No. 2013-1021, -1022

IN THE
UNITED STATES COURT OF APPEALS
FOR THE FEDERAL CIRCUIT

ORACLE AMERICA, INC.,

Plaintiff-Appellant,

v.

GOOGLE INC.

Defendant-Cross Appellant.

Appeal From The United States District Court For The
Northern District Of California In Case No. 10-CV-3561,
Judge William H. Alsup

BRIEF OF AMICI CURIAE EUGENE H. SPAFFORD, Ph.D., ZHI DING, Ph.D., AND LEE A. HOLLAAR, Ph.D. IN SUPPORT OF APPELLANT

Jared Bobrow, Esq.
Principal Attorney
Aaron Y. Huang, Esq.
WEIL, GOTSHAL & MANGES LLP
201 Redwood Shores Parkway
Redwood Shores, CA 94065
(650) 802-3000

DATED: FEBRUARY 19, 2013

Counsel for Amici Curiae
EUGENE H. SPAFFORD, PH.D., ZHI DING, PH.D., AND LEE A. HOLLAAR, PH.D.

CERTIFICATE OF INTEREST

Counsel for amici curiae, Eugene H. Spafford, Ph.D., Zhi Ding, Ph.D., and Lee A. Hollaar, Ph.D., certifies the following:

1. The full name of every party represented by me is:

EUGENE H. SPAFFORD
ZHI DING
LEE A. HOLLAAR

2. The names of the real parties in interest (if the party named in the caption is not the real party in interest) represented by me is:

NOT APPLICABLE.

3. All parent corporations and any publicly held companies that own 10% or more of the stock of the party represented by me are:

NOT APPLICABLE.

4. The names of all law firms and the partners or associates that appeared for the parties now represented by me in the trial court or agency or are expected to appear in this Court are:

Jared Bobrow
Aaron Y. Huang
WEIL, GOTSHAL & MANGES LLP

Dated: February 19, 2013

By: */s/ Jared Bobrow*
Jared Bobrow
WEIL, GOTSHAL & MANGES LLP

TABLE OF CONTENTS

Page

CERTIFICATE OF INTEREST i
I. STATEMENT OF INTEREST 1
II. SUMMARY OF ARGUMENT 6
III. ARGUMENT 8
A. Technology Background 8
1. APIs and software APIs 8
2. Java APIs 10
B. An API is a Work of Creative Expression. 12
1. An API can be expressed in innumerable different ways. 12
2. The expression of an API is not dictated by its function. 17
C. Android Is Not Interoperable With Java. 20
D. Copyright Protection Incentivizes Reliable and Secure APIs. 22
IV. CONCLUSION 24

TABLE OF AUTHORITIES

Cases

Computer Assoc. Int'l, Inc. v. Altai, Inc.,
982 F.2d 693 (2d Cir. 1992) 4, 21

I.

STATEMENT OF INTEREST

Dr. Eugene H. Spafford is a Professor of Computer Science at Purdue University, where he has been employed since 1987, and the founder and Executive Director of the Center for Education and Research in Information Assurance and Security at Purdue. He has over 30 years of experience in both practice and research in the field of computing and computer science, including with the use of application programming interfaces ("APIs"). Over the past decade, he has served in an advisory or consulting capacity on issues in computing and information systems with several U.S. government agencies and their contractors, including the National Science Foundation, the Federal Bureau of Investigation, the Government Accountability Office, the National Security Agency, the U.S. Department of Justice, the U.S. Air Force, the U.S. Naval Academy, U.S. SPACECOM, the Department of Energy, and the Executive Office of the President. He has served on the President's Information Technology Advisory Committee and has testified before Congressional committees nine times. He is a Fellow of five major scientific and professional organizations involved with computing, the Association for Computing Machinery, American Academy for the Advancement of Science (AAAS), Institute of Electrical and Electronic

Engineers ("IEEE"), International Information Systems Security Certifications Consortium, and Information Systems Security Association.

Dr. Spafford has published and spoken extensively about software engineering, information security, and professional ethics, and he has served on the editorial boards of several major journals of computer science. He was affiliated with the Software Engineering Research Center, a NSF University-Industry Cooperative Research Center located at Purdue. His current research is directed towards the architecture, construction, and public policy of secure information systems. He has been writing computer programs since 1972, including computer security programs that have been used internationally by government agencies and companies, and his programming experience includes Java and other programming languages. He also has taught undergraduate and graduate courses involving software engineering, information system security, and many programming languages.

Dr. Zhi Ding is a Professor of Electrical and Computer Engineering and the Child Family Professor of Engineering and Entrepreneurship at the University of California, Davis. He has over 25 years of experience in both practice and research in the field of electronic and electrical engineering, including with the use of APIs. He received his Ph.D.

degree in Electrical Engineering from Cornell University in 1990. He was a faculty member of Auburn University and the University of Iowa, and he has held visiting positions at Australian National University, Hong Kong University of Science and Technology, NASA Research Center (Cleveland, Ohio), and USAF Wright Laboratory.

Dr. Ding has published extensively about electrical engineering, and his research focuses on communications and systems. Dr. Ding is a co-author of the popular engineering textbook *Modern Digital and Analog Communication Systems* (4th ed.). Dr. Ding is a Fellow of the IEEE, and he has served on the technical committees of several workshops and conferences. He was the Technical Program Chair of the 2006 IEEE Global Telecommunication Conference. Dr. Ding's research includes active collaborations with researchers from several countries, including Australia, China, Japan, Canada, Taiwan, Korea, Singapore, and Hong Kong.

Dr. Lee A. Hollaar is a Professor of Computer Science at the School of Computing at the University of Utah, where he teaches courses in computer and intellectual property law and computer systems and networking. He has been programming computers since 1964 and designing computer hardware since 1969. He received his B.S. degree in electrical engineering

from the Illinois Institute of Technology in 1969 and his Ph.D. in computer science from the University of Illinois at Urbana-Champaign in 1975. Dr. Hollaar was a Fellow with the Senate Committee on the Judiciary and technical advisor to its chair, Senator Hatch, and a visiting scholar with Judge Randall R. Rader at the Court of Appeals for the Federal Circuit. During his time with the Senate, he helped the Judiciary Committee with the No Electronic Theft Act and Digital Millennium Copyright Act, as well as a number of other bills. He has been a Special Master in a number of software copyright cases in the Federal District Courts for the Eastern District of Michigan and the District of Puerto Rico, where he offered opinions and made recommendations on the scope of copyright protection in software and whether there was infringement.

In 1997, he submitted one of the first amicus briefs discussing how the *Computer Associates v. Altai* test should be applied, a brief cited with approval by Circuit Judge Jon O. Newman in his law review article *New Lyrics For An Old Melody: The Idea/Expression Dichotomy In The Computer Age*, 17 Cardozo Arts & Ent. L.J. 691 (1999). His amicus brief to the Supreme Court in *MGM v. Grokster* provided the inducement theory that became the basis of the Court's unanimous decision. Dr. Hollaar is the

author of *Legal Protection of Digital Information* (2002), which discusses the copyright protection of computer software and, in particular, the application of the abstraction-filtration-comparison test. He is presently creating an online course on intellectual property law for computer scientists and engineers based on his book.

Drs. Spafford, Ding, and Hollaar's interest in this appeal is to improve the intellectual property laws of the United States. A robust and balanced intellectual property regime promotes innovation, reliability, and security in software and information systems, and this brief explains why copyright in the area of APIs protects creative expression and helps promote interoperability, reliability, and security in information systems. They have no interest in any party to this litigation or stake in the outcome of this appeal.

Drs. Spafford, Ding, and Hollaar submit this *amici curiae* brief with the consent of all parties, who have stipulated their consent for all *amici* to file their briefs.

No party's counsel authored this brief in whole or in part. No party or party's counsel contributed money that was intended to fund preparing or submitting this brief. No person, other than the *amici* or their

counsel, contributed money that was intended to fund preparing or submitting this brief.

II.

SUMMARY OF ARGUMENT

At stake in this appeal is whether, and to what extent, software code constituting a type of work known as an application programming interface ("API")—and specifically, an API for the Java programming language—is a work of creative expression, such that it may be entitled to copyright protection. As professors and researchers in computer science, we have created, used, and taught others about software APIs—including in the software that we have written, the research that we have overseen, the companies and government agencies that we have advised, and the courses that we have taught. Based on our research and experience from over three decades in computer science, computational algorithm research and development, and the software industry, software APIs are widely used throughout modern, complex information systems. The public policy and legal treatment of APIs is therefore of great academic and practical interest to us and others in the computer science, computer engineering, systems engineering, and software engineering communities.

We submit this brief to draw the Court's attention to, and correct what appears to be, the district court's fundamentally erroneous understanding of the creativity underlying APIs. We do not address the district court's legal reasoning, but we do observe that its holding appears predicated on assumptions that there is only one or a few limited ways to express an API and that expression is dictated by a set of pre-determined functions. Those, assumptions however, are mistaken: for any given problem or use case, an API can be structured and expressed in a vast variety of ways, and that variety reflects the creative choices and subjective judgments of its author. In other words, contrary to the district court's opinion, a huge number and variety of APIs may be written to accomplish the same purpose, and the differences among those APIs are determined by each designer's creativity, experience, and personal preference, and only somewhat, if at all, by any requirements of functionality.

The specific API at issue in this case, a software API written in the computer programming language known as "Java," provides a good example of the creative expression inherent in an API: authoring the design of a Java API requires significant creativity and involves many subjective choices not dictated by function—perhaps more than is required for authoring

the code that implements that design. Because of the central role that well-designed APIs play in modern information systems, it is important that the law recognize and protect the creativity they embody. The incentive to develop robust APIs for integrated and interoperable—and, ultimately, reliable and secure—systems depends on it.

III.

ARGUMENT

A. Technology Background

1. APIs and software APIs

To assist the Court in understanding the broader context of the present case, and the potential scope of the district court decision's impact, we have provided the following summary of software APIs and APIs generally.

An "application programming interface" (API) is a protocol used to interact with a system. There are many types of APIs that are used to describe and enable interactions in various types of applications. APIs can vary greatly in size, complexity, and form. The term "API" may also be used loosely to refer to the documentation describing the API's specification.

A "software API" is a type of API used in computing. A software API provides an interface that allows a user, computer, or piece of software to communicate with another piece of software. A software API

prescribes the expected behavior or set of rules that the software embodies, as well as the format and nature of data communicated with that software in any interaction. For example, a software API may describe what routines, data, and variables are available in the software; it may describe the inputs, outputs, exceptions, and types of data used by the routines, along with their precision.

An API thus consists of two different types of source code: a design that prescribes the expected behavior and structure of the API; and an implementation that performs the design. A single software API may have one or more implementations, in the form of one or more software libraries that each implement the same functional API. Usually, the API is designed along with the rest of the software in an iterative fashion, and then the API is packaged and documented for others to use. An API may also have no implementation, in the case of an abstract API used in design or documentation.

By analogy, an API is similar to a blueprint. The blueprint may represent the design of a large structure, and may specify the building materials, clearances and dimensions, placement of utilities, and methods of entry and egress. The blueprint will incorporate features designed to meet the requirements of building codes, the strengths of the materials, and the

capacities of other items not explicitly listed on the blueprint, such as wiring chases that may be used for a security system. There are uncountable designs possible because architects may make many subjective choices to address the many requirements. The blueprint is therefore the creative expression of a unique design that is the product of the architect's many choices.

2. Java APIs

The software API for a computer programming language can describe the syntax, functions, variables, and data structures that a program written in that language may use. The software API for a particular computer programming language therefore prescribes the methods by which a user may interact with, structure, and handle data, exceptions, and the functions available to software in that programming language.

The software API at issue in this case is for one such programming language, named "Java." The Java programming language allows a user to organize data and functions into a hierarchy of nested and interrelated structures called "objects."[1] A Java API describes the objects,

[1] There are several general forms of computer languages, and not all of them use the "objects," "classes," and "methods" described here as abstract representations for structure and definition. Those that do use these abstractions are known as "object-oriented" languages. Java is one such object-oriented language.

their relationships to each other, and methods by which a program written in the Java language may interact with and control them. To accomplish this, a Java API defines a set of "classes," with each class definition prescribing a set of controls and data, such as "variables" and "methods,"[2] associated with that class. Each method may in turn have none or any combination of different inputs, known as "parameters"; outputs, known as "return values"; and values returned and/or actions taken in cases of error, known as "exceptions." The classes are organized into "packages"; the packages are organized into "libraries." A Java API may also contain "interfaces" and "fields" that may be used to share variables, methods, and data across otherwise unrelated classes.

The class definitions therefore may prescribe an array of possible inter-class relationships. By way of a simplified example, a Java API could be created to draw graphics on a screen. Such an API could be implemented using a "DrawScreen" library. The "DrawScreen" library may contain separate packages for "Foreground" and "Background." The "Foreground"

[2] "Method," as used in this brief, is a term of art for the Java programming language that means a portion of code that performs an operation. A method implements the behavior of an object. The use of the term "method" in this brief is limited to this technological meaning, and is not intended to be conflated with the district court's use of the legal term "method of operation."

package may contain classes for "Polygon," "Text," and "Photo." The "Polygon" class may have subclasses for "Ellipse" and "Rectangle." The "Ellipse" class could in turn have methods for "DrawCircle" and "DrawEllipse." At the same time, there may be an interface "FillColor," containing methods that the unrelated "Polygon" and "Background" classes may implement.[3]

The specific example at issue in this case is the software API implemented by the "core" software libraries that are packaged and distributed along with the Java programming language. This API discloses what packages, classes, methods, and variables are contained in the libraries, their organization and relationships, how to use them, and their expected behavior.

B. An API is a Work of Creative Expression.

1. An API can be expressed in innumerable different ways.

Because there are an extremely large number of ways in which an API may be designed to solve a problem or use case, the specific design and

[3] The names used here are of course examples only. The author of an API may create any name for the variables, methods, classes, and packages, and is not constrained by its function. Using a name such as "XX03A1" would work as well as "Polygon," but would be less mnemonic for programmers.

structure of an API is a work that expresses the creative choices of the author rather than solely functional requirements.

The flexibility and variability provided by the method, class, and package definitions in a Java API serve as a perfect example of the nearly limitless possible behaviors, combinations, and relationships in an API's design. The author of a Java API must come up with what methods, variables, classes, packages, fields, and interfaces to include. She must then define how each of those elements behave, how they relate to one another, and how they may be used, such as which classes will inherit from which others, and what methods and data to expose to what other classes. These iterative design choices—of selection and definition—exist at each and every level of a Java API, including not only the parameters in a method declaration, but also the class, package, interface, field, and so forth. For example, the author must decide what each class, field, and package represents and what problems/use cases it should solve.

Using the simplified graphics drawing example provided above, a software API for drawing shapes may be designed in many different ways. Figure 1 below shows one possibility, in which a "Polygon" class contains three separate methods for drawing each of three separate shapes:

"drawCircle," which takes as inputs the coordinates of the center of the circle and the length of the radius; "drawEllipse," which takes the coordinates of the center, the length of the major axis, and the length of the minor axis; and "drawSquare," which takes the coordinates of the center of the square and the length of a side.

Function	Inputs	Output
drawCircle	x,y coordinates radius	
drawEllipse	x,y coordinates major axis minor axis	
drawSquare	x,y coordinates length of a side	

Figure 1.

However, these functions are not pre-determined, as the designer may instead choose to design the "Polygon" class differently. As shown in Figure 2 below, instead of a separate "drawCircle" method, the API may only include the "drawEllipse" function. This approach may be more difficult to use and implement, because the user must be sure to input equal values for the

major and minor axes rather than simply the radius to draw a circle. However, it has the added advantage of greater flexibility and power, in that it allows a user to draw more complex shapes than a circle using the same method and to more simply modify and test code that uses the method.

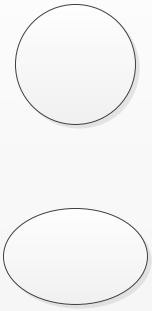
Function	Inputs	Output
drawEllipse	x,y coordinates major axis minor axis	
drawSquare	x,y coordinates length of a side	

Figure 2.

As a third alternative, instead of separate methods for drawing specific shapes, the author of the API may instead define within the "Polygon" class only a single method "DrawPolygon" that takes as its inputs the coordinates of the center of the shape and a complex variable called "ShapeType." The "ShapeType" input in turn is defined in an entirely new class, and represents a complex data object that includes variables for the

number of sides (*e.g.*, "0" for a circle or "4" for a square) and, for any given shape, the relevant size measurements (*e.g.*, the radius in the case of a circle or the length of the sides in the case of a square). While this method may be even more difficult to use and implement, it is of course even more flexible and powerful because it can draw more complex shapes (*e.g.*, a pentagon).

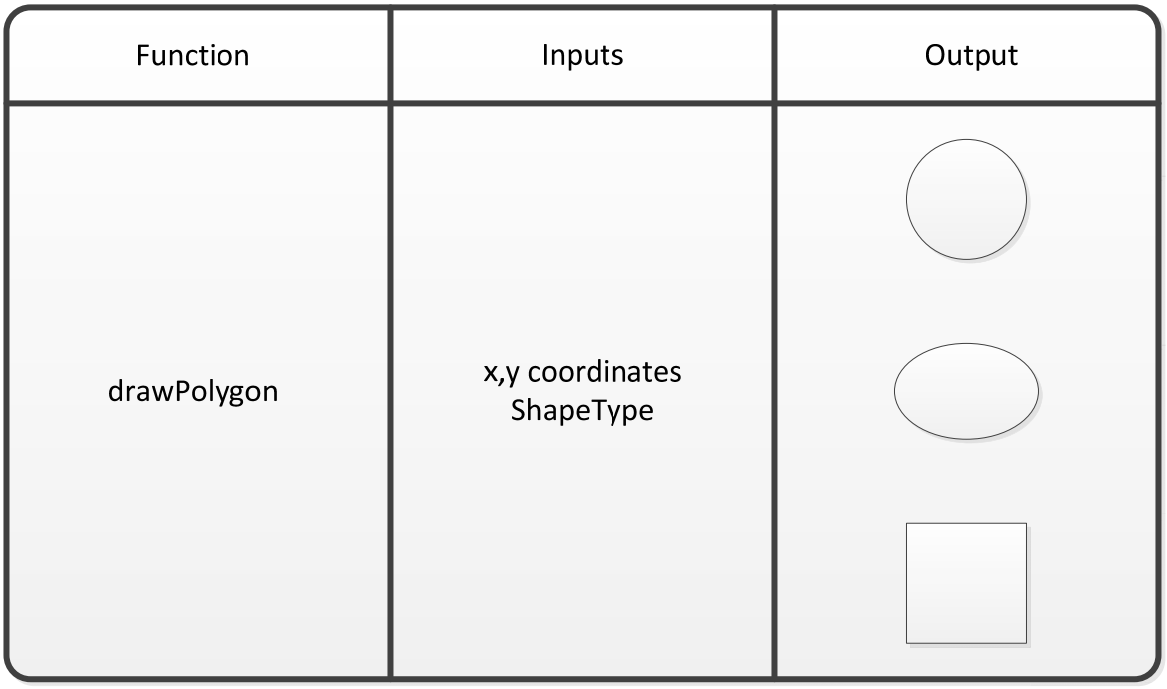


Figure 3.

Furthermore, each of these methods may have different outputs and exceptions. For example, the method may either throw an exception[4] or

[4] To "throw an exception" is a term of art in Java programming that means to detect an error that will cause the Java engine to interrupt the defined flow of control to signal and resolve the error.

return a value of true or false to indicate whether it was properly executed; or, it may return a pointer to a drawn object that then may be further manipulated.

Of course, these are only a few of the possibilities for a single method. There are many more ways that an author may design and structure the API to perform the same function. Moreover, the author must make similar decisions for each and every method and class in an API. For a complex API that may involve hundreds or thousands of methods and classes, these choices are multiplied hundreds or thousands of times over, resulting in an almost uncountable number of possible API designs.

2. The expression of an API is not dictated by its function.

As a consequence of this great design flexibility, the expression of an API reflects the author's imagination and creativity rather than rigid dictates of pre-determined functions. For a complex API, authoring the design may require greater creativity, talent, experience, and subjective judgment than authoring the underlying source code. First, the designer must successfully anticipate the future needs of the users of the API. Next, the designer must decide which structures best address those needs while balancing competing tensions of functionality, simplicity, and consistency. Finally, the designer must create an organization of the solution that appears

intuitive to users. The methods comprising an API class must work individually but also together, similar to how the classes in a package must work together.

This is more creative (and even less dictated by functionality) than a simple taxonomy or "command structure," as the district court's opinion states, because it does more than classify pre-existing elements with known logical relationships. The author of an API invents entirely new elements as well as new behaviors and relationships for each of those elements. An API's design thus embodies only one of a huge number of possible approaches to a problem, and it expresses the author's creativity and affirmative choices in solving that problem.

The many possible approaches to even the simple circle-drawing example above illustrates the myriad creative choices that an author must make, including balancing factors such as ease of use, flexibility, and difficulty of implementation, as well as the expected use and the expected users. This is analogous, for example, to the innumerable creative choices that an author makes in writing a story. The author must consider the various narrative elements to include, such as the characters, locations, and plot points, and the relationships and specific details of each. She must also consider

how to describe each element, and how to arrange them into the sentences, paragraphs, chapters, and so forth to create the order and structure of the story. Breaking down a use case into the functions, structure, and design of an API requires similar decisions regarding what to include, how to include them, and their internal structure and relationships.

Thus, the district court incorrectly assumed that there are "pre-assigned functions" that an API must include. *See* A164, A167. Because the structure and approach of an API—and any methods associated with that structure—are selected and defined by the author, there simply is no "pre-assigned" set of functions, even where the use case or problem to be solved is the same. Likewise, the district court incorrectly focused on a single method and failed to address the many other creative choices that go into software API design beyond simply the parameters in a method declaration, including: the class in which the method is defined; the method's exposure to other classes; and the containing class's relationship to other classes, interfaces, and packages. This is demonstrated by the differences in the APIs in Java and Android for performing the same function of drawing graphical user interfaces.

C. Android Is Not Interoperable With Java.

The district court's order assumed that "some" code written in Java could be run on Android in support of its contention that Google's copying of Java's API allowed "interoperability." A168-69. In the context of software and computer systems, "interoperability" refers to the ability of software designed for execution on one system to be executed on another. We note that the specific API copied in this case does not present such a situation: the court was incorrect, as a technological matter, in its assumption that Google's copying of the Java API did—or could—allow arbitrary programs written in Java to run on Android.

In fact, Android's internal product literature specifically states that Android could not run all programs written in Java. A2205 ("Does Android support existing Java apps? A. No. . . . Is Android Java compatible? A. No."). We understand that this was affirmed by the evidence and witness testimony presented during trial. A167; A21181:4-7; A21503:16-A21504:2; A22397:11-A22398:3; A22463:13-22. For example, many programs written in Java rely on popular packages that are missing from Android, such as JavaX.sound, JavaX.imageIO, JavaX.print, JavaX.swing, and Java.awt. Such

programs have to be substantially modified or even rewritten from scratch to run on Android.

Indeed, Google's engineers have stated that Google was motivated to copy the API, not for interoperability reasons, but merely to make Android more attractive to the pre-existing base of Java developers. A2033. (The packages copied were those useful for smartphones, Android's target market.) The fact that Google copied only certain packages and functionality of Java to include in Android is further evidence that its copying was not related to interoperability.

There is a practical concern that might be raised related to the copyrightability of APIs—that a developer could leverage its copyright primarily to prevent interoperable software from being developed. We do not advocate restraining copying that is required for purposes of interoperability, but we do not believe that concern applies in this case. One test that could help address that concern is to determine whether the copying is dictated by external technical considerations related to interoperability and not simply because the copying party chose to design its program in that way. *See Computer Assoc. Int'l, Inc. v. Altai, Inc.*, 982 F.2d 693, 709-10 (2d Cir. 1992) (explaining, in the context of non-literal copying of software, that

copyright protections should not attach where a programmer's design is dictated by extrinsic considerations, such as "compatibility requirements of other programs with which a program is designed to operate in conjunction"). For example, the creator of a word processor who wants to be able to open Microsoft Word files in the .doc file format so that they can be used in that creator's word processing program (*i.e.*, so that the programs are interoperable) must be able to copy the .doc file format. In such a case, the copier may argue that its behavior enables a greater good.

But that is not the case here. There was no technical reason, dictated by considerations related to interoperability, for Google to copy the declaring code of only certain packages of the Java API. If Google had needed to copy the Java API for purposes of interoperability, it would have copied all of it. But Google did not do that—Google cherry-picked the code that it copied. The fact that it was able to do so indicates that it was not subject to a technical constraint mandating copying for purposes of interoperability. And in fact, Android is not interoperable with Java.

D. Copyright Protection Incentivizes Reliable and Secure APIs.

APIs embody the creative expression, competence, and understanding of their original authors. To the extent that copyright

protection over the design and structure of APIs prohibits unauthorized changes to APIs, they may help prevent the introduction of anomalous problems, including security, compatibility, stability, and fragmentation that may result from such changes.

If an API is written by an author trained in proper software engineering, safety, and security, the resulting API and code may support critical features and dependencies not explicitly described in the documentation. If someone else with less knowledge of the features and dependencies modifies the API (and the underlying implementation), it may introduce unanticipated vulnerabilities and instability. The versions (and stability) of the software may become fragmented and of uncertain reliability.

Using the blueprint analogy given above, anyone not following the blueprint (*e.g.*, putting in larger or extra windows) may cause unexpected failures (*e.g.*, wall collapse because of loss of load-bearing structure) because not all of the design requirements and responses are explicitly shown in the blueprint. By the same token, we believe that a software API should receive copyright protection to help prevent such failures in information systems.

IV.

CONCLUSION

For the aforementioned reasons, this Court should reverse the district court's judgment that the Java API packages in this case are not protected by copyright.

Dated: February 19, 2013

Respectfully submitted,

/s/ Jared Bobrow

Jared Bobrow, Esq.
Principal Attorney
Aaron Y. Huang, Esq.
WEIL, GOTSHAL & MANGES LLP
201 Redwood Shores Parkway
Redwood Shores, CA 94065
(650) 802-3000

Attorneys for *Amici Curiae*
EUGENE H. SPAFFORD, PH.D., ZHI DING, PH.D., AND LEE A. HOLLAAR, PH.D.

No. 2013-1021, -1022

IN THE
UNITED STATES COURT OF APPEALS
FOR THE FEDERAL CIRCUIT

ORACLE AMERICA, INC.,

Plaintiff-Appellant,

v.

GOOGLE INC.

Defendant-Cross Appellant.

Appeal From The United States District Court For The
Northern District Of California In Case No. 10-CV-3561,
Judge William H. Alsup

AFFIDAVIT PURSUANT TO FEDERAL CIRCUIT RULE 47.3(d)

Pursuant to Federal Circuit Rule 47.3(d), I, Aaron Y. Huang of Weil, Gotshal & Manges LLP, hereby swear that I have the actual authority of the following counsel of record to sign the foregoing BRIEF OF AMICI CURIAE EUGENE H. SPAFFORD, PH.D., ZHI DING, PH.D., AND LEE A. HOLLAAR, PH.D. IN SUPPORT OF APPELLANT, and the accompanying Certificate of Interest and Certificate of Compliance on his behalf:

Jared Bobrow
WEIL, GOTSHAL & MANGES LLP
201 Redwood Shores Parkway
Redwood Shores, CA 94065
(650) 802-3000
Counsel for Amici Curiae
Eugene H. Spafford, Ph.D., Zhi Ding, Ph.D.,
and Lee A. Hollaar, Ph.D.

Dated: February 19, 2013

Respectfully submitted,

/s/ Aaron Huang

Aaron Y. Huang
WEIL, GOTSHAL & MANGES LLP
201 Redwood Shores Parkway
Redwood Shores, CA 94065
(650) 802-3000

CERTIFICATE OF SERVICE

In accordance with Fed. R. App. P. 25 and Fed. Cir. R. 25, I certify that on this day I served the foregoing via the Court's CM/ECF on:

Christa M. Anderson
Robert A. Van Nest
Steven Hirsch
Michael Kwun
Dan Jackson
KEKER & VAN NEST, LLP
633 Battery Street
San Francisco, CA 94111

Dorian E. Daley
Deborah K. Miller
Matthew Sarboraria
Andrew C. Temkin
ORACLE AMERICA, INC.
500 Oracle Parkway
Redwood Shores, CA 94065

E. Joshua Rosenkranz
Mark S. Davies
Andrew D. Silverman
Kelly M. Daley
ORRICK, HERRINGTON & SUTCLIFFE LLP
51 West 52nd Street
New York, NY 10019
(212) 506-5000

Annette L. Hurst
Gabriel M. Ramsey
Elizabeth C. McBride
ORRICK, HERRINGTON & SUTCLIFFE LLP
405 Howard Street
San Francisco, CA 94105-2669

Dale M. Cendali
Diana M. Torres
Sean B. Fernandes
Joshua L. Simmons
KIRKLAND & ELLIS LLP
601 Lexington Avenue
New York, NY 10022

Michael A. Jacobs
MORRISON FOERSTER LLP
425 Market Street
San Francisco, CA 94105-2482

Kenneth A. Kuwayti
MORRISON FOERSTER LLP
755 Page Mill Road
Palo Alto, CA 94304-1018

Dated: February 19, 2013

Respectfully submitted,

/s/ Aaron Y. Huang

Aaron Y. Huang, Esq.
WEIL, GOTSHAL & MANGES LLP
201 Redwood Shores Parkway
Redwood Shores, CA 94065
(650) 802-3000

Attorneys for *Amici Curiae*
EUGENE H. SPAFFORD, PH.D., ZHI DING, PH.D., AND LEE A. HOLLAAR, PH.D.

CERTIFICATE OF COMPLIANCE

V.

THIS BRIEF COMPLIES WITH THE TYPE-VOLUME LIMITATION OF FEDERAL RULE OF APPELLATE PROCEDURE 32(A)(7)(B).

☒ The brief contains 4,497 words, excluding the parts of the brief exempted by Federal Rule of Appellate Procedure 32(a)(7)(B)(iii).

VI.

THIS BRIEF COMPLIES WITH THE TYPEFACE REQUIREMENTS OF FEDERAL RULE OF APPELLATE PROCEDURE 32(A)(5) AND THE TYPE STYLE REQUIREMENTS OF FEDERAL RULE OF APPELLATE PROCEDURE 32(A)(6).

☒ The brief has been prepared in a proportionally spaced typeface using Microsoft Word 2010 in Times New Roman, 14 point font.

Dated: February 19, 2013

Respectfully submitted,

/s/ Jared Bobrow

Jared Bobrow, Esq.
WEIL, GOTSHAL & MANGES LLP
201 Redwood Shores Parkway
Redwood Shores, CA 94065
(650) 802-3000

Attorneys for *Amici Curiae*
EUGENE H. SPAFFORD, PH.D., ZHI DING, PH.D., AND LEE A. HOLLAAR, PH.D.