

# Source Code: A Solution to the Software Patent Problem?

Michael Libertin and Mike Mester

May 10, 2009

# Contents

<b>I</b>	<b>Source Code Disclosure Under Current Law</b>	<b>1</b>
1	Statutory Requirements	2
2	Case Law	5
3	Policy and Regulations	9
<b>II</b>	<b>The GIF Patent</b>	<b>11</b>
<b>III</b>	<b>Should Source Code Be Required?</b>	<b>14</b>
4	Is Disclosure Even the Problem?	14
5	Implementation Issues	17

# Introduction

Almost everyone seems to agree that the current patent system is not handling software inventions adequately, and proposed solutions run the gamut from subtle doctrinal changes [28] to wholesale abolition of software patents [21]. One more moderate idea is to simply require applications for software patents to include the invention's source code, effectively making patented software open source<sup>1</sup> [10]. Mandating source code in software patents is an attractive proposal in large part because of its simplicity, and it seems likely to better fulfill the patent system's promise of expanding technical knowledge through disclosure. Better yet, since patent law already requires that patented inventions be disclosed, a source code disclosure requirement might not require new law.

In this paper, we examine the question of source code disclosure in software patents. In Part I, we evaluate statutes, case law, and regulations to determine whether patent law as currently understood requires software patents to include source code. We conclude that disclosure requirements for patents deem non-source code means of disclosure sufficient for almost all software inventions and that current regulations in fact discourage submission of source code with patent applications. In Part II, we use a famous software patent as a case study for source code's effectiveness in disclosing software inventions, demonstrating that source code adds little to the patent. Finally, in Part III we consider whether the patent system should be reformed to require source code disclosure. Arguing that software patents are not systematically underdisclosed and that source code does little to aid disclosure, we conclude that a source code requirement for software patents is ill advised. As promising as a source code requirement might seem initially, it would do little to improve the quality of software patents and its implementation would introduce substantial problems.

## Part I

# Source Code Disclosure Under Current Law

When examining the legal aspects of software patents and source code disclosure, three elements must be considered. First is, of course, the patent statute itself, since it forms the basis of all patent law. We consider

---

<sup>1</sup>Of course, software patents with disclosed source code would not be free in the broader sense. Indeed, since the source code disclosed would likely remain under copyright of the inventor or her assignees, the patented software could not even be said to be properly open source. We consider the copyright issue more deeply in Section 4 below, but the relatively limited copyright protection given to source code means that programmers could likely rewrite a patented invention's source code to avoid copyright infringement (assuming of course the patent has expired, lest they infringe the patent itself).

its bearing on software patents and source code disclosure in Section 1. Second is the case law, which is particularly important here because courts have extensively augmented the relatively bare text of the patent statute with common law doctrine. We examine three seminal cases concerning source code disclosure in Section 2. Third are relevant regulations, as they shed light on how the statute is applied by the executive branch as embodied by the USPTO. We consider these regulations in Section 3. As it turns out, it is quite clear that as the law stands, there is no blanket requirement to disclose source code. In fact, there is even reason to believe that source code disclosure in patents is discouraged, if not forbidden.

## 1 Statutory Requirements

The most relevant statute by far to the discussion of source code disclosure in patent is Title 35, known colloquially as the Patent Act. The section describing the patent specification, §112, is most applicable, since the provisions of this section outline the basic disclosure requirements for patents in general. The relatively well-know excerpt below defines what a patent specification must contain:

The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same, and shall set forth the best mode contemplated by the inventor of carrying out his invention. [1]

Contained in this excerpt are all three of the disclosure requirements for patents applications: the written description requirement (Section 1.1), the enablement requirement (Section 1.2), and the best mode requirement (Section 1.3). We consider each of these requirements and their consequences for source code disclosure below.

### 1.1 Written Description

As its name suggests, the written description requirement requires patent applications to contain a written description of their invention. It is fairly clear that this requirement alone does not necessitate the disclosure of source code. While it is conceivable that one might think of source code as a “description” of a program,<sup>2</sup> it

---

<sup>2</sup>Interestingly enough, the claim that source code is a description of a program is apparently preemptive defense used by the makers of the LAME mp3 encoder, which is potentially infringing on patents held by a group called the Fraunhofer Society. The creators of LAME claim that by only providing the source code of their encoder, they are only providing a description of it and not the encoder itself, which would infringe. While this information was obtained from Wikipedia and thus unsubstantiated, it seems likely to be true, especially given the LAME’s own website describes LAME as “an educational tool to be used for learning about MP3 encoding”[4].

is certainly not the case that code is the only way in which a program can be described. In other words, source code may be sufficient for satisfying the written description requirement, but it is certainly not necessary.

There is reason to doubt that source code is even sufficient., however Note that the statute requires the description to be both “clear” and “concise” [1]. Someone unfamiliar with coding might think that source code satisfies both these requirements, especially to one “skilled in the art.” The reality of software proves otherwise: source code is often not very clear, and it is certainly not concise.

While a skilled computer scientist might be able to read source code, it is *very* unlikely that this is the clearest way of describing the software, for several reasons. First, source code is programming-language specific, and thus unclear to even skilled programmers unfamiliar with that particular language. Say, for example, that a given software innovation has been implemented in FORTRAN, a relatively old and complicated language. There are likely to be a fair number of people who, though “skilled in the art” of programming, are unable to read a language like FORTRAN. To these people, a description containing source code would be unclear at best and useless at worst. Secondly, there are other ways of describing a program that are likely to be more clear. Pseudocode (an informal list of code-like instructions that incorporates elements of plain language) is a good example of a better means of conveying the description of a software invention to a skilled practitioner. As an added benefit pseudocode is not programming language specific, and can be understood by any reasonably skilled programmer.

With regard to conciseness, source code disclosure produces even more problems. Source code is rarely—if ever—concise. It often consists of multiple libraries, each consisting of hundreds if not thousands of lines. A few lines of pseudocode, for example, might be able to effectively describe a great many lines of source code.

Given the fact that source code is often neither clear nor concise, it would seem that an application containing source code would not satisfy the written description requirement of §112. In any event, it is certainly not the case that the written description requirement necessitates source code disclosure.

## 1.2 Enablement

The enablement requirement further specifies what the specification of a patent application must contain. Specifically, it requires that the applicant disclose enough information so as to “enable any person skilled in the art. . . to make and use” what is described in the invention [1]. As with the written description requirement, it might seem at first glance that source code disclosure would not only satisfy this requirement, but would be an ideal way to do so. Again, however, closer inspection reveals otherwise.

A requirement of source code disclosure would indeed enable someone to use the program in question, but

it seems to go well beyond what the statute requires. If source code is included in a patent, anyone—not just someone skilled at coding—could simply go to the USPTO’s website, cut the source code out of the published patent, paste into a text editor, and compile it. Whereas the statute requires one disclose enough information on how one makes the invention so that a skilled person can recreate it, requiring source disclosure sets the bar much higher: it essentially requires the patent applicant to disclose so much information that nearly anyone can recreate the invention.

There is also the question as to whether or not if one discloses source code one has actually included instructions on how to make the invention, rather than just the invention itself. If one considers the “making” of a program to be the writing on source code, then by disclosing source code one is disclosing not the method of making the invention, but a product of that making. While this point is debatable, it casts further doubt on the legal standing of a source code disclosure requirement for patents.

Given these considerations, it is clear that while source code disclosure *might* satisfy the enablement requirement, it is not the only way to do so, and as such is certainly not mandated by the enablement requirement.

### 1.3 Best Mode

The statute also requires that the applicant disclose “the best mode contemplated by the inventor of carrying out his invention” [1]. The purpose of best mode is to prevent inventors from hiding better implementations of their inventions by disclosing only inferior modes, thereby frustrating the patent system’s goals of advancing knowledge. Of the three main elements of patent disclosure, best mode comes perhaps the closest to requiring source code because it deals specifically with implementation. Since software is implemented through source code, there is at least a plausible argument that best mode requires inventors to provide source code in some cases. When the inventor knows of some ingenious or obscure way of coding a part of her invention, best mode indeed demands that she disclose it. While source code is probably the most obvious means for that disclosure, suitably detailed pseudocode seems acceptable too. As with most other patent law doctrines, the test is merely whether a person of ordinary skill in the art could reconstruct the inventor’s best mode from the disclosure without undue experimentation. Nothing in this flexible standard would seem to mandate source code disclosure for best mode if pseudocode or something else adequately explain what the inventor intends. Moreover, only those parts of the inventor’s best mode that are not otherwise obvious to a person of ordinary skill in the art need be disclosed. The vast majority of most software’s source code is likely obvious enough from the rest of the patent’s specification to be omitted, leaving only small snippets of cleverness

that must be disclosed under best mode. Certainly, best mode as currently understood does not constitute a blanket requirement that source code for any software invention be provided in full.

Even if best mode were reinterpreted to require full source code disclosure, motivated inventors could easily evade the requirement. The best mode requirement only applies when the inventor has a preference for a mode of implementation at the time of filing. In the absence of that subjective preference, there is quite simply no best mode to disclose. An inventor who delays contemplation of any best mode before he files his patent application (perhaps by refraining from reducing the invention to practice) therefore need not worry about the best mode requirement at all. This subjective element of best mode severely limits its potential effectiveness as a means for requiring source code disclosure.

## 2 Case Law

It seems clear from the discussion in Section 1 that multiple interpretations could arise from the statute, and that it is somewhat difficult to choose amongst them. This is where case law are helpful, as they show how the law has been interpreted in practice. On the issue of source code disclosure in software patents, there are three foundational cases: *In Re John W. C. Sherwood* (Section 2.1), *In Re Hayes Microcomputer Products, Inc.* (Section 2.2), and *Fonar Corp. v. General Electric* (Section 2.3). Though relatively old (none is younger than ten years and the oldest was decided a quarter century ago), these cases set the precedent for source code disclosure that continues to control today. While none of the three goes so far as to say §112 forbids the disclosure of source code, all conclude that §112 does not require source code. In this section of the paper, we discuss these three seminal cases on source code in software patents.

### 2.1 *In Re Sherwood*

*Sherwood* was decided in 1980, a time when software and computers were not used to nearly the extent that they are now. Nevertheless, software was well known in scientific fields, and the case involved a patent on a method of collecting and processing seismological data.<sup>3</sup> The original application was denied for two reasons: first because it failed to satisfy the best mode requirement, and second because the subject matter was unpatentable. We will focus on the best mode issue, leaving the complex, interesting, and important issues concerning software as patentable subject matter for another day.

Sherwood's patent was initially denied in part because the examiner believed he failed to appropriately

---

<sup>3</sup>The details of the invention are both complex and highly technical, but they are fortunately immaterial to our discussion here.

disclose the best mode of carrying out his invention. After describing the invention, the patent specification included this somewhat vague statement intended to meet the best mode requirement [7]:

The best mode contemplated by the inventor of carrying out the invention is to perform the processing steps of the invention on a large scale digital computer with the processed data imprinted into visible form on any of the presently available plotting devices.

Perhaps not surprisingly, this was not considered sufficient for the patent examiner, who, among other things, believed source code should have been disclosed. In fact, the examiner's rejection even went so far as to describe the specification as "merely an invitation to others to see if they can obtain Applicant's objectives which Applicant does not disclose how to obtain" [7].

While the USPTO Board of Appeals agreed with the examiner, the court to which Sherwood appealed did not. In his opinion, Judge Baldwin explained that writing source code from a description like Sherwood's was something that could often be done by someone skilled in the art. At the same time, however, he did not say that this was always the case. In a passage quoted later in *Hayes*, Baldwin expressed the opinion that, depending on the program, writing source code could either be a tedious, uninventive process or a challenging, creative one. He explained that source code need not be disclosed, at least in the former case:

In general, writing a computer program may be a task requiring the most sublime of the inventive faculty or it may require only the droning use of a clerical skill. The difference between the two extremes lies in the creation of mathematical methodology to bridge the gap between the information one starts with (the "input") and the information that is desired (the "output"). If these bridge-gapping tools are disclosed, there would seem to be no cogent reason to require disclosure of the menial tools known to all who practice this art.

Baldwin also drew a cogent comparison between programming and translating, and in doing so reinforced the position that §112 does not usually require the disclosure of source code. While only a footnote in the case, it accurately expresses why the actual creation of source code is within the skill of the art and therefore does not need to be disclosed:

In assessing any computer-related invention, it must be remembered that the programming is done in a computer language. The computer language is not a conjuration of some black art, it is simply a highly structured language. Analogously, if a person were to express a complete thought in German, it would be no trick for a translator to convert that thought into a palpable English form. The thought, thus expressed, might not be worthy of Shakespeare, but it would be understandable to one who uses the English language. Similarly, the conversion of a complete thought (as expressed in English and mathematics, i.e., the known input, the desired output, the mathematical expressions needed and the methods of using those expressions) into the language a machine understands is necessarily a mere clerical function to a skilled programmer.

These two points show an admirably deep understanding of programming and give a great deal of force to the position the source code disclosure is not required.

*In Re Sherwood* is the first major case to deal with anything like a software patent, and thus in stating that the Patent Act does not require the disclosure of source code it set a precedent that future cases would follow.

## 2.2 *In Re Hayes*

This case is interesting because it does not involve a software patent in a strict sense, but rather a patent for a modem that contained and made use of software.<sup>4</sup> Nevertheless, *Hayes* represents another of the early cases involving the validity of software patents, particularly with regard to disclosure requirements, and it reaffirmed the logic laid out in *Sherwood*.

In 1985, Hayes Microcomputer Products had created and patented a modem that implemented, among other things, a complex timing system. A few years later, another company, Ven-Tel, began distributing a modem with similar features. When Hayes asked for licensing fees, Ven-Tel and others brought suit claiming that Hayes' patent was invalid because it did not meet the disclosure requirement. Specifically, Ven-Tel claimed that Hayes did not disclose the "structure" of the software behind the timing system [8]. The court (both the jury in the original case and the appellate court which affirmed the decision of the lower court) disagreed with this logic, and in doing so set precedent regarding the question of whether or not software patent specifications need to disclose source code. In support of its decision, the appellate court laid out several arguments.

First, most of the court's decision turned on the fact that the specification is not geared towards the general public and "that the specification [of Hayes's application] meets the requirements of section 112, first paragraph, recognizing that the specification is directed to one skilled in the art." The fact that the specification only needs to be enough for one skilled in the applicable field is key. In this particular case, the court found that the the patent was sufficient because "[o]ne skilled in the art would know how to program a microprocessor to perform the necessary steps described in the specification" [8]. Given this logic, Hayes did not need to disclose the actual firmware that ran on the modem.

Second, the patent application contained a diagram that described how the timing system worked, which the court found sufficient to satisfy the disclosure requirement. The case quotes the inventor's testimony during lower court proceedings in which he says in reference to the disclosed diagram that "if you had experience in doing microprocessor programming, you would know how to implement what's in that diagram" [8]. From this it is clear that the court believed that there are other ways of conveying the information

---

<sup>4</sup>The patent in question was for a "Modem with Improved Escape Sequence Mechanism to Prevent Escape in Response to Random Occurrence of Escape Character in Transmitted Data."

contained in source code.

Lastly, the court pointed out that “an inventor is not required to describe every detail of his invention” and that enough was disclosed in Hayes’s patent to meet the statutory requirements. This point is highly relevant and indicates that while there may be elements of an invention that, if disclosed, might make it easier to implement an invention, not all of these need be disclosed so long as enough is disclosed for the invention to be implemented. With regard to software, this bar can be met without source code.

Despite being a patent for a modem rather than a piece of software, *In Re Hayes* strengthened the precedent set by *Sherwood*, and helped clarify what qualifies as sufficient disclosure when programs are involved.

### ***2.3 Fonar v. General Electric***

The latest case that can be described as foundational, *Fonar Corp. v. General Electric*, was decided in 1997. The case involved two patents assigned to the Fonar corporation, each involving MRI machines and ways of using them. General Electric, which had been accused of infringing the patents and had in response claimed that Fonar’s patents were invalid, had been order to pay tens of millions of dollars<sup>5</sup> to Fonar in royalties and compensation.

General Electric’s claim that the Fonar’s patents were invalid was based largely on the fact that Fonar did not disclose the software on the machines and that because that software represented the best mode of carrying out the invention, Fonar had failed to satisfy the disclosure requirements of §112. Fonar disagreed, arguing that “providing a description of the software’s functions is what is important for a best mode disclosure, rather than actual source code.” The court agreed with Fonar, dealing another blow to any reading of §112 as requiring source code disclosure.

The reasons behind the court’s decision support many of the points raised in this paper’s discussion of the statutory issues regarding source code disclosure. First, the court found again that the description in the patent would have been enough for a skilled programmer to recreate the software on the machines. Second, the court recognized that there was a great deal of source code, which no doubt had a bearing on its decision. Lastly, the source code was apparently hardware-specific, and thus would not have worked on all machines. All three of these points are reflected in an excerpt of testimony from the lower court proceedings quoted in the opinion. Again, it illustrates many of the points raised in the earlier section describing the statutory concerns of clarity and conciseness:

---

<sup>5</sup>\$68,421,726, to be exact.

Q. From that written description, is there sufficient description to a software engineer, such as yourself, of what software needs to be written in order to perform the multi-angle oblique invention?

A. Yes.

...

Q. In any event, the software, itself, as we see in the hundred pages of Exhibit 816, is not reproduced in its entirety in the patent.

Is that right?

A. That's correct.

Q. Why is that?

A. For a few reasons.

First of all, it's large as you can see. It's several hundred pages. It wouldn't help someone to have that software anyway because that software only works on a Fonar machine.

What's much more important is to have a description of what the software has to do, and that is what you will find in the patent. [6]

This testimony serves to illustrate the problems that a source code disclosure requirement would pose. A hundred pages of source code disclosure would severely bloat patent applications, and machine-specific software is likely to be of little use to most patent readers in any case. We consider these difficulties in more depth in Section 5 below.

Fonar is important for its ratification of the proposition that source code is not required to satisfy §112's disclosure requirements. In fact, it does so more definitively than the cases that came before it:

Fonar's witnesses further testified that providing the functions of the software was more important than providing the computer code. We agree.

As a general rule, where software constitutes part of a best mode of carrying out an invention, description of such a best mode is satisfied by a disclosure of the functions of the software. [6]

*Fonar* states the law with rare clarity: except in extreme cases, source code disclosure is unnecessary even for satisfying the best mode requirement. Since best mode is likely the strictest of the three disclosure requirements, as discussed in Section 1.3 above, *Fonar* can be read to reject a requirement for source code disclosure more generally as well.

### 3 Policy and Regulations

The last legal aspect to consider is how the law regarding patent disclosure is enforced. This is important in that it illustrates how the law is applied before the any cases arising under the law are tried in court. In the case of software patents and source code disclosure, there are two areas that need to be examined: the USPTO's general policy and the Code of Federal Regulations. While the USPTO seems to have released little information regarding its policy on source code and software patents, it seems clear that they do not

require source code. The CFR goes further: it seems that, at least in some instances, it actually forbids source code disclosure.

### 3.1 USPTO Policy

The USPTO, like many federal bureaucracies, is a complex organization, and as such it is sometimes hard to find straightforward information about its policies. Nevertheless, there are resources from which one can glean the general position of the USPTO.

One of these is the Manual of Patent Examining Procedure, which is published by the USPTO, available on the USPTO website and used by patent examiners in reviewing applications. The document is hundreds of pages long, containing information of software patents. In particular, §2161.01 (“Computer Programming and - 2100 Patentability”) specifically deals with programs and their patentability. In this discussion, it does discuss whether or not source code disclosure is necessary. Quoting *Fonar*, it says that “source code listings are not a requirement for adequately disclosing the functions of software” [9].

The other way of gleaning information about USPTO policy is to examine how many software patent applications not containing source code have been accepted. If source code disclosure were necessary, there would be no approved patents that did not disclose source code. This is, however, obviously not the case. Even looking over just those cases reviewed in the previous section of the paper, it is clear that the latter two of them were initially approved without any source code disclosure. And while the first of them was initially rejected, that rejection was overturned in court.

Overall it seems that the USPTO does not require source code disclosure and will certainly approve patents that do not contain source code.

### 3.2 Code of Federal Regulations

The CFR is described by the Government Printing Office as “the codification of the general and permanent rules published in the Federal Register by the executive departments and agencies of the Federal Government” [5]. As such, it contains definitive rules on how the executive branch is to apply the laws that Congress has established. The CFR is therefore a good source for information on the ways in which statutes like §112 are applied.

With regard to software patents and source code disclosure, it turns out the CFR contains some fairly clear rules regarding the submission of software patents. One section, §1.96 of Title 37, deals specifically with

the software patent applications.<sup>6</sup> Interestingly enough, it not only does not require the disclosure of source code, but it also seems to discourage it and does not allow longer programs to be part of the printed patent.

First, §1.96 clearly does not require source code disclosure. This can be seen in its definition of “program listing”, a term which is used throughout the rest of the regulation:

A computer program listing for the purpose of this section is defined as a printout that lists in appropriate sequence the instructions, routines, and other contents of a program for a computer. The program listing may be either in machine or machine-independent (object or source) language which will cause a computer to perform a desired procedure or task such as solve a problem, regulate the flow of work in a computer, or control or monitor events.[3]

Given that this definition does not distinguish between source and object code, it seems that the USPTO is not actually interested in “program listings” being included for disclosure purposes, since object code, unlike source code, cannot be read by people. This suggests that even when “program listings” are included, it is done so merely for record-keeping purposes.

Second, the regulation seems to forbid the inclusion of a program of more than 300 lines in the actual specification itself.<sup>7</sup> Instead, it must be submitted “[a]s an appendix which will not be printed,” although, according to another section of the regulations, the appendix does become part of “become part of the permanent United States Patent and Trademark Office records” [3, 2]. The fact that the source code of larger programs *cannot* be submitted on the main part of the patent indicates that they are discouraged.

The CFR makes it fairly clear that source code disclosure is neither required nor encouraged.

## Part II

# The GIF Patent

In this section, we will examine one famous software patent to see in concrete terms what source code can and cannot disclose about an invention. The patent we will look at is U.S. Patent 4,558,302, better known as the patent on the Graphics Interchange Format (GIF) file format. Held by Unisys Corporation until its expiration in 2003, the GIF patent covered a data compression algorithm that forms part of the widely used GIF file format for images. The algorithm, called Lempel-Ziv-Welch (LZW) after its inventors, was also the

---

<sup>6</sup>Indeed, it is entitled “Submission of computer program listings”

<sup>7</sup>The full text of the relevant excerpt, which is somewhat unclear, is as follows: “As an appendix which will not be printed: Any computer program listing may, and any computer program listing having over 300 lines (up to 72 characters per line) must, be submitted on a compact disc in compliance with Sec. 1.52(e)” [3]. Sec. 1.52(e) describes compact disc submissions.

subject of another patent held by IBM, U.S. Patent 4,814,746.<sup>8</sup> Although never formally challenged, the Unisys patent achieved infamy as a result of Unisys’s aggressive attempts to force individuals hosting websites contains GIF images to license the format for fees reaching several thousand dollars [11].

What would become known as the GIF patent was issued in 1985 [12], but it does not mention the GIF file format or anything similar. Instead, GIF was developed independently in 1987 by CompuServe, apparently unaware of Unisys’s patent on the format’s core compression technology [22]. The file format quickly gained popularity because it produced image files of smaller size than competing formats could, conserving then-scarce bandwidth for webmasters. Around 1995, however, Unisys stepped in and threatened to enforce its rights to GIF’s compression algorithm against anyone who hosted GIF files on a website [23]. The resulting controversy spurred the creation of the alternative free format, PNG, and set off a debate about software patents. We chose the GIF patent to analyze here both for its history and for the wide variety of disclosure media it uses. In particular, by presenting the LZW algorithm as a state diagram, in prose, in pseudocode, and in complete source code, the GIF patent presents a unique opportunity to evaluate the quality of these different disclosure techniques. Since LZW is a well-known algorithm, we can also compare the patent’s disclosure to other descriptions of the invention in the technical literature. We hope the result will be a clearer understanding of the utility of source code disclosure.

While data compression techniques are not a focus of this paper, it will be useful for the following discussion to establish a basic understanding of the LZW algorithm. The goal of data compression is to take a file (which we can think of as a string of characters, like a text file) and encode it so that it takes less space, all without losing any of the file’s data. We accomplish this trick by finding common sequences of characters in the file and replacing them with shorter code sequences. In doing so, we keep a “dictionary” that associates each code with the sequence it represents. To decompress the file, we consult the dictionary for the meaning of the codes. If we were compressing an English text, for instance, we might create a dictionary that assigned very common words like “the” and “is” short codes but assigned long codes to less frequent words. Such a fixed dictionary might compress an English text well, but it would likely fare poorly if presented with, say, a French text. Fixed dictionaries are undesirable, then, because we would like to be able to compress a wide variety files without creating and storing dictionaries tailored to each type. The insight of LZW lies in its ability to create a custom dictionary for each file as it compresses. Better yet, LZW does not even require us to store the dictionary after we have compressed our file, because the algorithm reconstructs the dictionary as it decompresses. This feature and others made LZW an attractive option for the designers of GIF. Indeed,

---

<sup>8</sup>Neither patent appears to have been challenged and IBM was not aggressive in enforcing its rights, so whether these patents in fact conflict has not been settled definitively.

the algorithm remains popular today for data compression of all sorts, including in GIF files.

Perhaps most striking about the GIF patent's source code disclosure is its opacity. Although only 176 lines, including ample explanatory comments, the LZW code provided in the specification appears all but unintelligible. FORTRAN, the programming language in which it was written, was popular when the patent was filed but is archaic and obsolete now. Today, a skilled, experienced programmer might never have seen FORTRAN before and would therefore glean little from the code presented. Likewise, a patent examiner lacking FORTRAN fluency could do no more than take it on faith that the code actually did what the inventors claimed. Because the code presented only implements subroutines capable of running the algorithm and not a complete application, even testing the code requires considerable effort and skill (a tester would have to write code to process input and output, for instance).

Worse, the code does not separate the core of the invention from necessary but obvious bookkeeping. LZW's novel approach to creating and maintaining a dynamic compression dictionary is dispersed across the subroutines, sitting among trivial steps like initializing memory and checking preconditions. Sorting through the tangle of statements to find the actual algorithm would be an unenviable task. Indeed, even understanding how the LZW algorithm operates does not make deciphering the patent's source code an easy endeavor. To claim that the source code alone discloses the LZW invention, then, is beyond credibility.

Evidently the patent's authors recognized the insufficiency of the source code disclosure, since they included several other forms in the specification. Perhaps most telling, the authors provided translations of the FORTRAN code into pseudocode. Each of these translations runs to about 50 lines, and they are at least plausibly comprehensible for a person skilled in the art. Nevertheless, the translations leave much to be desired. Like the source code, they do not distill the invention down to its essence, instead bogging readers down in obvious trivialities like initializing memory, just as the source code does. While writing in pseudocode eliminates the barrier to understanding that FORTRAN imposes, the translations still include too much distracting detail.

A useful point of comparison for the patent's disclosure is Welch's 1985 journal paper introducing the algorithm [29]. Published in a well-respected, peer-reviewed technical journal, the article seems aimed at the very sort of people skilled in the art to which the patent specification should be directed. Whereas the patent's specification runs on for 24 pages (not including source code), however, Welch's paper fully describes the algorithm in less than five, including ample sections on prior literature and performance. Welch included no source code in his paper. He did provide pseudocode, but in a much less formal and mechanical style than in the patent's specification. The paper specifies the algorithm in three blocks of pseudocode, none

more than 20 lines long. The referees of the IEEE computer society that published Welch's paper evidently found that pseudocode sufficient to describe Welch's algorithm. Source code is therefore neither necessary nor sufficient for disclosing a software invention. Welch's paper discloses LZW without source code, and the patent needs a substantial volume of additional material on top of its code.

As little as the source code contributes to the GIF patent, there is good reason to believe source code would have even less utility for disclosing other software inventions. In many respects, the LZW algorithm is an ideal candidate for source code disclosure. It is a small, self-contained algorithm that can meaningfully exist independent of a larger application. It also can be implemented without reference to complicated, proprietary software libraries or specialized hardware (unlike the Fonar software discussed in Section 2.3). Many software inventions, however, enjoy neither luxury, and source code for them would be less enlightening than it was for LZW. The result is a gloomy picture for the utility of source code as a means for disclosing software inventions.

## Part III

# Should Source Code Be Required?

We now examine whether software patents should be required to disclose source code. Because source code disclosure is primarily a remedy for underdisclosed patents, we first consider whether software patents as a class could benefit substantially from more disclosure (Section 4). We then turn to the problems of formulating a source code disclosure requirement (Section 5), highlighting the difficulties of this apparently simple task. We ultimately conclude that source code would not address the core problems of software patents and that the implementation of a source code requirement would be too costly to justify.

## 4 Is Disclosure Even the Problem?

In this section, we examine current software patent disclosure practices in light of concerns over the quality of software patents in general. Put simply, we conclude that lack of disclosure does not seem to lie at the heart of the software patent problem. Studies of software patents have found no notable disclosure deficit, and none of the most infamous software patents have been widely criticized for underdisclosure. Even if software patents were typically underdisclosed, the benefits of requiring source code disclosure would likely be delayed

until a patent's expiration because of fears of willful infringement. Concerns over copyright infringement would also limit the usefulness of source code.

Although software patents do not exhibit ideal levels of disclosure, they seem no worse than patents in general. Computer scientists Martin Campbell-Kelly and Patrick Valduriez examined the fifty most cited U.S. software patents to get a qualitative view of the quality of software patents [18]. Their assessment is far from glowing, and they admit, "The level of disclosure is often less than optimal" [18, p. 3]. Nevertheless, underdisclosure is not unique to software patents. Campbell-Kelly and Valduriez note that mechanical patents typically include only the most general descriptions in their specifications [18, p. 13]. Source code being the software equivalent of detailed blueprints and part lists, requiring source code would force software patents to disclose far more than other patents must reveal. Other studies and anecdotal evidence suggest that this extra burden is unjustified.

A statistical analysis of 35,000 software patents [16] indicates that software patents generally do not suffer a quality deficit relative to other patents. Using quantitative metrics for patent quality, specifically number and type of prior art references, the analysis found that software patents fare well in comparison to other patents [16, p. 43]. Number of prior art references is an admittedly imperfect proxy for patent quality in general and might seem to have nothing at all to do with disclosure. What prior art references do measure, however, is applicant effort [16, p. 22]. Allison and Mann's work therefore suggests that negligent underdisclosure is probably no more prevalent in software patents than it is in other fields. Intentional underdisclosure is another issue, but source code disclosure is unlikely to frustrate a patent applicant dedicated to hiding her invention. Source code is after all famously amenable to obfuscation.<sup>9</sup> The main target of a source code requirement therefore appears to be negligence, and Allison and Mann show that lack of effort is not a particular problem for software patents.

None of the most infamous software patents seems to suffer from underdisclosure. While the GIF patent's specification is not a paragon of clarity, it would be hard to say that the invention presented therein is underdisclosed. Similarly, a patent on key technology for the MP3 standard [13] has never been challenged on disclosure grounds despite the trouble it has caused for free software developers and proprietary software companies alike.<sup>10</sup> Amazon's much-maligned One-click patent [14] is likewise criticized on many grounds, but disclosure does not appear to be one of them. Instead, the One-click patent captures what is perhaps the

---

<sup>9</sup>For nearly 20 years, programmers have even held an international competition to see who can best obfuscate source code (<http://www.ioccc.org/main.html>).

<sup>10</sup>Microsoft licensed the patent from its owner, Fraunhofer, for \$16 million and Fraunhofer has been aggressive in enforcing its rights (<http://www.neowin.net/news/main/07/02/22/microsoft-hit-with-15-billion-patent-verdict>). The failure of well funded, motivated parties like Microsoft to challenge the patent is strong evidence for its validity, particularly with respect to disclosure.

main line of attack against software patents: obviousness, not disclosure. Anecdotal though this evidence from prominent patents may be, it supports the empirical analyses showing that software patents do not generally suffer from underdisclosure.

In fact, the most common criticism of software patents is not that they are underdisclosed, but instead that they are obvious. Noted software patent opponent Richard Stallman has attacked software patent for creating patent thickets [24], for making property out of ideas generally [27], and for protecting trivial or obvious inventions [25]. Underdisclosure is not among his targets, and source code disclosure would do little to remedy the issues he identifies. In particular, requiring source code would do nothing to prevent obvious patents from slipping through the USPTO. Underdisclosure is in many ways the opposite of obviousness, and heavily obfuscated source code might even facilitate unscrupulous applicants making simple, obvious software look like patentable inventions.

Even if underdisclosure were a problem for software patents and even if source code were an attractive means for disclosing software inventions, source code in patent applications would likely be consulted only rarely. First, fear of willful infringement and the resulting treble damages make it unlikely that anyone would consult the source code in a patent until after the patent has expired. By the time the patent term has run out, a software invention might very well be obsolete and its source code useless. Moreover, programming languages evolve and die at a rapid enough pace that languages rarely retain their popularity for 20 years (see e.g. the GIF patent's code, written in the classically 1980s-era language FORTRAN). Second, the source code in patent applications would still be protected by copyright. Although copyright protection for computer code is thin, it certainly prohibits verbatim copying. The program would thus need to be rewritten to be used legitimately, and it might be easier for a programmer undertaking a rewrite to work from a clearer specification than raw source code. In the worst case, source code in the patent could induce others to intentionally write their versions differently, incurring avoidance costs in the process and perhaps encouraging suboptimal implementations.

Source code disclosure therefore seems to be an inapt solution to a problem from which software patents do not actually suffer. Neither empirical nor anecdotal evidence points to any great disclosure crisis among software patents. Efforts of software patent reformers would seem better spent on more pressing problems of software patents, particularly obviousness.

## 5 Implementation Issues

In addition to providing few benefits, a source code disclosure requirement would be hard to specify and implement. In this section, we examine why requiring source code is more complicated than it might appear. We first discuss the considerable problems of defining software patents. We then turn to the question of what source code must be disclosed, and we finally consider how a source code requirement would affect patent examiners.

The first hurdle in formulating a source code disclosure requirement would be determining what exactly constitutes a software patent. This question is surprisingly complex. None of the USPTO patent technology classes is either specific or exclusive to software. For instance, the GIF patent is classified in both the “Communication: Electrical” class and the “Registers” class. The Amazon One-click patent falls in two “Electronic Shopping” classes as well as a computer graphics class. In fact, the Amazon One-click is not even indisputably a software patent. Although Richard Stallman cites it as a paradigmatic case of a bad, trivial software patent [26], some patent professionals consider it a business method patent [20]. Under current patent law, the dispute is largely a matter of semantics, but a source code disclosure requirement could create a real controversy. Amazon would have ample incentive to argue for One-click as a business method if it felt its source code had value as a trade secret apart from its implementation of the One-click invention. Because patent law has generally avoided developing technology-specific doctrines, there is little legal basis for adjudicating such disputes.

Even patent scholars cannot agree on a good standard for defining software patents. Keyword-based search methods like the Bessen/Hunt technique [17] can be useful for broad statistical analyses, but any keyword search is bound to be both under- and over-inclusive [16, p. 12]. A source code requirement therefore cannot base itself on an objective, mechanical definition.

Subjective definitions suffer from two major problems, however. The first is the extra effort required of patent examiners or another disinterested party. A source code requirement could not simply rely on the applicant to self-classify, since the burden of providing source code would encourage misclassification into non-software classes. Classification could not profitably be left to litigation, either, since unlitigated patents could avoid the requirement without consequence. Adding another avenue for challenging patents would also inflate litigation costs, which are already too high. The effort required to classify inventions would not always be trivial, either. While many software patents might be easily identifiable, some classifications require considerable input from experts [16, p. 15]. The second problem is that the ambiguity inherent in any subjective standard will encourage costly avoidance behavior. Because source code disclosure represents

a substantially higher bar than patents are typically required to meet and because source code is often treated as a trade secret, patent applicants would likely be highly motivated to avoid disclosure if possible. Inventors might therefore tailor their descriptions to avoid a software classification, frustrating the goal of better disclosure. Given these problems, a subjective definition of software patents seems impractical.

An equally troublesome question arises when we consider what source code patent applicants would be required to disclose. The GIF patent had the luxury of describing a software invention that could be neatly encapsulated in less than 200 of code. Many other software patents cover inventions that cannot be delineated so clearly in source code.

Consider once again the Amazon One-click patent (assuming for the moment it is indeed a software patent): what source code would Amazon disclose if it were so required? The core of the invention involves an interaction between Amazon's web server software and a user's web browser, both of which consist of hundreds of thousands or even millions of lines of largely immaterial code. Certainly Amazon could not disclose all of this code, not least because it does not own most of it. On the other end of the spectrum, Amazon could conceivably include the HTML code that displays the One-click button on its website, but such a disclosure would be unenlightening. What Amazon actually included (in addition to the written specification) were a series of flow charts that describe the process and states of the One-click system as well as some schematic drawings of the web page. Whatever one's opinion of the One-click patent generally, both forms of disclosure better suit its subject matter than source code would. A blanket source code disclosure rule, however, would often force inventors to shoehorn their inventions into a block of source code small enough to be included in a patent application. When the inventions can be better disclosed through other means, providing the source code is a substantial waste of the inventor's and examiner's efforts.

Inventions that are more clearly software inventions than the One-click patent is can also present difficult questions over what source code must be disclosed. In 2004, Microsoft was issued a patent for tabbed web browsing [15]. The patent seems a clear case of a software patent, and if source code disclosure were required, surely a patent on a web-browser feature would qualify. As with the One-click patent, however, it is far from clear what code Microsoft could disclose. The source code that controls the creation and display of the tabs themselves is likely both voluminous and obvious. That piece of the code would also be nearly useless without the rest of Microsoft's web browser code, which Microsoft jealously guards as a trade secret. Instead, like Amazon with the One-click patent, Microsoft provided flow charts and pictures that clearly describe what tabbed browsing does and looks like. Including source code would be a distraction at best.

Yet another concern facing any source code disclosure requirement is its effect on overburdened patent

examiners. For a source code requirement to be effective, patent examiners must at least attempt to enforce it. Relying on litigation would be woefully ineffective, as substantially less than 1% of software patents are ever litigated [19]. Requiring patent examiners to closely inspect disclosed code would be infeasible, however. Even expecting them to be familiar with the programming language the code is written in would be unreasonable given the diversity of programming languages available for software development. Although one might consider having the examiners merely run the code to see if it does what is claimed, producing an executable program from source code is often a difficult task. The inventor could not easily provide such an executable with the application because of compatibility concerns. Moreover, for software patents covering inventions that improve upon much larger applications, the code provided will often not constitute an entire program in and of itself. The code provided simply could not be run in isolation from the entire application. An invention that depended on proprietary third party libraries would face difficulties as well, since the inventor might not possess a distribution license for the library on which her code depends. The situation would be similar for software inventions that depend on special hardware, like the Fonar patent (see Section 2.3). If an examiner wished to test Fonar's code, he would need a Fonar MRI machine costing millions of dollars. Surely, then, even testing code by running it is infeasible.

Although requiring source code disclosure seems on its surface to be a simple rule, adequate formulations prove elusive. There is no consensus definition of what constitutes a software patent, and current law is ill equipped to classify inventions based on technology. What source code to disclose is likewise unclear, particularly for inventions that improve upon large, pre-existing applications. Perhaps most troubling, no matter what standard we choose, examiners are likely to be unable to meaningfully enforce it. Checking source code by hand is beyond the capabilities of the USPTO and testing that disclosed code runs is infeasible for practical and technical reasons. Because a source code requirement unenforceable by patent examiners is likely to be completely ineffective, the inability of examiners to test disclosed source code perhaps constitutes a dispositive strike against requiring source code in patents.

## Conclusion

Unfortunately, a source code disclosure requirement for software patents appears both ineffective and infeasible. Current law does not support imposing such a high disclosure burden on software patents, and accepted doctrines permit invention descriptions with substantially less detail than full source code. Time and again,

courts have refused to demand that software patents as a class include source code, most recently and definitively in *Fonar* (see Section 2.3). Indeed, the USPTO and federal regulations (see Section 3) discourage the submission of source code in patent applications by relegating source code of any significant length to unpublished electronic appendices. This regulation is perhaps indicative of the USPTO's low opinion of source code's utility as a means for describing software inventions. Given the accepted understanding of the disclosure requirement, then, a source code disclosure rule would require new law and a change in patent office policy, both high hurdles for any reform to clear.

There is much evidence to suggest that source code disclosure would be largely unhelpful in reforming software patents. As our examination of the GIF patent demonstrated (see Part II), source code is often difficult to understand and rarely reveals any information a skilled programmer could not glean from written descriptions, diagrams, or pseudocode. Source code for larger software inventions, or code that relied on proprietary hardware and libraries, would be even less intelligible than the FORTRAN implementation of the LZW algorithm presented in the GIF patent. Moreover, underdisclosure is not a problem endemic to software patents. Patents on mechanical inventions generally omit many details from their specifications, leaving readers to rely on experimentation, experience, and intuition to implement the devices described. Although software patents may not achieve optimal disclosure in all cases, the ease of precisely specifying software inventions means that software patents are likely overdisclosed relative to their mechanical peers. In this light, a source code disclosure requirement seems misdirected.

Yet more problems lie in formulating a source code disclosure requirement with enough exactness to avoid the sorts of ambiguity that give rise to costly litigation. Neither which patents should be subject to such a regulation nor what code would be required can be well defined. Because patent law has tried to stay technology neutral so far, current doctrines are not equipped to make the fine distinctions often necessary to distinguish between software patents and business method patents. Requiring any invention that involves software in any way, however peripherally, to provide source code would be too broad, but a better objective definition of a software patent is hard to come by. Even if we could identify software patents as such, many software patents do not reduce easily to a reasonably sized, discrete chunk of source code, instead being dispersed across thousands or millions of lines of otherwise irrelevant code.

Despite its initial attractiveness, then, a source code requirement for software patents seems both ineffective and impractical. Current law correctly understands that source code often adds little to other forms of disclosure, and requiring code in addition would burden inventors and examiners alike. Reforming software patents therefore requires looking elsewhere.

## References

- [1] 35 U.S.C. §112.
- [2] 37 Code of Federal Regulations 1.52.
- [3] 37 Code of Federal Regulations 1.96.
- [4] About LAME. Available at <http://lame.sourceforge.net/about.php>.
- [5] Code of Federal Regulations (CFR): Main Page. Available at <http://www.gpoaccess.gov/CFR/>.
- [6] *Fonar Corp. v. GE*. 107 F.3d 1543; 1997 U.S. App. LEXIS 3402; 41 U.S.P.Q.2D (BNA) 1801; February 25, 1997, DECIDED; Rehearing Denied and In Banc Suggestion Declined May 8, 1997, Reported at: 1997 U.S. App. LEXIS 11727. Certiorari Denied October 6, 1997, Reported at: 1997 U.S. LEXIS 6020.
- [7] *In re Application of Sherwood*. 613 F.2d 809 ; 1980 CCPA LEXIS 298; 204 U.S.P.Q. (BNA) 537; January 10, 1980, Decided.
- [8] *In re Hayes Microcomputer Products, Inc. Patent Litigation*. 982 F.2d 1527 ; 1992 U.S. App. LEXIS 33579; 25 U.S.P.Q.2D (BNA) 1241; 93 Daily Journal DAR 1100; 93 Daily Journal DAR 1140; December 23, 1992, Decided.
- [9] Manual of Patent Examining Procedure, §2161.01. Available at [http://www.uspto.gov/web/offices/pac/mpep/documents/2100\\_2161\\_01.htm](http://www.uspto.gov/web/offices/pac/mpep/documents/2100_2161_01.htm).
- [10] The Disclosure of Source Code in Software Patents: Should Software Patents Be Open Source?
- [11] Unisys: LZW Patent and Software Information. Available at [http://www.unisys.com/about\\_\\_unisys/lzw/lzw\\_\\_license\\_\\_english.htm](http://www.unisys.com/about__unisys/lzw/lzw__license__english.htm).
- [12] U.S. Patent 4,558,302. Available at <http://www.google.com/patents?id=ZyoBAAAAEBAJ>.
- [13] U.S. Patent 5,579,430. Available at <http://www.google.com/patents?vid=USPAT5579430>.
- [14] U.S. Patent 5,960,411. Available at <http://www.google.com/patents?vid=USPAT5960411>.
- [15] U.S. Patent 6,785,865. Available at <http://www.google.com/patents?vid=USPAT6785865>.
- [16] John R. Allison and Ronald J. Mann. *The Disputed Quality of Software Patents*. *SSRN eLibrary*, 2007. Available at <http://ssrn.com/paper=970083>.

- [17] James E. Bessen and Robert M. Hunt. An Empirical Look at Software Patents. *SSRN eLibrary*, 2004. Available at <http://ssrn.com/paper=461701>.
- [18] Martin Campbell-Kelly and Patrick Valduriez. A Technical Critique of Fifty Software Patents. *SSRN eLibrary*, 2005. Available at <http://ssrn.com/paper=650921>.
- [19] Benjamin Hershkowitz. What Are My Chances? From Idea Through Litigation. *Kenyon & Kenyon LLP*. Available at <http://library.findlaw.com/2003/Oct/16/133092.html>.
- [20] Kenneth A. Liebman. One-Click or Two? The War Over Business Method Patents. *Faegre & Benson LLP*. Available at <http://library.findlaw.com/2000/Oct/1/126528.html>.
- [21] Gerard N. Magliocca. Blackberries and Barnyards: Patent Trolls and the Perils of Innovation. *Notre Dame Law Review*, June 2007. Available at <http://ssrn.com/paper=921252>.
- [22] Greg Roelofs. History of the Portable Network Graphics (PNG) Format. *Linux Journal*, April 1997. Available at <http://www.libpng.org/pub/png/pnghist.html>.
- [23] Pierre Sarrazin. Unisys/compuserve gif controversy. Available at <http://progfree.org/Patents/Gif/Gif.html>.
- [24] Richard Stallman. Fighting Software Patents – Singly and Together. Available at <http://www.gnu.org/philosophy/fighting-software-patents.html>.
- [25] Richard Stallman. The Anatomy of a Trivial Patent. Available at <http://www.gnu.org/philosophy/trivial-patent.html>.
- [26] Richard Stallman. Letter from Richard Stallman to Tim O’Reilly, 11 March 2000. Available at <http://www.gnu.org/philosophy/amazon-rms-tim.html>.
- [27] Richard Stallman. Patent Absurdity. *The Guardian*, 23 June 2005. Available at <http://www.guardian.co.uk/technology/2005/jun/23/onlinesupplement.insideit>.
- [28] Robert E. Thomas. Debugging Software Patents: Increasing Innovation and Reducing Uncertainty in the Judicial Reform of Software Patent Law. *Santa Clara Computer and High Technology Law Journal*, *Forthcoming*. Available at <http://ssrn.com/paper=1126450>.
- [29] T.A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, June 1984.

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 License.

For the full text of the license, please visit <http://creativecommons.org/licenses/by-nc-sa/3.0/us/>.

