Open to Inspection: Using Reverse Engineering to Uncover Software Prior Art, Part 2

ANDREW SCHULMAN
Software Litigation Consulting

COMPUTER SOFTWARE IS A TECHNOLOGY READILY turned into searchable text, yet ironically viewed as having a special prior-art problem. While descriptions of software are increasingly available in patents, patent applications, and the non-patent literature (NPL), software itself is regarded in the patent context as lost prior art, prior art that is inaccessible to searching.[1]

As the first part of this article (*New Matter,* Summer 2011) noted, this issue has less to do with "source code"—which when tightly held as a proprietary trade secret is not prior art[2]—than with the "object code" comprising most software products. Part 1 of this article showed that software products contain text which can be extracted with a simple reverse-engineering (RE) technique called "strings." This text can, in turn, be placed into a searchable prior-art database. The product may be mass-market software such as Microsoft Windows or Office, Adobe Photoshop, an iPhone app, an interactive Flash movie on a website, or the "firmware" located inside a camera.

This second part of the article will first describe several additional RE techniques to extract or generate text from software products, and will then compare this type of prior art with patent claim language. The focus of the article then turns to several legal issues raised by the use as prior art of reverse-engineered software products:

- How reliable, particularly as to date, is evidence of prior art which is later uncovered through RE?
- If a feature of an earlier public product is only uncovered later, via RE, can the feature reasonably be characterized for 36 U.S.C. § 102(a) or § 102(b) purposes as prior use or knowledge, on sale, or a printed publication? Must one ordinarily skilled in the art have been able to (readily?) uncover the feature at the time?
- If a feature can as a technical matter be uncovered with RE, but the product was accompanied by no-RE licensing language, is such language by itself sufficient to make the product's internals confidential, and thus not prior art?
- Is a database of strings extracted or generated from software products an infringing derivative work, or are the fragments so used subject under the merger doctrine to at most "thin" copyright protection? At the very least, is a database of strings from code a fair use, akin to Google's indexing or caching of text from websites?

GETTING INFORMATION FROM SOFTWARE PRODUCTS

The techniques noted in Part 1 of this article extract some text from software products or infer text from numbers readily found inside the product. The extracted text includes error messages, including internal error messages known as "assertions" (often containing source-code fragments); debugging information left in the product; names of APIs (application programming interfaces)[3] and protocols used by the product; pathnames of source-code files used to build the product; and embedded scripts. "Generated" text (text imputed to the product, based on other text or on numeric data in the product) includes names of services, protocols, and algorithms.

These techniques rely on extremely-simple algorithms such as "strings,"[4] and might not even be considered as RE by many programmers, nor fit within either the standard legal definition of RE[5] or an "end-user license agreement" no-RE clause. Even some fervent opponents of software RE have restricted their opposition to the more sophisticated techniques of "disassembly" or "decompilation" of code.[6]

Decompilation attempts reconstruction of the original source code, and is feasible for virtual-machine languages such as Java, Microsoft .NET code, and Flash ActionScript (found in the SWF

files ubiquitous on the web). Even then, decompilation can no more return to the original source code than one could retrieve a cow from a collection of hamburgers. When source code is "compiled" to form a product delivered to customers, much of the source code—particularly comments made by programmers in the source code—is unused.

With "native" machine code, to run directly for example on an Intel processor, disassembly rather than decompilation is employed. Disassembly does not attempt to recover the original source code, but instead displays machine code in a more readable form, and labels areas of the code based on cross-references found within the code.[7] While neither disassembly nor even decompilation can retrieve the original source code from a software product,[8] these techniques nonetheless can yield information relevant to a prior-art search.

In a patent infringement case, an expert may be asked to disassemble or decompile code, particularly in advance of the opponent's production of source code in discovery. Disassembly and decompilation typically yield too much material. In one case a medium-size Flash SWF file used on a website was found to be built from almost 400 ActionScript source-code files, totaling over 50,000 lines of code. The expert's analysis can be partially automated by reducing the file into a set of searchable "signatures," corresponding to each subroutine in code.[9] Such signatures can also be based on unusual code sequences[10] or on higher-level machine instructions whose names may be indicative of the elements or steps of an invention.

DYNAMIC REVERSE ENGINEERING

In the techniques described above and in Part 1, a software product is treated as a text to be examined without running the product, often without even installing it. One is *reading* a text, perhaps with some *translation* into a more readily-understandable language. These various ways of reading another's software product such as strings, metadata extraction, and disassembly are collectively referred to as *static* reverse engineering.

Another set of techniques, *dynamic* reverse engineering, involves running the software program of interest, under the control of a second program, which tests or tracks the first product's behavior. Dynamic RE tools include debuggers, network monitors (packet sniffers), file monitors, and API trace utilities.[11] Dynamic RE appears particularly appropriate for method/process claims, but also has application to apparatus claims.

Many public websites—whose front-end code is visible in the browser's "View Source" window, explored with tools available for web developers,[12] or (in the case of Flash SWF, .NET, or Java programs) decompilable—have large server backends (*e.g.,* ASPs), even whose object code is inaccessible, at least until discovery. The traffic between the client and the server can be monitored with a tool such as Fiddler or Wireshark, and various "black box" techniques can help draw inferences about server code. Note, however, if the code is outside public view, it is likely not prior art in the first place.[13] Whereas object code is a "non-informing public use" (see below), server code is closer to a secret use.

COPYRIGHT IMPLICATIONS OF DYNAMIC RE

From a copyright perspective, dynamic RE is sometimes viewed more favorably than static RE, because dynamic RE is merely observing what a product does, rather than looking inside (and in the course of looking, possibly temporarily copying bits of) the product to see how it does it. But because dynamic RE requires installing and running a software product, it is far more likely than static RE to require the motions of assent to a "clickwrap" license.

Dynamic RE carried out in a later period also presents the problem of reliably reconstructing the software environment, including network components, as it existed during the prior-art period. On the whole, static RE is better suited to uncovering prior art than is dynamic RE, which however is well suited to investigating claims of patent infringement.[14]

COMPARING STRINGS IN CODE WITH LIMITATIONS IN CLAIMS

Once searchable text is extracted or generated from software products and placed into a prior-art database, the elements or steps of patent claims can then be checked against the text, much as would text from any document. The software prior-art database will likely contain few complete sentences, and many small text fragments. Patent searching generally involves combinations of words or short phrases, corresponding to the elements or steps of a patent claim, together with likely synonyms. Such initial searches to produce "leads" for closer inspection are not a radical departure from what patent searchers already do.[15]

As an example, take a patent claim with the following limitations: media channel, predefined episodes, subscribing, downloading updates, checking available space, and deleting earlier episodes; the patent claims priority to applications filed in Nov. 2003.[16]

A search was conducted across a limited set of Microsoft products for combinations of text including the following terms: episode, channel, subscribe, update, and download. Results were found in a Windows 98 program named tvx.exe and in two modules it references, named commmstv.dll and msepg.ocx. These 1998 files refer to "Microsoft TV Services" and "WebTV for Windows". The tvx.exe file contains embedded SQL source code for manipulating a SQL database of television data. Further searching among Microsoft products, including a 1998 Microsoft "Platform SDK" (software development kit), yielded publically-available source-code files related to a TV project from the early 1990s named "Microsoft Guide", an EPG (Electronic Program Guide) with support for episodes, channels, subscriptions, and so on.

The patent in question appears not to refer to Microsoft explicitly, nor to cite patents assigned to Microsoft, so this may be a fruitful area for closer inspection. While hardly conclusive, this small example illustrates the basic approach of searching through product code to find potential prior art.

Even having found all the elements or steps together in a single prior-art reference, the searcher must determine if they fit together in an appropriate way. A patent claim is not a parts list or aggregation of components.[17] For example, a relational database with a built-in debugger does not anticipate a claim to a debugging system with a relational-database back-end.[18]

THE SOFTWARE LEXICON

How likely is it that software products contain patent-relevant strings, so that patent claims can be used to guide a search of text extracted from software products?

Claims may be broader than what an inventor has implemented, reflecting the creative role of patent attorneys in uncovering the actual invention in its broadest sense,[19] or narrower, so as to avoid prior art. Searching non-patent literature (NPL) often requires fleshing out claims with concrete examples, preferably proper nouns, from dependent claims or the patent specification. Method claims in particular tend to be more abstract, and more difficult to line up with NPL.

The language used by programmers often differs from that contained in software patents. Where a patent might refer to a copying a command or a packet, text extracted from a product may refer to cpy, cmd, or pkt. Programmers may use jocular terms, such as kill, crash, barf, and magic, whereas a patent might refer to deletion, failure, error message, and opaque numeric value. Code has even more misspellings (especially dyslexias) than patent documents.

At the same time, software patents include the same odd terminology as software itself. Terms resembling ordinary words but with a special software meaning, such as thread, spool, hash, salt, and thunk,[20] appear in software patents as well as in code, with (subject to claim construction) the same meaning. The titles of even entire classes of patents include terms programmers use, including VxD, Markov, and prefetch.

Patent specifications often contain code fragments useful to narrow searches for what might otherwise be mind-numbingly broad claim terminology. For example, while a step involving "an intercept software component technology" was difficult to align with what appears in code, the patent specification referred to "vnodeops," a code reference associated with intercepting file operations.[21]

Between the ability of a patentee to be "her own lexicographer,"[22] the preference for intrinsic over extrinsic (*e.g.*, dictionary) evidence of claim meaning,[23] and a tendency of computer programmers to use non-standard naming in their code, any software prior-art searching system requires an extensive lexicon, including synonyms, abbreviations, and class context. Frequently-occurring terms such as ROM, CPU, and DVD, are discounted.

Accordingly, the software lexicon presents difficult problems, but given the importance of software, these problems are worth tackling.

HOW MANY REFERENCES?

Use of software as prior art presents an issue distinct from printed documents: what constitutes a single reference in the software context? It is black letter law that "[a] claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference."[24] How does one determine the boundaries of a single reference in a software product?

A single software product such as Windows may include hundreds or even thousands of different modules, each a separate file on the disk. Is everything in Windows part of a single reference? This does not sound right,[25] at least when applied to parts of Windows not typically used together.[26] But neither should each module be regarded as a separate reference, such that, for anticipation, every limitation of a patent claim must be found within a single module. The better view is that a single software reference includes a given module, plus those the module relies upon.[27]

If it proves overly complicated to form a single prior-art reference from what are arguably different pieces of code—albeit comprising a single product, or used together to bring about a single result—the analysis can shift from 35 U.S.C. § 102 anticipation (where a single reference is required for lack of novelty) to § 103 obviousness (where multiple references may be combined to demonstrate the requisite motivation and thus obviousness). The POSITA (person of ordinary skill in the art) may have implicit motivation to combine references,[28] particularly ones already having a real-world interrelation.

DATING AND RELIABILITY OF SOFTWARE REFERENCES

Whenever code is used in litigation, it is crucial to accurately *date* the code. File date/timestamps found on manufactured CD-ROMs present few problems; likewise for file date/timestamps found on non-writeable floppy diskettes, or when an original-shrinkwrap copy of the software is producable. However, vendor-provided dates can be changed, and even some vendor-provided dates are incorrect.[29] Thus, a software prior-art database will collect as many dates as possible for each code file: creation date; date of last access and of last write; any © dates embedded in the file; compiler/linker dates built into the file; and dates on the physical media and packaging.

Reviews and advertisements in trade journals may relate back to a product date.[30] For more recent software, online resources can provide the date of a code file's first appearance "in the wild."[31]

When considered as a printed publication, software prior art is an electronic publication. Greater concern over the reliability of electronic publications appears in Europe than in the US, but even the EPO views dating rather than contents as the primary reliability concern.[32] Public accessibility of material posted on the internet is a separate concern.[33]

FINDING OLD CODE

Existing software-related prior art NPL (non-patent literature) libraries are based, not on the software products themselves, but on descriptions of software.[34] Further, this descriptive material is predominantly sourced from academic rather than trade publications, and from abstracts rather than full text.[35] To the extent that trade publications have been used, the source code that often accompanies articles in publications such as the *Microsoft Systems Journal, Dr. Dobb's Journal,* or *MacTech,*[36] is missing, as generally are the all-important advertisements. There are several useful "defensive publication" archives, including one specially focused on software prior art,[37] but again, these rely on printed publications several steps removed from the software itself.[38]

Given the incentive to smoke out prior art, where can litigants find older software products?

Several code preservation societies and computer museums have large collections of software products.[39] But with few notable exceptions, these generally are indexed at best by product name, not by filenames within the product, much less by the textual content of each file.[40] General libraries, whose catalogs have in turn been cataloged at WorldCat, surprisingly often have software titles, though presumably not for rental.[41] At one point, the Library of Congress had a Machine Readable Collections Reading Room, which later was closed.

Products which were sold to the public can often be found on eBay, in the backs of old computer books available on Amazon.com, and in stores such as Weird Stuff Warehouse in Sunnyvale, CA. The basements and closets of experts working on a case are natural sources of relevant prior art, with caveats regarding dating, reliability, and chain of custody.

In addition to its "Wayback Machine" (a source of dated object code if one already knows the filename of interest),[42] the Internet Archive has a "Software Archive" project, the impetus for the software-archiving exemption to the Digital Millennium Copyright Act anti-circumvention clause.[43] Unfortunately, while the archive itself apparently contains a large amount of older Apple, Atari, and Commodore products, for now the public collection is dominated by the TUCOWS and CD BBS archives, already available elsewhere.[44]

One issue faced by software archivists is uncertainty as to what this material is good for; in some cases, old code is preserved solely so that old data can still be accessed.[45] These archives have not been designed with patent prior-art searching in mind; indexing does not include sufficient detail beyond product name.[46]

The US government's National Software Reference Library (NSRL), part of the National Institute for Science and Technology (NIST),[47] is a collection of approximately 21,000 different software products from about 1,400 vendors,[48] whose contents have been processed to obtain digital signatures uniquely identifying the files in the software packages. NIST distributes these signatures[49] largely for use by law-enforcement investigators as indicators of what to *ignore* when inspecting computer systems seized in criminal cases.[50] These lists of filenames associated with products may help in some prior-art situations, but the NSRL's goal of filtering out commercial code is almost the exact opposite of what software prior-art searchers need.[51]

IF NOBODY SAW IT AT THE TIME, IS IT STILL "PRIOR" ART?

Vendor A puts feature X in its product P, sold to the public, but only some employees of A know about X. This occurs at time T1. Some time later, at T2, vendor B receives a patent on X, and B sues C for patent infringement. Now C, at time T3, hires an expert, who reverse engineers P and uncovers X. Is P prior art which invalidates B's patent? In other words, if a feature of an earlier public product is only uncovered later, via RE, can the feature reasonably be characterized as prior art which may invalidate a later patent?

It is black-letter law that prior art is *public*; a drawing made on the underside of a tablecloth in someone's kitchen, unknown at the time to the tractor-pulling art, is not prior art.[52] As befits black-letter law, there are important exceptions of "secret prior art."[53]

Yet it happens all the time: at T3, C uncovers a remarkably obscure document dating from time T1, and this is prior art, so long as it's not an internal memo of B (or of A, for that matter).[54]

Between a secret on the one hand, and ready availability on the other, there are only degrees of convenience or obscurity. Prior art status depends, not on whether the public actually accessed a reference at the time,[55] but whether the relevant public *could* have. The prior art need not have been inspected, so long as it was open to inspection.

Thus, the question becomes, if a feature is only uncovered later using RE, could it have been uncovered earlier? Must one ordinarily skilled in the art have been able to (readily?) uncover the feature at the time?

Note that this question need not be asked of the patentee's own pre-patent code, because even secret use *by the inventor* is subject to the 102(b) on-sale bar.[56]

Further, the question becomes irrelevant, at least under a § 102(g) concealment analysis, if a third party's earlier inventive activity "benefitted" the public. For example, a golf ball whose covering is made of a secret material is prior art, even if the secret to the golf-ball covering only becomes public afterwards, because the golf ball itself (and certainly its exterior) is public.[57] In contrast, a

hidden process used by a third party to produce a public product—such as a machine to make quilts, where the quilts are sold, but the machine is not, where the inventive workings of the machine cannot be discerned from the quilts—is not prior art.[58]

There is an important distinction between a hidden inventive process to produce a public product on the one hand (not prior art), and a public product with an invention hidden inside it on the other (prior art). In the quilting case, the public can enjoy the fruits or output of the invention, but has no ability to access the invention itself, because the invention and its output are separable, and the invention can't be discerned from the output. In the golf ball case, the invention itself is what's in public, and the public merely failed to observe at the time.

Where do software products fit here? Some are like the secret quilting machine used to produce public quilts. In particular, in client/server products, code runs on the server, with only its output delivered to clients. Even object code on the server is inaccessible, except to the extent that clients can understand the server code via dynamic RE (see above). Software held back from the public, whether in source or object form, is generally not prior art.[59]

Much object-code software, however, is disseminated to the public, and what the public could learn through RE is prior art, just as the "secret ingredient" of a golf ball is prior art, if the golf ball itself is in sufficiently plain view. As noted in the golf-ball case, "even though there may be no explicit disclosure of the inventive concept, when the article itself is freely accessible to the public at large, it is fair to presume that its secret will be uncovered by potential competitors long before the time when a patent would have expired".[60]

This area comprising the undocumented portions of public products is known, in the 102(g) context, as "non-informing public use."[61] The concept dates back to the 19th century cases when, for instance, an arrangement of bolts in a safe was held to be prior art, even though "[t]hey were, it is true, hidden from view after the safes were completed, and it required a destruction of the safe to bring them into view. But this was no concealment of them or use of them in secret. They had no more concealment than was inseparable from any legitimate use of them."[62]

But is it really correct that a well-hidden component of a public product, perhaps requiring destructive testing of the product, is not a secret but merely a non-informing public use, at least until the secret is actually found out? Having noted that secrets are generally not prior art, aren't the internals of even very public software products, such as Microsoft Windows, regarded as trade secrets? If vendor C learns of the secret at time T3, wasn't it still a trade secret at T1? Isn't it still a secret until someone actually learns it, not merely when she could have conceivably done so?

A secret, after all, is never more than a matter of delay or timing.[63] That X was uncovered, even easily, at time T3, does not make it non-secret at T1. An important case involving software prior art, *In re Epstein*,[64] teaches that the date for software prior art can relate back from a later abstract. While the abstracts in *Epstein* may relate to more superficial aspects of a product than what one learns from even the simplest RE, perhaps similar relation-back applies to some inherent[65] feature of the code, merely waiting to be examined.

This question—whether learning a secret at T3 relates back to a finding of non-secrecy at T1—implicates much larger questions: trade secret/patent interaction, the prior-user defense,[66] and the basic disclosure vs. invention policy conflict that the patent system should reward one who discloses an invention, over one who keeps it secret, but patents should be granted for inventions, not that which has already been done.[67]

Trade-secret law provides a simple answer. Most enactments of the Uniform Trade Secrets Act (UTSA) include as part of the definition of a trade secret that the information's value derives from "not being readily ascertainable by proper means"[68] (and RE is generally seen as a proper means).[69] Information's status as a trade secret depends, not on whether it has been actually ascertained, but on whether it was readily ascertainable, by proper means.

So, the question becomes whether information ascertained at T3 could "readily" have been ascertained earlier, at T1. Could a POSITA have been able, before the invention or critical date, to readily uncover the information, without (to analogize from a somewhat different area) undue experimentation and fresh inventive activity? That depends:

- Strings found verbatim in object code at T3 were inherently present at T1, plainly visible if anyone had decided to view the object code in a word processor.[70] Readily ascertainable.
- Given numbers or strings inherent in object code on the one hand, and corresponding text from external documentation on the other, the text is background to (rather than a separate reference from) the code, if the documentation existed at the time, and the POSITA would have known to look there. For example, a software product's use of an API named "QueryPerformanceCounter," together with contemporaneous documentation for QueryPerformanceCounter, likely forms a single reference, at least to the extent of the product's use of the API. Likewise for numbers and corresponding documentation, such as the example given in part 1 of the numbers 243FF6A88 and 85A308D3 indicating use of the well-documented Blowfish algorithm (though given the possibility of false positives, there is room for debate on whether "Blowfish," and all that implies, is inherent to code which contains these numbers but not the name).[71]
- Information available at T3 through disassembly or decompilation, if the disassembler or decompiler were available at T1, is readily ascertainable at T1, at least so long as

understanding the listings is within the POSITA's skills, which it may not always be.[72]

- Information from dynamic RE, such as a network monitor, is unlikely to become an issue in the first place because of the difficulty of reliably reconstructing, at time T3, the T1 computing environment. But if an expert at T3 can show that he has reproduced the T1 environment, then almost by definition, the POSITA at T1 would have been able to do so.

At some point, it will be sufficiently difficult to glean information through RE at T3, indicating it was undetectable at T1.[73] If the requisite RE could not have been achieved at the time (including for legal reasons; see next section), the technology is effectively the same as a secret machine locked away from public view. Information only gleaned through heroic RE efforts, for example where the time and cost to RE exceeds a trade secret owner's reasonable expense to prevent RE,[74] is not "readily ascertainable." Some RE is sufficiently creative, uncovering features of a product unknown even to the vendor,[75] that it might be viewed as making a fresh inventive discovery rather than merely finding prior art. Even here, however, there is no requirement that prior art be known to its owner.[76]

Further, to be prior art, a reference must have been available or accessible, not only physically, but logically. In the college theses cases, it was insufficient that a thesis reside on a library shelf; the reference must be cataloged or indexed so that the POSITA at the time could have found it—not merely have found it if he already knew with hindsight exactly where to look, but actually found it if he didn't already know its location.[77] At the same time, only the title need have been cataloged, not the full text contents. The analogy to software is that a competitor, knowing of a public product, would have been able to examine the object code.

Apart from accessibility, operability is another prior-art requirement. If prior-art software is being characterized as a printed publication—which is quite reasonable given the prior art status of electronic publications[78]—then it must at the time have enabled the POSITA to practice the invention,[79] which a working product in the market would presumably do.

To summarize, something normally hidden may nonetheless be public. Of course; every student in an IP survey course knows this from the 19th century corset case, in which even part of an undergarment was in public use because it was outside the inventor's control.[80]

IS "SHRINKWRAPPED" CODE REALLY OUT IN PUBLIC?

Software products generally come with "shrinkwrap license agreement" or end-user license agreement (EULA) language restricting reverse engineering. Because software is a form of text, restricting RE in effect restricts reading the text, at least reading it too closely. Indeed, some vendors recognize this, and not only attempt to restrict the usual triumvirate of "reverse engineering, disassembly, and decompilation," but assert the user will not "analyze" the product.[81]

How do these no-RE clauses relate to the previous section's conclusion that technology uncovered later through RE can constitute prior art? The immediate concern is not whether such language prohibits reverse engineering for the purpose of uncovering prior art (see next section). Rather, the concern goes back to cases such as *Gillman v. Stern*, where Judge Learned Hand noted how the earlier machine "was always kept as strictly secret as was possible, consistently with its exploitation. In general, everybody was carefully kept out of Haas' shop where the four machines were used."[82] Similarly in a well-known case on oil prospecting as prior art, the court emphasized the lack of any deliberate concealment.[83] The question is whether a software vendor's placement of "you may not analyze" language on the product is sufficiently deliberate action to make the product's internals secret, such that the product's internals are not prior art.[84]

With one exception, such language is widely regarded as unenforceable,[85] often even carrying indicia of its possible unenforceability: "you may not analyze, reverse engineer, decompile, disassemble the Software or seek to obtain a source code of the Software in any way, except for the scope stipulated by applicable law."[86] As for the applicable law, as noted earlier, California's Uniform Trade Secrets Act explicitly states reverse engineering by itself is not improper means of learning a trade secret.[87] The important exception is non-public software, such as a limited "beta" test for which the user must sign.[88] Even here, simple methods such as string extraction may not even constitute "reverse engineering" as the license language defines this term.

But does the general non-enforceability of such clauses really dispose of the question whether the mere presence of the clause renders information hidden inside object code a secret, at least until someone decides to ignore the clause? Doesn't the presence of a no-RE clause make information buried in otherwise-public software that much less accessible or available to the POSITA, akin to an internal document, which is not a printed publication "no matter how many copies are distributed?"[89]

However, while a company's proprietary source code is an internal document, information buried inside object code available in the marketplace, with no-RE shrinkwrap, is something entirely different: a widely-available public document which is marked "Confidential." If it can be had by anyone for the asking, by paying some amount, from whom is it being kept confidential?

AKIN TO CLASSIFIED TECHNOLOGY?

The software scenario—activity designated as confidential but carried out in the marketplace—can be analyzed under cases relating to the prior-art status of government-classified technology.

First, classified technology is not prior art.[90] Even if later Wikileaked, it would not be prior art during the period it was still confidential.[91]

But second, the holder himself must do something to keep the material confidential. This is akin to reasonable security precautions (RSP) in trade-secret law. For example, documents in a library on a military base, with security and subject to the librarian's approval, were nonetheless printed publications because they were distributed to commercial companies and private individuals without restriction.[92] In general, information is publicly accessible if "interested members of the relevant public could obtain [it] if they wanted to,"[93] even for a fee.

A recent application to software is *Ex parte ePlus*, in which the BPAI considered ePlus's argument that a user manual was proprietary: "[m]ost software and the manuals that come with such software would contain restrictions on copying and further distribution, but that would not rise to the level of those items being considered confidential disclosures...[which] would require restrictions on who could purchase or otherwise obtain the specific documentation.... The requirement of purchasing software to obtain a manual goes to its cost and not to its accessibility."[94]

The BPAI distinguished *IMX v. Lendingtree*,[95] in which the user manual for mortgage-related software was found not to be a printed publication, because in *IMX* there was no independent evidence that the manual (stamped "Highly Confidential") was publicly accessible, whereas in *ePlus* the examiner "clearly show[ed] that the software packages had been sold."

An analogy may be made to the experimental use exception to public use under patent (or trade secret) law, in which the inventor's control over the invention is the key factor in determining whether use of the invention in public is truly experimental, or a public use.[96] Just as a statement of experimentation, without control, would not cancel public use, likewise a statement of confidentiality, without a restriction on dissemination, does not create confidentiality. Secrecy has an objective as well as a subjective component, and a vendor cannot cast a spell of secrecy over object code present on perhaps millions of machines (including open-access internet cafes) by mere *ipse dixit* or by a no-RE clause in a EULA (End User License Agreement).

COLLECTING PRIOR ART AS COPYRIGHT VIOLATION?

Another potential concern is whether a database built from text extracted from software products would constitute a copyright-infringing derivative work.

First, the database contains extracts from code, not the entire code; it is not as a collection of discrete pieces of software, but a concordance of words and phrases, linked back to information on the file in which the word or phrase is found.[97] In many cases, the same string is located across files from multiple vendors; in the case of some vendors, few of their strings will be unique.

Because the database contains fragments from code, most of these fragments served some functional purpose in the product from which they were extracted, and would thus be analyzed under the copyright merger doctrine.[98] In *Lotus v. Borland*, even an extensive set of 469 strings comprising a menu hierarchy was found to be functional, akin to the labels on VCR buttons.[99] The prior-art software database discussed here is, in large part, collecting such labels, such as names of API calls, error messages, debug text, and menu and dialog items. Copyright protection would be "thin," largely confined to non-constrained portions of the text, such as unused "dead code."

Alternatively, analysis would follow fair use in library preservation, and of caching by search engines such as Google.[100] Given that the database's purpose is patent related, and can also be used to find software copyright infringement and open-source violations, litigation privilege would be a further defense.[101]

A DATABASE OF PRIOR-ART SOFTWARE

The author has used preliminary versions of this database of prior-art software in several cases, and a group of software engineers including the author is building a more extensive database, CodeClaim.com. The code is out there; it's prior art; it can be indexed and searched. ◀◀

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 LL.M. program at Golden Gate University. His website is 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. In addition to examples given in part 1 of this article, note the similarity between older and recent concerns, *e.g.*, in 1995 "[u]seful examples of prior art are most likely buried within application programs or system software, and known only to the authors in any detail" (Bernard Galler, SOFTWARE AND INTELLECTUAL PROPERTY PROTECTION 32 (1995)) and in a 2009 online discussion of i4i's metacode-

mapping patent (U.S. Patent 5,787,449, recently the subject of *i4i v. Microsoft*, 564 U.S. ___ (2011)), "it can be impossible to do prior art search for software. There were numerous word processing systems in the 70s and 80s that took [this]...the companies that created these systems are gone and none of the systems [are] running. There was no Web so the only articles are in old magazines. Companies didn't publish the details of their implementations...." (comment at http://broadcast.oreilly.com, Aug. 15, 2009).

2. As discussed in detail later, while a third party's truly confidential or secret (as opposed to merely hidden or obscure) technology is not prior art, the patentee's own secret activities may raise the on-sale bar.
3. An API is a service provided by one piece of software and used by another. APIs generally have names, or numbers for which there are well-known names. These APIs are typically documented by the vendor, but there is also a large category of "undocumented" APIs. In Microsoft's case, some of these were the subject of an antitrust consent decree. *See* part 1, and Geoff Chappell, *Missing Settlement Functions*, at http://www.geoffchappell.com/viewer.htm?doc=studies/windows/shell/missing.htm.
4. The "strings" algorithm counts consecutive sequences of printable characters; when the end is found of a sequence longer than some minimum such as four or five characters, the sequence is printed, the count reset to zero, and the search for printable characters resumed.
5. "A trade secret law, however, does not offer protection against discovery by fair and honest means, such as by independent invention, accidental disclosure, or by so-called reverse engineering, that is, starting with the known product and working backward to divine the process which aided in its development or manufacture." *Kewanee Oil Co. v. Bicron Corp.*, 416 US 470, 476 (1974), citing a 1902 case, though the earliest judicial use of the term "reverse engineering" appears to be *Koehring v. Etnyre & Co.*, 254 F. Supp. 334, 339 (N.D. Ill. 1966); *see also Thompson Ramo Woolridge Inc. v. United States*, 175 Ct. Cl. 527, 532 (Ct. Cl. 1966).
6. Jonathan Band and Masanobu Katoh, Interfaces on Trial 2.0, 138-9 (2011; available online at http://mitpress.mit.edu/band) (IBM stating in 1990 that, among RE techniques, only decompilation is "regarded as objectionable"). IBM had claimed these techniques assist software pirates, before itself feeling compelled to reverse engineer Microsoft Windows. *Id.* at 56 (IBM reverse engineering Windows) and 138–139 (earlier IBM demonstration of decompilation facilitating piracy).
7. *E.g.*, Clive Turvey, Win32 disassembler in dumppe at http://www.tbcnet.com/~clive/vcomwinp.html#DUMPPE; IDA Pro disassembler at http://www.hex-rays.com/idapro/; SWF Flash decompiler at http://www.sothink.com/product/flashdecompiler/; and .NET decompiler at http://www.reflector.net/.
8. This point is emphasized in a classic article, Andrew Johnson-Laird, *Software Reverse Engineering in the Real World*, 19 U. Dayton L. Rev. 843 (1994); today, the point would perhaps rather be how much information (though not the original source code) can be gleaned from widely-available software products.
9. Andrew Schulman, *Finding Binary Clones with Opstrings and Function Digests*, Dr. Dobb's Journal, July 2005 (Part I), August 2005 (Part II), and Sept. 2005 (Part III). The opstrings technique enables sequence searching of object code, similar to biological sequence searching and other forms of non-word searching (see Stephen Adams, Information Sources in Patents 200-201 (2006)).
10. Andrew Schulman, *Notes on Similarities between QTVHDW.DLL (Apple QuickTime for Windows) and DCISVGA.DRV (Microsoft/Intel Video for Windows*, Feb. 22, 1995, at http://www.sonic.net/~undoc/apple_ms.txt (sequence of unusual API calls used to show identicality between code from two different vendors).
11. *E.g.*, Fiddler web debugger at http://www.fiddler2.com; Wireshark (formerly Ethereal) at http://www.wireshark.org/; and Mark Russinovich and Bryce Cogswell's SysInternals monitoring tools, including RegMon, FileMon, PortMon, and ProcMon, now distributed by Microsoft at http://technet.microsoft.com/en-us/sysinternals.
12. *See* Chris Pederick's Web Developer (http://chrispederick.com/work/web-developer/) and the similar tool built into Microsoft Internet Explorer version 8.
13. *But see* the SABRE airline reservation system in Lockwood v. American Airlines, 107 F.3d 1563 (Fed. Cir. 1997).
14. Andrew Schulman, *Hiding in Plain Sight: Using Reverse Engineering to Uncover Patent Infringement*, Intellectual Property Today (Nov. 2010).
15. *See* Gregory Stobbs, Software Patents (2010 revision) § 5.10 (Software Prior Art Searching).
16. This example was selected, more or less at random, from U.S. Patent 7,568,213, which has been reported as a "patent for podcasting." *See http://w2.eff.org/patent/wanted/volomedia/EFF_volomedia_prior_art.pdf.*
17. *See* Robert Faber, Mechanics of Patent Claim Drafting (2010 revision) § 3.22 (Tying the Elements Together); *NetMoney v. Verisign*, 545 F.3d 1359 (Fed. Cir. 2008); *Therasense v. Becton Dickinson*, 593 F.3d 1289 (Fed. Cir. 2010).
18. U.S. Patent 7,117,483.
19. Ronald Slusky, Invention and Analysis and Claiming: A Patent Lawyer's Guide (2007).
20. A uspto.gov search for "ACLM/thunk" returns 25 hits in patents, and 13 in published patent applications.
21. U.S. Patent 6,606,685.
22. *See Interactive Gift v. Compuserve*, 231 F.3d 859 (Fed. Cir. 2000), in which the term "point of sale" (POS) location in U.S. Patent 4,528,643 referred, not merely to cash registers and the like, but even to a home computer. The venerable "own lexicographer" phrase appears, *e.g.*, in *Chicago Steel Foundry v. Burnside Steel Foundry*, 132 F.2d 812 (7th Cir. 1943) ("and we add, his own grammarian").
23. *Vitronics v. Conceptronic*, 90 F.3d 1576 (Fed. Cir. 1996).
24. *Verdegaal Bros. v. Union Oil Co. of California*, 814 F.2d 628, 631 (Fed. Cir. 1987), cited in the Manual for Patent Examination Practice ("M.P.E.P.") § 2131.
25. Though some software vendors claim their products are "tightly coupled" or "integrated," which if true (or by estoppel) would suggest all modules constitute a single reference.
26. Of the thousands of items found in the "kitchen sink" that is Windows, few are used all the time, and the others tend to form sub-groups of modules used together; this was the basis for a Microsoft effort known as "MinWin" (minimal Windows)

and for Windows Embedded; other software exhibits similar locality of reference.

27. But the M.P.E.P., at § 608.1(p), expressly states, of incorporation of reference by a patent, "[a]n incorporation by reference by hyperlink or other form of browser executable code is not permitted." Trying to find a single reference from multiple interrelated modules also raises the issue of whether module reliance is determined statically or dynamically; static reliance (*i.e.*, a "hard" link from A to B, found in a module import/export table) can yield a bright-line determination, whereas dynamic reliance (in some amount of testing, *e.g.*, with a "dependency walker," it was found that A invokes B) depends on the particular configurations tested. These issues resemble those in the determination of which modules Microsoft was obligated to document under its consent decrees with US and European antitrust/competition authorities.
28. *KSR Int'l Co. v. Teleflex, Inc.*, 550 U.S. 398 (2007).
29. Older MS-DOS floppy-disk software is often dated Jan. 1, 1980, which is default DOS filedate, but which also predates MS-DOS 1.0 by two years.
30. *In re Epstein*, 32 F.3d 1559 (Fed. Cir. 1994) (software products in public use or on sale, more than one year before filing date, based on descriptions of products in abstracts appearing after critical date).
31. *See* the Internet Archives's "Wayback Machine" (http://www.archive.org; http://waybackmachine.org); see also "first seen on" file information, *e.g.*, at http://www.prevx.com/filenames.
32. *See* M.P.E.P. § 2128 (Electronic Publications as Prior Art); EPO, T 1134/06 (Internet Citations), 16 Jan. 2007 (lengthy discussion of prior-art status of http://web.archive.org/web/20000620174023/http://www.boersenspiel.de); David Rogers, *Documents on the Internet as Prior Art*, 2 J. IP Law & Practice 354 (2007).
33. *SRI Int'l, Inc. v. Internet Security Systems, Inc.*, 511 F.3d 1186 (Fed. Cir. 2008) (paper posted to un-secured, but also un-indexed, FTP site not prior art).
34. However, there are searchable archives of open source code, including Black Duck's http://www.koders.com (3.4 billion lines of open source code); http://codase.com (250 million lines of code); Google Code Search (http://www.google.com/codesearch); and the Netlib/GAMS (Guide to Available Mathematical Software) repository at http://www.netlib.org/bib/gams.html and http://gams.nist.gov/. There is an "Open Source as Prior Art" (OSAPA) project to make repositories such as sourceforge.net accessible to prior-art searching (see http://www.linuxfoundation.org/programs/legal/osapa).
35. *See, e.g.*, INSPEC at http://www.theiet.org/publishing/inspec.
36. As noted in Stobbs, Software Patents § 5.09, CD-ROMs of such material is available, including the Microsoft Developer Network (MSDN; http://msdn.microsoft.com/en-us/subscriptions), the Dr. Dobb's Developer Library DVD (http://store.ddj.com), and the MacTech DVD (http://www.mactech.com/cd/). However, these CDs and DVDs as shipped are generally not immediately searchable, as contents are often compressed and archived. This is perhaps why the USPTO's "search templates" (http://www.uspto.gov/web/patents/searchtemplates/searchtemplates.htm) do not refer to MSDN or to software development kits (SDKs).
37. Software Patent Institute at http://spi.org. Besides defensive publications, SPI's collection also includes "computer manuals, older textbooks, journal articles, conference proceedings, computer science theses, and other such materials which may contain valuable prior art." For reliability concerns about such material see E. Robert Yoches and Terry Callaghan, *The Next Battle: New Forms of Software Prior Art*, 2 U. Balt. Intell. Prop. L.J. 115 (1994), but the article predates *In re Klopfenstein*, 380 F.3d 1345 (Fed. Cir. 2004) (slides mounted on poster boards for two and a half days at conference are prior art, even though no copies disseminated).
38. In addition to http://spi.org, *see* ip.com's http://priorartdatabase.com/; and http://www.researchdisclosure.com.
39. *See* Doron Swade, *Preserving Software in an Object-Oriented Culture*, History of Electronic Artifacts 195–206 (Edward Higgs ed. 1998), Software Preservation Group (http://www.softwarepreservation.org/); Computer History Museum, Mountain View, CA (*e.g.*, dBase II software for Osborne 1 cataloged at http://www.computerhistory.org/collections/accession/102626801 and, atypically, MacPaint and QuickDraw source code at http://www.computerhistory.org/highlights/macpaint/); software collection of Munich Computer Museum (http://www.computermuseum-muenchen.de/software.html); Charles Babbage Institute (http://www.cbi.umn.edu); and product-specific archives such as Dan Bricklin's Visicalc site (http://www.bricklin.com/visicalc.htm).
40. A notable exception is the superb PDP-10 software archive at http://pdp-10.trailing-edge.com/ in which files have been extracted from tape images, each file given its own webpage, and contents even of executable files included on the page, enabling a Google search for strings (*e.g.*, "HOLLERITH constant used where numeric expression required"), leading to, *e.g.*, an old FORTRAN compiler containing that string (http://pdp-10.trailing-edge.com/k20v7b/01/field-image/fortra.exe.html). Another is Frank van Gilluwe's FaultWire File Analysis Lookup at http://www.faultwire.com/solutions/file_solutions.php, indexed by filename, *e.g.*, http://www.faultwire.com/file_detail/shimgvw.dll*1356.html, and containing some executable strings (*e.g.*, "GDI+ file thumbnail extractor").
41. *See* WorldCat advanced search for Format: "Computer file" at http://www.worldcat.org/advancedsearch, with tiles such as "2.0", "3.0", dBase, AutoCAD, etc.
42. *E.g.*, code at issue in Judge Jackson's Findings of Fact in *United States v. Microsoft Corp.*, 84 F. Supp. 2d 9 (D.D.C., 1999) ¶¶ 90–92 ("Withholding Crucial Technical Information")(Nov. 1999) can be found, with Aug. 1995 file date/timestamp, tucked away inside a ZIP archive file at http://replay.waybackmachine.org/19990209175113/http://www.microsoft.com/WIN32DEV/APIEXT/ RAS1244B.ZIP. *Also see* discussion of the Wayback Machine in EPO, T 1134/06 (Internet Citations), 16 Jan. 2007.
43. *See* Internet Archive Gets DMCA Exemption to Help Archive Vintage Software, http://www.archive.org/about/dmca.php; Software Archive at http://www.archive.org/details/software; description of Classic Software Preservation Project (CLASP) and copyright issues at http://web.archive.org/web/20100722072350/http://www.archive.org/details/clasp; Sam Williams, *Prowling the Ruins of Ancient Software*, Salon, July 30, 2003, at http://www.salon.com/technology/feature/2003/07/30/software_archaeology.
44. *See* http://www.archive.org/details/cdbbsarchive and http://www.archive.org/details/tucows; an especially useful archive

of material from CDs of material from older computer "bulletin board" systems (BBS) is http://cd.textfiles.com/ ("The Past on Plastic;" "Who knew that the companies looking for a quick buck through the late 1980's and early 1990's with "Shovelware" CDs would become the unwitting archivists of the BBS age?")

45. *See* Elizabeth Kaplan, *A Response to "Preserving Software: Why and How,"* 1 ITERATIONS: AN INTERDISCIPLINARY JOURNAL OF SOFTWARE HISTORY 2–3 (Sept. 13, 2002), at http://www.cbi.umn.edu/iterations/kaplan.html.
46. For example, the phrase "low order memory fragmenter" appears in source code accompanying an article in PC MAGAZINE, March 28, 1995. The source code file was named 1MBFort.c, and can still be found compressed inside an archive file named 1mbfort.zip. If one already knows the 1mbfort.zip filename, it is easy to find the file. But if one had only the phrase "low order memory fragmenter," how would one find the file in which it appeared (and thereby the date when it was public), without "deep" content indexing? (Or an explicit mention, as at http://oreilly.com/centers/windows/feat/softram/sr-start.html.)
47. http://www.nsrl.nist.gov/.
48. NSRL's NSRLProd.txt and NSRLMfg.txt indicate a much larger collection than does http://web.archive.org/web/20100527095702/http://www.nsrl.nist.gov/index/prodname.index.txt.
49. http://www.nsrl.nist.gov/Downloads.htm.
50. *See* the graphic of Sherlock Holmes throwing away files in http://www.nsrl.nist.gov/Documents/NSRL-CFS-April-2009.pdf.
51. However, one NSRL project includes "Smart Unpacking" of files; *see* Benjamin Long, *Smart Unpacking: Understanding Files through Patterns* (2009) at http://www.nsrl.nist.gov/Documents/aafs2009/blong-aafs2009.pdf); extracting individual source and object code files from archives is crucial for large-scale use of software products as prior art.
52. *Nat'l Tractor Pullers v. Watkins*, 205 U.S.P.Q. 892 (N.D. Ill., 1980), though this relates largely to the need for contemporaneous corroboration of later oral testimony regarding earlier non-doc prior art (*see also Woodland Trust v. Flowertree Nursery*, 148 F.3d 1368 (Fed. Cir. 1968)).
53. *See* so-called "secret prior art" (eventually printed patent applications, prior art as of the application date) in 35 U.S.C. § 102 (e); *Hazeltine v. Brenner*, 382 U.S. 252 (1965); and the inventor's own secret prior activity, noted below.
54. M.P.E.P. § 2128.01 (Level of public accessibility required) III (internal documents intended to be confidential are not "printed publications") ("[d]ocuments and items only distributed internally within an organization which are intended to remain confidential are not 'printed publications' no matter how many copies are distributed," citing, *e.g,. Garret v. United States*, 422 F.2d 874 (Ct. Cl. 1970)). Confidential source code is a perfect example, but publicly-sold object code produced from the source code, and any source fragments which find their way into the object code, is not.
55. M.P.E.P. § 2128 (examiner need not prove anyone actually looked at the document).
56. *Metallizing Engineering Co., Inc. v. Kenyon Bearing & Auto Parts Co., Inc.*, 153 F.2d 516 (2d Cir. 1946)(L. Hand, J.); *see also Auld v. Chroma Graphics*, 714 F.2d 1144 (Fed. Cir. 1983) (method patentee's secret method used to prepare samples offered for sale before critical date).
57. *Dunlop v. Ram Golf*, 524 F.2d 33, 37 (7th Cir. 1975), discussed below.
58. *Gillman v. Stern*, 114 F.2d 28 (2d Cir. 1940) (L. Hand, J.; secret method not discernible from public product); *see also Gore v. Garlock*, 721 F.2d 1540 (Fed. Cir. 1983) (public could not have learned crucial parameters even from observing the machine, nominally held in confidence, much less from the output of the machine); *Metallizing Engineering v. Kenyon Bearing & Auto Parts*, 153 F.2d 516 (2d Cir. 1946) (L. Hand, J., citing numerous cases of secret machines producing public output).
59. *But see* the SABRE airline reservation system in *Lockwood v. American Airlines*, 107 F.3d 1563 (Fed. Cir. 1997).
60. *Dunlop v. Ram Golf*, 524 F.2d 33, 37 (7th Cir. 1975) ("[i]n this case, for example, it is not unreasonable to assume that competing manufacturers of golf balls in search of a tough new material to be used as a cover, might make inquiries of Wagner's Surlyn supplier that would soon reveal his secret ingredient"); a supplier inquiry is analogous to (and may even form part of) RE. *See also Kewanee Oil Co. v. Bicron Corp.*, 416 U.S. 470, 491 (1974) ("If the invention, though still a trade secret, is put into public use, the competition is alerted to the existence of the inventor's solution to the problem and may be encouraged to make an extra effort to independently find the solution thus known to be possible"). But note "still a trade secret"; *see below.*
61. Philip Burke, *The "Non-Informing Public Use" Concept and its Application to Patent-Trade Secret Conflicts*, 45 ALB. L. REV. 1060 (1980-81); Ami Patel, *Advocating a Totality of the Circumstances Test to Analyze a Non-Informing Use of an Invention*, 48 WAYNE L. REV. 1287 (2002-03).
62. *Hall v. Mac*neal, 107 U.S. 90 (1883).
63. Except for what are called "subjective secrets," exemplified by "I'm thinking of a number between 1 and 1000." *See* Arvin Quist, *Security Classification of Information*, Vol. 2, PRINCIPLES FOR CLASSIFICATION OF INFORMATION (1993), Ch. 2, Major Types of Classified Information – Subjective and Objective Secrets, at http://www.fas.org/sgp/library/quist2/chap_2.html. Even as to objective secrets, so-called "security through obscurity" works better in practice than many purists admit; *see* COLLBERG ET AL., SURREPTITIOUS SOFTWARE: OBFUSCATION, WATERMARKING, AND TAMPERPROOFING FOR SOFTWARE PROTECTION (2009).
64. *In re Epstein*, 32 F.3d 1559 (Fed. Cir. 1994).
65. M.P.E.P. § 2112 (Requirements for rejection based on inherency; burden of proof), in particular, § 2112 (II) (inherent feature need not be recognized at the time of the invention).
66. *Kewanee Oil Co. v. Bicron Corp.*, 416 U.S. 470 (1974); F. Andrew Ubel, *Who's On First?—The Trade Secret Prior User or a Subsequent Patentee*, 76 J. PAT & TRADEMARK OFF. SOC'Y 401 (1994); Leslie Hill, *Prior User Defense: The Road to Hell is Paved with Good and Bad Intentions*, 10 FED. CIR. B.J. 513 (2000-01).
67. *Compare Brenner v. Manson*, 383 U.S. 519 (1966) (no patent for invention without disclosure of utility; "a patent is not a hunting license") *with Oak Industries v. Zenith Electronics*, 726 F. Supp. 1525 (N.D. Ill. 1989) (possibility of patent even when likely that invention was already known to industry).
68. Uniform Trade Secrets Act § 1 (4); California omits this phrase from its UTSA adoption Cal. Civ. Code § 3426, but ready ascertainability is a defense, and may affect damages.
69. California's UTSA explicitly notes that "[r]everse engineering or

independent derivation alone shall not be considered improper means." California UTSA, Cal. Civ. Code § 3426.

70. *See* Fig. 1 in Andrew Schulman, *Open to Inspection: Using Reverse Engineering to Uncover Software Prior Art, Part 1*, NEW MATTER, Summer 2011, at 29.

71. *Compare In re Robertson*, 169 F.3d 743 (Fed. Cir. 1999) (inherency "may not be established by probabilities or possibilities. The mere fact that a certain thing may result from a given set of circumstances is not sufficient.") *with Schering Corp. v. Geneva Pharm.*, 339 F.3d 1373 (Fed. Cir. 2003) ("inherent anticipation does not require that a person of ordinary skill in the art at the time would have recognized the inherent disclosure").

72. *E.g.*, Geoff Chappell's unraveling of Microsoft's deliberately-obfuscated AARD code; *see* Schulman, *Examining the Windows AARD Detection Code*, DR. DOBB'S JOURNAL, Sept. 1993; and Chappell letter to editor, DR. DOBB'S JOURNAL, Jan. 1994.

73. *In re Seaborg*, 328 F.2d 996 (C.C.P.A., 1964) ("the claimed product, if it was produced in the Fermi process, was produced in such minuscule amounts and under such conditions that its presence was undetectable").

74. *See E.I. duPont deNemours & Co. v. Christopher*, 431 F.2d 1012 (5th Cir. 1970) (trade secret not lost when DuPont failed to take unreasonable security precautions to prevent aerial photography of chemical plant under construction). As one data point on RE costs, see Honeynet Project, *Reverse Challenge Results, Time/Cost Analysis* (2002) at http://old.honeynet.org/reverse/results/#cost.

75. One RE organization's definition of RE even emphasizes this: "[r]everse engineering is the process of generating new information about software...." (http://www.program-transformation.org/Transform/UVicReverseEngineeringTutorial). When a third-party analyst discovers a security bug in another's product, it's debatable that this is the creation of new information, because the bug already existed, though (hopefully) unknown to the vendor. Whether RE can create as well as discover facts is related to the copyright issue of "created facts," including academic response to the naïve view of facts in *Feist v. Rural Telephone*, 499 U.S. 340 (1991), and "numeric expression" in cases such as *CDN Inc. v. Kapes*, 197 F.3d 1256 (9th Cir.1999) and *NY Mercantile Exchange v. Intercontinental Exchange*, 497 F.3d 109 (2d Cir. 2007). Justin Hughes, *Created Facts and their Awkward Place in Copyright Law*, INTELLECTUAL PROPERTY PROTECTION OF FACT-BASED WORKS 186 (Brauneis ed. 2009).

76. *E.g., Abbott Labs v. Geneva Pharmaceuticals*, 182 F.3d 1315 (Fed. Cir. 1999) (earlier third-party sales of a product no one, not even the seller, knew at the time happened to include the later-to-be-disputed compound; only years later, in litigation, was it revealed the compound had in fact been on sale before plaintiff's patent application; even the seller of a product need not have recognized the significance of its product, for that product to constitute prior art, if the product is later found to inherently possess all claim limitations).

77. *Compare In re Hall*, 781 F.2d 897 (Fed. Cir. 1986) (doctoral thesis, indexed in library catalog, is prior art) with In re Cronyn, 890 F.2d 1158 (Fed. Cir. 1989) (doctoral thesis, only indexed by student name in shoebox in department library, is not prior art).

78. M.P.E.P. § 2128 ("Printed Publications" as Prior Art; Electronic Publications as Prior Art); electronic content can qualify as a "printed publication" because this term of art requires neither printing nor publication in the traditional sense, requiring rather dissemination and availability. *In re Wyer*, 655 F.2d 221, 226 (C.C.P.A., 1981); JOSEPH ROOT, RULES OF PATENT DRAFTING (2011), Ch. 13 (Printed Publication).

79. *In re Donohue*, 766 F.2d 531 (Fed. Cir. 1985) (mere naming of claimed compounds would be insufficient to constitute enabling disclosure, but article in J. CHEM. SOC'Y sufficiently describes invention because POSITA could have combined with his own knowledge to make claimed invention); M.P.E.P. § 2121.01 (use of prior art in rejections where operability Is in question).

80. *Egbert v. Lippmann*, 104 U.S. 333 (1881), concerning the patentee's own public use; for a third-party analogy, *see Baxter Int'l v. COBE Laboratories*, 88 F.3d 1054 (Fed. Cir. 1996) (Newman, J. dissent: "[h]eretofore, § 102(e) was the only source of so-called 'secret prior art,'" majority allows "unknown, private laboratory work to create a new bar to patentability").

81. *E.g.*, http://www.teac.co.jp/dspd/download/driver_agreement.html ("[e]xcept to the extent expressly permitted by the laws of the jurisdiction where you are located, you may not analyze, decompile, disassemble or otherwise reverse engineer the Software"). *Compare with* http://www.caseta.com/legal.html ("If you do not agree with these terms and conditions, then you may not read...information on this Site").

82. Gilman, *supra* note 58, at 30.

83. *Rosaire v. Baroid Sales*, 218 F.2d 72, 74 (5th Cir. 1955) ("[t]he work was performed in the field under ordinary conditions without any deliberate attempt at concealment or effort to exclude the public and without any instructions of secrecy to the employees performing the work").

84. *See* Mark Flagel and Steven Weiner, *Existing Programs Could Be Considered Prior Art; Publicly Available Software with a Ban on Reverse-Engineering May Invalidate Future Registrations*, 17 NATIONAL LAW JOURNAL C32 (May 8, 1995); Flagel and Weiner, *Trade Secret Software As Prior Art: Litigation Strategies*, 11 COMPUTER LAWYER 8 (Dec. 1994); George Gates, *Trade Secret Software: Is It Prior Art?*, 6 COMPUTER LAWYER 11 (Aug. 1989).

85. *See, e.g., Vault v. Quaid*, 847 F.2d 255 (5th Cir. 1988), but note shrinkwrap license enforceability in other contexts, *e.g., ProCD v. Zeidenberg*, 86 F.3d 1447 (7th Cir. 1996); *see also* Jonathan Brand and Masanobu Katoh, *Interfaces on Trial 2.0* (2011); Pamela Samuelson and Suzanne Scotchmer, *The Law and Economics of Reverse Engineering*, 111 YALE L.J. 1575 (2001-2).

86. http://mousecloner.com/eula.html; similarly, http://www.microsoft.com/windowsxp/eula/home.mspx (Microsoft: "You may not reverse engineer, decompile, or disassemble the Software, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation.").

87. Cal. Civ. Code § 3426.1 (a) ("[r]everse engineering or independent derivation alone shall not be considered improper means"); the "alone" is important, and assumes one has legal access to the item being reverse engineered.

88. *See, e.g.*, Microsoft's assertion of trade-secret misappropriation against Stac Electronics, in counter-claim to Stac patent infringement suit, based on Stac's reverse engineering of preload interface during a beta test of MS-DOS 6.0, for which Stac signed, DR. DOBB'S JOURNAL, May 1994; http://drdobbs.

com/184409244. If the beta itself is widespread, though (some Microsoft beta tests are larger and more public than many product releases from other companies), or if the vendor engages in extensive publicity for the beta, it may look more like an early product release than a product test intended to be maintained in confidence.

89. M.P.E.P. § 2128.01.

90. *Del Mar Engineering v. United States*, 524 F.2d 1178 (Fed. Cir. 1975); *Lockheed v. United States*, 553 F.2d 69 (Ct. Cl. 1977).

91. Of course, once a secret is disclosed, it is no longer a secret, even if the disclosure was improper. *Religious Technology Center v. Lerma*, 908 F. Supp. 1362 (E.D. Va. 1995); in particular, the Section 102(b) bar still operates despite misdisclosure. *Evans Cooling Systems, Inc. v. General Motors Corp.* 125 F.3d 1448 (Fed. Cir. 1997).

92. *Siemens-Elema AB v. Puritan-Bennett Corp.*, 1989 U.S. Dist. LEXIS 16609, 5–7 (S.D. Cal., 1989).

93. *Constant v. AMD*, 848 F.2d 1560 (Fed. Cir. 1988). *But see National Semiconductor v. Linear Technology*, 703 F. Supp. 845 (N.D. Cal., 1988) (widespread "grabbing of papers" at conference and showing them to competing companies, contradicting an *implied* confidentiality policy, did not constitute publication).

94. *Ex parte ePlus*, BPAI Appeal 2010-007804, Reexamination Control 90/008,104, Patent 6,023,683 (May 2011) at http://des.uspto.gov/Foia/ReterivePdf?system=BPAI&flNm=fd2010007804-05-18-2011-1.

95. *IMX v. Lendingtree*, 405 F. Supp.2d 479 (D. Del. 2005).

96. *Lough v. Brunswick*, 86 F.3d 1113 (Fed. Cir. 1996) (extent of control inventor maintained over testing is "critically important").

97. The difference between a copy and a concordance is the difference between, *e.g.*, a copy of the Bible and an index of key words in the Bible. The concordance contains words and short phrases, which (apart from their original selection or arrangement) are not protectable, and the original generally cannot be recreated from the indexing (but see reconstruction of Dead Sea Scrolls from concordance in, *e.g., Lim et al.,* On Scrolls, Artefacts and Intellectual Property (2001)). There is not a copy of the software as such, except in some cases a temporary copy created during software analysis, and likely existing for less than the 1.2 seconds noted in *Cartoon Network v. CSC*, 536 F.3d 121 (2d Cir. 2008); but *see* the older *MAI v. Peak*, 991 F.2d 511 (9th Cir. 1993) (loading program into RAM is making a copy; congressionally overruled for computer repair in 17 U.S.C. § 117).

98. *Nichols v. Universal Pictures*, 45 F.2d 119 (2d Cir. 1930) (L. Hand, J.), applied to software in *Computer Associates v. Altai*, 982 F.2d 693 (2d Cir. 1993) (elements dictated by efficiency, dictated by external factors, and taken from public domain; the court states that its abstractions test resembles RE).

99. *Lotus v. Borland*, 49 F.3d 807 (1st Cir. 1995).

100. *Field v. Google*, 412 F. Supp. 2d 1106 (D. Nev. 2006); *Parker v. Google*, 422 F. Supp. 2d 492 (E.D. Pa. 2006) (characterizing Google as indiscriminately collecting material). A potential difference is that Google's raw material (with the important exception of Google Books) is already online, with an implied license for indexing, if the material is linked-to by another page (*i.e.*, part of the "web"), and not excluded by a robots.txt file. *eBay v. Bidder's Edge*, 100 F. Supp. 2d 1058 (N.D. Cal. 2000). It might be argued that there is less fair use for unpublished works (*Harper & Row v. Nation*, 471 U.S. 539 (1975)), but software products are published, albeit in an obscure language.

101. A software patent owner suspecting infringement "may have no choice but to reverse engineer the rival's software in order to gain the evidence it needs to file suit." Julie Cohen and Mark Lemley, *Patent Scope and Innovation in the Software Industry*, 89 CAL. L. REV. 1, 28 (2001). The same point has been made regarding the impact of DMCA anti-circumvention on litigant's ability to research their cases. Jeffrey Sullivan and Thomas Morrow, *Practicing Reverse Engineering in an Era of Growing Constraints under the Digital Millennium Copyright Act and Other Provisions*, 14 ALB. L.J. SCI. & TECH. 1, 6 (2003).