harborlabs
CYBER.SCIENCE

LITIGATION SUPPORT

Guidelines for Source Code Comparison in Litigation

Abstract

In over a dozen years of working as testifying and consulting experts in technology-related matters, we often find it necessary to conduct source code comparisons for our clients. There is currently no standard methodology recognized by the courts for performing such comparisons. Nonetheless, Harbor Labs has developed its own set of techniques and methodologies to perform codebase comparisons across a wide variety of common litigation scenarios. This whitepaper aims to present those basic guidelines for source code comparisons to assist in the understanding the scope of such work, facilitate cost and schedule estimates, and enable technical experts to streamline their analysis in order to achieve the desired outcomes.

Introduction

Source code comparison is a computer science discipline that Harbor Labs consulting clients often require in support of their software-related litigation. There are many reasons why such a comparison may be necessary, but we find that code comparison is most commonly performed in support of litigation related to trade secret misappropriation or breach of contract disputes.

Any project requiring a code comparison will typically have its own set of unique circumstances and characteristics, which in turn dictate the process and scope of the comparison to be performed. In some cases, wholesale copying in part or in whole of the codebase might be alleged. For instance, one party may claim that their codebase was copied and reused by another party in order to produce a competing product. In such a case, a comparison of the entire codebase will be necessary. The goal of the comparison would be to demonstrate a high degree of overlap in the two codebases to substantiate the allegation. Or similarly, if defending against such an allegation the objective would be to show a high degree of uniqueness in the code sets.

In other cases, a more a subtle copying may be suspected. For instance, it could be alleged that a party used another party's software as a "reference codebase," enabling them to develop their own unique software product using the reference codebase as a blueprint. This allegation is not as straightforward as simple copying, and there may be little to no direct evidence of source code reuse. In this case, other more complex computer science techniques must be used to reveal any circumstantial evidence that might support the allegation.

Before beginning any code comparison exercise, it is crucial to first understand the facts of the case and the assertions that are being made. This will determine the most effective analytic approach and the best set of techniques necessary to support the client's case. In this white paper, we outline the several methods for performing source code comparisons.

whitepaper

The Process of Comparing Codebases

Source code review engagements begin with an assertion or a hypothesis. Common hypotheses are that the code was used as a reference, or that the code is a direct copy of another codebase, or that the code was not copied at all and is original work. Our analytic process is designed to prove or disprove the hypothesis, typically comprising a code review phase followed by a documentation phase, ultimately resulting in a detailed report containing evidence-based conclusions.

There are several avenues of investigation to explore in a source code comparison. In order to determine whether and if so, how, one codebase has been derived from another, it is necessary to use all available information, including:

1. Developer documentation
2. Manual source code analysis
3. Automated source code comparison

Developer documentation often provides a useful description of when and how source code was developed. If such documentation exists, it may be possible to use it as a guide to prove that two codebases were developed independently. Alternatively, documentation may reveal that a later codebase could have been derived from another codebase as a reference. In all cases, any available developer documentation can be useful in a source code comparison engagement. Developer documentation may:

1. Assist in the understanding and flow of the code comparison
2. Imply that two codebases were developed independently
3. Imply that one codebase was used as a reference to build another codebase

Manual source code analysis typically involves combining several different techniques to provide decisive information about how much two codebases differ. Examples of such techniques include:

1. Comparing number and composition of lines of code
2. Comparing external and internal dependencies between systems; comparing strings from within the codebase (looking for key information such as email addresses, similar typos, similar text, etc.)
3. Comparing the overall architecture of the codebases (e.g., one is primarily comprised of microservices whereas the other is monolithic.)
4. Comparing the cyclomatic complexity of the codebases

This type of analysis can provide specific instances of codebase overlap. It is also helpful in providing a large amount of additional evidence to support the results of automated analysis.

Automated source code comparison is useful because it can provide evidence of direct source code copying across the entire codebases, regardless of the size of those codebases. However, such a comparison typically also uncovers many false positives (e.g., the appearance of code copying when there is none) that scale with the size of the codebase. Because of this challenge, it is beneficial to design tests where similar functional components from the codebases are matched up and then compared using a code comparison algorithm such as Rabin-Karp. There are two common pitfalls of automated comparison:

whitepaper

1. Potential for many false positives
2. Algorithms may require large amount of computing resources and time

Binary analysis is necessary when the source code is not available for one or more of the software products being compared. There are a variety of reasons why this may occur. For example, source code may be lost or destroyed over time or in a company reorganization. In other cases, it may be necessary to analyze how a product works before having access to the source code. Regardless of the reasons, comparing codebases in which one or more of the programs has already been compiled is much more difficult than comparing codebases in which source code is present as structural, syntactic, and semantic information about the source code is lost in the compilation process.

In many cases, it is best to identify specific functionality, such as the implementation of a specific algorithm, to be analyzed or compared. After identifying the functionality at issue, a reviewer will usually first manually disassemble the software products using a tool such as Ghidra to identify and analyze the assembly-language. After isolating the desired functionality, there are a variety of techniques that can be useful to determine whether the two implementations are derived from a common source. These include:

1. Automated analysis of implementation of the algorithm (e.g. using structure and design of basic blocks)
2. Manual analysis of the implementation of the algorithm
3. Analysis of cyclomatic complexity
4. Identification of errors or bugs common to both implementations
5. Decompilation to a higher level language in order to use source code analysis techniques

The Harbor Labs Approach

The investigative techniques outlined in this document are useful for producing many possible leads in making a determination. Ultimately, a human must review and select the most pertinent information from these leads to provide an accurate assessment. After all the analysis is complete, it is crucial to manually review the results to identify and remove false positives and draw supportable and reproducible conclusions.

To deal with some of these pitfalls, Harbor Labs has developed a variety of internal tools that we use to generate and filter data and to automate many of the manual analysis steps described in this report. Harbor Labs has found that by using these custom tools it is able to scale its analysis to extremely large codebases that would have been infeasible to analyze in the past. Furthermore, Harbor Labs has also found that its tools dramatically cut down on the time needed to analyze codebases of any size.

Despite the need for manual review of the data, it is also important that raw results of all tests be provided with any reports produced from source code review to maintain scientific integrity of the process and to demonstrate that data was not cherry-picked to support any particular conclusion.

The following list provides an example of the types of information that may be request or reviewed for each step of the source code comparison process:

whitepaper

1. **Review supporting developer documentation**
    - Version control history
    - Design and architectural diagrams
    - Build instructions
    - Development process documentation including
        - User stories
        - Requirements documents
        - Sprint plans
        - Progress and milestone assessments
        - Meeting minutes
        - Test documentation and results
2. **Review the source code**
    - Composition of codebases
    - Source code architecture
    - Exported APIs
    - Build environment and dependencies
    - Bugs and design flaws
    - Unit tests
    - Strings including
        - URLs
        - Email addresses
        - Developer names
        - Copyrights
        - Dates
        - UI messages including error messages
        - Code symbols including class, function, variable names and namespaces
        - Typos

whitepaper

3. **Test the source code**
    - Programmatic source code similarity comparison either
        - Applied across codebases
        - Applied across similar subcomponents
4. **Conclusions**
    - All tests generate false positives; results must be filtered
    - Draw supportable and reproducible conclusions from the findings.
5. **Create a Code Comparison Assessment document**. The conclusion of the analysis will culminate in a report detailing the findings. The report should contain the following general sections:
    a. Executive Summary
    b. Summary of the questions being addressed
    c. Summary of the codebases
    d. Source code comparison process design and methodology
    e. Results of development process review
    f. Results of source code review
    g. Results of empirical testing
    h. Conclusions

Estimating Cost

The cost of completing a source code comparison may be difficult to predict. If, for example, the client provides incorrect information about how similar or different the codebases may be, the code comparison might require many more (or many fewer) resources than initially projected. Software projects tend to be complex, and intuition about them is often wrong.

Some comparisons can be completed in a single day by a single person, while other reviews require multiple weeks with a larger team. Some of the factors that influence the time requirements are:

- **The complexity of the inquiry:** Some code comparisons are performed to answer simple questions, such as whether a particular function exists in both codebases. It may be possible to definitively answer such questions in a very short time, especially if the answer is yes.

whitepaper

- **The quality of the code:** A well-developed software package is much easier to review and understand than a system developed poorly. Comparison source code can take much 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 comparison is to prove that a certain algorithm (but not a specific function) 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 producing 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 compare programs written in high-level languages. In general, it is easier to compare code written Python than in C, and it is easier to review code written in C than assembly. Anytime the software must 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 comparison process. However, it is our experience that most projects tend to be poorly documented.

Conclusions

It is common for clients to require a comparison of source code to support litigation or investigations related to trade secrets or contract disputes. In this white paper, we address the common issues that arise during litigation support engagements and provide recommendations for efficient and effective performance of code comparisons. This document is intended as a guide so that source-code comparison exercises may be in any applicable engagement.