harborlabs
CYBER.SCIENCE

EXPERT TESTIMONY

Guidelines for Source Code Review in Hi-Tech Litigation

Whitepaper

Abstract

In a dozen years of working as testifying and consulting experts in technology-related litigation, we often find ourselves conducting source code reviews for our clients. Unfortunately, in many cases, we find that the parties have already negotiated the terms of the review without a sufficient understanding of the nature of the code in question, without reasonable assumptions about the resources required for a proper analysis, or without realistic expectations about the time necessary to adequately answer the relevant questions. The purpose of this white paper is to present basic guidelines for source code reviews to assist litigation teams in negotiating the terms of protective orders, to improve cost estimates, and to enable technical experts to streamline their analysis.

Introduction

Source code reviews are often required in technology-based litigation. Some courts provide limited guidance to the parties, but many details remain unaddressed. For example, the US District Court of Delaware provides a Default Standard for Access to Source Code, and the US District Court of Northern California addresses source code review in Section 9 of an Interim Model Protective Order. However, neither of these documents cover many of the practical challenges that confront the litigating parties when faced with implementing an actual source code review.

In reviews that involve source code belonging to the opposing party we often run into at least one of the following problems:

- Challenges related to conditions of the protective order
- Unrealistic expectations of time and resources
- Insufficient review tools
- Relevant portions of source code not produced

These issues may arise before independent technical experts are involved in the litigation. Consequently, counsel for the plaintiffs and defendants often have to negotiate the details without the benefit of technical advice. We find that this often causes problems during the actual review that make the process slower, more difficult, and more costly.

In the following sections, we address each problem in turn and provide some basic guidelines. We specifically avoid hard and fast recommendations because each review is unique and has its own idiosyncrasies. What we do provide, however, are basic concepts that emphasize trade-offs and can be tailored to specific circumstances. Our goal is to help legal teams make appropriate decisions early in the litigation process to ensure that source code reviews by their technical consultants are effective and efficient, and that appropriate protections of intellectual property are maintained.

Conditions of the Protective Order

When outside experts review the opposing side's source code, the source code owner is rightly concerned about protecting its intellectual property. The reviewer typically signs a "Protective Order" from the court that outlines the conditions under which he or she may see the code, how the code can be copied or printed, and general stipulations of non-disclosure.

The exact requirements and limitations outlined under a protective order are usually the result of negotiations of the parties and their counsel. While there are some standard provisions, both sides often press for customization in order to seek a tailored balance between ease of access for the reviewer on the one hand, and security of the intellectual property on the other. In general, increasing security decreases ease of access and vice versa.

Location

One of the first issues to resolve is where the review will take place. There are typically three alternatives.

- *Producing Party* - The review itself takes place on the premises of the law firm representing the owner of the source code. This is, in our experience, the most common scenario. Because the source code is in "friendly" hands, it is at a reduced risk of exposure. For the reviewer, however, this may require traveling, and it means only having access to the source code on pre-arranged days and during pre-arranged hours. Reviewers must be careful not to inadvertently reveal work-product or confidential information through unguarded conversations, notes left on the review computer, or other materials left behind.
- *Neutral Party* - The review itself takes place at a neutral location agreed upon by both parties. In terms of source-code security, this location is almost as secure as having the code with the producing party. This security can be increased by limiting control of the physical review machines to the producing party by, for example, using tamper-proof boxes. This arrangement may reduce the chance that he or she might inadvertently reveal sensitive information to the opposing side. These benefits are offset by the increased costs and coordination required.
- *Requesting Party* - We have participated in reviews conducted on the premises of the law firm of the requesting party as well. The code was provided via remote desktop over a VPN to a reviewing computer in a secure room. Although this approach places the produced code in a higher risk environment, there are times when the level of access required favors this solution. The requesting party should be aware that the producing side can, from a technical perspective, monitor the reviewer's actions when remote desktop connections are used. It is a good idea for the requesting party to request a stipulation in the protective order that the producer will not monitor their reviewer's actions.

Physical Setup of the Review Room

Once the location is selected, the next issue is determining the parameters of the actual source code review room.

- *Monitoring* - Almost all code reviews require the consultants to sign in and sign out. This is a basic and effective step. There are a variety of other mechanisms employed. Some parties require that the source code machine be monitored at all times either by a trusted person or by video equipment. While this does not limit the ability of the reviewer to access the code, it does increase the risk of the consultant revealing sensitive information to the opposing side.

- *Permissible Items* - Typically, no electronic items are permitted in the review room but generally the reviewer can take notes on paper. Recently, however, we participated in reviews where a laptop was authorized for entry into the code room so long as the laptop had no camera or wireless hardware. On the other hand, we also participated in a review where the code was so sensitive (e.g., single lines of code were considered to be worth millions of dollars) that paper was permitted into the room only under special circumstances.

- *"War Room"* - Most reviewers greatly benefit from a room where they can set up their laptop with Internet access for research and contact their side's counsel. These rooms are useful components of the review and very often overlooked by both sides. The requesting party should communicate this need when negotiating the protective order.

- *Printer* - Source code printing is an essential part of the review providing the reviewer with segments that can be incorporated into expert reports. One common method we utilize is to print to PDF files that are placed in a designated folder. Another option is for the printer in the review room to be loaded with bates-stamped paper. In this case, the reviewer can have immediate access to the printed sequences. The trade-off here is enforcement versus disclosure. When the producing side performs the printing, it can ensure that the printed materials conform to the conditions of the protective order. On the other hand, when the reviewer can print directly, the producing party cannot see what segments the other side thinks are important.

The Physical Review Hardware

The machines containing the source code are secured to prevent unauthorized copies.

- *Restricting physical access* - The source code review laptop often has its ports (e.g. USB, Ethernet, etc.) disabled. If the entire machine needs to be disabled, the easiest way to do this is with tamper-evident tape wrapped around the body of the laptop physically blocking access to these ports. However, this also prevents authorized users, such as the IT department of the producing party, from installing missing tools or newly produced code. An easier way to block access to the reviewer is to provide the reviewer with only a limited account that is blocked from using specified hardware ports.

- *Restricting wireless access* - The code review machine often has no wireless hardware at all. Authorized users may still use the Ethernet port for legitimate requests that require networking. The machines containing the source code should also be configured to allow reasonably comfortable use over extended periods of time. Ideally they should be configured with dual monitors, an external keyboard and a mouse. The workstation should be set up in an ergonomically correct fashion.

receive incomplete information from their clients about how easy it should be to find a particular feature in the code, the code review might require orders of magnitude more resources than initially projected. Software projects tend to be complex, and intuition about them is often wrong.

In our experience, a baseline code review involves the consultant examining the code at least twice, with each of these two reviews lasting at least two days. The goal of the first review is to familiarize the consultant with the "big picture" and determine if the code requires any special considerations. If it turns out that the code and the questions about the code are both straightforward, then the reviewer may cancel the follow up review. It should be noted that for very large systems, the full review in some instances might require multiple consultants reviewing source code for weeks.

When the goal of the review is to identify source code evidence for legal arguments, such as infringement contentions, the space between reviews also gives the consultant a chance to discuss the source code with counsel. These discussions, with the benefit of some exposure to the source code, often assist counsel in refining the arguments and theories in the case, and clarify for the consultant what he or she is looking for. The subsequent review is significantly more effective.

Another reason for breaking the review into two phases is to provide time for additional discovery. Frequently, counsel will need to request production of additional code or code resources after an initial review. Moreover, insights from the code can provide new insights into additional documents to search for, or tests to perform on the product.

Of course, some reviews can easily be completed in a single day by a single person, while other reviews require multiple weeks with multiple reviewers. Some of the factors that influence the time requirements are:

- *The complexity of the inquiry*: Some code reviews are performed to answer simple questions, such as whether a particular function exists in a software package. It may be possible to definitively answer such questions in a very short time period, especially if the answer is yes. Other reviews require much more complicated tasks such as comparing versions, or determining whether one software package was copied from another.
- *The quality of the code*: A well-developed software package is much easier to review and understand than a system developed poorly. Answering a question about the code can take an order of magnitude longer for the same size code, based on how well the code is designed and written.
- *The size of the code*: The impact of the size of the code depends in large measure on other factors. For example, if the goal of the review is to prove that a certain algorithm does not exist in the code, then size is extremely important. When size matters, getting a line count across all files is a good approximation, although the number of files, the number of modules, and the number of versions are all relevant. In making an estimate, legal teams need to be aware of non-code files (such as data files, etc.) that may be reviewed as well as support files such as Makefiles, build scripts, source control meta data, and third party modules that are incorporated in the system.
- *The programming language:* It is much easier to review programs written in high-level languages. In general, it is easier to review Python than C, and it is easier to review C than assembly. Anytime the

software has to interface with hardware, regardless of the programming language, there is additional complexity that takes time to figure out, especially if the hardware is customized.

- *Documentation:* The availability of documentation related to the source code speeds up the review process. However, it is our experience that most projects tend to be poorly documented.
- *The review environment:* The format of the code (e.g. paper printouts versus an electronic format with developer tools) makes a tremendous difference in the efficiency of the review. A proper development environment with a language-aware editor can reduce review time significantly.

Software Review Tools

As described above, the tools available to the software reviewer are important. The success or failure of the review may hinge on whether the analyst has the appropriate means to perform the analysis.

One of the best source code analysis tools is a fully functioning build environment. A build environment allows the reviewer to compile the source code into a working program. This is extremely useful for at least the following reasons:

- It gives the reviewer reason to believe that all the source code is available. Missing source code prevents the successful building of the code. In some cases, building the code is not feasible because large software projects may have unrelated components that are unavailable or irrelevant for review. Nevertheless, compilation provides an indication of completeness.
- The reviewer can test the software with breakpoints, a debugger, or even debug output. This enables the reviewer to more quickly understand how the code functions, especially with specific inputs.
- The reviewer can trust that the running code being tested is the same as the source code that is reviewed. Such assurance is not possible when the reviewer is provided with an executable and source code, but without the ability to compile and build the system.

Unfortunately, it is rarely practical to provide the reviewer with a complete build environment because they generally require highly tuned, specially configured software. Even when it is practical to have such an environment, there are often a number of configurable options in the build process and it may be impossible to tell if the reviewer has built the software correctly. Beyond these problems, the reviewer may be unlikely to have the right hardware or system setup to even run the software once built. Finally, the opposing side may object to the reviewer building and running code.

If, however, the requesting party wants the consultant to build code, this needs to be factored into the setup of the review machine. The operating system and installed software need to meet the build specifications. Similarly, if the requesting party wants the reviewer to actually run the software after building it, the review environment needs to support this, and in some instances the requesting party may need to obtain specialized hardware.

A typical source code review project in a patent case is not likely to require building and running the software. In these scenarios, it is less critical to arrange all of the software tools up front as long as the appropriate tools can be installed later. It is important to ensure that there is appropriate technical support for whatever environment is ultimately installed. As part of the negotiations of the code review, the requesting party's counsel should have a clear local point of contact for solving these problems. We experienced a difficult review where the machines were delivered to a law office in one city, but the IT support was located elsewhere in the country.

For code reviews to be effective, particularly given that the result of the review will most likely be described in a report rather than in a computer language, some tools with special capabilities are required. Below is a list of these capabilities, with sample lists of applications that can be provided to enable them. It is important to note that in many cases it is useful to have multiple tools.

The list below is oriented toward Windows, as this is the most popular platform. A similar list could easily be produced for other operating systems. Here are the capabilities:

- *Searching and cross-referencing code* - code does not connect in a straight line - having good tools for searching and cross-referencing will save time and enable more thorough analyses.
    - Microsoft Visual Studio (Required in the case of Microsoft languages)
    - Eclipse IDE with plugins for specific languages
    - dnGrep
    - pdfgrep
    - Xcode (Required in the case of Apple languages)
- *View documents* – documentation is often produced alongside source code. This documentation may be in a variety of formats but it is most commonly in the form of Microsoft Office documents or PDFs. In order to read these files, appropriate viewing tools must be installed. Similarly, oftentimes a source code repository conatins compressed files that must be expanded in order to be analyzed.
    - Adobe Acrobat Reader
    - Microsoft Office Document Viewers
    - 7-Zip
- *Compose plain text* - reviewers often want to produce a manifest containing all of the files printed and the mapping from Bates numbers to file names. This makes it easier to report on the code after the review is complete. Similarly, occasionally electronic note-taking, even if later discarded, can be extremely helpful during a review. These editors are well-known for their power and flexibility, but the editor a reviewer prefers to use will largely be a matter of taste and familiarity.
    - Vim/gVim
    - Nano
- *Print files with line numbers* - including line numbers is a critical capability. The following are known to have this capability built in with a high degree of configurability:

- Notepad++
- Kate
- TextWrangler

- *Print to a file* - paper is often inserted into a printer pre-labeled with Bates numbers. This can be convenient, but can also make it hard to get things right the first time. The ability to first print to a file is essential, especially since it makes it easier to predict how many pages will be produced for a given review. This can be important for staying within the parameters set in a protective order. While Windows typically has a print-to-XPS feature, this can be cumbersome to use compared to the alternatives:
    - CutePDF printer driver
    - A PDF reader (e.g., Acrobat)

In addition to the above essentials, reviews can often be made even more effective with supplementary tools. When using Windows, it is extremely useful to have the Ubuntu Sbusystem for Windows or Cygwin installed, as these provide some basic but flexible and powerful command-line utilities with which most reviewers are already familiar (e.g., grep, find, bash, sed, awk, and vi). Additionally, wherever possible, having a scripting language installed such as Python or Perl can make a review much easier to accomplish, particularly when faced with unusual or unforeseeable tasks. A reviewer, using these tools, can often create a new tool very quickly that increases efficiency and reduces the review time substantially.

The Source Code

One problem that we commonly experience is incomplete code production. In many cases, reviews are delayed because critical pieces are missing. Below, we identify three common problems that prevent the production of necessary code:

Assumed Irrelevancy. Generally, the court only requires the production of relevant code. The parties may dispute which parts of the code are relevant, and the producing party may base their code production on unreasonable assumptions.

As a simple example, the producing party may assume that they need only produce the files or modules that are directly involved in some particular operation. The problem with this assumption is that modern software is highly interconnected. The produced code almost certainly makes use of unproduced code, the functionality of which may be crucial to understanding how the code works in general, and answering questions specific to the case. For these reasons, the requesting party might consider asking for the complete source code of the software project. If the producing party absolutely rejects this request, then they need to spend the time necessary to ensure that all the code used by the relevant module is also produced. In technical terms, having no "unresolved symbols" from their code base is the goal.

Unproduced Versions. Most legal cases are tied to specific versions of a particular piece of software. For example, patent infringement cases generally apply to a particular time period. Yet, we frequently find that the requesting party does not explicitly request the versions for the indicated years.

Understanding the version numbers of the accused products or the prior art systems is very important, especially for the requesting party, who must know what to ask for and whether or not they have received the code that they need.

Most modern software companies use what is called either a “source code control system” or a “version control system.” This software keeps track of the source code as it changes. Changes to the code are recorded by date, and certain versions such as those actually released in product form are often tagged with major numbers. It should be noted that the internal versions of the software may not be exactly related to public version numbers. It is a good idea for the requesting party to learn as much as possible about the versions of the software and what they mean. A good opportunity to do that is in the depositions of the corporate representatives of the producing parties, or through interrogatory questions. We recommend that the requesting party identify the released versions of the software that pertain to the years in question, and any internal version numbering or tagging that correspond. Once identified, the requesting party should ask for all released versions of the relevant code to be produced. Ideally, each version resides in a separate, but parallel, directory to make cross-version comparisons easy.

Unproduced Non-code Components. In addition to the raw source code, additional data is often required for a complete analysis. Three common examples are build data, source code control metadata, and data files or databases.

Build files and build instructions are sometimes not included but are significant for many reasons. First, some languages such as C can actually disable entire sections of code based on certain build parameters. Without the build files, it may not be clear which code is active. Second, the build files can identify all source files, so a reviewer can tell if something is missing. Finally, the build files can also give some practical indications of how the source code operates.

As mentioned previously, modern software is generally tracked using a source code or version control system. These systems keep track of how the software changes, and who made the changes. This information, called the code metadata, provides valuable insights into why and how code changes, and how the code sections interact.

Finally, many software products make use of extensive configuration files, code templates, execution scripts, data files and databases. These non-code components often dictate how the code operates or even (e.g., in the case of code templates) produce code automatically. While the precise composition of these files may vary, even from customer to customer, the software developers generally keep canonized sample files for testing purposes.

We recommend that the reviewing party request all the build files, all the source code control metadata, and all data files that are used for testing the relevant source code. In the case of generated code, it is very useful to see both the original template (with data necessary to instantiate it) and at least some sample code output. When sample output is not available, but templates are, it is beneficial to the reviewer to have access to the source of the system used to generate the final code. This can help to reduce or eliminate ambiguities or leaps of logic in subsequent reporting.

When possible, it is also helpful to have access to any third-party source code that is heavily relied upon. For example, many web sites use jQuery as a critical component of their Javascript-based systems. There is nothing stopping them from making individual changes to that library, since it must be included in source form. Having access to the actual version used, even though it is third-party software, can be helpful in determining what, precisely, a given system is doing. In all cases, human-readable versions are better than compressed or “minified” versions.

It should be noted that essential data files might be stored in a format that is not text. The producing party should ensure that they provide the technical documentation and/or tools necessary to understand and interpret the data files.

Conclusions

Parties in litigation often require source code reviews. In this white paper we address some of the common issues that arise during source code review and provide recommendations for important considerations. It is our hope that this document will serve to guide discussions about the source-code review process in high tech litigation.

About Harbor Labs

Harbor Labs is a leading provider of cyber science consulting services, specializing in cryptography,, network security audits, software vulnerability assessments, and secure programming. Our elite staff of cyberscientists comprises many of the industry’s foremost experts in their respective cyber disciplines and are among the first to be contacted when a high-profile, national-level cyber event occurs.

info@harborlabs.com
1-833-CYBR SCI
106 Old Court Rd,
Suite 305, Pikesville,
MD 21208