



Annual ADFSL Conference on Digital Forensics, Security and Law

2010
Proceedings

May 20th, 11:00 AM

Measuring Whitespace Patterns as an Indication of Plagiarism

Ilana Shay

Zeidman Consulting, ilana@zeidmanconsulting.com


Nikolaus Baer

Zeidman Consulting, nik@zeidmanconsulting.com

Robert Zeidman

Zeidman Consulting, bob@zeidmanconsulting.com

Follow this and additional works at: <https://commons.erau.edu/adfsl>

 Part of the [Computer Engineering Commons](#), [Computer Law Commons](#), [Electrical and Computer Engineering Commons](#), [Forensic Science and Technology Commons](#), and the [Information Security Commons](#)

Scholarly Commons Citation

Shay, Ilana; Baer, Nikolaus; and Zeidman, Robert, "Measuring Whitespace Patterns as an Indication of Plagiarism" (2010). *Annual ADFSL Conference on Digital Forensics, Security and Law*. 10.

<https://commons.erau.edu/adfsl/2010/thursday/10>

This Peer Reviewed Paper is brought to you for free and open access by the Conferences at Scholarly Commons. It has been accepted for inclusion in Annual ADFSL Conference on Digital Forensics, Security and Law by an authorized administrator of Scholarly Commons. For more information, please contact commons@erau.edu.

EMBRY-RIDDLE
Aeronautical University™
SCHOLARLY COMMONS

(c)ADFSL



Measuring Whitespace Patterns as an Indication of Plagiarism

Ilana Shay

Zeidman Consulting
15565 Swiss Creek Lane
Cupertino, CA 95014 USA
phone: 408-203-9715
fax: 408-741-5231
Ilana@ZeidmanConsulting.com

Nikolaus Baer

Zeidman Consulting
15565 Swiss Creek Lane
Cupertino, CA 95014 USA
phone: 805-699-6452
fax: 408-741-5231
Nik@ZeidmanConsulting.com

Robert Zeidman

Zeidman Consulting
15565 Swiss Creek Lane
Cupertino, CA 95014 USA
phone: 408-741-5809
fax: 408-741-5231
Bob@ZeidmanConsulting.com

ABSTRACT

There are several different methods of comparing source code from different programs to find copying¹. 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 and tabs 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.

Keywords: Copyright Infringement, Intellectual Property, Litigation, Open Source, Plagiarism, Source Code, Source Code Similarity, Whitespace.

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” except where we are quoting from other papers.

INTRODUCTION

When writing code, the programmer is focused on the visual elements: statements, comments, variable names, 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.

The significant problem of copied code in academia and industry, and ways to address the problem, has been discussed in a number of papers (Abraham, S. and Milligan, G. 2008). Papers that survey the various copy detection methods for detecting copying discuss the examination of language tokens like comments, data types, identifiers and strings but do not mention whitespace (Whale, G. 1990) (Sallis, P., Aakjaer, A., and MacDonell, S. 1996) (Clough, P. 2000) (Goel, S. Rao, D., et al. 2008). Some papers that discuss specific techniques for copy detection mention whitespace in passing, but do not address it in their algorithms (Parker, A. and Hamblin, J. 1989). Several significant papers in the field describe copy detection algorithms that ignore or remove all whitespace before performing a comparison (Baker, B. 1995) (Hamilton, J. 2008). Early papers by one of the authors of this paper replaces all sequence of whitespace characters with a single space character (Zeidman, R. 2006, 2008). Only one paper investigated copy detection methods by specifically looking at whitespace (Aliefendic, S. 2009), but the methods presented in the paper were not successful at separating copied code from independently developed code. 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.

HYPOTHESIS

Our hypothesis was that if a whitespace pattern is a good method of evaluating file similarity, then a file copied from another file will have similar whitespace patterns, while two independently developed files will have very different whitespace patterns. We would not expect different files to have similar whitespace patterns.

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.

To test this hypothesis 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 very similar to exactly one file in the set, itself. If we graph the similarity scores of the whitespace patterns for all file pairs, and if whitespace pattern matching is a good method, the similarity scores should produce a low average similarity score with one large, narrow peak close to 0% and a small peak, representing files compared to themselves, at 100%.

We also compared source code files from one program against another unrelated program. If we graph the similarity scores of the whitespace patterns and obtain one peak, it can be understood 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 as similar to each other, which means that the method does not work well.

If we graph the similarity scores of the whitespace patterns and obtain more than one peak, 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 one peak.

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 in the 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 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.
- Newline characters are analyzed as line separators.

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.

Tools for Analyzing CodeDiff Results

Four programs were required to manipulate CodeDiff database and calculate the average score and standard deviation (“STD”):

1. DB Skimmer: parses the CodeDiff database and removes all entries except the n^{th} highest scoring file pairs for each file in the first directory. We created this program.
2. CalcAvrScore: calculates the average of file pair similarity scores in the CodeDiff database. The user can select the number of highest scoring file pairs to count in the average calculation, and can also disregard file pair scores based on the file size, or the number of lines in the file. We created this program.
3. Filter DB: removes files from a CodeDiff database, based on their extension, size, or the number of lines that they contain. This program calculates the average and standard deviation for the remaining scores in the database. We created this program.
4. The CodeSuite tool produces summary spreadsheets with the distribution of scores in a CodeDiff database.

TESTS

Tests were done on code that was written only in the C programming language. We ran FileFormatter on the source code and converted every single file to whitespace format. When running CodeDiff, we selected the following parameters:

- File reporting threshold: 512.
- Count all scores including 0%.
- Comparison option: percent of first file. We selected this option since we want to have the first directory, as a base for our comparisons. First we compare the same directory to itself then we compare the same directory to another directory.

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 expect to see low scores, but we may 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 distributions and also the average and standard deviation (STD).

Compare One Program to Itself, All Files

The graph in Figure 1 counts all possible file pairs; we can see that it includes pairs with 100% similarity. All 487 files were compared against each other, for a total of 237,169 file pairs.

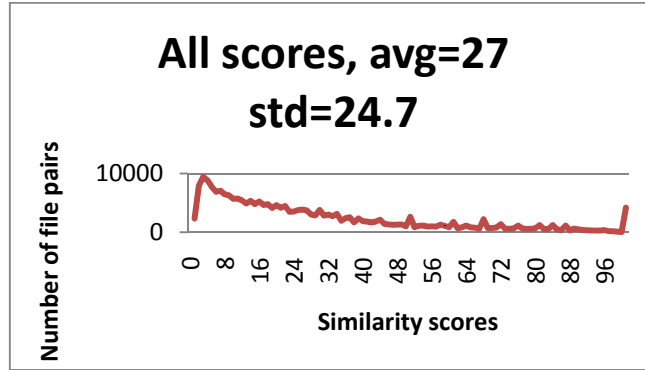


Figure 1: Compare one program to itself, all scores.

There were more than the expected 487 files pairs with scores of 100. There were 4,163 file pairs with 100% score and 2,421 file pairs with scores between 90% and 100%. A few reasons for this could be that header files may be disproportionately similar to each other, small files may be disproportionately similar to each other, and Linux may have duplicate code in different files or duplicate files in different directories that were identified as similar.

We assumed that header files may be similar and disproportionately contributing to the high scores, so we filtered out the scores of the files with extension .h. 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. Filtering both small files and header files left 276 files in the directory and 76,176 total file pairs, as shown in Figure 2. Although the majority of file comparisons fall at the low end, the average score is fairly high (35%) and the large standard deviation (25.2) means there is a big spread among the results as can be seen. Also there are still some 100% matching files.

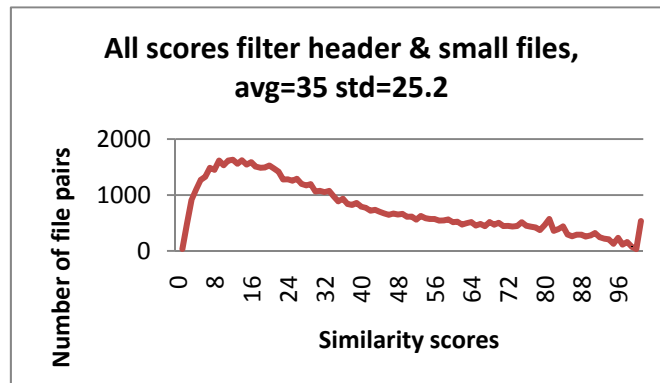


Figure 2: Compare one program to itself, all scores; filter both small files and header files.

Compare One Program to Itself, Highest Similarity File

Looking for copying the investigator looks for the highest matching score for each file in order to find files that are more likely to have been copied. A high matching score indicates high similarity. When comparing one program to itself, for each file we will have at least one 100% matching score. We might have more than one 100% matching scores; this can happen if the same file was copied to another sub directory for example.

We wanted to see what happen when we take out the identical matching scores of each file to itself so we took out the top score from the database and examined this case, which is called "second highest similarity." This case represents a real life scenario when there maybe some similar code, or similar functions, but files are not identical.

Figure 3 is a graph of the highest scoring file pairs for each file in the first directory. All but the highest scoring file pairs were filtered out of the database. The average score is 100 and the STD is 0, which was expected.

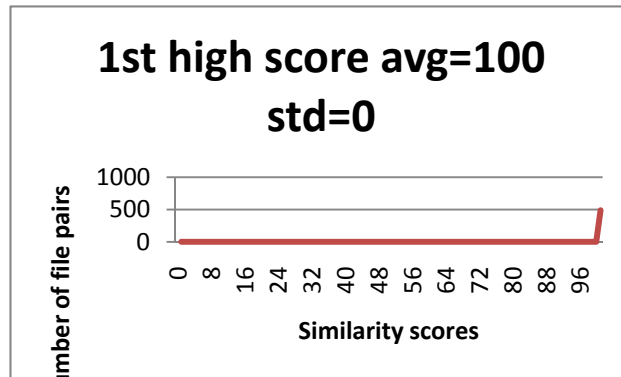


Figure 3: Compare one program to itself, 1st highest score.

Compare One Program to Itself - Second Highest Similarity

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. Even after removing the top matching files, we can see that we have some 100% matching scores.

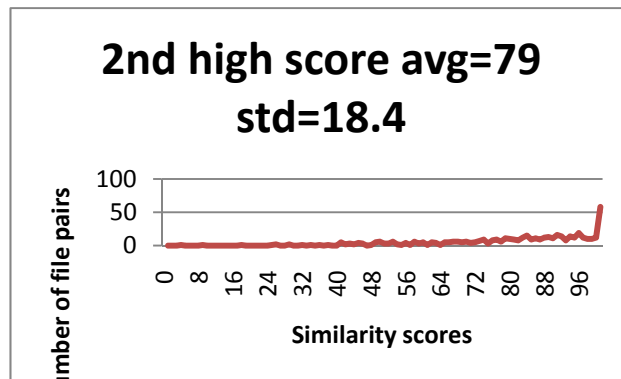


Figure 4: Compare one program to itself, 2nd highest score.

We then filtered out both header files and files with less than 10 lines to produce the data that contain 276 file pairs score for Figure 5. The average is still high, but we can relate this to the fact that since we compared the same program to itself, maybe the same writer wrote the other files so there still may be some similarity in the code. At this point, however, this whitespace pattern matching has not produced a low, narrow peak near zero that would allow us to confirm that these files are not copied from each other.

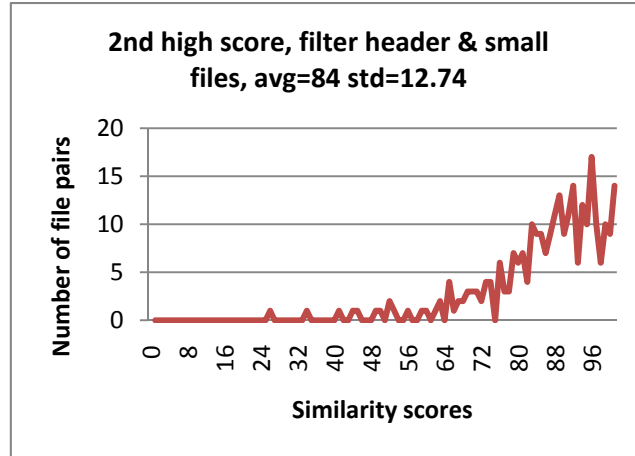


Figure 5: Compare one program to itself, 2nd highest score; filter both small files and header files.

Compare One Program to a Different Program, All Files

The comparison of the 653 Apache server files to the 487 Linux Kernel files produced 318,011 file pairs. The results are shown in Figure 6. The average is low but interesting to see that we still have 100% matching scores.

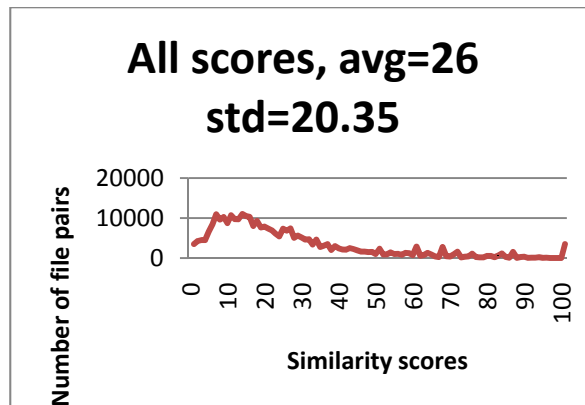


Figure 6: Compare one program to a different program, all scores.

Filtering out small files and header files resulted in a database with 118,956 total file pairs. This shows a low average and low standard deviation, as we would like, but again this is looking at all file combinations, which does not help us detect copied files.

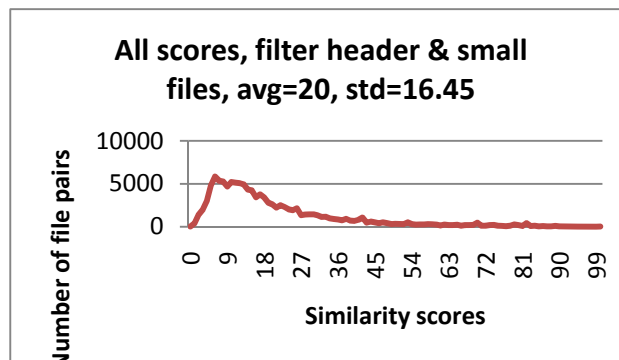


Figure 7: Compare one program to a completely different program, all scores; filter both small and header files.

Compare One Program to a Completely Different Program – First Highest Similarity

We have assumed that the Linux source code files are completely different from the Apache source code files, so the whitespace similarity scores should be very low. We have also assumed that Linux and Apache have few if any identical files, so the first high similarity graph of two different programs should be comparable to the second high similarity graph of a program against itself that was shown in Figure 4. Figure 8 is a graph of the highest scoring file pairs for each of the 487 files in the Linux directory, with 487 file pairs.

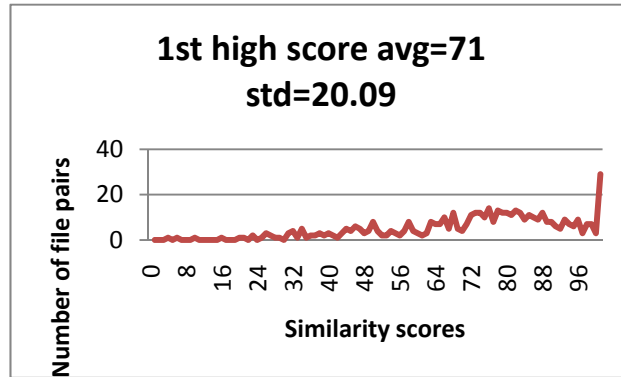


Figure 8: Compare one program to a different program, 1st high score.

The average is high but it is lower than the average when comparing one program to itself. This graph still contains 100% scores.

Filtering out small and header files resulted in a smaller database that has 274 file pairs. The graph in Figure 9 has a high average considering the fact that the compared files are from different programs and we believe them to be developed independently.

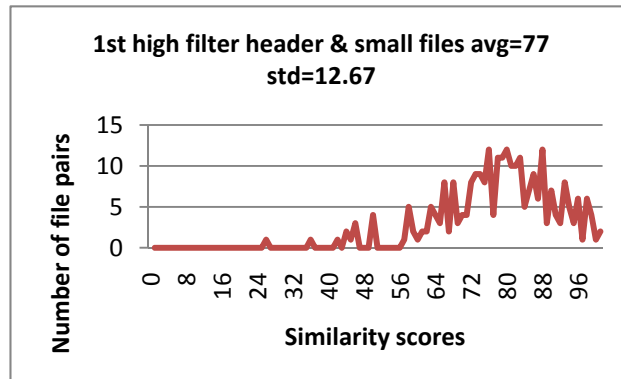


Figure 9: Compare one program to completely different program, 1st high score; filter both small and header files.

Reasons for High Similarity of Different Programs

Looking at the original source code of the similar files, we found that some of the high similarity scores were generated because of:

- Header files. Most of the lines in header file contain variable types and variable names; when translating these lines to whitespace format, these lines are similar.
- Small files. Similarity is calculated as a percentage of the file size, small files will have higher similarity percentages.
- Macros and definition statements. Files that extensively use macros and definitions

may look similar.

- Common program statements. The whitespace translation of two very different lines of code may be the same, especially when common programming statements like 'if' or 'for' are being used.
- Common author. It is possible that the programs actually have some common authors that worked on both programs.
- Third party code. Using third party code in different programs increases the similarity scores.
- Automatic code generation. Using automatic code generator tools like Visual C++ wizard will probably increase the similarity scores.
- Copied routines. Some common routines may have been copied between the two programs.
- Copied files. Some files may have been copied between the two programs.

That may all be reasons for high similarity but probably cannot account for this significant amount of similarity.

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 low end of the graph, but there was a wide distribution rather than a thin peak. Even after attempting various filtering methods, we often had a large spread and a number of different files with identical or nearly identical whitespace patterns.

Even 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 also have high scores. Low scoring file pairs indicate a low level of similarity, and high scoring file pairs do not imply that copying occurred. High scoring file pairs indicate where further investigation for copying might focus.

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. For this we will need to use CodeMatch® function of CodeSuite® instead of CodeDiff®.
- 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 the same program to itself, filter out files with the same name. In this paper we eliminate the top score, which was 100%. But if the same file was copied to a different directory we will get other file pairs with similarity scores of 100%. This filtering will eliminate comparing identical files that were copied to different directories.
- Examine all unseen characters in addition to space and tab.

- Test this whitespace pattern matching method on different 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. We cannot currently eliminate code from code generation tools because we don't know how to identify it from its whitespace pattern.
- Compare two different versions of the same program as a more rigorous test of whitespace pattern comparisons.

AUTHOR BIOS

Ilana Shay Ilana Shay is 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.

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.

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.

REFERENCES

- [1] Abraham, S and Milligan, G. (2008). 'Software Plagiarism in Undergraduate Programming Classes'. Information Systems Education Conference. 11/6/2008-11/9/2008. Phoenix.
- [2] Aliefendic, S. (2009). 'Using whitespace patterns to detect plagiarism in program code'. Department of Computer Science. School of Computer Science and Informatics, University College. Dublin.
- [3] Baker, B. (1995). 'On Finding Duplication and Near-Duplication in Large Software Systems'. Second Working Conference on Reverse Engineering. 7/14/1995-7/16/1995. Toronto, Canada.
- [4] Clough, P. (2000). 'Plagiarism in natural and programming languages: an overview of current tools and technologies, Research Memoranda'. CS-00-05, Department of Computer Science, University of Sheffield. UK.
- [5] Goel, S., Rao, D., et al. (2008). 'Plagiarism and its Detection in Programming Languages'. Technical Report, Department of Computer Science and Information Technology, JIITU.
- [6] Hamilton, J. (2008). 'Static Source Code Analysis Tools and their Application to the Detection of Plagiarism in Java Programs'. Department of Computing at Goldsmiths, University of London.
- [7] Parker, A. and Hamblen, J. (1989). 'Computer Algorithms for Plagiarism Detection'. IEEE Transactions on Education Vol. 32, No. 2. May, pages 94-99.
- [8] Sallis, P., Aakjaer, A., and MacDonell, S. (1996). 'Software forensics: old methods for a new science'. International Conference on Software Engineering: Education and Practice. 1/24/1996-1/27/1996. Dunedin, New Zealand.

- [9] Whale, G. (1990). "Identification of Program Similarity in Large Populations", *The Computer Journal*, Vol. 33, Issue 2: Pages 140-146.
- [10] Zeidman, R. (2006). 'Software Source Code Correlation'. 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR'06). 7/10/ 2006-7/12/2006. Honolulu, HI.
- [11] Zeidman, R. (2008). 'Multidimensional Correlation of Software Source Code', Third International Workshop on Systematic Approaches to Digital Forensic Engineering. 5/22/2008. Oakland, CA.

