

SOFTWARE PATENTS: WHAT ONE-CLICK BUY AND SAFE AIR TRAVEL HAVE IN COMMON

Have you ever sat in an airplane, typing on your laptop, when the darn thing crashes for the one-millionth time? Have you ever then thought about how the airplane you are sitting in is controlled by software, too—the technical term being “fly by wire”¹—and then started sweating uncontrollably? Software controls not only air traffic but plenty of other safety-critical technologies: the tightrope walk of controlling the chain reaction of radioactive elements in nuclear power plants;² the navigation and activation of missiles;³ the moves and cutting-depth of a surgical laser when correcting eye-sights;⁴ the list goes on... With such reliance on software, malfunction due to errors in the program code becomes unacceptable. Software patents help heighten the standard by supporting the re-use of the code of established and tested systems.⁵

Introduction

Many people associate software patents with the case of *Amazon.com v. Barnes & Noble*⁶ and the dispute over the ‘411 patent, which granted Amazon a property right in a “[m]ethod . . . for placing a purchase order.”⁷ This association is generally coupled with skepticism about whether a single-click shortcut for online purchases is worthy of two decades of government-enforced protection.⁸ While this surely is a simplification of the case, and valid arguments have been advanced by both sides,⁹ software patents have also been said to serve higher purposes: they positively affect software quality.¹⁰ Since higher software quality means safer software and with software controlling medical devices, airplanes and nuclear power plants, software patents affect all of us.

¹ See e.g. James E. Tomayko, *Computers Take Flight: A History of NASA's Pioneering Digital Fly-By-Wire Project* (2000), available at <http://www.dfrc.nasa.gov/History/Publications/f8ctf/index.html>.

² See e.g. Katie Walter, *Reliable Software for Protection Systems*, Science & Technology, March 1998, at 21, 21, available at <http://www.llnl.gov/str/03.98.html>.

³ See e.g. *Hi-Tech, Low Cost*, Global Defence Review, 1999, at <http://www.global-defence.com/99/missiles/missile4.htm>.

⁴ See e.g. Brian R. Will, *Locking On: Tracking the Technology*, Ophthalmology Management, Sept. 2000, at 40, available at <http://www.ladarvision.com/pros/library/default.asp>.

⁵ See Mark A. Lemley & David W. O'Brien, *Encouraging Software Reuse*, 49 Stan. L. Rev. 255 (1997).

⁶ *Amazon.com, Inc. v. Barnesandnoble.com, Inc.*, 239 F.3d 1343 (Fed. Cir. 2001); see also Stephen Dirksen et al., *Who's Afraid of Amazon.com v. Barnesandnoble.com?*, 2001 Duke L. & Tech. Rev. 3 (discussing the prospective impact of the case).

⁷ U. S. Patent No. 5,960,411 (issued Sept. 28, 1999).

⁸ 35 U.S.C. § 154(2) (1994).

⁹ See e.g. Larry Graham, *Debunking Software Patent Myths*, IEEE Software, July-Aug. 2000, at 122, 123.

¹⁰ See Lemley, *supra* note 5, at 262-63, 265, 294-99.

All Software Contains Errors

Virtually all, that is - statistically speaking, there are generally between one and five undetected defects, i.e. coding errors that evade review and are released to the public, per one thousand lines of code.¹¹ Y2K corrections did not catch all date-critical code statements either; one to fifty errors per ten thousand lines of code were left in a system.¹² The problems caused by the same percentage of defects also increase with system size because of the added complexity.¹³ Microsoft Windows 2000, admittedly an example of particularly bad code quality, contains about forty million lines of code and weighs in with fifteen errors per thousand lines, an estimated six hundred thousand bugs.¹⁴ (Still puzzled why your laptop crashes all the time?) Modern medical devices generally contain smaller, embedded software systems, but still involve between ten and hundreds of thousands of lines of code.¹⁵

Air traffic control systems involve millions of lines of code.¹⁶ Raytheon's radar airport approach system STARS (Standard Terminal Automation Replacement System), for example, which was adopted by the Federal Aviation Administration and the Department of Defense for civil and military airports, respectively, involves 1.6 million lines of code.¹⁷ Modern airplanes add further millions to the equation: Boeing's newest flagship, the 777, has four million lines of software on board, 2.5 million of which were newly developed (the rest being reused from existing systems).¹⁸ The Airbus A380, scheduled to begin commercial service in 2006,¹⁹ is expected to put a solid one billions lines of code to use.²⁰

Software Safety and Patents

Software has been error-prone since its early childhood, when its complexity outgrew the diligence of human programmers. In 1968, scholars from around the globe met for the first International

¹¹ See e.g. Watts S. Humphrey, *The Future of Software Engineering: I*, news@sei interactive, 1st Q. 2001, at http://interactive.sei.cmu.edu/news@sei/columns/watts_new/2001/1q01/watts-new-1q01.htm.

¹² See e.g. Doug Black, *Barnes & Noble Conducting Extra Y2K Due Diligence; Data Integrity IV&V Tool Checking for Lingering Y2K Bugs*, Y2K Wire, April 7, 1999, at http://www.year2000.com/releases/dii04_07_1999.html.

¹³ Humphrey, *supra* note 11.

¹⁴ Bernhard Cole, *Will Microsoft Win The Embedded Code Quality Battle?*, IApplianceWeb, April 1, 2002, at <http://www.iapplianceweb.com/story/OEG20020104S0084>.

¹⁵ See e.g. Terry Costlow, *Medical design's a bloody long haul*, EE Times, August 15, 1997, at <http://www.eetimes.com/news/97/967news/medical.html>.

¹⁶ See e.g. Tekla S. Perry, *In Search of the Future of Air Traffic Control*, IEEE Spectrum, Aug. 1997, at 18, available at http://sdg.lcs.mit.edu/atc/D2/spectrum_aug97_atc.htm.

¹⁷ See e.g. *FAA Runs Shakedown Cruise of its Approach System*, Government Computer News, March 6, 2000, at http://www.gcn.com/vol19_no5/news/1443-1.html.

¹⁸ Ron J. Pehrson, *Software Development for the Boeing 777*, Cross Talk, Jan. 1996, at 2, available at <http://www.stsc.hill.af.mil/CrossTalk/1996/jan/Boein777.asp>.

¹⁹ *A 380 Family*, Airbus web site, at http://www.airbus.com/product/a380_backgrounder.asp.

²⁰ See e.g. Pierre de Chazelles, *The A380 Challenges for Software Engineering*, Keynote address at the 8th Eur. Software Eng'g Conf. (September 14, 2001).

Conference on Software Engineering, which targeted the following problems: software development is late, over budget, and fails to perform as expected.²¹ Software engineers subsequently developed numerous approaches attempting to solve these problems, mostly aiming at the software development process, i.e. (1) the art of specifying a software's desired functionality before its development, (2) general approaches to software design and implementation, and finally, (3) testing whether the software does what it promises and whether it contains errors.²²

Today, the software community has widely given up on the idealistic view that completely error-free software can be created anytime soon.²³ Instead, the concern has shifted to safety-critical systems. Acknowledging that any modern system contains errors in its code, the goal is to ensure that none of these errors result in behavior that puts people's lives or health at risk.²⁴

Since the best way to put software to the test is to release it to mankind, one approach to address the problem is to reuse program code that has been successfully run in existing systems. The system can not only show whether it is reliable in the first place, but user feedback also leads to the detection and correction of errors. Program code that has proved dependable in one system is therefore likely to cause less trouble when reused than new code written from scratch.²⁵

Reusable program code preferably comes in the form of components, i.e. blocks of software where the inner workings do not concern the recipient, who simply plugs the component into her system and accesses its functionality via an interface. Similar to a car manufacturer buying most components from suppliers and only putting them together, the vision for creating software in the future is to buy components “off-the-shelf” and build a system from parts.²⁶

Sadly, this is still a vision.²⁷ First, components cannot interact unless they share a common interface.²⁸ Unfortunately, the lucrative perspective of being the developer of a standard interface has led to each major player, including the Microsoft Corp., Sun Microsystems and an independent conglomerate

²¹ *Highlights*, 1 Int'l Conf. on Software Eng'g 3 (1968), available at <http://www.cs.ncl.ac.uk/people/brian.randell/home.formal/NATO>

²² See e.g. Dino Mandrioli et al., *Fundamentals of Software Engineering* (1991).

²³ See Frederick P. Brooks, *No Silver Bullet: Essence and Accidents of Software Engineering*, IEEE Software, July-Aug. 1987, at 10.

²⁴ See e.g. Robyn R. Lutz, *Software Engineering for Safety: A Roadmap*, in *The Future of Software Engineering* (2000), at 213, available at <http://www.softwaresystems.org/future.html>.

²⁵ Lemley, *supra* note 5, at 265.

²⁶ See e.g. Vincent Trass & Jos van Hillegersberg, *The Software Component Market on the Internet: Current Status and Conditions for Growth*, Software Eng'g Notes, Jan. 2000, at 114.

²⁷ See e.g. Butler Lampson, *How Software Components Grew Up And Conquered The World*, Keynote address at the 22nd Int'l Conf. on Software Eng'g (May 21, 1999) (transcript available at <http://research.microsoft.com/lampson/slides/ReusableComponentsAbstract.htm>).

²⁸ David Garlan et al., *Architectural Mismatch or Why it is Hard to Build Systems out of Existing Parts*, 17 Int'l Conf. on Software Eng'g 179 (1995).

called the Object Management Group, to develop its own, thereby leading to a proliferation of standards.²⁹ Second, unlike manufacturing hardware components, implementing software does not require specialized, expensive technology. Any company can develop its own code, which is generally economically more viable—at least in the short run—than purchasing existing components.³⁰

Patents target both problems. While most software is still only protected by copyright, a type of protection that discourages reuse,³¹ patents can give the first developer of an interaction standard or of a software component, which executes tasks in a novel and patent-worthy way, exclusive rights.³² The owner of the patent can not only keep other developers from re-inventing the same technology, but will generally sell licenses where he forces the licensee to reuse his tested, high quality code rather than to write cheap re-implementations.³³

The Law

Title 35 of the United States Code defines which inventions are patentable: “[a]ny new and useful process, machine, [article of] manufacture, or composition of matter, or any new and useful improvement thereof.”³⁴ Laws of nature, scientific phenomena, or mathematical formulae are excluded.³⁵ Exclusive rights to such fundamental “scientific truths” of our world would grant unreasonable control to individuals.³⁶ In this light, the courts treated software suspiciously at first. In the 1970s, the Supreme Court held that software was essentially mathematical formulae, not patentable under U.S. law.³⁷

In 1981, the Supreme Court decided in *Diamond v. Diehr* that an invention could not be denied a patent solely because it contained a computer program.³⁸ Diehr had filed a patent application for a process for molding synthetic rubber—the technical term being “curing.”³⁹ Curing rubber is apparently a quite sophisticated art, depending on multiple factors such as rubber size, thickness, cure time and

²⁹ Michael S. Guntersdorfer & David G. Kay, *How Software Patents Can Support COTS Component Business*, IEEE Software, May-June 2002, at 78 - 80, available at <http://www.computer.org/software/homepage/2002/03COTS/index.htm>.

³⁰ *Id.* at 81.

³¹ Lemley, *supra* note 5, at 290-93.

³² *Id.* at 295.

³³ Guntersdorfer, *supra* note 29, at 81.

³⁴ 35 U.S.C. § 101.

³⁵ See *Mackay Radio & Tel. Co. v. Radio Corp. of Am.*, 306 U.S. 86, 94 (1939); *Funk Bros. Seed Co. v. Kalo Inoculant Co.*, 333 U.S. 127, 130 (1948).

³⁶ See *Mackay Radio*, 306 U.S. at 94.

³⁷ *Gottschalk v. Benson et al.*, 409 U.S. 62, 72 (1972) (holding that a mathematical algorithm itself is not patentable but adding that it may be that the patent law should be extended to cover programs); *Parker v. Flook*, 437 U.S. 584, 596 (1978) (refusing to overrule or expand *Gottschalk* without a clear signal from Congress).

³⁸ *Diamond v. Diehr et al.*, 450 U.S. 175, 187 (1981).

³⁹ *Id.* at 177.

pressure, and the temperature inside the mold.⁴⁰ The temperature factor had been viewed as an “uncontrolled variable” before Diehr used a computer program that constantly measured the inside of the mold and controlled the press accordingly.⁴¹

The court held that the patent could not be denied solely because its claims contained mathematical formulae.⁴² Instead, the court looked at the whole invention, the “industrial process for the molding of rubber products,” and granted the patent.⁴³ As a result, the U.S. Patent and Trademark Office developed guidelines and began issuing patents that were explicitly software related.⁴⁴ Two exceptions remained in place, however: the mathematical algorithm exception⁴⁵ and, arguably, the business method exception.⁴⁶

In 1998, the Court of Appeals for the Federal Circuit threw out both exceptions in *State Street Bank & Trust Company v. Signature Financial Group*.⁴⁷ The court held that the mathematical algorithm test was misleading⁴⁸ and that the business method exception had never existed as such, but rather that prior business method inventions had always been denied on other grounds.⁴⁹ The court reasoned that instead of focusing on the category of the subject matter of an invention, its practical utility is essential, which should be tested together with the requirements of novelty and non-obviousness.⁵⁰ Signature Financial Group's “Data Processing System for Hub and Spoke Financial Services Configuration”⁵¹ satisfied the requirements and the court upheld the patent.⁵²

The broad holding in *State Street* was arguably limited subsequently in *WMS Gaming Inc. v. International Game Technology*, where the court returned to its prior holding of *In re Alappat* that

⁴⁰ Id.

⁴¹ Id. at 178-81.

⁴² Id. at 187.

⁴³ Id. at 192-93.

⁴⁴ U.S.P.T.O., *Examination Guidelines for Computer-Related Inventions*, 61 Fed. Reg. 7,478 (Feb. 28, 1996).

⁴⁵ The courts created the Freeman-Walter-Abele test to determine whether an algorithm only represents an abstract idea. *In re Pardo*, 684 F.2d 912, 915 (CCPA 1982); *see also* *In re Freeman*, 573 F.2d 1237, 1246 (CCPA 1978); *In re Walter*, 618 F.2d 758, 767 (CCPA 1980); *In re Abele*, 684 F.2d 902, 907 (CCPA 1982).

⁴⁶ The business method exception had been implied to exist but never explicitly upheld. *See e.g.* *In re Howard*, 394 F.2d 869, 870-72 (CCPA 1968) (mentioning the alleged business method exception but stopping short of deciding whether business methods are inherently unpatentable as suggested by concurring Judge Kirkpatrick).

⁴⁷ *State St. Bank & Trust Co. v. Signature Fin. Group*, 149 F.3d 1368, 1373-77 (Fed. Cir. 1998).

⁴⁸ Id. at 1373 (*quoting* *In re Alappat*, 33 F.3d 1526, 1544 (Fed. Cir. 1994)).

⁴⁹ Id. at 1375.

⁵⁰ Id.; 35 U.S.C. § 101 contains the usefulness requirement and classifies inventions as processes, machines, manufactures, or compositions of matter. § 102 demands novelty. § 103 requires that an invention is non-obvious.

⁵¹ U.S. Patent No. 5,193,056 (issued March 9, 1993).

algorithms are patentable because they limit a general purpose computer to a specific purpose performing functions pursuant to the software.⁵³ This statement seems narrower than *State Street's* useful versus abstract distinction.⁵⁴ Still, the fact that software is patentable seems untouched.⁵⁵

Conclusion

With legal barriers out of the way, software patents can now be deployed to enforce software reuse. As a result, software quality could increase for two distinct reasons. First, license-enforced reuse leads to the establishment of patented code in multiple systems rather than each vendor cooking its own soup. The more the same program code is used, however, the more feedback will be created, the more errors uncovered and eliminated. Second, patent protection makes diligent code development economically viable again. The prospect of twenty years of protection against infringement makes higher lead costs due to time and resource consuming software engineering methods worthwhile. Quick and dirty implementations in order to win time-to-market races become unnecessary.

Higher software quality means, of course, increased software safety. The fewer errors a system contains, the less likely it is that it contains safety critical ones. In the end, software patents might help solving a problem as a by-product that software engineers in an unregulated market never could.

By: Michael Guntersdorfer

⁵² *State St.*, 149 F.3d at 1377.

⁵³ *WMS Gaming, Inc. v. Int'l Game Tech.*, 184 F.3d 1339, 1348-49 (Fed. Cir. 1999) (*quoting* Alappat, 33 F.3d at 1445).

⁵⁴ *State St.*, 149 F.3d 1373-75.

⁵⁵ *See e.g. Hughes Aircraft Co. v. United States*, 148 F.3d 1384, 1385 (Fed. Cir. 1998) (Clevenger, J., dissenting) (relying on *State Street* when noting that “[t]oday . . . virtually anything is patentable”).