THE FEDERAL COURTS LAW REVIEW

Volume 1 2006

Discovery in Computer Software Patent Litigation

Andy Johnson-Laird*

* Mr. Johnson-Laird is a Forensic Software Analyst with Johnson-Laird Inc. He can be reached at the following address: 850 NW Summit Avenue, Portland, OR 97210; telephone number: (503) 274-0784; fax number: (503) 274-0512; email address: andy@jli.com; or website: http://www.jli.com/. Portions of this paper were first published in the August 1994 issue of *The Computer Lawyer* in the article, *Smoking Guns and Spinning Disks.* The author gratefully acknowledges the assistance of G. Gervaise Davis III Esq. with the legal topics addressed in this paper. This article was published electronically in March 1998.

I. Rules Applicable to Computer-Based Evidence

A. Rule 34 of Federal Rules of Civil Procedure – Production of Documents

Federal Rule of Civil Procedure 34(a) states:

> (a) **Scope**. Any party may serve on any other party a request (1) to produce and permit the party making the request, or someone acting on his behalf, to inspect and copy, any designated documents (including writings, drawings, graphs, charts, photographs, phono-records, and other data compilations from which information can be obtained, translated, if necessary, by the respondent through detection devices into reasonably usable form), or to inspect and copy, test, or sample any tangible things which constitute or contain matters within the scope of Rule 26(b) and which are in the possession, custody or control of the party upon whom the request is served; or (2) to permit entry upon designated land or other property in the possession or control of the party upon whom the request is served for the purpose of inspection and measuring, surveying, photographing, testing, or sampling the property or any designated object or operation thereon, within the scope of Rule 26(b).

Writing in *Federal Practice and Procedure*, Wright and Miller take the first step in applying Rule 34(a) to computerized information:

> One seemingly small but actually quite important change made in the 1970 amendments of Rule 34 brought the federal rules, in some ways for the first time, into the computer age. The original rule had spoken only of documents and tangible things. Today much information of importance in litigation is no longer on documents stored in files but is on punched data cards, electronic disks or tapes, or otherwise stored in a computer. Plainly this information must be subject to discovery and the 1970 amendment clearly provided for ediscovery.[1]

Wright and Miller reason that Rule 34(a)'s reference to "other data compilations from which information can be obtained, translated, if necessary, by the respondent through detection devices into reasonably usable form," makes it clear that the requesting party is entitled to have the computerized data presented in a form such as a hard copy printout. However, in computer software patent litigation, that would be inadvisable, since a printout often lacks significant additional information only available by forensic examination of the computer-based version of the document.

1. Charles A. Wright & Arthur R. Miller, Federal Practice and Procedure § 2218 (2d ed. 1994).

B. *Rule 106 of Federal Rules of Evidence – Related Writings*

Federal Rule of Evidence 106 states: "When a writing or recorded statement or part thereof is introduced by a party, an adverse party may require him at that time to introduce any other part of any other writing or recorded statement which ought in fairness to be considered contemporaneously with it."

Wright and Miller comment:

> Computerized materials present difficulties of analysis. The use of a document prepared by a computer involves a writing and should be treated in the same fashion as other documents under the Rule. The problem arises where a party asserts that the data stored within the computer constitutes the "writing" and the print-out is an incomplete part. Or the opponent may assert that the print-out requires the use of the program to be correctly understood. This Rule offers no guidance on such questions.[2]

The proper treatment of photographs and motion pictures under Rule 106 is equally obscure. But the fact that Article X of the Rules, unlike the state codifications, treats motion pictures and photographs as a distinct category, rather than a species of "writing," might lead to an inference that writing is to be given the same restricted meaning for purposes of Rule 106.

Since a picture is worth a thousand words, to a liar as well as to an honest man, there would seem to be an even greater danger of misleading photographs than of documents taken out of context. It is possible to crop photographs and edit motion pictures to alter their significance.[3]

To some extent these dangers can be avoided by manipulation of the "best evidence" rule, but even though photographs are "writings" for purposes of that rule, there are many uses of photos to which the "best evidence" rule does not apply. It can also be argued that the same policy reasons that justify treating photographs as "writings" for purposes of the "best evidence" rule are applicable to the same issue in connection with the completeness doctrine. Moreover, photographs and motion pictures do not raise the administrative problems thought to justify excluding conversations from Rule 106.[4]

---

2. 21A CHARLES A. WRIGHT & ARTHUR R. MILLER, FEDERAL PRACTICE AND PROCEDURE § 5073 (2d ed. 1977).

3. This was written before the advent of digital photography and digital motion pictures, the advent of which makes a complete mockery out of concepts that 'the camera never lies.' Today, the photograph and videotape can lie as easily as the redacted or out of context document.

4. 21A WRIGHT & MILLER, *supra* note 2, § 3.

With the arrival of audio-visual computing, often referred to as "multimedia," even small personal computers can store textual documents, digitized photographs, motion pictures, and high-fidelity sound. This digitized information can be manipulated easily and in a way that, in the future, will doubtless cause much consternation to attorneys and judges alike as they struggle with the application of Rules 34 and 106 to computer-based information. When digitized information is involved, there is actually no one original document. There are few technical means to determine the authenticity of a given document, or to detect whether someone has tampered with a digitized photograph, video recording, or audio recording.

II. Computer Software and Design and Production

Attorneys who have a good mental model of the nature of computer software, what it is, and how it is produced, are far more effective in motivating an opposing party's counsel to produce the relevant software in a form in which it can be analyzed cost effectively. Therefore, it is appropriate to begin a discussion about the discovery of computer software with an introductory overview of how we design and produce software. This overview helps to answer the question: What information exists that *can* be discovered? What follows is, of necessity, a brief introduction to the world of computer software design and production.[5]

A. Source Code

To visualize computer source code requires little more than a simple thought experiment which one has to commit to writing. The essential problem is to translate a real-world act into a simple series of steps and then write those steps down in the proper order.

As an example, imagine sending an email message or a letter to a child telling him or her how to calculate a 15% tip on a restaurant bill. The age of the child will determine the level of abstraction at which one must write the instructions. A younger child will require more detailed instructions, perhaps at the level of specifying which buttons to press; for an older child a simple instruction of "enter the total" will suffice. We can tell an even older child merely to "multiply 0.15 times total, and add the result to the total."

5. A more detailed introduction to the basic concepts of computer software is available on the World Wide Web at http://www.jli.com/papers.htm.

Now imagine a more complicated set of calculations: calculating a consultant's billing based upon hours worked (converting to decimal hours), client-specific billing rates and so on. At some point the complexity and scope of the problem to be solved exceed one's ability to hold the entire problem in one's head, and it becomes easier to create a written *specification* of the problem to be solved. Real-world programming has many of the characteristics described above. Typically a written specification will describe the problem to be solved and its solution at a high level of abstraction; the programmer then will translate this into a more detailed level of abstraction as he or she writes the human readable form of the code, the *source code.* This source code is then *compiled* using a *compiler* to translate the source code into *object code* to form an *executable program* that can be run on the computer.

Source code can be written in a so-called high-level language such as "C."[6] C is one of the more popular computer programming languages. C has the advantage, to programmers at least, of shielding the programmer from having to understand the technical aspects of the computer system upon which the final program will run. The C language does this by creating a world in which the programmer can operate in ignorance of technical details. The secret is in the translation program, the compiler, which generates the correct code for the computer.

For the previous restaurant example, the program would need to declare a *variable*, a working storage location akin to a note pad upon which temporary results are jotted for the bill's total. The next step would be to set the contents of this variable to its initial value of zero (a step called initializing the variable). In C, this might appear as:[7]

```
float bill_total;
bill_total = 0.0;
```

This fragment of C source code illustrates several aspects of program source code. The word "float" indicates to the C compiler that it should set aside an area of memory that will contain a *floating point number.* A floating point number is expressed in the form 123.456 x

6. The folklore is that the language "C" was derived from a language called "B." B had been developed at Bell Labs, and may have been named for Bell Labs. Do not confuse a high-level language with a high level of abstraction. A high-level language is a comparative term. Early computer languages were so low-level that they (and many of their proponents) dragged their intellectual knuckles on the ground.

7. Experienced programmers will be quick to point out that this fragment lacks the appropriate pre- and post-amble to be valid C; however, the intent is to communicate the flavor of C, not to demonstrate syntax.

10^n, where "n" is chosen to represent the number with appropriate accuracy, ranging from very small numbers to very large numbers. One important point to note is that the programmer can, like a brattish child, simply state "I want that!" and let the C compiler take care of the details. A secondary point is that "float bill_total;" does not cause any *executable* instructions to be created in the resulting program—it merely requests that the compiler set aside some memory for storing this value and associates the name "bill_total" with this location in memory.

The name "bill_total" is created from whole cloth. If the programmer complies with the grammatical rules of the C language, which are specified in excruciating detail in reference books, he or she may call this variable *anything,* for example, "x," "Wayne_Newton," or "This_Is_Where_I_Want_To_Put_The_Bill_Total." The underscore is used to comply with C's grammar, which states that *symbolic variable names* such as these will be terminated by a space character (among other things). The underscore therefore serves to break up a long name without terminating it. Programmers learn (the hard way) that meaningful names that *hint* at the purpose of the data stored therein are much better than those that either give no information or tend to mislead. Programmers usually abbreviate symbolic variable names and there are different styles of *naming conventions*: "bt," "total," "tot_bill," "sum," "tot_no_tip" would all be valid names. These naming conventions are part of an individual programmer's *coding style.*

The semicolon shows the C compiler that the programmer has made a complete utterance, similar to the use of the period in English. The period cannot be used in C for this purpose, because it represents the decimal point.

The second line, "bill_total = 0.0"; *assigns* a value of 0.0 to the area in memory (assigned by the compiler) to correspond to "bill_total." To mathematicians this is a curious use of the equal sign; in contrast to mathematical notation, this is not an assertion that "bill_total" *is* equal to 0.0. Instead, it states that "bill_total" *will become equal to* 0.0. In the mind of the programmer, the value on the right of the equal sign is copied into the symbolic variable name on the left.

B. Comments

An important part of source code, though not used by the computer itself, is the *comments* added by the programmer. These are tex-

tual annotations intermixed with the actual programming statements that are discarded during the *compilation* of C code into object code.

In C, comments are delineated by two special character sequences, /* and */, which serve as parentheses. For example, here are the previous two lines of code with comments added:

```
/*
This is an example of a coment
Block comments typically precede the code they
describe and consist of several contiguous lines of comments like
this.
*/
float bill_total; /* Declare variable for the total */
bill_total = 0.0; /* Initialized variable to zero */
```

Note the deliberate spelling error of the word "coment." It is not at all unusual to see spelling errors in source code. In comments they do not matter, and in the active source code, providing the programmer is consistent, they have no effect. These spelling errors, and the actual writing style that programmers use in comments, can give good clues to authorship. Even the style of *block comments* at the top of the example above gives stylistic hints. The physical formatting of the comment lines, their indentations, and their spacing may provide clues to software authorship.

C. *Symbolic Constants*

In most of the examples of source code thus far, when a numeric value has been required it has been written in the actual code. This is a very bad practice. It is much harder to maintain programs because the same number may be scattered throughout a larger program, but not all of the instances where it occurs might have the same meaning. Any attempt to change this number will almost certainly create a bug of major proportions.

Therefore, most programming languages support the use of *symbolic constants*. For example, in C, one could write:

```
#define MAX_ITEMS 20 /* The maximum number of items */
:float item[MAX_ITEMS]; /* Declare array of floating point num-
bers */
:
while (count < MAX_ITEMS)
. . . and so on. . .
```

The C compiler automatically substitutes the value 20 from the "#define" line in all instances where "MAX_ITEMS" appears. This means that only a single line of code (the "#define" line) need be changed to alter the total number of items.

D. Object Code

Object code is the product of compiling source code and linking it with other object code to form an *executable binary* version of the program. Object code is lacking much of the information in the original source code:

> a) All of the comments have been removed during the compilation process.
> b) All of the symbolic variable names and symbolic constants have been replaced by memory locations (as numbers) and the constant values. The one exception is symbolic names of certain parts of the computer program. The linker (the program that links the object code) needs these as it connects discrete chunks of object code together to form the finished program.
> c) The symbolic names are normally removed from *production object code* sent out into the market. There is almost no information of high-level abstraction in the resulting executable binaries. That factor becomes very relevant when considering the topic of *reverse engineering* software (studying the object code to learn how it works). The only immediately comprehensible information in the object code is the *embedded character strings* that are created when the programmer needs to reference an error message or to display information on the screen.

One can see these embedded fragments of text if one examines object code using specific programming utility programs. These embedded fragments of text have a certain forensic value: for example, they can be used to relate certain object code back to the source code from which it may have been compiled. However, unless the embedded fragments contain unusually bizarre text, this is a rather approximate method that can be misleading. It is akin to recognizing people by the shadow they cast.

E. Software Production Phases

Discovery of computer software is greatly aided if one understands the phases in which it is produced. Knowing *how* it is produced leads to understanding *what* work product will be created, and what information and documentation may exist and be discoverable.

There are several very discrete phases in the journey from idea to retail shelf. Some phases may be omitted depending upon the task, the

experience levels of the programmers, and the size (and budget) of the company doing the work. The phases described below have significant overlap: writing the source code often starts before all of the programs have been specified, and testing starts before all of the program's source code has been written. Software development has been described as a process of *step-wise refinement.*

F. Specification Phase

The initial stage is to create documents describing the intended product. Various types of specifications can be produced consistent with the practices used by each development company. Usually each successive specification is more detailed in the level of technical detail it includes. Specifications include:

a) Marketing Requirements: The perceived requirements of the marketplace as to the particular features a new product should have to be desirable to users, particular performance requirements such as speed and upon what computers it should run, and so on.

b) Design Requirements: The specific design requirements for the program at a more detailed level. For example, it might describe that the product must be written in C, using particular supporting third-party object code libraries. These would not be matters of concern to the marketing of the program.

c) Functional Specification: A description of what a program or group of programs must do. At this level, overall functionality is apportioned to specific programs with a description of how these programs will interoperate and exchange information.

d) Detailed Design: A more detailed description of the input information, processing, and output information that will be produced by each program in the system. This level of specification is intended to guide the programmer who will create the source code. This level therefore usually includes descriptions of data structures, the major functionality within each program, algorithms to be used to compute results, and other detailed programming notes. Where necessary, algorithms will be described using logic flow diagrams (that show the decisions and processing to be performed depending on the outcomes of those decisions) or pseudo-code (source code written in part English and part programming language so that the underlying algorithm is visible to the programmer even though, as source code, it is not grammatically correct).

e) Test Plan: Describes how the program(s) will be tested before being made available on the market. Often this involves both general testing strategies and specific *scripts* that will guide someone through the specific interaction with the program. Testing is a particularly thorny issue, yet test plans are quite rare in the software industry. Testing can never confirm an absence of errors; testing can only reveal the presence of errors.

G. Implementation, or Coding, Phase

This is the phase where programmers systematically convert the detailed design specifications into working programs. A modern approach often used by smaller software development companies where, for time and cost reasons, the detailed design specification may not be produced, is to perform *rapid application development* (RAD). In RAD, working prototypes of programs are created to serve as the specification. Prototypes have an advantage in exposing users to real, but minimally functional, programs early, and in providing reaction to the design and functionality early enough to affect the design of the finished product.

H. Initial Testing

The coding phases consist of creating the human readable source code, converting it into object code, and linking object code modules to form executable programs to be tested using specimen data files. This testing of individual programs is often called *unit testing.*

Any mistakes found in testing must be investigated (sometimes, just getting the mistake to happen again is the hardest part). Once the programmer understands what went wrong, the source code must be changed and the modified program must be re-tested to ensure that the change corrects the original problem but does not introduce any additional errors.

I. Revision Control

Source code for a given program undergoes many changes from the very first version created to the final version. More complex programs require several programmers working collaboratively on the source code, and this creates an additional problem of coordinating all of the changes within the programming team.

Specialized *revision control programs* are used to manage and coordinate these changes by automating the job of a source code librarian. Revision control programs record every change made by all programmers to all source code files as the source code evolves. The most popular of these are Microsoft's SourceSafe, the Polytron Version Control System (PVCS), the Unix standards Source Code Control System (SCCS), and Revision Control System (RCS). Revision control programs work by creating *cumulative history files* for every source code file under their management. These history files contain details of

every version of every source code file that has ever existed from the time the source code was first put under version control.

J. Production Documentation

There are two general classes of documentation that are created as part of software production:

a) User documentation to tell users how to install the software, how to set it up for production use, and how to use all the features of the software.
b) Maintenance documentation to tell the software developer's own personnel how the program works so the developer can make changes to the program. In some cases, this documentation is embedded in the source code itself in the form of comments.

Documentation can also be managed by revision control systems or document management systems similar to those used in law firms.

K. Testing Phase

As more and more of the programs in a system become ready, they can be tested in combination to ensure that the output of one program can be accepted correctly by the next. The process is called *string testing* (the programs are "strung together"). String testing usually requires more complex test data files and more complex testing scripts. String testing is usually followed by *system tests,* in which all of the programs in the system are run as though in the finished product.

Once all of the programs are believed to be working, they will be *stress tested.* This is a process whereby the system is pushed to its limits. Huge data files are created and manipulated, and massive numbers of different operations are performed. The intent is to flush out any mistakes that may be revealed only when the programs have to perform many operations on large amounts of data.

When programs are deemed to be ready, they will be released to a small number of trusted clients for *alpha testing.* This is the first testing by actual users in the marketplace and usually generates many reports of problems that slipped through the in-house testing.

Once all relevant errors are corrected, the revised programs are released to selected users for *beta-testing*, and the cycle of testing, problem reporting, and correcting the programs is repeated. Some organizations even resort to *gamma-testing* to ensure more reliable software.

L. Maintenance Phase

There are two types of maintenance: *adaptive maintenance* and *corrective maintenance*. Taken together over the useful life of the programs, adaptive maintenance and corrective maintenance may cost tens or hundreds of times more than the original development cost.

M. Computer Programs in Firmware

When a computer program is stored directly in a computer chip so that it remains there even when the power to the chip is switched off, then it is called *firmware* rather than software. Firmware exists in every modern computer. It is used to perform the initial self-test of the computer when it is first switched on, and it is also responsible for loading the operating system *software* that will then take overall supervisory control of the computer until it is switched off. Firmware exists in every device that claims to be microcomputer controlled, from wristwatches to electric toasters, microwave ovens, cellular phones, and automobiles.

Firmware differs from software only in its embodiment. All of the above narrative regarding the production of computer software also applies to the production of firmware. Firmware, in the context of litigation, differs from software only in that its analysis is likely to require extraction of the actual zeroes and ones from the physical chip in which it is contained.

III. What Might Exist to be Discovered?

Creating a computer program leaves a broad wake of evidence as the development proceeds. During discovery, it is important to be aware of what relevant evidence might exist, and the form in which each type of evidence might be embodied.

A. Design and Specification Documentation

It is perhaps true that the larger the organization creating computer software, the greater the amount of design and specification documentation that will be created. Larger companies are more prone to regimentation and standardization, and this tends to spawn more formal approaches. A one person programming shop may be a much more ad hoc affair. Countering this thesis is the fact that programmer education is improving, the need for documentation is becoming more widely known, and there is greater pressure to produce it. There is, therefore,

no predictability as to the amount, quality, or type of documentation that may exist for a given project.

Most formal design and specification documentation will exist as some kind of computer-based evidence, and it will usually exist in multiple versions in multiple places. It is not unusual to find that what would, in an earlier era, have been handwritten marginalia scribbled on specification documents has been replaced by word processing documents containing embedded annotations, or even digitized audio.

There is an increasing trend to create graphical representations of computer software, be they computer-based versions of flowcharts or more modern dataflow diagrams or other representations of the required program logic. All these representations serve the same purpose: translation of thought into the sequential steps to be performed by a computer program.

B. Source Code Documentation

Usually source code exists as hundreds, sometimes thousands, of computer files containing text. Certain programming regimes,[8] such as Microsoft's Visual Basic, keep the source code in a quasi-textual form in association with other computer data that prevents the source code from being read directly. Whatever the specific form, the most important single fact about source code discovery is that a printed listing of the source code is demonstrably a redacted version of the information available in the computer-based source code.

Apart from the redaction that occurs when source code is printed out, a printed listing is a maximally inconvenient form in which to receive source code. One of the most frequent analyses of source code is to search for the place where a particular symbolic variable is defined or set to some value. A search of a million text lines of computer source code will take a few seconds if that source code is on a computer's hard disk, but it will take several hours if that source code is printed out. The risk of error is far higher with analysis of the printed source code than with analysis of a source code on a computer's hard disk.

There are also specialized source code analysis tools that, provided one has the source code on computer media, can build *calling trees* that

8. Formatting regimes are self-contained source code development environments. As an analogy, WordPerfect would be considered a self-contained word processing environment.

show how one part of the program invokes another part, which in turn invokes another. The calling tree is like a street map. Special text editing programs can be used to move within the source code rapidly, following cross-references to other parts of the program. None of these tools are available if all one has is the source code on paper.

Furthermore, if the source code has been placed under the aegis of a revision control system, there is a wealth of forensic information contained within the computer-based form of the source code that is almost entirely absent from the printed version. The revision control system's cumulative history file may reveal which programmers changed which lines of source code, the specific changes, the date and time each change was made, and additional annotations about why the changes were made. Revision control software acts as a time machine, winding the clock back so that one can see earlier versions of the computer software.

C. Header Files

One often overlooked aspect of source code is the so-called *header* or *copybook* files. (The name used varies according to the programming language in question; header file is a term in C and C++, copybook is a term used in COBOL). These are separate source code files that contain definitional material. Rather than being repeated in hundreds of different source code files, the definitional information is merely included during compilation from a single file.

Header/copybook files are often stored in separate directories on the computer system. When source code is gathered for production in litigation, header/copybook files are easy to overlook. However, because of their definitional content, they are vital to understanding a program. For example, one might find a source code file containing the lines:

```
if (status == ALLDONE) exit.
```

Absent knowledge of the significance of the symbol "ALLDONE" (and knowledge that the use of capital letters is a convention to imply that this is a symbolic constant defined elsewhere), this code is meaningless. However, earlier in the source code file, one might find a source code line that reads:

```
#include </projectX/includes/STATDEFS.H>.
```

This is an instruction to the compiler that, at compile time, it is to read in the entire source code contained in file STATDEFS.H that can be

found in the "includes" directory in the "projectX" subdirectory. On examination of this included file, STATDEFS.X, one might find the source code line:

#define ALLDONE 43.

Armed with this knowledge one can determine that the original source code line, in effect, reads, by substituting the value of 43 in place of the symbol ALLDONE:

if (status == 43) exit;

in this example, "STATDEFS.X" is a header/copybook file.

D. Object Code

Certain computer languages do not result in object code being created for a program; these *interpreted* languages control a computer by means of an *interpreter* reading in each line of the computer program, and "executing" each line. In this sense, WordPerfect "interprets" the embedded codes in a document, modifying the appearance of the text in response to the "instructions" given by these codes. Microsoft Word, and all of the current Microsoft Office products, are built upon Microsoft's Visual Basic for Applications. They all "interpret" instructions in the documents they create and manage.

Despite interpretive languages such as Basic and Lisp (which is used in association with the computer-aided design program, AutoCAD), compiled languages do require object code with which to control the behavior of the computer.[9] As each source code file is compiled, a corresponding object code file is created. These object code files must then be *linked* together, along with any previously compiled object code, to form the complete executable form of the program.

During the process of compilation, all symbolic variable names and source code comments are discarded. However, the names of particular functions (a word used to describe specific *subroutines* within a program) often survive in the object code as a byproduct of the linking process and, as such, can provide some forensic insights into the origins of the object code. By matching the symbolic names found in the object code back to the symbolic variable names declared in the source

9. A compiled language must be translated into object code before it can control a computer. An interpretive language requires a special language interpreter (such as a BASIC interpreter) that is capable of examining each source code statement and causing the appropriate action to take place.

code, it may be possible to establish a link, and thereby to establish an association between source code and object code.

E. Firmware

If the object code to prior versions of the firmware program no longer exists, it may be possible to discover prior versions of the firmware "burned" into hardware chips. It is usual for these chips to be kept by a firmware developer either for technical support reasons (an earlier version can be tested to see if it exhibits a reported problem), or because certain chips, once burned, cannot be reused, and the technician keeps them rather than discarding them. (The practice of *dumpster diving*—the searching of dumpsters outside offices—has been known to reveal developmental copies of a company's firmware in discarded chips.)

F. "Make" Files

Even moderate-sized programs can become quite complex because of the number of different source code files that must be compiled into object code. It would be grossly inefficient and time-consuming if a single change to one source code file were to require that all source code files be recompiled. Therefore, it is usual to use another specialized program to determine which source code files must be recompiled and to automate the entire process of compilation and linking. Recompilation can be quite complex if an "include" file is changed because all source code files that contain this file must be recompiled. This system, first popularized under the Unix operating system, is called "make." The files that define the interdependencies between the source code files, and contain directives to control the compilation and linking, are called *makefiles* (usually written as a single word).

Unless a makefile is provided, it is almost impossible to rebuild a modern computer program's object code from its underlying source code. There is simply not enough information in the source code from which to infer how the process must be done.

G. Libraries

Many software developers, in the interests of completing a working program sooner, will purchase prefabricated libraries to provide prefabricated solutions for specific tasks. Examples of these include telecommunications libraries, database management libraries, and digi-

tized image manipulation libraries. These libraries are licensed either as source code files that are to be compiled and linked with the developer's own object code, or as precompiled libraries that only require linking.

Given the finished object code for a developer's program, apart from an embedded textual copyright message, it may be impossible to tell that one or more components of the program was created by the inclusion of third party source or object code.

H. Development Documentation

Developing today's software is a complex, time-consuming, and expensive task. Therefore, in all but the smallest development companies, the software development *process* has to be managed carefully. While separate from the technical aspects of the actual software development, documentation of the commercial aspects of software development often includes useful forensic information relating to programmers who worked on the project and the overall schedule under which the software was developed.

Associated with the management of the technical development process is another wake of forensic evidence that will include budgetary estimates, schedules showing milestone dates, and project reporting information revealing development schedules and milestones.

Although the quantity, quality, and nature of design and specification material will vary wildly from one software developer to another, there is usually *some* kind of development documentation. It may range from early prototypical versions of the source code, to handwritten notes in a programmer's notebook, to a series of different versions of specification materials.

Electronic mail is very commonly used in the software development industry. Programmers have been sending email messages to each other since the early 1970's, long before the Internet became known to the public. It is quite common for employees in software development companies, or in software development groups of large companies, to communicate extensively by email among themselves. It is also quite common for software development employees to communicate by email to the marketing and sales departments, and even to end users of the computer software.

Software development is a demanding art. Computers are notorious for doing what programmers ask them to do rather than what programmers meant them to do. The maintenance phase of a computer

program can cost more than ten times the original development cost, as the next generation of programmers struggles to understand what was going on the in the minds of their programming forebears.

To combat these problems, and to reduce the cost of developing and maintaining software, companies usually try to impose some level of uniformity in the thinking and writing of their programmers. This uniformity will be documented as requirements for standardized ways of designing, coding, and testing software. That documentation can be a useful source of understanding of the software itself and of forensic information about the development process.

IV. Unique Characteristics of Computer-Based Evidence

From an evidentiary point of view (and to the delight of forensic software analysts such as this author), computer-based evidence has characteristics that, if not understood and anticipated, may cause a requesting party to fail to discover relevant evidence and a producing party to fail to preserve or produce it. These characteristics are:

a) Even small companies and individuals usually keep a vast amount of information on computers (or make redundant copies for safekeeping).
b) Computer-based evidence is usually stored in a less organized manner than is paper evidence. (It does not matter what sequence documents are stored in, for example, if the computer only takes three seconds to find any one of them.)
c) Documents can be easily stored on the computer in a form that makes it extremely tedious to search for particular documents before production. This is especially true for digitized images, motion pictures and audio recordings.
d) Redundant copies made deliberately for safekeeping (called 'backup copies') are rarely managed appropriately and end up being scattered around an office or at employees' homes.
e) Inadvertent copies of documents are often made during customary computer usage; these are rarely managed appropriately and tend to propagate from one computer to another.
f) Backup copies of documents can be imperfect copies of the originals, sometimes omitting information, and sometimes including more than is required.
g) Electronic networks (both in the office and around the world) make it easier and quicker to transmit several copies of 400 page documents to other computers around the world than to walk to the nearest coffee pot.
h) Electronic mail, because of its informal nature, encourages its authors to write unwise and inappropriate things that they might never say to a recipient directly, and might regret writing in a letter or facsimile.

These characteristics demand that extraordinary steps be taken during the early stages of litigation, perhaps before filing, if relevant

evidence is to be preserved. Ignorance of these characteristics may mean a requesting party will fail to obtain appropriate evidence, and a producing party may inadvertently waive privilege or produce documents in ignorance of their significance to the case.

To optimize its litigation position, a requesting party must:

a) Take early and aggressive action to ensure that computer-based evidence is properly preserved by the opposing party. This may include putting the opposing party on notice to preserve all relevant evidence by making immediate backup copies of computer hard disks and by cessation of reuse of tapes during customary backup cycles. Retaining a competent computer expert to help convince a magistrate judge why such early and forceful action must be taken may be advantageous.

b) Formulate discovery requests that demand the production of all relevant forms of computer-based evidence. The requesting party must insist on a complete examination of relevant computer-based evidence.

c) Retain a competent expert to help sift through the computer-based evidence produced by defendant. An expert can often find 'smoking guns' that the opposing party unwittingly produced.

To optimize its litigation position, a producing party must:

a) Take early action to ensure that the client (probably with the assistance of an expert) *preserves* existing evidence, including, if necessary, disabling any automated procedures on the computer that will destroy such evidence as part of a routine janitorial function on the computer's storage. It is most unusual for commercial backup software to preserve a *complete* backup of a computer's storage media suitable for evidentiary analysis. Backup software only preserves *active* files, and not those recently deleted. It may be a separate question whether deleted files that still exist on hard disks and tapes are discoverable. Some might assert that deleted files are analogous to paper documents that have been discarded.

b) Retain a competent expert to sift through any computer storage media that is to be produced. If one produces a magnetic tape to an opponent, and the magnetic tape contains information protected by privilege or by the attorney work product doctrine, one may thereby waive privilege on entire classes of evidence.

V. Redundant Copies of Computer-Based Evidence

A. Global Distribution of Computer-Based Documents

Litigation attorneys routinely recognize that multiple copies of every conventional business record may exist in many locations and in different embodiments. Similarly, multiple copies of every item of computer-based evidence may exist on some or all of the parties' computers, and possibly on computers under the control of third parties.

It is usual for a modern company to have a personal computer for each employee; more sophisticated companies will network these computers together, with 'server' computers providing a central repository for computer software and documents, accessible from all other computers on the 'local area network' (LAN) or 'wide area network' (WAN). Such networks, augmented by worldwide networks like the Internet, make it inevitable that computer-based evidence will travel widely. In contrast to paper documents, which require a certain amount of effort to distribute, computer-based versions of documents can be propagated around the world almost effortlessly. For example, assuming that this author had access to the source code for Microsoft Windows and the Internet, here is the command that could send hundreds of copies (depending on the number of addressees) of Microsoft's 'crown jewels' around the world in a few seconds:

```
mail mailing-list < msw.src.tar.uu
```

This command would complete its function within five to ten seconds. The command is irrevocable without highly specialized software knowledge, but even that would not stop the command unless remedial action were taken within a few seconds of the command's execution.

Even a sole proprietor with a single personal computer can connect to worldwide networks and can create evidence relevant to litigation on the hard disks of computers many miles away. Modern computer networks render geographic distance irrelevant; evidence created on a computer 5,000 miles away is logically as close as the hard disk inside your own computer. It is therefore vital for both defendants' and plaintiffs' attorneys to consider all possible locations where relevant evidence might be stored and to not limit their consideration merely to the geographical locations where the parties happen to have their offices.

B. Deliberate Backup Copies

Backups are redundant copies of information resident on the computer at the time the backup copy was made. If (or when) the computer hardware fails, a backup copy can be used to recreate the information that otherwise would be lost. Backup copies are best made at frequent and regular intervals. Larger companies typically employ individuals whose responsibility it is to ensure that central computers,

be they local area network servers or central mainframe computers, are backed up on a daily schedule.

Most organizations use a backup *cycle* as a compromise between spending too much time making backups copies and failing to backup all relevant information. An example of a simple, but frequently used, backup cycle illustrates its evidentiary significance:

a) Every Friday a complete backup copy is made of every file on the computer system (except those files that have been deleted or remnants of old files not completely overwritten by new files).

b) Every day other than Friday, only those active files that have been changed are backed up. Each day uses a different set of backup media.

When this backup cycle is used, if the computer system fails on Tuesday, all of the data files can be restored to their most recent state by reloading the computer from the previous Friday's backup, followed by the Saturday, Sunday, and Monday backups.

To give greater disaster-recovery capability, many organizations use additional strategies:

a) Each Friday's backup is written to one of four sets of backup media, using these Friday sets in rotation. If a computer-based file is found to have been corrupted three weeks ago, with a little effort it can be restored by using the appropriate Friday tape.

b) Every fourth Friday, a new set of Friday backup media is used to replace the set that would otherwise be used. The set removed from use can then be stored in a vault for some number of months, or perhaps in perpetuity.

Thus, a complete set of backup media acts as a time machine; restoring a backup copy on to the computer winds the clock back to the moment in time when the backup was made, be that earlier that same day or several months or even years earlier. In patent infringement cases, prior versions of a computer program may be relevant to issues of willfulness. In trade secret cases, prior versions of computer programs may show use of trade secrets. In copyright cases, prior versions of computer programs may be relevant to issues of derivation.

The cost of magnetic media has dropped dramatically, and it is now usually more cost-effective to use different media for each successive backup copy, or at least to have multiple sets of media that are used in rotation. At any instant in time, most organizations have redundant copies of their computer-based pre-litigation evidence for preceding days, weeks, months or years.

C. *Inadvertent Backup Copies*

Preservation of deliberate backup copies is a sound business practice. Even without formal procedures (or their enforcement), many inadvertent backup copies of computer-based evidence are made by individual computer users, either to protect their own data files created by programs such as word processors,[10] or because of giving copies of data files on diskettes to colleagues. In smaller companies where there is no central server, each computer user will be responsible for maintaining his or her own backup copies in case a computer fails. However, it is usual for users in this context to maintain less than complete backup copies. Many users learn the hard way (when their computer fails or when they inadvertently delete a file) the perils of failing to keep adequate backups. Similarly, users learn the peril of overwriting the media used for the previous backup with the current backup; a data file that is corrupt now might not have been corrupt the previous time a backup copy was made, but reusing the media obliterates the last good version of the data file.

Because of private backups, application program backups, or data file sharing, almost every computer user has many floppy diskettes or small tape cartridges (depending on the backup device on their computer) in his or her possession, and those diskettes or cartridges typically contain a hodgepodge of data files. Much computer-based evidence is backed up almost inadvertently, swept along with other data files as they are copied to a backup. It is easier to backup all active files[11] from the hard disk to a tape cartridge than it is to choose individual files. Most users take the path of least resistance and, if they make backups at all, will backup entire disks or directories.

D. *Off-Site Backup Copies*

To guard against dramatic catastrophes such as fire or flood, many companies either encourage or require the storage of backups off-site, away from their offices. This off-site storage might be in the vault of another company specializing in the safekeeping of backup media or could be at the homes of senior management. If a company permits employees to use their home computers outside of office hours, or if it

10. Word processors and other application programs usually make a backup copy of whatever file is currently being created or modified. In the event of a power failure, or inadvertent modifications, the user can revert to the original data file.

11. Commercial backup software rarely, if ever, has the capability to backup 'deleted files' or other remnants of prior documents.

employs telecommuters who routinely work at home during the day, off-site backups will be created on these home computers as an inevitable by-product. However, these informal or inadvertent backups are unlikely to be as complete as intentional corporate backups.

E. Source Code Escrow

Software companies that license their software are often required by their licensees to deposit computer-based copies of their human readable source code and machine readable object code in escrow to guard against the possibility that a licensee will be unable to continue to use the software if the licensor goes out of business. There are companies that specialize in source code escrow. Escrow copies also act as backup copies. Most escrow agreements call for each successive version of the source and object code to be placed into escrow, but few agreements require earlier versions to be destroyed or returned to the software company.

F. Source Code Licensees

Often software companies will license their source code to their customers and will deliver this software in computer-based form. This is a more common practice among companies developing software for mainframe computers than among those who produce products for the mass personal computer market. Each source code licensee, in effect, becomes an off-site backup site and usually retains many versions of the vendor's source and object code. Often, the licensor's source or object code melds with the licensee's computer-based information and is swept along with whatever intentional and incidental backups are created.

VI. Backup Copies are Incomplete

The preceding section implies that with adequate backups, computer-based evidence can be adequately preserved. While this is true from a typical computer user's point of view, it falls very short from the forensic software analyst's point of view. Most commercial backup software makes backup copies only of active documents and data files on the computer's hard disks, and excludes deleted files and remnants of old files partially overwritten by new ones. There is usually no purpose to be served by preserving this excluded information, so there is no motivation for the backup software vendors to provide this capabil-

ity. Most typical backup or copy programs can be irrefutably shown *not* to preserve all of the computer-based evidence that is available on the original storage device. Fortunately, there are now special backup programs that preserve an entire image of a hard disk on tape, and special hard disk duplication programs that make a complete copy of one hard disk onto another.

The knowledge of the inadequacies of common commercial backup software provides a strong impetus for a requesting party to demand access to the producing party's actual computer systems to inspect or preserve (using special 'mirror image' backup software) the original disks, diskettes, and tapes. A party who, using a commercial backup program, knowingly makes an incomplete backup copy for production to the adverse party, might expose itself to Rule 34 sanctions for destruction of (or failure to preserve) computer-based evidence.

A. Magnetic Tapes

Magnetic tapes of various sizes are often used for backup copies, as they can store a large amount of data in a relatively small space. Like home audio and video tapes, information is written from the beginning of the magnetic tape sequentially toward the end of the tape. Like home audio and video tapes, computer tapes are usually reused when the data recorded on them becomes outdated.

The key evidentiary question is whether any prior residual information that has not been overwritten by new information continues to exist. This is analogous to a two-hour video tape that originally contained two hours of *Gone With The Wind*, but was reused to record a one hour episode of *Masterpiece Theater*. Does the second hour of *Gone With The Wind* still exist on the tape? From the technical point of view, residual information on a tape that has not been overwritten continues to exist. It takes very little effort to access this residual information "off the end" of the active information.

Though this could be key evidence, in this author's experience, standard commercial tape duplicating software fails to copy "off the end" information, and most counsel are unaware of its possible existence.

B. Computer-Based Evidence Must be Preserved Immediately

It is important to note that remnants of documents and data files residing on computer media have no guaranteed life-span; they could

be overwritten from a few seconds of using the computer, or could remain for months or years. The *only* sure preservative method is to make a complete *image* backup[12] of hard disks before the computer system is used for anything else and to make complete image backups of entire diskettes and tapes before they are reused. In this author's experience, companies do not do this. When litigation is filed, a producing party may make a diligent effort to preserve paper versions of documents but, because of lack of technical knowledge, the producing party is extremely lackadaisical in preserving computer-based evidence.

VII. The Vast Quantity of Computer-Based Evidence

A typical company has many computers, each with one or more hard disk drives. The cost of high-capacity hard disks has dropped dramatically in the last decade,[13] and even portable laptop computers are available with hard disks that can store 3,000,000,000 characters of information (approximately 1,000,000 pages of typewritten material, or a stack of paper 350 feet high).

To appreciate the enormousness of computer-based information, consider a hypothetical small software company, SmallSoft Inc., with fifty employees. Assume forty programmers have computers and that ten senior managers and salespersons also have laptop computers. Further assume that ten employees have home computers they use to tap into the company's computers when they work from home. Finally, assume that the company uses a local area network and so has a server computer acting as a central repository for in-house documents and as a switching yard for electronic mail messages. SmallSoft Inc., has a total of sixty computers used by individuals, and a central server computer.

Individual computers, and even laptop computers, are likely to have hard disks of two or three gigabytes or larger. Central network computers usually have at least five or ten gigabytes or more of disk storage. The total disk storage space of SmallSoft Inc. is likely to be, at a minimum, 130 gigabytes of information. That is the equivalent of 4,300,000 pages of typewritten material. Of course, this is the maximum and assumes 100% of all disks are used for data storage. Add to this the number of diskettes and tape cartridges that might be used for

12. "Image backup" is a term of art that describes the entire contents of a given hard disk. An image backup includes the data contained in previously deleted files, unless the data has been completely obliterated by new files which have overwritten the existing data.

13. The current retail cost of hard disk storage is about $150 per gigabyte (that is, per 1,000,000,000 characters of storage).

backup copies and the equivalent of typewritten pages is likely to double or triple. If SmallSoft Inc. enforces rigorous backup procedures, the total storage may even quadruple, reaching a stunning 17,000,000 pages of typewritten information.

In paper form, these vast volumes of evidence would require hundreds of storage boxes, but today's technology permits five gigabytes of information to be stored on an 8-mm camcorder tape. A handful of these tapes could easily store 25 gigabytes of information. New devices will store up to 25 gigabytes on a single camcorder cassette; all of SmallSoft Inc.'s evidence could be held in a jacket pocket. Absent adequate computer security, a disgruntled employee could walk past a security guard with all of SmallSoft's intellectual property, and all computer-based information, in a purse or a shirt-pocket.

VIII. The Disorganized State of Computer-Based Evidence

Although computer diskettes and tapes are readily organized in desk drawers or storage cabinets, and hard disks are tidily hidden inside personal computers, the evidence they contain is rarely organized coherently. Most computer users settle for just getting useful work done, rather than attempting to maintain a neat and tidy hard disk. In fact, many computer users do not even organize their data files into subdirectories to impose an organizational hierarchy on their files. Since the disorganized state of a computer's hard disk is not visible externally, there is little motivation for tidiness.

Individual backup media are rarely catalogued. In testament to the general reliability of modern personal computers, backup copies are rarely used once created. Usually only the most recent backup copy is relevant to disaster recovery. Off-site backups, copies of materials in escrow, and copies sent to licensees are rarely accounted for. If they are, only the most recent material is usually of interest to the computer user. The true chaos of a company's organization of computer-based evidence becomes apparent only when litigation is filed. Both parties are confronted with the challenges of:

a) finding all relevant evidence, and
b) sifting through that evidence to discard irrelevant or privileged information.

For the defendant, it is easy to take a lazy approach and claim that much computer-based evidence was never created, no longer exists, or cannot be found. In this author's experience, some companies assert:

a) They do not keep more than thirty days' worth of electronic mail messages. Centralized backup copies of email for only thirty days is plausible, but it is almost certain that individual computer users preserve their own private copies of email correspondence for far longer than this in either printed form or magnetic media.
b) They do not keep more than one or two prior versions of their software products. This begs the question of whether these companies provide user support for earlier versions of their products. If they do, then it would be reasonable that they keep both the source code and object code for these products. Absent the source and object code, it would be almost impossible for a company to meet its customer support obligations. To fix a problem, or to help a user around it, demands the recreation of the problem to diagnose and correct it.

Taking the lazy way is also extremely risky because a typical company leaves a wide swath of evidence (paper, computer-based or anecdotal) regarding its procedures. One set of backup disks or tapes usually contains telltales that speak of specific policies or procedures, or hints of the existence of, or cross references to, particular documents or versions of source code. The court (and opposing counsel) may be skeptical if an otherwise well-organized company appears to have only recently developed patchy backup procedures.

IX. Drafting Document Requests

This section offers some technical suggestions for crafting precise document requests, although it is worded colloquially rather than in the style one normally finds in such requests.

A document request should demand production of the following information:

a) All relevant computer source code. If the source code is managed by revision control software, the revision control version of the source code must be requested, rather than specific versions of the source code *derived* from the revision control software. The software requested must include all makefiles and other control files used to create the finished object code from source code, along with any source code header or copybook files that are included by reference into other source code files.
b) If the documentation is managed by a document control system (such as many law firms use), the actual document files (and perhaps the document control software, as well) must be requested if all forensic information is to be produced.
c) If third party software products are required to unlock or gain access to the source code, copies of these products must be requested unless the same versions of these products are still currently available on the market.
d) If any information is encrypted or protected by security access software, then the security access software and appropriate decryption keys and passwords must be provided.

The production of third party software, either for revision control, document management, or as software components used by the producing party, is almost always a disputed issue. The producing party usually takes the position that production of this software would be an act of copyright infringement. On the other hand, unless the particular *version* of the software is still currently available, the lack of this software will almost certainly deny the requesting party access to relevant information. In the case of the revision control and document management software, the requesting party will be denied highly relevant forensic information. In the case of the third party products, the requesting party may be unable to access all of the source code produced.

At first blush, a reasonable approach might be to simply purchase current versions of the third party products. However, by the time a dispute's discovery phase has begun, it is more likely than not that any relevant third party product will be unavailable on the market.

The principal issue is whether a producing party is obligated to produce copies of third party materials used in the production of the computer software. This might at first appear to be a clear example of copyright infringement, but there are often extenuating circumstances (beyond the fact that the copying is being done in connection with litigation), the most usual of which is that the third party software used in the creation of the software is no longer available. Unless the producing party makes a copy of this third party material, the requesting party will not have a complete copy of the software.

In certain cases, most notably with Microsoft's Visual Basic products, third party extensions ("VBX") must be present for the Visual Basic program to present the producing party's source code. The third party, VBX, acts as a lock that will block access to the producing party's source code, or at least to that part of it that interoperates with the third party VBX.

A. Embodiment of Documents Produced

A document request should make it clear that, where any information (be it source code, documents or other information exists on computer media) it must be produced as computer media rather than being printed out. The only condition under which a printed document might be acceptable is if the producing party does not have a copy of that document on computer media.

B. Information Necessary to Access Information Produced

To recover and analyze the information produced, one needs to understand some important technical details about the information's original habitat, and details about the particular embodiment in which it has been produced. Concerning the information's original habitat—the environment in which the information was created and managed—one needs to know:

> a) the manufacturer, model, and configuration of the computer system upon which the information was created and managed (for example, a Toshiba Tecra 730 CDT with 64 MB of RAM);
> b) the manufacturer, name, and version number under which the information was created and managed (for example, Windows 95, Release 950A with Service Pack 1 installed); and,
> c) the manufacturer, program name, and version number of the software used to create and manage the information. An example response for source code might be Microsoft Visual C++, version 5, or, for word processing documents, WordPerfect 6.1. The response to this question might bifurcate if the information in question is source code that is maintained under revision control software. Then, besides the details about the source code itself, one also needs to know the manufacturer, program name, and version number of the revision control software itself.

Regarding the embodiment in which the information is being produced, one needs to know:

> a) The manufacturer and model number of the hardware device in which the media was placed to write the information being produced. This is particularly important for magnetic tapes, or for any device other than a standard IBM PC compatible floppy diskette (which is one of the few true standards in the industry). An example response would be a WangDAT 3400DX 4mm tape drive.
> b) The manufacturer, program name, version number, and specific operational parameters used with the software that was employed to write the information to the media being produced. For example, this might be Arcada Backup version 2.0.

This embodiment-related data is particularly critical for minicomputers such as the IBM AS/400, for mainframe computers, or for those running the Unix operating system. Without this data, the information produced will be, for all practical purposes, inaccessible.

C. Ensuring a Complete Production

Though one of the first tasks for a forensic expert is to determine whether a document production is complete, it is quite rare to see a document production request that defines a means by which both par-

ties can know whether all relevant computer software files have been produced.

A complete production of computer source code is easily defined as production of all relevant source code, makefiles, and additional information (the details of which were described above), necessary for the requesting party to access all files from the media upon which they were produced and to recreate the entire executable program from scratch using the source code provided. If the computer software is written in Microsoft Visual Basic, a further condition is required: the requesting party must be able to access the entire source code embedded within all of the Visual Basic files provided (the inability to do this shows that not all third party software components have been provided).

A complete production of object code can be defined as all aspects of the program used on a computer, where the aspects of the program's capabilities are described in an appropriate user manual.

X. Drafting Interrogatories

While source code and its associated design and specification documents usually provide a wealth of forensic information (at least when produced on computer media), there is usually a need to understand specific details of the way in which the software was created. These questions can be addressed through interrogatories.

The three most important areas of information required are usually:

a) Which programmers worked on which parts of the program.
b) By what means the document production requested was compiled. What was searched? How? By whom?
c) Identification of the various versions of the software that were created.

A. Software Authorship

A given source code file is usually developed by a single computer programmer, at least during the initial stages of its creation. Thereafter, many different hands and minds may touch it, changing certain lines of text, inserting others, and deleting others.

If the source code has been placed under revision control, in the ideal case, every change can be related to a specific human author. More often than not, there will be questions of program authorship that are best determined before any depositions if pertinent questions are to be asked of the appropriate deponents.

B. Methods Used for Document Production

Before the computer revolution, a business' records were in view, in filing cabinets or desk drawers, and a physical search was all that was required to comply with a document request. Computers have changed that, providing ultra-compact means for storing millions of pages of printed paper on devices that can fit into a pocket.

Therefore, when a producing party makes a document production involving computer-based evidence and software, a very relevant series of questions needs to be asked. These questions should include how the party prepared the document production, on what computers the party searched, by what means the party searched, what keywords were used in any keyword searches, what magnetic tapes were searched, and whether the search included any off-site vaults and storage facilities.

C. Identification of Versions

Software evolves rapidly in today's industry, and one version may be current for only a few weeks or months. Usually, a given version of software is identified by its version number such as 3.11, or in some cases by its "vintage," such as Windows 95. Software version numbers to the left of the decimal point are usually increased when a new version contains significant new capabilities, whereas the numbers to the right of the decimal point are increased to reflect minor increments.

While a finished program may have a specific version number, such as WordPerfect 6.1, there can be hundreds of components, each with their own version number, none of which is the number 6.1. It is therefore important to establish precisely which versions of the software were released to the public or put into service, and, if possible, identify some means for determining the individual components' version numbers included in those specific releases. Only after this version number history has been ascertained can one determine the specific source code used to create each release of the overall software.

XI. Analysis of Computer Software

Once the producing party has made a production, the initial challenges for a forensic software analyst are:

a) Testing for completeness of the production of software and object code.

b) Identifying specific versions of the software and the versions of the individual components of which it is comprised.

c) Correlating the specific versions of the software with the specific versions of the object code.
d) Creating a chronology of the software's evolution.
Only when these important preparatory steps have been completed can the analysis of the software with respect to the patent, copyright, or trade secrecy claims begin.

XII. Conclusion

The computer geeks of the world have effectively hidden many conceptual nooks and crannies that exist with respect to computer software and its associated computer-based evidence. The problems of evidence preservation, production, and forensic analysis are further exacerbated by each quantum leap that occurs in computer storage and processing power.

Computers, it could be argued, have made life easier for many who use them. However, nothing could be further from the truth for those who deal with the preservation, production, and analysis of computer-based evidence.

A COMPILATION OF ARTICLES PUBLISHED ELECTRONICALLY FROM 1998-2006

PART II: MILITARY TRIBUNALS AND TERRORISM

I. TIME FOR CONGRESSIONAL ACTION: THE NECESSITY OF DELINEATING THE JURISDICTIONAL RESPONSIBILITIES OF FEDERAL DISTRICT COURTS, COURTS-MARTIAL, AND MILITARY COMMISSIONS TO TRY VIOLATIONS OF THE LAWS OF WAR

II. MILITARY TRIBUNALS, THE CONSTITUTION, AND THE UCMJ