No. 13-298

IN THE

Supreme Court of the United States

ALICE CORPORATION PTY. LTD.,

Petitioner,

v.

CLS BANK INTERNATIONAL AND
CLS SERVICES LTD.,

Respondents.

ON WRIT OF CERTIORARI TO THE UNITED STATES COURT OF APPEALS FOR THE FEDERAL CIRCUIT

BRIEF OF *AMICUS CURIAE* RED HAT, INC., IN SUPPORT OF RESPONDENTS

ROBERT H. TILLER
Counsel of Record
RED HAT, INC.
100 East Davie Street
Raleigh, NC 27601
(919) 754-4232
rtiller@redhat.com

Counsel for Amicus Curiae
Red Hat, Inc.

252140

COUNSEL PRESS
(800) 274-3321 • (800) 359-6859

TABLE OF CONTENTS

Table of Contents

TABLE OF CITED AUTHORITIES

Page

CASES

Page

STATUTES AND AUTHORITIES

Cited Authorities

Page

Page

Page

Page

STATEMENT OF INTEREST OF *AMICUS CURIAE* RED HAT, INC.

Red Hat, Inc., is the world's leading provider of open source software and related services to enterprise customers.[1] Its software products are used by Wall Street investment firms, hundreds of Fortune 500 companies, and the United States government. Based in Raleigh, North Carolina, Red Hat has offices in 33 countries.

Red Hat's interest in this proceeding is based on its experience in the software industry and its commitment to the free and open source software community. Open source software is growing at an exponential rate,[2] and is already of strategic economic importance. It provides the technological backbone of many large corporations and supports essential functions of many national and regional governments. In addition, it is used daily by millions of individuals for such activities as web searching, email, online shopping, social media, and banking. It is found in devices as varied as mainframe computers, desktop

1. No counsel for a party authored this brief in whole or in part, and no such counsel made a monetary contribution intended to fund the preparation or submission of this brief. No person other than amicus curiae or its counsel made a monetary contribution to its preparation or submission. Petitioner and Respondents have consented to the filing of this brief through blanket letters of consent filed with the Clerk's Office.

2. According to a 2008 study, the amount of open source code was doubling every fourteen months. Amit Deshpande & Dirk Riehle, The Total Growth of Open Source, *in* PROCEEDINGS OF THE FOURTH CONFERENCE ON OPEN SOURCE SYSTEMS, 197, 207 (Springer Verlag 2008), *available at* http://dirkriehle.com/2008/03/14/the-total-growth-of-open-source/.

computers, tablets, mobile phones, medical devices, aircraft, automobiles, and warships.[3]

Open source software involves not only innovative products, but also a transformative means of production, and understanding it is important in considering the patent eligibility issue at bar. In brief, all software begins as plain text "source code." Programmers write and edit source code in human-readable programming languages that allow specification of software features and behavior at a high level of abstraction. Source code is typically translated by a program called a compiler into "object code" form, which basically consists of a series of instructions to be executed on a computer. Since object code consists of unintelligible strings of 1s and 0s, software is effectively illegible and unmodifiable without access to its source code. Open source software permits modification and improvement by making the source code available to the user.

The open source model produces software innovation through a mechanism of collaborative development that relies on free communication of ideas among large numbers of independent individuals and companies. It uses a combination of technological and legal means. Typically, an open source program originates as a community-

3. There are numerous widely used open source software programs. Examples include the Linux operating system kernel, the Apache web server, the Firefox web browser, the Eclipse integrated development environment, the OpenStack cloud computing platform, the Git distributed version control system, the WordPress blogging tool, and the GNU Compiler Collection (GCC). Open source software is essential for Google, Amazon, Facebook, and nearly every other modern technology company.

based project whose members work together using tools such as email, mailing lists, Internet Relay Chat, bug reporting systems, wikis, source code repositories, and source code version control systems. These tools enable rapid communication among geographically dispersed software developers, and make it possible for large numbers of developers from many different backgrounds and organizations to work collaboratively. A community open source project makes its software publicly available in source code form, under licensing terms that grant very broad, royalty-free copyright permissions allowing further use, copying, modification, and distribution.

In making source code available and conferring broad copyright permissions, open source differs significantly from traditional proprietary software. A vendor of proprietary software generally develops the software in-house and provides only object code to the user subject to restrictive licenses that allow no rights to copy, modify, or redistribute that code. Such vendors also retain the source code as a trade secret.

The open source development model has proven to be highly effective in producing software of superior quality. Because there are many developers working as collaborators in a distributed fashion, innovation happens rapidly. Because of the many who volunteer their time,[4]

4. The profit motive is, of course, an important incentive for software development, but it is not the only one. Open source software developers often contribute to open source projects on a voluntary basis. *See* Steven Weber, The Success of Open Source 129-30 (2004). Some of the motivations for their contributions include improving the functioning of a product for business or personal use, enhancing programming skills,

and the availability of the source code under royalty-free licenses granting generous modification and distribution rights, the cost of producing and improving software is low. Software bugs and security problems are quickly identified and remedied. Moreover, because users have access to the source code, those users can diagnose problems and customize the software to suit their particular needs.

The scope of patentable subject matter is an issue of critical importance to the future development of all software, including open source. Because open source innovation depends on sharing source code and free collaboration, open source community members do not generally seek to prohibit or control use of open source software through patents, and most open source software developers view software patents as hindering innovation. Red Hat respectfully submits that this Court should evaluate the issues at bar with a view to the importance of open source software and the bright promise of future open source innovation.

---

reputation, philosophical commitment to free software, and personal enjoyment. *Id.* at 134–36. Copyright protects authors against the copying of their software. Patents, of course, block independent invention of patented technology. Although some patent advocates use the rhetoric of "theft" of ideas to support their arguments, there is evidence that the great majority of patent lawsuits are not against defendants who copied inventions but rather against independent inventors. Dan L. Burk & Mark A. Lemley, The Patent Crisis and How the Courts Can Solve It 28 (2009) [hereinafter "The Patent Crisis"].

SUMMARY OF ARGUMENT

This Court has long recognized that patents do not always promote innovation, and they may substantially hinder it. Thus in evaluating patent eligibility under 35 U.S.C. § 101 ("Section 101"), the Court has consistently held that certain types of abstract ideas are excluded. Two types of excluded abstract ideas are illustrated by the patent claims at issue here. The first is broadly stated economic practices prevalent in our system of commerce like the one in issue in *Bilski v. Kappos*, 130 S. Ct. 3218, 3229–31 (2010). The second is mathematical algorithms like the one found unpatentable in *Gottschalk v. Benson*, 409 U.S. 63, 68 (1972).

Software is properly understood as involving layers of abstraction. Examples of such layers include the idea of a program or its function, the methodology of how the program achieves the idea, the source code of the program (that is, the human readable computer instructions), and the object code of the program (the machine readable instructions). The claims at issue here describe a methodology for financial intermediation that is abstract in the same manner as the patent claims rejected in *Bilski*. It also effectively includes mathematical algorithms, because only through such algorithms in source and object code could the claims be implemented.

To be sure, when mathematical algorithms or other abstract ideas are augmented by an inventive concept, they may satisfy the requirements of Section 101. However, merely adding a general purpose computer to an otherwise unpatentable algorithm is far from an inventive concept. The computer adds nothing meaningful

to the abstract idea, and amounts to insignificant post-solution activity. Viewing a general purpose computer as a different machine each time it runs a different program is a highly artificial, because such machines are not in any material way changed by running software.

In confirming the boundaries for software patents, it is significant that such patents were generally not issued until the mid-1990s. Between the 1940s and 1990s, without patent protection or threats, extraordinary innovation occurred, and software grew from an interesting idea into a major industry. In the early 1990s, key leaders in the software industry were opposed to software patents. However, following a series of decisions by the Federal Circuit in the mid-1990s, there was an explosion in software patenting, and now there are hundreds of thousands of such patents.

Far from encouraging innovation, this proliferation of patents has seriously encumbered it. Because software is an abstract technology, translating software functions into patent language generally results in patents with vague and uncertain boundaries. Software products are often highly complex, created by combining hundreds or thousands of discrete elements in a cumulative process.

Because the boundaries of software patents are exceedingly vague and the numbers of issued software patents is now enormous, it is virtually impossible to rule out the possibility that a new software product may arguably infringe some patent. Creating a new software product therefore always entails an unavoidable risk of a lawsuit that may cost millions of dollars in legal fees, as well as actual damages, treble damages, and an injunction that terminates a business.

This case offers an opportunity to restore the historical and well-founded boundaries for patentable subject matter that exclude abstract ideas of the type generally involved in software from patent eligibility. This course correction will both bring clarity to the law and remove a significant barrier to technology innovation.

ARGUMENT

I. THIS COURT'S DECISIONS ESTABLISH THAT THE TYPES OF ABSTRACT IDEAS THAT ARE CHARACTERISTIC OF SOFTWARE FALL OUTSIDE THE SCOPE OF SECTION 101.

This Court has long recognized that patents have costs as well as benefits. *See Benson*, 409 U.S. at 68–72 (explaining *O'Reilly v. Morse*, 56 U.S. (15 How.) 62 (1853)). Patents do not always promote innovation, and they may substantially hinder it. *See id.* A patent on a process excludes others from using that process, and thus may block or discourage technological progress. Therefore, defining the proper subject matter limits under 35 U.S.C. § 101 is of critical importance.

Thus this Court has determined that "laws of nature, natural phenomena, and abstract ideas" are excluded from patent protection. *Diamond v. Diehr*, 450 U.S. 175, 185 (1981). The Court's prior cases have established guidelines for two types of unpatentable abstract ideas that are applicable here.

The first type is a broad concept of a "practice long prevalent in our system of commerce… ." *Bilski*, 130 S. Ct. at 3231. *Bilski* rejected an attempt to patent a

business method for reducing the risk of price changes for commodities in the energy market. The Court determined that "[t]he concept of hedging . . . is an unpatentable abstract idea . . . ." *Id.* Allowing such a patent "would effectively grant a monopoly over an abstract idea." *Id.*

The second type of unpatentable abstract idea is a mathematical algorithm. *Diehr*, 450 U.S. at 185. For example, in *Benson*, the patent application at issue concerned an algorithm for converting signals from binary-coded decimal numerals into pure binary form. The Court rejected the attempt to patent "the algorithm itself." 409 U.S. at 72. Similarly, in *Parker v. Flook*, 437 U.S. 584, 585–87 (1978), the Court rejected a patent application describing methods of updating alarm limits for use in the petrochemical and oil-refining industries.

Software involves several different levels of abstract ideas. At a high level there is the function of the software, or the problem that the software is designed to solve. An example is the idea for a program that creates textual documents, spreadsheets, or calendars. At another level is the general methodology for fulfilling the functions. For word processing, for example, the software may be thought of as storing in computer memory the necessary letters and symbols in various fonts, receiving signals from an input device such as a keyboard, and as a result of this input, storing the particular letters in a file in memory. This level can be expressed in mathematical terms, though it can also be described in conventional language similar to the business method claims rejected in *Bilski*.

Underneath the abstract ideas for what the software does and the general methodology is the human-readable

source code, and underneath that, there is the object code that implements the program. Computer scientists view source and object code as equivalent to mathematical algorithms. Ben Klemens, *The Rise of the Information Processing Patent*, 14 B.U. J. Sci. & Tech. L. 1, 9–11 (2008); Ben Klemens, Math You Can't Use 26–44 (2006) [hereinafter "Math You Can't Use"].

As Donald E. Knuth, Professor Emeritus at Stanford University and one of the world's most respected computer scientists, has explained, "[it is not] possible to distinguish between 'numerical' and 'nonnumerical' algorithms, as if numbers were somehow different from other kinds of precise information." Letter from Donald Knuth, Professor Emeritus at Stanford Univ., to the Comm'r of Patents and Trademarks (Feb. 23, 1994), *available at* http://progfree.org/Patents/knuth-to-pto.txt; *see also Diehr*, 450 U.S. at 219 n.47 (Stevens, J., dissenting) (citing scholarly authority for the proposition that computer programs are algorithms).

The claims at issue in this case describe in high-level terms the steps involved in financial intermediation. Escrow arrangements are fairly viewed as "a fundamental economic practice." *Bilski*, 130 S. Ct. at 3231. The type of abstraction at issue is comparable to that in *Bilski*. Thus the claims here plainly fall outside the scope of Section 101.

Moreover, the claims here involve mathematical algorithms, inasmuch as implementing them would require source code and object code. That is, although the code is not described in any detail, the technology could never be used unless source code were written and compiled into object code. Source code and object code are properly understood as essentially mathematical and algorithmic in

nature. Thus the invention here is also correctly viewed as the type of abstract idea found unpatentable in *Benson*. For this reason as well, the claims here fall outside the scope of Section 101.

II. AN ABSTRACT IDEA MAY BE PART OF A PATENTABLE INVENTION, BUT SUCH AN INVENTION MUST CONTAIN MORE THAN MERE POST-SOLUTION ACTIVITY TO SATISFY SECTION 101.

A. ABSTRACT IDEAS MAY BE COMBINED WITH INVENTIVE CONCEPTS AND BECOME PATENT ELIGIBLE.

Although a claim to an algorithm, law of nature, or other abstract idea by itself will not satisfy the requirements of Section 101, it may be part of a patentable invention. In other words, an application of an abstract idea may be patentable, even though the abstract idea alone is not. *Mayo Collaborative Servs. v. Prometheus Labs., Inc.*, 132 S. Ct. 1289, 1293–94 (2012). There must be "elements or a combination of elements, sometimes referred to as an 'inventive concept,' sufficient to ensure that the patent in practice amounts to significantly more than a patent upon" the abstract idea itself. *Id.* at 1294.

This is illustrated by *Diehr*, which concerned a process for molding uncured synthetic rubber into cured precision products. 450 U.S. at 177. This process involved continuously measuring the temperature in a mold and feeding the results into a computer, which would use the Arrhenius equation to recalculate the appropriate cure time. The Court concluded that when the patent was

considered as a whole, it was not an attempt to patent the Arrhenius equation, but rather a patent on an industrial process for molding rubber.

Thus *Diehr* recognized that an algorithm that is plainly unpatentable by itself may be a part of a process that is patentable when it involves a physical transformation of the sort that has traditionally been considered patentable, such as an industrial process for curing rubber. *Id.* at 184–88; *see also Flook*, 437 U.S. at 589–95. The Court has also recognized that an otherwise abstract idea may be patentable when "tied to a particular apparatus." *Flook*, 437 U.S. at 588, n.9 (citing *Cochrane v. Deener*, 94 U.S. 780, 787–88 (1876)).

B. A CLAIM ELEMENT INVOLVING ABSTRACT SOFTWARE THAT MERELY ADDS A GENERAL PURPOSE COMPUTER DOES NOT INCLUDE AN INVENTIVE CONCEPT, BUT INSTEAD INCLUDES ONLY INSIGNFICANT POST-SOLUTION ACTIVITY.

Petitioner has argued that the patent at issue here is likewise "tied to a particular apparatus," because it involves a general purpose computer. This is an argument that proves too much. If running software on a computer were sufficient to overcome the prohibition on patenting abstract ideas, all software written to run on a general purpose computer would satisfy Section 101. This result is foreclosed by *Benson*.

In *Benson*, the patent application covered "a method of programming a general-purpose digital computer to

convert signals from binary-coded decimal to pure binary form." 409 U.S. at 65. The Court found that the procedure at issue was a mathematical algorithm, and amounted to an unpatentable abstract idea. *Id.* at 65–66, 71–72. The Court noted that the algorithm had "no substantial practical application except in connection with a digital computer… ." *Id.* at 71. The claims were intended to "cover any use of the claimed method in a general-purpose digital computer of any type." *Id.* at 64.

Thus the algorithm claimed in *Benson* could have been viewed as "tied to a machine," inasmuch as it was functionally tied to a digital computer. Nevertheless, the Supreme Court held that the claims were abstract ideas outside the scope of Section 101. *Benson* therefore precludes an interpretation of Section 101 that views abstract ideas as patentable based on their implementation in software running on a general purpose computer.

Diehr is consistent with this understanding. The rubber curing process in *Diehr* involved an algorithm and a computer, but the Court's analysis of subject matter turned on the claim as a whole, which concerned the physical transformation of the rubber—not the implementation of the algorithm in a computer program. 450 U.S. at 184–88. *Diehr* explained that use of the computer did not render the process unpatentable, but the decision makes clear, by its focus on the transformation of rubber, that use of the computer alone does not suffice to make the process patentable. *See id.* at 187.

The *Diehr* Court cautioned against allowing the prohibition on patenting of abstract formulas to "be circumvented by attempting to limit the use of the

formula to a particular technological environment." *Id.* at 191 (citing *Parker v. Flook*, 450 U.S. 584). "To hold otherwise would allow a competent draftsman to evade the recognized limitations on the type of subject matter eligible for patent protection." *Id.* at 192. In view of this caution, this Court should make clear that the test for patentable subject matter cannot be satisfied by the mere drafting device of adding components of a general purpose computer as limitations of a claim that is otherwise directed to an unpatentable algorithm.

Although computers are extraordinary machines, in the last half century they have become as fundamental and common as the telephone once was, or before that the telegraph, and countless earlier human technologies that were major advances in their time. It is clear that a general purpose computer by itself cannot amount to an "inventive concept." *See Mayo*, 132 S. Ct. at 1294. When software has been written for a general purpose computer, it is reasonable to view running that software on a computer as "insignificant post-solution activity." *Diehr*, 450 U.S. at 191.

However, some have taken the view that a general purpose computer running a software program becomes a special purpose computer. *See In re Alappat*, 33 F.3d 1526, 1545 (Fed. Cir. 1994) (en banc), *abrogated by In re Bilski*, 545 F.3d 943 (Fed. Cir. 2008). This view is highly artificial, if not metaphysical.

Modern computers are designed to be capable of running software of varying designs and purposes. Running a program does not materially change the physical hardware. To be sure, each program has an

ephemeral effect on the voltage levels that implement the ones and zeros of machine language. Viewing such an effect as creating a special purpose computer would be like saying a telephone becomes a "special purpose telephone" when used for a particular call, or a television becomes a 'special purpose television" when used to watch a particular channel or other content. In short, viewing a computer as being fundamentally transformed each time a program is run is untenable.

III. BY CORRECTING LOWER COURT CASE LAW PERMITTING WHOLESALE PATENTING OF SOFTWARE, THIS COURT WILL REMOVE A BURDEN ON INNOVATION THAT HAS CAUSED SIGNIFICANT ECONOMIC HARM.

A. SOFTWARE INNOVATION LONG PREDATED SOFTWARE PATENTS.

The importance of the software industry to the United States economy is well recognized. What is less well recognized is that major innovations and enormous economic successes in the software industry occurred without the encouragement or threat of software patents. Such extraordinarily successful software products as Microsoft Word, Oracle Database, Lotus 1-2-3, the Unix operating system, and the GNU C compiler all date from the 1980s or earlier—well before the proliferation of software patents that began in the 1990s. Market forces, rather than patents, spurred development of these products. *See* Fed. Trade Comm'n, To Promote Innovation: The Proper Balance of Competition and Patent Law and Policy Ch. 3, § V(C) (2003) [hereinafter "FTC Innovation Report"], *available at* http://www.

ftc.gov/sites/default/files/documents/reports/promote-innovation-proper-balance-competition-and-patent-law-and-policy/innovationrpt.pdf; *see also* James Bessen & Robert M. Hunt, *An Empirical Look at Software Patents* 6 (Fed. Reserve Bank of Philadelphia, Working Paper No. 03-17/R, 2004) [hereinafter "*An Empirical Look*"], *available at* http://www.researchoninnovation.org/swpat.pdf.

In the 1972 *Benson* decision, this Court took note of the exclusion of software from patenting, of problems caused by attempts to patent software, and of the industry's impressive growth without patents. "Direct attempts to patent [software] programs have been rejected on the ground of nonstatutory subject matter." *Benson*, 409 U.S. at 72 (quoting The President's Comm'n on the Patent Sys., To Promote the Progress of Useful Arts, S. Doc. No. 90-5, at 13 (1st Sess. 1967) [hereinafter "President's Comm'n Report"]). "Indirect attempts to obtain patents and avoid the rejection, by drafting claims as a process, or a machine or components thereof programmed in a given manner, rather than as a program itself, have confused the issue further and should not be permitted." *Id.* (quoting President's Comm'n Report, *supra*, at 13).

The *Benson* Court noted "that the creation of programs has undergone substantial and satisfactory growth in the absence of patent protection and that copyright protection for programs is presently available." *Id.*; *accord Diehr*, 450 U.S. at 217 (Stevens, J. dissenting) (noting commentary that the software industry "is growing by leaps and bounds without [patent protection].").

Software patent policy changed in the mid-1990s. In 1996, the Examination Guidelines for Computer-Related

Inventions changed the PTO's approach and authorized such patents. Examination Guidelines for Computer-Related Inventions, 61 Fed. Reg. 7478, 7479–92 (Feb. 28, 1996). Around this time, the Federal Circuit expanded its view of patent eligibility for software. *See An Empirical Look*, *supra*, at 3; *In re Alappat*, 33 F.3d at 1544–45; *State St. Bank & Trust Co. v. Signature Fin. Grp., Inc.*, 149 F.3d 1368 (Fed. Cir. 1998), *abrogated by In re Bilski*, 545 F.3d 943 (Fed. Cir. 2008). Software patenting greatly expanded in the following years. *An Empirical Look*, *supra*, at 3, 5.

The point here is that the software industry began and reached maturity without the benefit of patent monopolies. This is not to say there was no legal protection for software products. As the *Benson* Court noted, copyright law provided (and it still provides) substantial protection for software products.[5]

---

5. Although it is frequently assumed that patents encourage technological innovation, there is little empirical evidence supporting this view. In a 2003 report entitled "Patents in the Knowledge-Based Economy," the National Research Council conducted a comprehensive review of the United States patent system, and concluded that "[t]here are theoretical as well as empirical reasons to question whether patent rights advance innovation in a substantial way in most industries." Nat'l Research Council, Patents in the Knowledge-Based Economy 2 (2003). Scholarly studies have called into question the basic assumption that patent protection in general spurs innovation. *See* Michele Boldrin & David K. Levine, *The Case Against Patents* 1 (Fed. Reserve Bank of St. Louis, Working Paper No. 2012-035A, 2012), *available at* http://research.stlouisfed.org/wp/2012/2012-035.pdf ("there is no empirical evidence that [patents] serve to increase innovation and productivity"); Andrew W. Torrance & Bill Tomlinson, *Patents and the Regress of Useful Arts*, 10 Colum. Sci. & Tech. L. Rev. 130, 133–34 (2009), *available at* http://www.stlr.org/html/volume 10/Torrance.pdf.

This recent history, by itself, calls into serious question whether software patents serve the primary purpose of the patent system of encouraging innovation. *See* U.S. Const. art. I, § 8. Many of the world's most successful software companies and software products originated and grew strong without incentives from patents. Instead, these successes arose from the dynamics of the competitive market place. FTC Innovation Report, *supra*, at Ch. 3, § V(C). That is, prior to the expansion of patentability for software in the mid-1990s, survival in the market place for software depended primarily on the ability to innovate better and more quickly than competitors. Competition, without patent monopolies, resulted in a remarkably dynamic software industry with an impressive record of innovation.

B. THE PROLIFERATION OF SOFTWARE PATENTS HAS RESULTED IN NEW RISKS THAT DISCOURAGE INNOVATION.

Since the mid-1990s, software patents have proliferated. At present in the United States there are hundreds of thousands of issued software patents. *See* James Bessen & Michael J. Meurer, Patent Failure 22 (2008) [hereinafter "Patent Failure"]. They continue to increase at the rate of tens of thousands per year. *An Empirical Look*, *supra*, at 3. This proliferation has raised significant risks for software developers.

For years, far-sighted industry leaders, scholars, and software developers have warned of these risks and opposed software patents. Patent Failure, *supra*, at 189. These include Bill Gates, co-founder of Microsoft. In 1991, Mr. Gates, stated, "If people had understood how

patents would be granted when most of today's ideas were invented, and had taken out patents, the industry would be at a complete standstill today… ." Kirk D. Rowe, *Why Pay for What's Free?: Minimizing the Patent Threat to Free and Open Source Software*, 7 J. Marshall Rev. Intell. Prop. L. 595, 595 (2008).[6]

Similarly, Professor Knuth, of Stanford University, wrote in 1994, "When I think of the computer programs I require daily to get my own work done, I cannot help but realize that none of them would exist today if software patents had been prevalent in the 1960s and 1970s." Letter from Donald Knuth to the Comm'r of Patents and Trademarks, *supra*. Dr. Knuth also stated, "I strongly believe that the recent trend to patenting algorithms is of benefit only to a very small number of attorneys and inventors, while it is seriously harmful to the vast majority of people who want to do useful things with computers." *Id.*

The views of Mr. Gates and Professor Knuth were shared by many firms and developers in the 1990s. *See* Patent Failure, *supra*, at 189. The risk that they articulated—that patents tend to hinder software innovation—relates to at least two different aspects of software: the incremental nature of software development

6. Adobe and Oracle likewise opposed software patents in 1994. *See* Public Hearing on the Use of the Patent System to Protect Software-Related Inventions Before the U.S. Patent & Trademark Office 16–18 (1994) (statement of Douglas Brotz, Principal Scientist, Adobe Systems, Incorporated), *available at* http://www.uspto.gov/web/offices/com/hearings/software/sanjose/sjhrng.html; *id. at* 23–25 (statement of Jerry Baker, Senior Vice President, Oracle Corporation).

and the near impossibility of establishing clear boundaries for software patents.

In general, software innovation is cumulative in nature—that is, new products typically build on products built previously. *See* FTC Innovation Report, *supra*, at Ch. 3, § V(B); The Patent Crisis, *supra*, at 47. Innovation is rapid and product cycles are short. Major software products are complex, involving many thousands or even millions of lines of code and many different components. Components are normally developed using many earlier-developed sub-components. Some software products contain thousands of distinguishable components, any number of which could (in view of erroneous patenting practices) already be patented. *See* FTC Innovation Report, *supra*, at Ch. 3, § V(E)(2)(c); The Patent Crisis, *supra*, at 53–54.

It is, however, practically impossible to know with reasonable certainty whether a new software product could be said to infringe some prior software patent. Patents are conventionally referred to as intellectual property. However, as James Bessen and Michael Meurer have explained in detail, patents differ substantially from tangible property in that their boundaries are often fuzzy and unpredictable. Patent Failure, *supra*, at 46–72. If patents do not give clear notice of their limits, they create a risk of inadvertent infringement. Vague patents also enable opportunistic behavior. For example, a patentee may, based on vague language, claim ownership of a technology unknown to the inventor, but instead first conceived by someone else.[7] *Id.* at 199.

7. The unreliability of indexing for software patents also means that software patents are of little use in advancing

This problem of uncertain patent boundaries is particularly acute with software patents. As discussed in Section I, software is an abstract technology, with various levels of abstraction. Software algorithms can be represented in numerous different ways, and even computer scientists sometimes disagree over whether two software technologies are equivalent. *See* PATENT FAILURE, *supra*, at 22, 203. Thus it is not surprising that software patents are typically framed in abstract language with uncertain boundaries. *See id.* at 22–23, 203; THE PATENT CRISIS, *supra*, at 27, 58. As a result, a software developer, when shown a software patent, often cannot be sure whether the patent reads on newly developed code. *See* FED. TRADE COMM'N, THE EVOLVING IP MARKETPLACE 80–83 (2011) [hereinafter "FTC IP MARKETPLACE REPORT"], *available at* http://www.ftc.gov/sites/default/files/documents/reports/evolving-ip-marketplace-aligning-patent-notice-and-remedies-competition-report-federal-trade/110307patentreport.pdf.

---

innovation by disclosing new technology. From a software developer's point of view, it is completely impractical to seek new ideas in patents, and few if any do so. Most avoid reading patents, for fear that a chance encounter may increase the risk that they will one day be accused of willful infringement. *See* THE PATENT CRISIS, at 32; Mark A. Lemley, *Ignoring Patents*, 2008 MICH. ST. L. REV. 19, 21–22 (2008). A further indication of the ineffectiveness of the patent system for software is that there is little patent licensing prior to development and distribution of products. *See* THE PATENT CRISIS, *supra*, at 59. Because of vague patent boundaries and unreliable search methods, it is not possible to determine all possible rights at issue and strike bargains as to those rights.

This difficulty is multiplied hundreds or thousands of times with regard to a complex software product combining hundreds or thousands of discrete components. A separate but related problem faces all software developers—that of the impossibility of patent clearance, or determining whether there are existing patents that may be said to read on a new product. There is no reliable, economical method for searching the hundreds of thousands of existing software patents. *See* PATENT FAILURE, *supra*, at 50, 69–70; *see also* Christina Mulligan & Timothy B. Lee, *Scaling the Patent System*, 68 N.Y.U. ANN. SURV. AM. L. 289, 289, 297–305 (2012); FTC IP MARKETPLACE REPORT, *supra*, at 90–91; Klemens, *supra*, at 79–80. The clearance problem is made even worse by the existence of tens of thousands of applications that, for eighteen months after filing, are unpublished.

Thus, simply by virtue of producing and marketing an innovative software product, a software developer assumes the risk of a costly patent infringement lawsuit. *See* FTC INNOVATION REPORT, *supra*, at Ch. 3, § V(E). In the U.S., software patents are more than twice as likely to be the subject of a lawsuit than other patents and account for one quarter of all patent lawsuits. PATENT FAILURE, *supra*, at 22, 192. The cost of defending a patent lawsuit frequently amounts to several million dollars. AM. INTELL. PROP. L. ASS'N, REPORT OF THE ECONOMIC SURVEY 35, I-145–46 (2013). Such lawsuits involve technical issues that are difficult for judges and juries to understand, and so even with a strong defense the outcome is usually far from certain. If there is a judgment of infringement, the penalty may be an injunction ending further production and enormous monetary damages. Defense costs and litigation risks are so large that in most cases defendants

agree to some payment to settle such cases. Even when claims appear to have no valid basis, targets frequently agree to pay for licenses based on the mere threat of litigation. Michael J. Meurer, *Controlling Opportunistic and Anti-Competitive Intellectual Property Litigation*, 44 B.C. L. Rev. 509, 513–15 (2003).

Some large technology companies have addressed the risk of inadvertent infringement of patents by seeking as many patents as possible, on the theory that a large patent portfolio signals the possibility of a countersuit and thus will deter other companies from bringing a patent lawsuit.[8] *See* FTC Innovation Report, *supra*, at Ch. 2, § (III)(C)(2)(b); The Patent Crisis, *supra*, at 55. Companies with such portfolios often enter into cross-licensing agreements with other large companies that have their own patent portfolios in an attempt to obtain a modicum of patent peace. FTC Innovation Report, *supra*, at Ch. 3, § V(E)(2)(c)(i); *see also* Richard M. Stallman, Speech at the University of Cambridge, London (Mar. 25, 2002), *in* Free Software, Free Society: Selected Essays of Richard M. Stallman 97, 103–04 (2002); Math You Can't Use, *supra*, at 83–85.

While such defensive measures are understandable from an individual enterprise's perspective, they are far from optimal. They create a vicious cycle: to defend

8. Red Hat, like some of its competitors, has built a patent portfolio. This portfolio is designed to be used only for the purpose of defending against patent aggression. Red Hat has extended a public Patent Promise under which it pledges not to enforce its patents against parties that infringe those patents through their use of software covered by designated open source licenses. *See* https://www.redhat.com/legal/patent_policy.html.

against a multitude of vague patents, companies obtain still more vague patents. Resources expended on this strategy are, of course, unavailable for research and development or for other more productive purposes. FTC INNOVATION REPORT, *supra*, at Ch. 3, § V(E)(2)(c). Moreover, although established companies may be able to bear the cost of this deterrence strategy, small companies and potential new competitors generally lack the resources to do so. Thus the system discourages new entry into the market, and thereby hinders innovation. *See id.*; *see also* CLAYTON M. CHRISTENSEN, THE INNOVATOR'S DILEMMA 9, 52 (2002) (showing that small firms generally lead in new technologies).

Moreover, even for companies with the financial resources to build patent portfolios, the deterrence approach is not always effective. With the proliferation of software patents has come the expansion of a class of businesses created expressly for the purpose of exploiting vague patents. *See* FTC IP MARKETPLACE REPORT, *supra*, at 60–62; Klemens, *supra*, at 27–31 (2008); Meurer, *supra*, at 509.

These are sometimes referred to as non-practicing entities ("NPEs"), patent assertion entities ("PAEs") or, less politely, as patent trolls. These entities acquire vague patents at low cost with a view to threatening or bringing lawsuits against operating businesses. They frequently conceal their identities and holdings until the companies that are their targets, which have no knowledge of the relevant patents, are locked in to a product and business strategy. Then they demand ransom. Because such entities produce no products, they are not deterred by the possibility of a countersuit.

NPE lawsuits have greatly increased in frequency in recent years and now account for more than 60 percent of all U.S. patent litigation. James Bessen & Michael J. Meurer, *The Direct Costs from NPE Disputes* 4 (Boston Univ. Sch. of Law, Working Paper No. 12-34, 2012), *available at* http://www.bu.edu/law/faculty/scholarship/workingpapers/2012.html; Colleen V. Chien, *Patent Trolls by the Numbers*, (Santa Clara Univ. Sch. of Law Research Paper No. 08-13, 2013), *available at* http://ssrn.com/abstract=2233041. NPE lawsuits are concentrated in the technology sector, and between 62 and 94 percent involve software patents. Bessen & Meurer, *supra*, at 7. Professors Bessen and Meurer have calculated the direct costs of NPE assertions in just one year (2011) at $29 billion. *Id.* at 2. They point out that much of this amount is social loss, and amounts to a tax on innovation. *Id.* at 5.

Recently a hybrid type of NPE has emerged which is sometimes referred to as a patent privateer. Operating companies that might otherwise be deterred from suing a competitor by the threat of a patent countersuit instead recruit an NPE to bring suit. The resulting lawsuit may cost the target company millions to defend, as well as distracting it from innovating and other business activities. *See* Edith Ramirez, Chairwoman, Fed. Trade Comm'n, Opening Remarks at the Computer & Communications Industry Association and American Antitrust Institute Program (June 20, 2013), *available at* http://www.ftc.gov/sites/default/files/documents/public_statements/competition-law-patent-assertion-entities-what-antitrust-enforcers-can-do/130620paespeech.pdf; Michael A. Carrier, *Patent Assertion Entities: Six Actions the Antitrust Agencies Can Take*, 1 CPI Antitrust Chron. 2, 4 (2013), *available at* http://ssrn.com/abstract=2209521.

In sum, all software companies, developers, and users face substantial risks from software patents. These risks include whether an unknown patent may cover a newly written code or some other preexisting code in a complex product, whether such a patent could be the basis of a lawsuit, whether a relevant patent is in the hands of an aggressive party dedicated to bringing patent lawsuits, and whether a trial may result in an injunction or damages award.

These risks have obviously not brought the software industry to a standstill. But scholars have found that they reduce research and development and reduce entrances of competitors into the market. *See An Empirical Look*, *supra*, at 38–39; Iain M. Cockburn & Megan J. MacGarvie, *Entry and Patenting in the Software Industry* 21 (Nat'l Bureau of Econ. Research, Working Paper No. 12563, 2006), *available at* http://www.nber.org/papers/w12563.pdf. This evidence that software patents are hindering, rather than promoting innovation and growth is a strong signal that the patent law requires a course correction.

To be sure, correcting the patent law to apply this Court's precedents to software patents will disadvantage those who currently profit from such patents, such as NPEs, companies with software patent licensing programs, and some lawyers. But a large share of existing software patents sit in large firms' portfolios and have little economic effect other than (at times) deterring competitor's patent lawsuits.[9] Invalidating these defensive

9. One study found that "[t]wo-thirds of all software patents are obtained by a small group of industries known for strategic patenting," and that this group "appears to have substituted these patents for R&D." *An Empirical Look*, *supra,* at 38.

patents will accord with the original aim of those portfolios of lessening litigation risk. Moreover, this Court's precedents have provided ample foreshadowing of the statutory problem with such patents. Any temporary costs should be more than counterbalanced by the increased economic benefits created by removing a major hindrance to software innovation.

Respectfully submitted,

ROBERT H. TILLER
Counsel of Record
RED HAT, INC.
100 East Davie Street
Raleigh, NC 27601
(919) 754-4232
rtiller@redhat.com

Counsel for Amicus Curiae
Red Hat, Inc.