Measuring Whitespace Patterns In Computer Source Code As An Indication Of Copying

BY LLANA SHAY, NIK BAER, AND BOB ZEIDMAN OF ZEIDMAN CONSULTING

Ilana Shay *was a research engineer at Zeidman Consulting. She has developed software for advanced printers, e-commerce, image processing, and semiconductor equipment. She was the head of the computer science department in Ort Holon, technical high school. Ilana is certified in the use of CodeSuite®. Ilana has won recognition for excellence from Tencor Instruments. She holds a bachelors degree in mathematics and computer science from Tel Aviv University. She can be reached at Ilana@ZeidmanConsulting.com.*

Nikolaus Baer *is a research engineer and projects manager at Zeidman Consulting. He has developed software for marine research, optical testing equipment, military terrain databases, mobile applications, and medical devices. He has written articles and given presentations about software trade secret theft and how to analyze source code for detecting it. Nik is certified in the use of CodeSuite®. He placed first in the Start Cup 2004 business plan competition. He holds a bachelors degree in computer engineering from UC Santa Barbara, where he attended on a Regents Scholarship. He can be reached at Nik@ZeidmanConsulting.com.*

Robert Zeidman *is the President of Zeidman Consulting. He is the author of the books Designing with FPGAs and CPLDs, Verilog Designer's Library, and Introduction to Verilog. Bob is a Senior Member of the IEEE and was the recipient of the 1994 Wyle/EE Times American by Design Award and the 2003 Jolt Reader's Choice Award in addition to other engineering and scholastic awards. He holds seven patents and has an MSEE degree from Stanford University and BS degrees in physics and EE from Cornell University. He can be reached at Bob@ZeidmanConsulting.com.*

There are several different methods of comparing source code from different programs to find copying[1]. Perhaps the most common method is comparing source code statements, comments, strings, identifiers, and instruction sequences. However, there are anecdotes about the use of whitespace patterns in code. These virtually invisible patterns of spaces, tabs and newlines have been used in litigation to imply copying, but no formal study has been performed that shows that these patterns can actually identify copied code. This paper presents a detailed study of whitespace patterns and the uniqueness of these patterns in different programs. We decided to investigate whitespace file patterns and determine whether comparing whitespace patterns in different files is a reliable method to measure code similarity and thus detect copying.

When writing code, the programmer is focused on the visual elements: statements, comments, variable statements, comments, variable names, and strings. During the writing process the programmer also uses non-printing characters to separate the programs visual elements. The non-printing characters can be spaces, tabs, or newlines. The sequence of these non-printing characters is the whitespace pattern.

We will score file pairs based upon a percentage of similarity of their whitespace patterns. An effective method should meet the following criteria:

- The whitespace pattern of a file compared to itself should produce a 100% similarity score.
- The whitespace pattern of a file compared to a completely different file should produce a low similarity score.

We compared and determined the percentage of similarity of the whitespace patterns of source code files from one program against themselves. Each file should be 100% similar to exactly one file in the set, itself, and should be very dissimilar to all the other files. The similarity scores should produce a low average similarity score with most file comparisons close to 0% similarity and a small group, representing files compared to themselves, at 100%.

We also compared source code files from one program against a different program. If the similarity scores follow a normal distribution[2] then we examine the similarity scores of the whitespace patterns according to these conditions:

- Low average means that the whitespace method works really well because low similarity scores indicate that most of the files are different from each other.
- High average and high standard deviation means that the method may still work, but we need to investigate why we are getting high correlation for different files. We may need to filter out some of the files.
- High average and low standard deviation means that the whitespace method finds different files similar to each other, indicating that the method does not work well.

If the similarity scores of the whitespace patterns do not follow a normal distribution, then this implies that there may be files that are similar for reasons we have not taken into account such as a common author or the use of third party code. For this method to be a good way of finding copied code, we need to find a way to filter out files to get a normal distribution.

METHODOLOGY

There are no off-the-shelf whitespace comparison tools, so we had to develop our own. The steps to measure the whitespace similarity are:

- Convert each file containing source code to a whitespace file format.
- Compare whitespace formatted files using the CodeDiff® function of CodeSuite®, a program from Software Analysis and Forensic Engineering Corporation.
- Analyze the results.

The CodeDiff tool in CodeSuite compares and scores the similarity of files, but we had to develop tools to convert the source code to a whitespace format, and to analyze the CodeDiff database results. These tools are described below.

CONVERTING SOURCE CODE FILE TO WHITESPACE FILE FORMAT

We created the FileReformatter program that inverts the text in the file. By this we

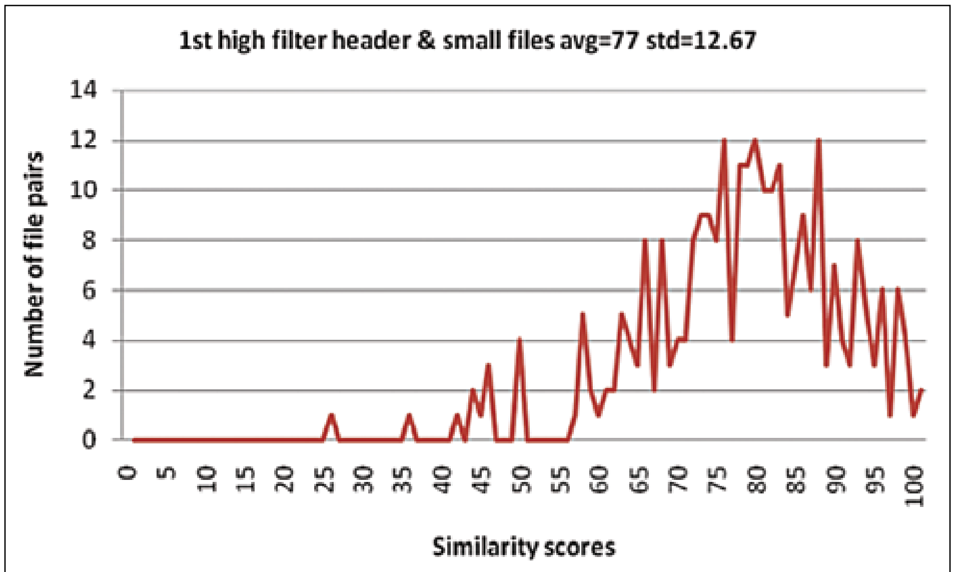

Figure 1: Compare one program to itself, 2nd highest score; filter both small files and header files.

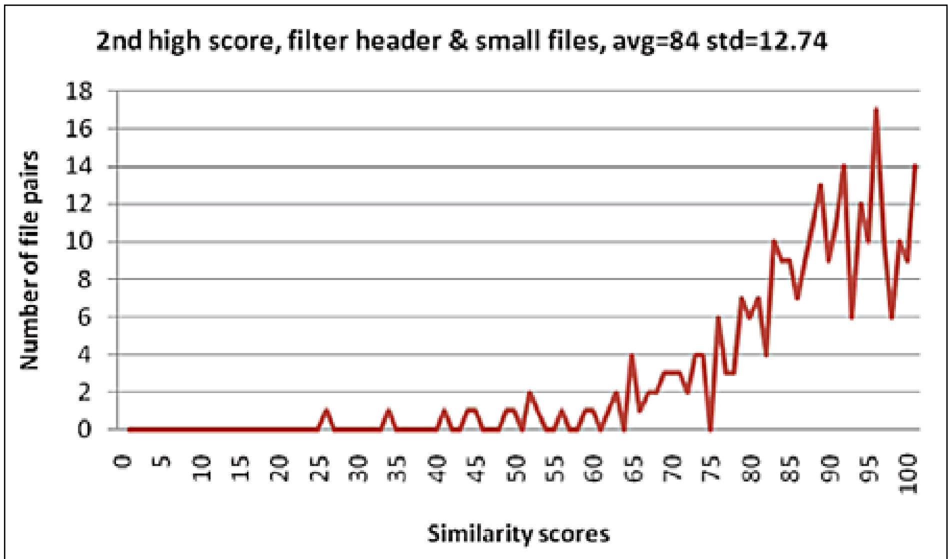

Figure 2: Compare one program to a different program, 1st high score, filter header and small files

mean that the invisible characters become visible, and the visible characters become invisible, so that CodeDiff can evaluate the whitespace characters.

FileReformatter converts source code files to whitespace file format according the following rules:

- Every continuous sequence of printable characters is converted to one space.
- Every space is converted to the character 'S'.
- Every tab is converted to the character 'T'.
- Newline characters are not converted.

The output file contains only the characters 'T', 'S', space and the original newline characters.

For example, the following line of C code:

int var = prevValue + 5;

can be thought of as:

int(S)var(T)=(S)prevValue(S)+(S)5;

where (S) and (T) represent space and tab characters; so FileReformatter will translate the line into:

S T S S S

COMPARING FILES LINE BY LINE

We used CodeDiff to compare directories that have been translated by FileReformatter. It compares files in pairs; one file from the first directory and one file from the second directory. Each line from the file in the first directory is compared to each line in the file from the second directory. CodeDiff calculates the percentage of matching non-blank lines to the total number of non-blank lines in the first file. This percentage is the file pair similarity score. The output is a database containing all of the file pairs and their similarity scores.

Tests

Tests were done on code that was written only in the C programming language. We ran FileReformatter on the source code and converted every single file to whitespace format; then we ran CodeDiff that calculated the similarity scores.

COMPARE ONE PROGRAM TO ITSELF

The open-source Linux Kernel version 1.0 was selected to be tested. This directory has 487 different source code files in many sub-directories. CodeDiff was set up to compare the same directory against itself, which means each file is compared against itself as well as all the other files in the directory tree. We expected to see 100% similarity when comparing each file to itself, and low similarity when comparing a file to all the other files.

COMPARE ONE PROGRAM TO A COMPLETELY DIFFERENT PROGRAM

Comparing two different programs should result in low similarity scores. We expected to see low scores, but we thought we might also get some high similarity scores if there is code from a third party or a code generation tool. Using CodeDiff, we compared the Apache HTTP version 2.0.35, which contained 653 files to the Linux Kernel version 1.0, which contained 487 files.

Results

In the results we display the percentage of score spread along with the average and standard deviation (STD).

COMPARE ONE PROGRAM TO ITSELF - SECOND HIGHEST SIMILARITY

We compared all files in the one directory to itself; this produced a huge database of 237,169 file pairs. When looking for copying, an investigator will examine the highest scoring comparisons. When comparing a program against itself, each file will have at least one 100% matching comparison with itself. In order to model a real life scenario when there may be some similar code, or similar functions, but files are not identical, a new database was created by filtering out all but the second highest scoring file pairs for each file in the first directory. The newly created database has only one file pair for each file—the highest matching file other than itself. Using CodeSuite we created a summary spreadsheet that had 487 file pairs with an average similarity score of 79 and a standard deviation of 18.4. Even after removing the top matching files, we had some 100% matching scores, which may be the result of a file being copied into another directory. We assumed that header files, small files containing simple definitions for the program, may be similar and disproportionally contributing to the high scores, so we filtered out the scores of these files. Because the CodeDiff score is calculated as a percentage of the file size, we thought that filtering small files with less than ten lines may also help to reduce the number of high similarity scores.

Change paragraph to "Filtering out both header files and files with less than 10 lines left 276 file pairs, see Figure 1. The average similarity score was 84, which is still high. However, since we compared the same program to itself, some of the similarities could be caused by a common author. At this point the whitespace pattern matching has not produced a narrow spread near zero that would allow us to confirm that these files are not copied from each other."

COMPARE ONE PROGRAM TO A COMPLETELY DIFFERENT PROGRAM – FIRST HIGHEST SIMILARITY

We assumed that the Linux source code files are very different from the Apache source code files, so the whitespace similarity scores should be very low. The comparison of the 653 Apache server files to the 487 Linux Kernel files produced 318,011 file pairs. Since an investigator will examine the highest scoring comparisons, we do the same as before. When we examined the results of comparing one program to a different program and looked at the highest similarity scores, the database still had some 100% similarity scores. The average similarity score is 71 and the standard deviation is 20.09.

Filtering out small and header files resulted in a smaller database that had 274 file pairs, with an average (see Figure 2) of 77 and standard deviation of 12.67. The filtered database has a high similarity average despite the fact that the compared files are from different programs and we believe them to be developed independently.

CONCLUSION

This whitespace pattern matching method can be used to focus a search for evidence of similarity or copying, but this method cannot stand by itself. When we compared a set of files to themselves, there were many files at the with low similarity scores, but there was a large standard deviation. Even after attempting various filtering methods, we often had a wide distribution of scores and a number of different files with identical or nearly identical whitespace patterns.

When we compared completely different files from completely different software projects, and filtered out header files and small files with less than 10 lines, there were still many files with high similarity scores. Therefore, this method is not precise. High whitespace comparison scores do not necessarily mean that there is similarity between programs, since it has been shown that completely different programs may have high scores. Low scoring file pairs indicate a low level of similarity, and high scoring file pairs only indicate where further investigation for copying might be focused; they do not imply that copying occurred.

FUTURE WORK

Future work can be done in the following areas:

- Examine sequences of whitespace. Perhaps similar sequences of whitespace in lines of code are better indicators of copying than line-by-line patterns.
- When filtering out small files, vary the number of lines in a file that we define as a small file until whitespace pattern matching is effective.
- When comparing a program to itself, filter out files with the same name, which is very likely to be the same file in a different directory.
- Examine all unseen characters in addition to space and tab.
- Test this whitespace pattern matching method on different programming languages.
- Compare code generation tools. When using a code generation tool, like MFC Wizard, the whitespace similarity score is expected to be higher than for manually created code.
- Compare two different versions of the same program as a more rigorous test of whitespace pattern comparisons.

ENDNOTES

1. Although many in the field refer to plagiarism, this is not accurate. Plagiarism is unauthorized copying. The algorithms defined in this field of computer science can detect copying but not whether the copying was authorized. We will refer to "copied code" instead of "plagiarized code."
2. Normal distribution refers to a bell-shaped distribution that indicates that there is a high probability that measurements occur close to the mean.