Open to Inspection: Using Reverse Engineering to Uncover Software Prior Art, Part 1

ANDREW SCHULMAN
Software Litigation Consulting

SOFTWARE PATENTS ARE OFTEN CRITICIZED on the ground that patent examiners fail to examine the claims against the full scope of prior art. Software patent applications are of course examined in light of earlier software patent documents and printed publications, which are the type of documents that populate the databases used by the United States Patent and Trademark Office. However, unlike most other technologies, the state of the art in software is often found in "undocumented" aspects of products that do not find their way into standard prior-art libraries.

Part 1 of this article discusses the software prior-art problem and the reverse-engineering solution, along with the nature of software products as readable texts, as illustrated in the recent California case of *Silvaco v. Intel*.[1] Part 2 will complete the description of reverse-engineering methods for finding prior art in software products, and will address three legal issues presented by the use of reverse engineering to reveal prior art: (1) if earlier use of a given technology is only uncovered later, via reverse engineering, was the technology truly public or on sale in the earlier period?; (2) if a product's internal operation was technically susceptible to reverse engineering, but the product came with licensing language restricting reverse engineering, were the product's internals still public or on-sale, or is such language sufficient to render secret a widespread product's internals?; and (3) is a searchable database of software prior art an infringing derivative work, or is its relationship to the underlying copyrighted works a fair use, analogous to Google's indexing and caching of websites?

COMPUTER PROGRAM = INSTRUCTIONAL TEXT

Given the nature of software, it is odd that prior-art libraries are based on descriptions of computer programs, but not on computer programs themselves. A standard legal definition of a computer program is "a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result."[2] Whereas copyright law views computer programs as texts ("literary works"), the patent system—while quite sensibly viewing computer programs as devices or methods—appears to forget that that they are also sets of instructions. A set of statements or instructions is a type of document. It may be written in an unfamiliar language, but it is text which exists naturally in a searchable form. Thus software, perhaps uniquely among inventive subject matter, is already in a form amenable to integration into prior-art libraries.

SOURCE CODE OR OBJECT CODE?

Software takes basically two forms: source code and object code. Software products used by consumers generally consist of object code, which is a set of instructions *directly* used in a computer. Source code contains a more *indirect* form of instruction; it is akin to a software product's blueprints, and is (apart from "open source" products) often held as a proprietary "crown jewel" inside a software company.

Importantly, it is not only source code that is searchable text. Software products themselves—that is, the object code that comprises a product such as Windows or an iPhone app used by a consumer—are also texts which can be searched, and in which prior art can be located. The functions represented by the source code may be inferred from the object code, even though the source code itself is not typically a part of the software product delivered to consumers.

The first part of this article discusses how several simple reverse-engineering techniques can help bring software products—that is, text derived directly from the object code itself, not merely manuals, books, or articles—into the patent system as searchable prior art.

Is there any larger issue at stake here, apart from augmenting prior art with bits of the real world? In addition to helping apply the constitutional mandate of only granting patents to novel inventions, using object code as prior art will also enhance the disclosure role of the patent system, by codifying previously tacit knowledge.[3] Promoting progress in the useful arts requires incentivizing not

merely invention as such, but also its disclosure, codification, classification, and indexing as part of a larger system.

REVERSE ENGINEERING OF SOFTWARE

The legal definition of reverse engineering is "starting with the known product and working backward to divine the process which aided in its development or manufacture."[4] In the case of software, reverse engineering almost never involves trying to reconstruct the original source code from a product in object-code form. Rather, the goal is generally to learn something about a product's design, from lower-level details of the product. For example, it may be important to know which compression method a product uses internally; the product probably doesn't disclose this in its documentation or during use, and the source code is likely inaccessible, but the presence of certain numbers within the product's object code may act as a "fingerprint" of a given method.

That reverse engineering can be used to investigate claims of patent infringement is well established;[5] indeed, the Federal Rule of Civil Procedure 11(b) requirement of reasonable pre-filing inquiry may mandate it.[6] With one important exception to be discussed in part 2 of this article, the techniques and tools employed to uncover software prior art are largely similar to those used to investigate claims of infringement. As a practical matter, these techniques are more likely to be employed by the defense in patent infringement cases, than by patent examiners. The building of an extensive prior-art software library would make the results of these techniques usable by patent examiners.

THE PROBLEM OF UNDOCUMENTED PRIOR ART

In an oft-cited statement on software patents, one scholar back in 1995 referred to the problem of "'common industry knowledge' that does not appear in the scholarly literature:" "[m]any new developments in computer programming are not documented in scholarly publications at all. Some are simply incorporated into products and placed on the market; others are discussed only in textbooks or user manuals that are not available to examiners online. In an area that relies so heavily on published, 'official' prior art, a rejection based on 'common industry knowledge' that does not appear in the scholarly literature is unlikely."[7]

Six years later, this situation was unchanged: "[P]rior art in this particular industry may simply be difficult or, in some cases, impossible to find because of the nature of the software business. Unlike inventions in more established engineering fields, most software inventions are not described in published journals. Software innovations exist in the source code of commercial products and services that are available to customers. This source code is hard to catalog or search for ideas."[8]

Even with the growing professionalization of software development, and narrowing of the gap between academic computer science and industry software engineering, a large amount of software practice is not reflected in academic publications. Rather, it exists as "folklore" carried out electronically but not written down on paper.[9]

This problem of undocumented prior art may partially self-correct itself as software companies increasingly patent their inventions, and consequently document internal aspects of their technology which previously would have been shipped to customers inside products, but otherwise left uncodified. These patent documents (both granted patents and published patent applications) then become readily-searchable prior art against later patent applications.

For example, Microsoft described in a patent application from 2001 an otherwise-undocumented Windows interface known as DirectUI, and as early as a 1994 patent application provided an appendix disclosing an interface (well-known among programmers at the time both for its importance and lack of formal documentation) to create so-called "namespace extensions."[10] These interfaces were important parts of the Windows product itself, present on millions of computers worldwide, but Microsoft for a time only documented them as part of the patent system.[11] Were developers to more regularly consult the patent literature (often discouraged by employers fearful of willful infringement findings, at least pre-*Seagate*),[12] such patent-application disclosures at least would be superb examples of the patent system performing its disclosure and codification functions.

Similar improvement may also be expected as a result of initiatives for public submission of prior art, such as Peer To Patent (a joint project of the USPTO and New York Law School),[13] and from Google Code Search, which enables searching of a large body of software source code.[14] Important undocumented software features are sometimes documented by third parties who reverse engineer the products;[15] such third-party documentation has been cited in patents.[16] Indeed, whether software patent quality has suffered due to inadequacy of prior-art libraries remains an open question.[17]

But to whatever extent a searchable database of software prior art has gradually accumulated, there is one large and rather obvious hole in the prior art both during initial examination and in later invalidity contests during infringement litigation.

SOFTWARE PRODUCTS — PRIOR ART?

This obvious piece of prior art is the software itself, which in many cases (such as iPhone applications, Windows and Microsoft Office, or the "firmware" embedded inside devices such as cameras or internet routers) runs on tens of millions of machines, and is therefore on sale and in public use. Software products often contain information not described in public documents. While the workings of the software are described in the vendor's source code, this source code is generally proprietary and inaccessible to the public (even with the increasing reliance on "open source"), thereby not

only making it unsearchable but also potentially not prior art in the first place.[18] And while manuals, non-academic trade publications,[19] trade ephemera,[20] defensive publications,[21] and books are increasingly searchable, and full-text searchable at that (pre-selected keywords are becoming less important), the fact remains that many aspects of software products are not reflected in sources typically referred to as printed publications, even under that term's expansive definition as any item (even a single copy) obtainable with reasonable diligence by those concerned.[22]

It is uncontroversial that software itself (in contrast to documentation *about* the software) can constitute prior art, as of the date the software was accessible to the public.[23] The issue is collecting it and putting it into a form useful to assess novelty of later software.

Given that the world's patent systems generally regard as prior art that which has already been done before, as well as that which has been written down,[24] the problem of incorporating unpublished material into the prior art is hardly limited to software. While the vast majority of countries purport to consider prior art in all forms, including unpublished use, from anywhere in the world, as a practical matter examination is limited to published literature.[25] As noted in a global study of patent systems, while the European Patent Office (EPO), for example, operates under an absolute-novelty statute, "EPO examiners, however, do not tour the world in an anthropological search for oral prior art. Instead, much like their US counterparts, they search databases consisting of patented and non-patented literature...the search for novelty is at its most robust for technology areas where the patent literature is the thickest and at its weakest for prior art that exists in some non-written form of social practice."[26] Similarly, another scholar notes that historically examiners "could check a text against another text they could find in their library, but could not travel the world looking for machines."[27]

Software is an unusual type of machine, however. It is made out of the same stuff as texts; meaning, not ink and paper, but symbols. Unlike physical machines or chemicals, software is of course already in electronic form. It is already naturally in the form into which Google, for example, is laboring to put the world's books. Thus, the omission of software itself from prior-art libraries is especially curious.

THE MYTH OF ONES AND ZEROES

Why, then, is software an untapped prior-art resource? Perhaps having incomplete prior-art databases is a rational practice for the United States Patent and Trademark Office,[28] given that bringing actual software products into the patent system would further burden examiners, but this hardly explains the lack of better resources for patent defense litigators who are seeking invalidating prior art.

Software prior art likely remains an untapped resource because of a misconception about software, firmly held even by otherwise-knowledgeable non-programmers. Lawrence Lessig for example has acknowledged that his remarkable book *Code and Other Laws of Cyberspace* labored under the misimpression that object code is necessarily non-informing, with source code uniquely disclosing information regarding the protocols, interfaces, and rules that code employs and enforces.[29] According to this conception, a software product as shipped from a vendor or found on a customer's computer consists merely of binary code such as 010101 (looking presumably much as code does in the movies). Usable information about the product may be found in documentation or in "help" files or in the product's visible display,[30] and would be contained in the vendor's proprietary "source code," but is assumed to be absent from the "object code" that comprises the product open to the public.

SILVACO V. INTEL

The recent California case of *Silvaco v. Intel* provides a good example of this assumption.[31] Silvaco develops software used in circuit design. In an earlier suit, Silvaco obtained a judgment against a competitor (CSI), which was found to have, in conjunction with ex-Silvaco employees, misappropriated Silvaco trade secrets in developing its competing product. Intel was a licensee of that competing product. Silvaco sued Intel, maintaining that Intel's mere use of the CSI product—known (because of industry publicity for the *Silvaco v. CSI* judgment) to contain Silvaco's trade secrets misappropriated by CSI—was sufficient to constitute trade-secret misappropriation by Intel. Intel moved for summary judgment, contending that it never possessed Silvaco's trade secrets, because the CSI product was distributed only in object-code form, not source code. Silvaco did not allege that Intel knew the trade-secrets themselves, merely that it was using CSI's product, which Intel knew contained Silvaco's trade secrets.

The trial court granted Intel's summary judgment motion, and the appeals court affirmed, with the following striking analogy: "[o]ne who bakes a pie from a recipe certainly engages in the 'use' of the latter, but one who eats the pie does not, by virtue of that act alone, make 'use' of the recipe in any ordinary sense, and this is true even if the baker is accused of stealing the recipe from a competitor, and the diner knows of that accusation." Intel had merely eaten the pie baked by CSI.

While this is surely the correct result, the factual basis is misplaced: an object-code product may well incorporate trade-secret functionality originally found in the source code. Software recipes from source code are "baked into" software products built from that source code. In fact, if the product is placed on the market, and is both technically and legally open to reverse engineering, it is questionable whether information contained in the product remains a trade secret, even absent actual reverse engineering by any third party. Silvaco's president stated in deposition, apparently to show that it took reasonable security precautions to protect its trade secret, that the trade secrets inside binary-code products "are

protected by the binary code. Natural protection. You cannot do anything with binary code."[32] This statement supports the court's conclusion, but it is remarkably incorrect. Hidden functionality originally specified in source code may also be discernible in the object code. This should not be surprising, given that the object code shipped to customers is a *translation* of the source code held by manufacturers.

This "you cannot do anything with binary code" misconception persists even in the face of knowledge that software is sufficiently open to reverse engineering that software vendors regularly desire legal protection from this common industry practice (which their developers meanwhile perhaps carry out on competitors' products). Given what is generally regarded as the unenforceability of such restrictions in the context of mass-market shrinkwrap licenses—indeed, these restrictions often carry the explicit caveat that the restriction is "except to the extent expressly permitted by applicable law"—it appears that software has been avoided as prior art based on the assumption that object or binary code is "naturally" protected from inspection.

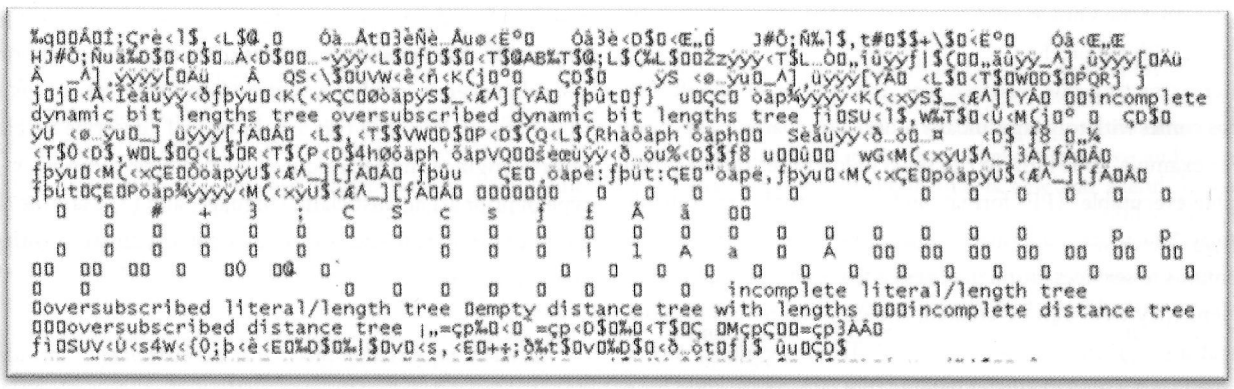

Figure 1: A portion of the Windows XP object code, viewed inside a word processor

TREATING SOFTWARE PRODUCTS AS TEXTS

A computer program typically contains different components or modules. Each of these modules typically corresponds to a file on a computer's disk, just as memos one has written reside on a disk. The code file has a name, just as word-processing documents such as memos and briefs have filenames. For example, much of the code that produces graphics in Microsoft Windows is found in files called "gdi32.dll" and "gdiplus.dll" (GDI standards for "graphics device interface"; DLL stands for "dynamic link library"), and much of the user-interface code is in a file called "user32.dll."

Don't actually do this, but if one were to take a word processor and, instead of opening a memo or brief, instead opened one of the files such as user32.dll that comprises the computer's operating system, or even one of the files (such as winword.exe or mso.dll) that comprise the word processor itself, one would mostly see "garbage" characters. While the word processor would do its best to display the contents of the code file as a word-processing document, not surprisingly the result would be mostly unreadable.

Mostly, but not entirely. Among the garbage, one would almost certainly also see some recognizable text. Early on in the Windows user32.dll file, there are snippets of text such as "CreateDesktop," "CreatePopupMenu," "DestroyAcceleratorTable," "GetClipboardViewer," and "RegisterUserApiHook." Though jammed together without spaces, note the presence of verbs (steps such as create, destroy, get, and register) and nouns (elements such as desktop, popup menu, accelerator table, and "user API hook").

In the gdiplus.dll file, one would see error messages, such as: "oversubscribed dynamic bit lengths tree," "incomplete literal/length tree," and "empty distance tree with lengths." See Figure 1. There are over 1,500 such text fragments in this one file; Windows contains thousands of files like this.

While hardly earth-shaking in themselves (this particular text inside Windows happens to correspond to well-known compression software known as "zlib"), these snippets at any rate demonstrate that software products contain readily-searchable text. To emphasize the point, this text came from inside Windows itself, rather than from a document describing Windows. Similar examples could be given of Mac OS X, Adobe Photoshop, an iPhone application, or the ROM or firmware embedded inside a device such as a camera.[33] This text can be extracted from the code with the simplest of reverse engineering tools, a program known as "strings," which finds and displays consecutive sequences (strings) of human-readable text buried in files otherwise not immediately readable.[34] (A software reverse-engineering "tool" is simply a computer program used to examine or manipulate other programs.)

Strings are present in code in part as text the program can display, such as error messages, menu selections, "help" descriptions, and dialog-box items. Other strings are not intended to be seen,

but are included in the code file as necessary for operation (such as names of operating-services such as "CreatePopupMenu" used by the program), for field testing, as internal error messages,[35] or as left-over artifacts of the software development process. Even a vendor that tightly guards its source code as its "crown jewels" will often send out its products with debugging information, right inside the readily-observable object code,[36] that contains significant source-code fragments,[37] including the names of functions implemented and used by the product. And, to the extent the code is found to use documented interfaces, the elements and steps recited in the documentation can be associated with the code file, and in turn correlated with the elements or steps or patent claims.

Remarkably, the code in software products often contains the names of source-code files from which the product was built. In addition to using these names to focus discovery requests for the source code, and gleaning whatever knowledge may be gained from the filename itself,[38] surprisingly often (particularly with code for hardware devices),[39] these filenames correspond to open-source code,[40] so that the known public contents of the source file can be associated with the product.

After extracting raw readable text, the next level of software analysis comes with tools specifically tailored to a particular type of file. For example, code built to run on Windows uses the so-called "portable executable" (PE) format, and various tools are available to display "metadata" contained in such files, such as the links the file contains to services found in other code files. Given such links, a "dependency tree" can be created, showing the names of modules that use a given module of interest.[41] This dependency tree is remarkably similar to the list of patents that cite, or are cited by, a given patent,[42] or (as in "Shepardizing") to the cases that cite or are cited by a given case.

The names noted earlier, such as "CreateDesktop" and "CreatePopupMenu," are among hundreds of services (also known as APIs, or application programming interfaces) provided by the user32.dll module of Windows. A modern operating system provides (in fact, is largely a collection of) thousands of such APIs. Programs are written by invoking these API services; the services themselves have been written by invoking other services. Metadata-dumping utilities display the names of services "exported" by a module, as well as the names of services it "imports" from other modules.[43] As seen earlier, such names often have a verb/noun form referring to elements of a software device, or steps of a software method.

Text can also be "imputed" to a given code file, based on numerical data contained in the file. The example was given earlier of learning which compression method a product is using, based on some numbers found in the code. Certain numbers or sequences of numbers are a nearly-certain indication that a particular algorithm is employed. For example, presence in code of the base-16 numbers 243FF6A88 and 85A308D3 would, in the absence of any other information, point to use of Blowfish. While perhaps evoking dangerous sushi, to a software engineer, "Blowfish" would bring to mind an encryption algorithm created by Bruce Schneier. Similarly, some long sequences known as GUIDs or UUIDs (Globally or Universally Unique Identifiers) are often associated with a specific protocol or service. With a database associating UUIDs with names, the protocols or services used by a software product can be derived from UUIDs. Utilities are available to search out such "magic" numbers or "signatures" found in code.[44] Naturally, the longer the signature, the less likelihood of "false positives."

Having inferred the use of a well-known algorithm to a code file, textual descriptions of the algorithm's operation can be imputed to the file; this is especially helpful when the algorithm in turn uses other steps or elements whose names are likely to appear in patent claims.

CONCLUSION

Code is not only ubiquitous, used to control a remarkable amount of modern activity; code is also text which is readable and searchable at various levels. This text includes the object or binary code deployed in public. A vast body of prior art is thus waiting to be tapped. Part 2 of this article will: (1) describe several more reverse-engineering techniques; (2) describe comparison of this type of prior art against claim language; and (3) examine the legal issues of prior art found (or some might say, created) with reverse engineering. ◀◀

The views expressed in this article are personal to the author and do not necessarily reflect the views of the author's firm, the State Bar of California, or any colleagues, organization, or client.

Andrew Schulman is an attorney and computer programmer based in San Francisco. He is the author and editor of several books on the internal operation of Microsoft operating systems. Since 1994, his company Software Litigation Consulting has provided technical expertise in litigation involving computer software, including patent, copyright, trade secret, antitrust, and privacy cases. He is a candidate in the Intellectual Property LLM Program at Golden Gate University. His website www.softwarelitigationconsulting.com; his email is undoc@sonic.net. *The author would like to acknowledge the assistance of Keith Hutchinson (Pfizer) and Kenneth Wilson (Carr & Ferrell).*

Endnotes

1. *Silvaco v. Intel*, 184 Cal. App. 4th 210 (2010).
2. 17 U.S.C. § 101.
3. Dan Burk, *The Role of Patent Law in Knowledge Codification*, 23 Berkeley Tech. L.J. 1009 (2008).

4. *Kewanee Oil v. Bicron*, 416 US 470 (1974).
5. *See, e.g.*, Henry Heines, *Determining Infringement by X-Ray Diffraction*, Chemical Engineering Progress (Jan. 1999); Julia Elvidge, *Using Reverse Engineering to Discover Patent Infringement*, Chipworks (available at http://www.photonics.com/Article.aspx?AID=44063) (Sept. 2010); Andrew Schulman, *Hiding in Plain Sight: Using Reverse Engineering to Uncover Patent Infringement*, Intellectual Property Today (November 2010).
6. *See Judin v. U.S. and Hewlett-Packard*, 110 F.3d 780 (Fed. Cir. 1997) (attorney sanctions for reliance upon inventor of micro-optical imaging patent, without attempt to inspect Postal Service bar-code scanners); *Antonious v. Spalding and Evenflo*, 275 F.3d 1066 (Fed. Cir. 2002) (failure to cut open and examine all relevant golf clubs); *Bender v. Maxim Integrated Products*, 2010 WL 2991257 (N.D. Cal., 2010) (reverse engineering not an absolute requirement, but here RE may be only method yielding sufficiently specific infringement contentions); but see *Intamin Ltd. v. Magnetar Technologies*, 483 F.3d 1328 (Fed. Cir. 2007) (no sanctions for failure to obtain and cut open metal casing in roller-coaster braking system; distinguished from ease of obtaining sample in *Judin*).
7. Julie E. Cohen, *Reverse Engineering and the Rise of Electronic Vigilantism: Intellectual Property Implications of "Lock-Out" Programs*, 68 S. Cal. L. Rev. 1091, 1179 (1995).
8. Mark A. Lemley and Julie E. Cohen, *Patent Scope and Innovation in the Software Industry*, 89 Cal. L. Rev. 1, 13 (2001).
9. The "Jargon File" is a massive codification of programmer "folklore." *See* http://www.netsoc.tcd.ie/~inky/jargon/; it appears in book form as Eric S. Raymond, The New Hacker's Dictionary (3d ed. 1996); U.S. patent citations to the jargon file can be found with uspto.gov search OREF/"jargon file."
10. U.S. Patent No. 6,961,945 (Microsoft DirectUI widget API); U.S. Patent No. 5,831,606 (namespace extensions).
11. "Supplemental Expert Report of Andrew Schulman," *Comes v. Microsoft*, Iowa, Dec. 19, 2006 (available at http://www.sonic.net/~undoc/comes_v_microsoft/Supp_Rpt_Andrew_Schulman.pdf).
12. *See, e.g.*, Christopher Lanks, *In Re Seagate: Effects and Future Development of Willful Patent Infringement*, 111 W.Va. L. Rev. 607 (2008-9).
13. Peer to Patent (http://PeerToPatent.org) and the curiously-named "Patent Busting Project" of the Electronic Frontier Foundation (http://w2.eff.org/patent/).
14. http://www.google.com/codesearch ("search public source code," but not object code).
15. *See, e.g.*, Andrew Schulman, David Maxey, and Matt Pietrek, Undocumented Windows: A Programmer's Guide to Reserved Microsoft Windows API Functions (1992); Matt Pietrek, Windows Internals: The Implementation of the Windows Operating Environment (1993); Geoff Chappell, DOS Internals (1994) and over 2,000 pages of documentation at http://www.geoffchappell.com; Schulman, et al., Undocumented DOS: A Programmer's Guide to Reserved MS-DOS Functions and Data Structures (2d ed. 1993). vendors themselves sometimes provide such information, apart from their formal documentation, e.g., Mark Russinovich and David Solomon, Windows Internals (5th ed. 2009); Kalen Delaney, Inside Microsoft SQL Server 2005: The Storage Engine (2006).
16. U.S. patent citations to such material can be found, *e.g.*, with uspto.gov search OREF/"undocumented dos" or OREF/"windows internals."
17. *See, e.g.*, Martin Campbell-Kelly and Patrick Valduriez, *A Technical Critique of Fifty Software Patents*, SSRN 650921 (2005); John R. Allison and Ronald J. Mann, *The Disputed Quality of Software Patents*, 85 Wash. U. L. Rev. 297 (2001), and the saga of the Amazon one-click patent, particularly the changing responses of software industry leader Tim O'Reilly (available at http://www.oreillynet.com/policy/2001/03/14/bounty.html).
18. Whether a trade secret can constitute prior art is surprisingly controversial. *See, e.g.*, George Gates, *Trade Secret Software: Is It Prior Art?*, 6 Computer Lawyer 11 (Aug. 1989).
19. *See, e.g.*, uspto.gov search for OREF/("dr. dobb's journal" or mactech or "windows dos developers journal" or "microsoft systems journal").
20. *Jockmus v. Leviton*, 28 F.2d 812 (2d Cir. 1928) (printed catalog in French; L. Hand: "[W]hile it is true that the phrase, 'printed publication', presupposes enough currency to make the work part of the possessions of the art, it demands no more. A single copy in a library, though more permanent, is far less fitted to inform the craft than a catalogue freely circulated, however ephemeral its existence; for the catalogue goes direct to those whose interests make them likely to observe and remember whatever it may contain that is new and useful."); see, e.g., collection of 100 million datasheets from 7,500 manufacturers available at http://datasheetarchive.com/; document archive at http://bitsavers.org (with code at http://bitsavers.org/bits/); textfile archive available at http://textfiles.com/directory.html (with code from "shovelware" CDs at http://cd.textfiles.com/; see also uspto.gov search oref/"textfiles.com"); software descriptive material (e.g. manuals) at Software Patent Institute, http://spi.org/ (*see* uspto.gov search oref/"spi.org").
21. IBM Technical Disclosure Bulletin (TDB) and Siemens Disclosure Journal available at http://priorartdatabase.com.
22. *In re Klopfenstein*, 380 F.3d 1345 (Fed. Cir. 2004) (slide presentation posted for three days at conference is printed publication); "Electronic Documents as Prior Art / Status as 'Printed Publication,'" Manual of Patent Examining Procedure (MPEP) § 2128 (updated 2010); uspto.gov search for OREF/wikipedia (with 3,470 hits in granted patents); E. Robert Yoches and Terry Callaghan, *The Next Battle: New Forms of Software Prior Art*, 2 U. Balt. Intell. Prop. L.J. 115 (1994); and Neil Pierotti, *Does Internet Information Count as a Printed Publication?*, 42 IDEA 249 (2002).
23. *In re Epstein*, 32 F.3d 1559 (Fed. Cir. 1994) (earlier software product, described in later textual abstract, is prior art under 35 U.S.C. § 102 (b) for earlier date at which software itself was public); MPEP § 2133.03 (b) ("nonprior art publications can be used as evidence of sale before the critical date"); *Eolas v. Microsoft*, 399 F.3d 1325 (Fed. Cir. 2005) (Viola browser made by Perry Wei, shown to Sun engineers, but abandoned, suppressed, or concealed).
24. The United States' "in this country" clauses in 35 U.S.C. § 102 appear to uniquely disregard foreign prior use/knowledge; however, the ability to patent known foreign unpublished inventions (in other words, so-called "traditional knowledge")

is limited by 35 U.S.C. § 102(f). See Margo A. Bagley, *Patently Unconstitutional: The Geographical Limitation on Prior Art in a Small World*, 87 MINN. L. REV. 679 (2003) and Craig Allen Nard, *In Defense of Geographic Disparity*, 88 MINN. L. REV. 221 (2003).

25. Not surprisingly, the traditional knowledge (TK) databases considered by the USPTO (available at http://www.uspto.gov/web/offices/dcom/olia/dictionaries.html) appear to be based on previously-published codifications of TK.
26. PETER DRAHOS, THE GLOBAL GOVERNANCE OF KNOWLEDGE: PATENT OFFICES AND THEIR CLIENTS 65 (2010).
27. Mario Biagioli, *Patent Republic: Representing Inventions, Constructing Rights and Authors* (2007) in The *Law of Patents* 202 (Craig Nard ed., 2008).
28. Mark Lemley, *Rational Ignorance at the Patent Office*, 95 NW. U.L. REV. 1495 (2001).
29. Open Code = Closed Code?, http://www.lessig.org/blog/2002/08/open_code_closed_code.html (August 22, 2002) (it "was a bit of a shock" to be told by a respected coder that binaries "reveal all"; open source distinguished for ability to modify code, not merely inspect it). The second edition of the book *Code v2* (2002), is available at http://codev2.cc/.
30. *See, e.g.,* uspto.gov search OREF/(screenshot OR "screen shot").
31. *Silvaco v. Intel*, 184 Cal. App. 4th 210 (2010).
32. Cited in Respondent's Brief, *Silvaco v. Intel*, Nov. 3, 2008, at 14 (available at http://pdfserver.amlaw.com/ca/silvaco0406_02.pdf).
33. While reading strings in firmware is trivial, extracting the firmware from a device in the first place is not trivial (*see, e.g.,* ANDREW HUANG, HACKING THE XBOX (2003); DERENGEL, HACKING THE CABLE MODEM (2006)); however, in many cases vendors place firmware updates on the internet, and these firmware updates can be inspected without touching the device.
34. There are numerous versions of the strings utility. *See, e.g.,* Microsoft's version at http://technet.microsoft.com/en-us/sysinternals/bb897439.
35. Which sometimes incorporate source-code fragments, such as "Assertion cmd_ptr > cmd.rmnet.sn.cmd_type == RMNET_SM_CMD_PTR_CHANGED failed" as one of many such strings found in the firmware for a wireless modem.
36. Matt Pietrek, *Remove Fatty Deposits from Your Applications Using Our 32-bit Liposuction Tools*, MICROSOFT SYSTEMS JOURNAL (Oct. 1996) (available at http://www.microsoft.com/msj/archive/s572.aspx). Debugging information is sometimes also provided in separate downloadable files (see, e.g., Microsoft PDB symbol files (available at http://msdn.microsoft.com/en-us/windows/hardware/gg463028)).
37. In some cases, object code files contain *complete* pieces of source code, in the form of embedded scripts or macros (such as embedded SQL database code, JavaScript, PostScript, XML, or HTML), and are often accompanied by plain-text files containing such scripts.
38. *E.g.,* "ds707_sec_pkt_mgr_handoff.c", "ps_stat_phys_link.c", or "ps_iface_rx_qos_fltr.c" found in Novatel firmware.
39. A famous example is the Linksys WRT54G router, whose firmware in 2003 was found to be based on Linux components; following this discovery, Linksys (now Cisco) released its code under the GNU General Public License; see http://homesupport.cisco.com/en-us/gplcodecenter.
40. *E.g.,* the filenames "pistachio/kernel/src/api/v4/schedule.cc" and "iguana/server/src/memsection.c" found inside a piece of firmware corresponds to the L4Ka microkernel project (http://www.l4ka.org/).
41. *See, e.g.,* Steve Miller's Dependency Walker program at http://dependencywalker.com/ (also widely distributed by Microsoft).
42. In addition to commercial services such as Ocean Tomo's PatentRatings and IPQ, which rank patents (with varying degrees of usefulness) based in part on citations to a given patent from other patents, there is extensive academic literature on both the benefits and limitations of using patent citations; see ADAM JAFFE, ET AL., PATENTS, CITATIONS, AND INNOVATIONS: A WINDOW ON THE KNOWLEDGE ECONOMY (2005) and associated website, http://www.nber.org/patents/.
43. *See, e.g.,* Clive Turvey's DumpPE at http://www.tbcnet.com/~clive/vcomwinp.html.
44. *See, e.g.,* Luigi Auriemma's SignSrch at http://aluigi.org/mytoolz.htm; the YARA signature-matching language at http://code.google.com/p/yara-project/; the author's GUIDFind (forthcoming); antivirus and anti-malware signatures in products such as ClamAV (http://www.clamav.net); and signatures for detecting code compression (http://peid.info); PETER SZOR, ART OF COMPUTER VIRUS RESEARCH AND DEFENSE, ch. 11 (2005). There is a rough similarity to searching DNA, RNA, and protein sequences at, *e.g.,* Patent Lens (http://www.patentlens.net/sequence/blast/blast.html), chemical syntheses and reactions in CASREACT (http://www.cas.org), or chemical structures (genus, subgenus, species) at *e.g.,* ChemFinder (http://chemfinder.camsoft.com).