEE Computer Pioneer Award, NEC C&C Prize, Reed College Vollum Award, Takeda Award, Lovelace Medal, EFF Pioneer Award, and inducted into the Internet Hall of Fame. Torvalds holds an M.S. in computer science from the University of Helsinki.

65. Andrew Tridgell. Dr. Andrew Tridgell is a computer scientist and free software developer in Canberra, Australia. Best known for his contributions to the development of the award winning Samba suite of networking software that enables interoperability with Microsoft networking services, he has been actively developing in the area of interoperability for more than 20 years.

66. Jeffrey Ullman. Jeffrey Ullman is the Stanford W. Ascherman Professor of Engineering (Emeritus) in the Department of Computer Science at Stanford and CEO of Gradiance Corp. He received a B.S. degree from Columbia University in 1963 and a Ph.D. from Princeton in 1966. Prior to his appointment at Stanford in 1979, he was a member of the technical staff of Bell Laboratories from 1966-1969, and on the faculty of Princeton University between 1969-1979. From 1990-1994, he was chair of the Stanford Computer Science Department. Ullman was elected to the National Academy of Engineering in 1989, the American Academy of Arts and Sciences in 2012, and has held Guggenheim and Einstein Fellowships. He has received the Sigmod Contributions Award (1996), the ACM Karl V. Karlstrom Outstanding Educator Award (1998), the Knuth Prize (2000), the Sigmod E. F. Codd Innovations award (2006), and the IEEE von Neumann medal (2010). He is the author of 16 books,

including books on database systems, compilers, automata theory, and algorithms.

67. Andries van Dam. Andries van Dam is a Professor of Computer Science at Brown University, and has served on Brown's Computer Science faculty since 1965. He is the author of the widely used reference books *Computer Graphics: Principles and Practice* and *Object-Oriented Programming Java: A Graphical Approach*. In 1967, Andries co-founded ACM SIGGRAPH, the precursor to SIGGRAPH. Andries is an IEEE Fellow, an ACM Fellow, and has been a member of the National Academy of Engineering since 1996. Andries has won multiple awards, including the Information Display's Special Recognition Award (1974), the IEEE Centennial Medal (1984), the National Computer Graphics Association's Academic Award (1990), the ACM SIGGRAPH Steven A. Coons Award (1991), the L. Herbert Ballou University Professor Chair (1992), the ACM Karl V. Karlstrom Outstanding Educator Award (1994), the Thomas J. Watson, Jr. University Professor of Technology and Education Chair (1995), the IEEE James H. Mulligan, Jr. Education Medal (1999), and the ACM SIGCSE Award for Outstanding Contributions to Computer Science Education (2000). Andries received a B.S. with honors in Engineering Science form Swarthmore College, a M.S. and Ph.D. from the University of Pennsylvania, and holds four honorary Ph.D. degrees.

68. Guido van Rossum. Guido van Rossum created the open-source programming language Python, and is its lead developer and thought leader. Python is widely used in industry, and is the most popular introductory teaching language at top U.S. universities. Guido developed the Python language while at CWI in Amsterdam. After moving to the United States, he worked as a guest researcher at NIST, at CNRI, and at several start-up companies. He became a Senior Staff Engineer at Google, and is currently a principal engineer at Dropbox. Guido is an ACM Distinguished Engineer and a recipient of several awards including the USENIX STUG Award, the NLUUG Award, the Free Software Foundation Award, and the Dr. Dobb's Journal 1999 Excellence in Programming Award. In 2013, Python was awarded the Dutch National ICT COMMIT/

Award. Guido holds an M.S. in Mathematics and Computer Science from the University of Amsterdam.

69. John Villasenor. John Villasenor is on the faculty at UCLA, where he is a professor of electrical engineering, public policy, and management, as well as a visiting professor of law. He is also a nonresident senior fellow at the Brookings Institution and a visiting fellow at the Hoover Institution. Professor Villasenor's research considers communications and information technologies and their broader ramifications, and has addressed topics including cybersecurity, autonomous vehicles, and digital media policy. Professor Villasenor is a member of the Council on Foreign Relations and a former vice chair of the World Economic Forum's Global Agenda Council on the Intellectual Property System. He holds an M.S. and Ph.D. in electrical engineering from Stanford University, and a B.S. in electrical engineering from the University of Virginia. Professor Villasenor has previously served as, though is not currently serving as, a consultant to Google in relation to the *Oracle v. Google* matter.

70. Jan Vitek. Jan Vitek is a Professor of Computer Science at Northeastern University. He is the past Chair of the ACM Special Interest Group on Programming Languages (SIGPLAN), the vice chair of AITO and of the IFIP WG 2.4, and is Chief Scientist at Fiji Systems. He holds a Ph.D. from the University of Geneva and an MSc from the University of Victoria. He works on various aspects of programming languages including virtual machines, compilers, software engineering, real-time and embedded computing, concurrency and information security. Professor Vitek led the Ovm project which resulted in the first successful flight test of real-time Java virtual machine. With Noble and Potter, Vitek proposed the notion of ownership for alias control, which became known as ownership types. He chaired PLDI, ISMM and LCTES and was program chair of ESOP, ECOOP, VEE, Coordination, and TOOLS.

71. Philip Wadler. Philip Wadler is a Professor of Theoretical Computer Science at the University of Edinburgh and Senior Research Fellow at IOHK. He is an ACM Fellow and a Fellow of the Royal Society of Edinburgh, past chair of ACM SIGPLAN, past holder of a Royal

Society-Wolfson Research Merit Fellowship, winner of the SIGPLAN Distinguished Service Award, and a winner of the POPL Most Influential Paper Award. Previously, he worked or studied at Stanford, Xerox Parc, CMU, Oxford, Chalmers, Glasgow, Bell Labs, and Avaya Labs, and visited as a guest professor in Copenhagen, Sydney, and Paris. He has an h-index of 66 with more than 22,000 citations to his work, according to Google Scholar. He contributed to the designs of Haskell, Java, and XQuery, and is a co-author of Introduction to Functional Programming (Prentice Hall, 1988), XQuery from the Experts (Addison Wesley, 2004) and Generics and Collections in Java (O'Reilly, 2006). He has delivered invited talks in locations ranging from Aizu to Zurich.

72. James H. Waldo. James "Jim" Waldo is the Gordon McKay Professor of the Practice of Computer Science in the School of Engineering and Applied Sciences at Harvard, where he is also the Chief Technology Officer. Jim is also a professor of technology policy at the Harvard Kennedy School. Previously, Jim designed clouds at VMware, and was a Distinguished Engineer at Sun Microsystems, where he investigated next-generation large-scale distributed systems. He was the lead architect for Jini, a distributed programming system based on Java. Before joining Sun, Jim spent eight years at Apollo Computer and Hewlett Packard, working in the areas of distributed object systems, user interfaces, class libraries, text and internationalization. While at HP, he led the design and development of the first Object Request Broker, and was instrumental in getting that technology incorporated into the first OMG CORBA specification. Jim edited the book *The Evolution of C++: Language Design in the Marketplace of Ideas* (MIT Press), co-edited *Engaging Privacy and Information Technology in a Digital Age* (National Academies Press), and was one of the authors of *The Jini Specification* (Addison Wesley). More recently, he authored *Java: The Good Parts*. He is currently a member of the editorial boards of Queue magazine and Communications of the ACM. He holds over 50 patents. Jim received his Ph.D. from the University of Massachusetts (Amherst). He holds two M.A. degrees from the University of Utah.

73. Dan Wallach. Dan Wallach is a professor in the Department of Computer Science and a Rice Scholar at the Baker Institute for Public Policy at Rice University in Houston, Texas. His research considers a variety of different computer security topics, ranging from web browsers and servers through Java security, electronic voting technologies, and smartphones. Wallach is a former member of the Air Force Science Advisory Board and a former member of the USENIX Association Board of Directors.

74. Peter Weinberger.* Peter Weinberger is a computer scientist at Google. Previously, he was Chief Technology Officer at Renaissance Technologies and held many positions at Bell Labs, including Information Sciences Research Vice President where he was responsible for computer science research, math and statistics, and speech. As a scientist at Bell Labs he worked on Unix, contributing to the design and implementation of the AWK programming language, the IO library for f77, the fast factoring program qfactor, the B-tree library cbt, a code generator for C, and a network file system. He did research on topics including operating systems, compilers, security, and number theory. Before joining Bell Labs, he taught in the Department of Mathematics at the University of Michigan, Ann Arbor. He holds a B.S. from Swarthmore and a Ph.D. from the University of California, Berkeley.

75. Steve Wozniak. Steve Wozniak co-founded Apple and invented the Apple I and Apple II computers. He holds a B.S. in Electrical Engineering and Computer Science from UC Berkeley, and honorary doctorates from twelve universities. Wozniak is Innovator in Residence at High Point University. He founded many companies including CL 9, which brought the first programmable universal remote control to market in 1987, Wheels of Zeus (WOZ), and Acquicor Technology. He was Chief Scientist at Fusion-io and at Primary Data. He designed calculators for Hewlett-Packard and taught computer science to elementary school students and their teachers. Wozniak won numerous awards including the ACM Grace Murray Hopper Award, the National Medal of Technology (with Steve Jobs), the IEEE Hoover Medal, the Heinz Award for Technology, the American Humanist Association Isaac Asimov Science Award, the Global Award of the President of Armenia for

Outstanding Contribution to Humanity Through IT, the Young Presidents' Organization Lifetime Achievement Award, the Cal Alumni Association Alumnus of the Year Award, and the Legacy for Children Award from the Children's Discovery Museum in San Jose. He was named a Fellow of the Computer History Museum "for co-founding Apple Computer and inventing the Apple I personal computer," and inducted into the National Inventors Hall of Fame, the Manufacturing Hall of Fame, and the Consumer Electronics Hall of Fame.

76. Frank Yellin.* Frank Yellin has spent over a decade working on runtime systems for interpreted and compiled languages. As a Staff Engineer in Embedded and Consumer at Sun Microsystems, he was an original member of the Java project. Yellin is co-author of The Java Virtual Machine Specification (Addison-Wesley, 1999), and co-authored the first version of the Java API specification. Previously he worked at Lucid, where he focused on multitasking, garbage collection, interrupts, and the compilation of Common Lisp. Yellin currently is a Staff Software Engineer at Google, where he works on automatic scalable security testing. He holds an A.B. in Applied Mathematics from Harvard and an M.S. in Computer Science from Stanford. He is the inventor or co-inventor of sixteen patents.