Full citation: Frantzeskou, G., Gritzalis, S., & MacDonell, S. (2004) Source code authorship analysis for supporting the cybercrime investigation process, in Proceedings of the 1st International Conference on E-Business and Telecommunication Networks. Setúbal, Portugal, INSTICC Press, pp. 85-92.

SOURCE CODE AUTHORSHIP ANALYSIS FOR SUPPORTING THE CYBERCRIME INVESTIGATION PROCESS

Georgia Frantzeskou, Stefanos Gritzalis
Laboratory of Information and Communication Systems Security, Aegean University
Department of Information and Communication Systems Engineering
Karlovasi, Samos, 83200, Greece
{gfran, sgritz}@aegean.gr

Stephen MacDonell
SERL, Auckland University of Technology
Private Bag 92006, Auckland 1142, New Zealand
stephen.macdonell@aut.ac.nz

Abstract

Cybercrime has increased in severity and frequency in the recent years and because of this, it has become a major concern for companies, universities and organizations. The anonymity offered by the Internet has made the task of tracing criminal identity difficult. One study field that has contributed in tracing criminals is authorship analysis on e-mails, messages and programs. This paper contains a study on source code authorship analysis. The aim of the research efforts in this area is to identify the author of a particular piece of code by examining its programming style characteristics. Borrowing extensively from the existing fields of linguistics and software metrics, this field attempts to investigate various aspects of computer program authorship. Source code authorship analysis could be implemented in cases of cyber attacks, plagiarism and computer fraud. In this paper we present the set of tools and techniques used to achieve the goal of authorship identification, a review of the research efforts in the area and a new taxonomy on source code authorship analysis.

Keywords: Authorship Analysis, Software Forensics, Plagiarism.

1. INTRODUCTION

Computers and networks have played an important role in peoples' everyday life over the last decade. But while computers have made our lives easier and have improved our standard of living, have also introduced a new venue of criminal activities.

Cyber attacks in the form of viruses, trojan horses, logic bombs, fraud, credit card cloning, plagiarism of code have increased in severity and frequency. Once forensic investigators have identified the piece of software responsible for the attack we might want to try to locate its source (Krsul, and Spafford, 1996).

In an attempt to deal in a more formal way to tackle these problems, Spafford and Weeber suggested that a technique they called *software forensics* could be used to examine and analyze software in any form, source or executable code, to identify the author (Spafford, and Weeber, 1993).

But why do we believe it is possible to identify the author of a computer program? Humans are creatures of habit and habits tend to persist. That is why, for example, we have a handwriting style that is consistent during periods of our life, although the style may vary, as we grow older. Does the same apply to programming?

Although source code is much more formal and restrictive than spoken or written languages, there is still a large degree of flexibility when writing a program (Krsul, and Spafford, 1996). This flexibility includes characteristics that deal with the layout of the program (placement of comments, indentation), characteristics that are more difficult to change automatically by pretty printers and code formatters, and deal with the style of the program (comment lengths, variable names, function names) and features that we hypothesize are dependent on the programming experience (the statistical distribution of lines of code per function, usage of data structures). Research studies on this field have proved that many of these features (layout, style, structure) of computer program can be specific to a programmer. Section 2 contains a revised categorisation on applications areas of the field, section 3 is an overview of tools and techniques available, section 4 contains a review of the area and section 5 a new taxonomy.

2. BACKGROUND

2.1 Motivation

As the incidence of computer related crime increases it has become increasingly important to have techniques that can be applied in a legal setting to assist the court in

making judgements (Gray et al. 1997). Some types of these crimes include attacks from malicious code (such as viruses, worms, trojan horses, and logic bombs) and computer fraud.

Another widely known example of authorship analysis is plagiarism detection. In the academic community, it is considered unethical to copy programming assignments (MacDonell et al. 1999). Using this technique, assignments can be compared to see if some are "suspiciously similar". Authorship analysis could also be applied in psychological studies of the relationship between programmer attributes and their code (Spafford 1989).

In the commercial world when a specific program module or program needs to be maintained the author may need to be located. It would be convenient to be able to determine the name of the programmer from a set of several hundred programmers, which is not otherwise recorded or may be incorrect

Some of these problems are already faced with a variety of techniques (Gray et al. 1997). The creation of a new field with its own methods and tools, called *software forensics*, has helped to tackle these issues in a proper way and not in an *ad hoc* manner. The term software forensics implies the use of these tools and methods for some legal or official purpose.

2.2 Where could it be used?

Source code authorship analysis can be divided into 5 sub-fields according to the application area. This categorisation is an extended version of Gray's et al. (1997) work.

1. *Author identification*. The aim here is to decide whether some piece of code was written by a certain programmer. This goal is accomplished by comparing this piece of code against other program samples written by that author. This type of application area has a lot of similarities with the corresponding literature where the task is to determine that a piece of work has been written by a certain author.

2. *Author characterisation*. This application area determines some characteristics of the programmer of a piece of code, such as cultural educational background and language familiarity, based on their programming style.

3. *Plagiarism detection*. This method attempts to find similarities among multiple sets of source code files. It is used to detect plagiarism, which can be defined as the use of another person's work without proper acknowledgement.

4. *Author discrimination*. This task is the opposite of the above and involves deciding whether some pieces of code were written by a single author or by some number of authors. An example of this would be showing that a program was probably written by three different authors, without actually identifying the authors in question.

5. Author intent determination. In some cases we need to know whether a piece of code, which caused a malfunction, was written having this as its goal or was the result of an accidental error. In many cases, an error during the software development process can cause serious problems.

3. THE PRACTICE OF PROGRAM AUTHORSHIP ANALYSIS

3.1 Overview

The essence of authorship analysis is locating some features that most likely remain constant among a set of programs written by the same author (*Metrics extraction*). The next step is using these source code features to develop models that are capable of discriminating between several authors (*Data analysis & classification*).

3.2 Metrics extraction

Based on general appearance of the code or the use of programming idioms, expert opinion can, potentially, be given on the degrees of similarity and difference between code fragments (MacDonell et al. 1999). However, a more scientific approach may also be taken since both quantitative and qualitative measurements can be made on computer program source code and object code. These measurements are referred to as *metrics* and most of them are borrowed and/or adapted from the field of software metrics and have primarily been used for software process estimation.

Authorship analysis is based on the construction of an *author profile* (Sallis et al. 1996), using a comprehensive set of such metrics. The profile for a given programmer is likely to include metrics relating to product size, structure, layout, and expression. Ideally, such metrics should have low within-programmer variability, and high between-programmer variability.

We could divide the metrics used for authorship analysis into 4 sub-categories. The first three belong to quantitative metrics category and the last on the qualitative metrics category: (Krsul and Spafford, 1996), (Kilgour et al., 1997).

Programming layout metrics include those metrics that deal with the layout of the program. For example metrics that measure indentation, placement of comments, placement of braces etc. These metrics are fragile because the information required can be easily changed using code formatters. Also many programmers learn programming in university courses that impose a specific set of style rules regarding indentations, placement of comments etc.

Programming style metrics are those features that are difficult to change automatically by code formatters and are also related to the layout of the code. For example such metrics include character preferences, construct preferences, statistical distribution of variable lengths and function name lengths etc.

Programming structure metrics include metrics that we hypothesize are dependent on the programming experience and ability of the programmer. For example such metrics include the statistical distribution of lines of code per function, ratio of keywords per lines of code etc.

Fuzzy logic metrics include variables that they allow the capture of concepts that programmers can identify with, such deliberate versus non deliberate spelling errors, the

degree to which code and comments match, and whether identifiers used are meaningful.

Measurements in the first three categories are automatically extracted from the source code using pattern matching algorithms. These metrics are primarily used in managing the software development process, but many are transferable to authorship analysis. Fuzzy logic metrics cannot be extracted in an automatic way and expert intervention is required.

It is possible to perform authorship analysis on the executable code, which is the usual form of an attack in the form of viruses, trojan horses, worms etc. In order to perform such analysis executable code is decompiled (Gray et al., 1997), a process where a source program is created by reversing the compiling process. Although there is a considerable information loss during this process there are many code metrics still applicable, such as compiler and system information, level of programming skill and areas of knowledge.

3.3 Data analysis & classification

Once these metrics have been extracted, a number of different modelling techniques, such as neural networks, discriminant analysis, case based reasoning can be used to develop models that are capable of discriminating between several authors (MacDonell et al., 1999).

3.3.1 Discriminant Analysis

Discriminant analysis (SAS) is a statistical technique that uses continuous variable measurements on different groups of items to highlight aspects that distinguish the groups and to use these measurements to classify new items. This technique is the most widely used for source code authorship classification.

An important advantage of the technique (MacDonell et al., 1999) is the availability of stepwise procedures for controlling the entry and removal of variables. By working with only those necessary variables we increase the chance of the model being able to generalize to new sets of data.

3.3.2 Neural Networks

Artificial Neural Networks (ANN) are computational models that try to emulate the behavior of the human brain (Mair et al., 2000). They are based on a set of simple processing elements, highly interconnected, and with a massive parallel structure. Some of the characteristics of neural networks are their learning, adapting and generalization capabilities. Feed-Forward Neural Networks (FFNNs) are the most commonly used form of ANNs and have been used in source code authorship analysis (MacDonell et al., 1999).

3.3.3 Case Based Reasoning

CBR is a machine learning method originating in analogical reasoning, and dynamic memory and the role of previous situations in learning and problem solving (Schank, 1982). Cases are abstractions of events (solved or unsolved problems), limited in time and space.

Aarmodt and Plaza (1994) describe CBR as being cyclic and composed of four stages, the *retrieval* of similar cases, the *reuse* of the retrieved cases to find a solution to the problem, the *revision* of the proposed solution if necessary and the *retention* of the solution to form a new case.

When a new problem arises, a possible solution can be found by retrieving similar cases from the case repository. The solution may be revised based upon experience of reusing previous cases and the outcome retained to supplement the case repository. One particular case-based reasoning system that has been previously used for software metric research and in source code authorship analysis is the ANGEL system (Shepperd and Schofield, 1997).

3.3.4 Manual Approach

This approach involves examination and analysis of a piece of code by an expert. The objective is to draw conclusions about the authors' characteristics such as educational background, and technical skill. This technique can also be used also in combination with an automated approach (Kilgour et al., 1997), in order to derive fuzzy-logic linguistic variables to capture more subjective elements of authorship, such as the degree to which comments match the actual source code's behaviour etc.

3.3.5 Similarity Calculation

This approach uses a set of numeric metric values or token strings (Verco and Wise, 1996) to represent each program. Based on these values programs are being compared in order to produce a measure that quantifies how close these programs are (Jones, 2001).

4. REVIEW OF RELATED WORK

Primarily authorship analysis studies have been performed in text and later this technique has been applied to computer programs. We now review previous research done in each of these areas keeping our focus on source code authorship analysis.

4.1 Text authorship analysis

The most extensive and comprehensive application of authorship analysis is in literature. One famous authorship analysis study is related to Shakespeare's works and is dating back over several centuries. Recently Elliot and Valenza (1991) compared the poems of Shakespeare and those of Edward de Vere, 7th Earl of Oxford, where attempts were made to show that Shakespeare was a hoax and that the real author was Edward de Vere, the Earl of Oxford. In this study, specific author features such as unusual diction, frequency of certain words, choice of rhymes, and habits of hyphenation have been used as tests for author attribution. The results indicated significant differences between the works of the two authors, which denied the claim that Edward de Vere was indeed Shakespeare. A similar study had been carried out on the disputed Federalist papers (Mosteller and Wallace, 1964), (Bosch, and Smith, 1998). Mosteller and Wallace (1964) adopted a statistical inference method to analyze the paper contents, while Bosch and Smith (1998) used linear programming techniques to find a separating hyperplane based on various combinations of 70 function words.

Both studies reached the same conclusion that the papers were written by Madison, one of the two authors in dispute. Diederich (2000) applied for first time a machine learning technique called Support Vector Machine (SVM) to this problem. He performed a number of experiments with texts from a German newspaper. With nearly perfect reliability the SVM was able to reject other authors and detected the target author in 60-80% of the cases.

Text authorship analysis has also been applied in the context of criminal investigation. The analysis of the Unabomber manifesto is an example of using linguistics metrics (e.g. word usage) along with manual and statistical analysis to attribute a piece of work to a particular author. In this case, the manifesto and the suspect terrorist, Theodore Kaczynski, shared similar characteristics, such as a distinctive vocabulary, irregular hyphenations, etc (Foster, 2001).

A new area of study is the identification and characterisation of electronic message authors based on message contents. De Vel et al (2001) evaluated author attribution performance in the context of multiple e-mail topic categories. The same authors have also undertaken authorship characterization and in particular authorship gender (male or female) and language background (English as first or second language) cohort attribution. In both cases they used structural and stylometric features and in the later experiment they used in addition, a set of gender-preferential language attributes. A machine learning approach was adopted and the SVM was employed as the learning algorithm. The experiments gave promising results.

4.2 Source code authorship analysis

On the evening of 2 November 1988, someone infected the Internet with a *worm* program. Spafford (1989) conducted an analysis of the program using three reversed-engineered versions. Coding style and methods used in the program were manually analyzed and conclusions were drawn about the author's abilities and intent. Following this experience, Spafford and Weeber (1993) suggested that it might be feasible to analyze the remnants of software after a computer attack, such as viruses, worms or trojan horses, and identify its author. This technique, called software forensics, could be used to examine software in any form to obtain evidence about the factors involved. They investigated two different cases where code remnants might be analyzed: executable code and source code. Executable code, even if optimized, still contains many features that may be considered in the analysis such as data structures and algorithms, compiler and system information, programming skill and system knowledge, choice of system calls, errors, etc. Source code features include programming language, use of language features, comment style, variable names, spelling and grammar, etc.

Cook and Oman (1989) used "markers" based on typographic characteristics to test authorship on Pascal programs. The experiment was performed on 18 programs written by six authors. Each program was an implementation of a simple algorithm and it was obtained from computer science textbooks. They claimed that the results were surprisingly accurate.

Longstaff and Shultz (1993) studied the WANK and OILZ worms which in 1989 attacked NASA and DOE systems. They have manually analyzed code structures and features and have reached a conclusion that three distinct authors worked on the worms. In addition, they were able to infer certain characteristics of the authors, such as their educational backgrounds and programming levels. Sallis et al (1997) expanded the work of Spafford and Weeber by suggesting some additional features, such as cyclomatic complexity of the control flow and the use of layout conventions.

An automated approach was taken by Krsul and Spafford (1995) to identify the author of a program written in C. The study relied on the use of software metrics, collected from a variety of sources. They were divided into three categories: layout, style and structure metrics. These features were extracted using a software analyzer program from 88 programs belonging to 29 programmers. A tool was developed to visualize the metrics collected and help select those metrics that exhibited little within-author variation, but large between-author variation. A statistical approach called discriminant analysis (SAS) was applied on the chosen subset of metrics to classify the programs by author. The experiment achieved 73% overall accuracy.

Other research groups have examined the authorship of computer programs written in C++ (Kilgour et al., 1997); (MacDonell et al. 1999), a dictionary based system called IDENTIFIED (integrated dictionary- based extraction of non-language-dependent token information for forensic identification, examination, and discrimination) was developed to extract source code metrics for authorship analysis (Gray et al., 1998). Satisfactory results were obtained for C++ programs using case-based reasoning, feed-forward neural network, and multiple discriminant analysis (MacDonell et al. 1999).

Ding (2003), investigated the extraction of a set of software metrics of a given Java source code, that could be used as a fingerprint to identify the author of the Java code. The contributions of the selected metrics to authorship identification were measured by a statistical process, namely canonical discriminant analysis, using the statistical software package SAS. A set of 56 metrics of Java programs was proposed for authorship analysis. Forty-six groups of programs were diversely collected. Classification accuracies were 62.7% and 67.2% when the metrics were selected manually while those values were 62.6% and 66.6% when the metrics were chosen by SDA (stepwise discriminant analysis).

4.3 Plagiarism Detection

Plagiarism detection is another field closely related to the problem of authorship analysis, especially authorship categorization and similarity detection. Jones (2001), offered a useful definition of plagiarism detection, characterising it as a problem of pattern analysis, based on plagiarising transformations, which have been applied to a source file. Such transformations include "verbatim copying, changing comments, changing white space and

formatting, renaming identifiers, reordering code blocks, reordering statements within code blocks, changing the order of operands/operators in expressions, changing data types, adding redundant statements or variables, replacing control structures with equivalent structures".

One of the earliest set of techniques for plagiarism detection in software is the *attribute counting* techniques which count the level of a certain attribute contained within a piece of code. These systems use a number of metrics such as Halstead's software science metrics (Halstead, 1977), McCabe's cyclomatic complexity (McCabe, 1976), the nesting depth (Dunsmore, 1984) etc. The first automated system used Halstead's metrics for plagiarism detection and has been developed by Ottenstein (1979). Other examples of attribute counting system include the work of Berghell, and Sallach, (1984), Grier's Accuse system (1981). This approach was at best moderately successful (Verco, and Wise, 1996), because "summing up a metric across the whole program throws away too much structural information".

More recent approaches named *structure metrics techniques*, which as Clough (2000) writes, "compare string representations of the program structure", are assessing "the similarity of token strings". Examples of these include Plague, Sim, YAP, and JPlag.

The sim plagiarism detection system (Grune, 1991) developed by Dick Grune converts the source programs into token strings and then in pairs finds matching substrings of decreasing lengths The YAP family approaches (Wise, 1992), (Wise, 1996), uses the source code to generate token sequences by removing comments, translating upper case letters to lower case, mapping synonyms to a common form, reordering the functions into their calling order and by removing all tokens that are not from the lexicon of the target language. The next step is to apply an algorithm where each token string is (non-redundantly) compared with all the others. The biggest change that has occurred in the latest version of YAP, YAP3, is a switch to the underlying use of the *Running-Karp-Rabin Greedy-String-Tiling* (*RKR-GST*) algorithm which allows the system to detect transposed subsequences, JPlag (Prechelt, 2002) uses the same basic comparison algorithm, the Greedy-String-Tiling (GST) as YAP3, but uses a different set of optimizations for improving its run time efficiency. Plague (Whale, 1990) works in a similar fashion to the *YAP3* method discussed previously, but without using the *RKR-GST* algorithm.

A different set of approaches include the work proposed by Jankowitz (1988) on a model for detecting plagiarism in student Pascal programs, where a template was constructed for each program, using elements like programming style features and the order in which procedures are referenced during static execution. All templates were compared against each other and similar regions were extracted from the programs. Statistical analysis was then performed on those common regions to characterize the students' programming styles. Jones (Jones 2001) in order to detect program similarities has created metrics based physical and Halstead program profiles. Closeness was computed as the normalized Euclidean distance between profiles.

5. TAXONOMY

A new taxonomy of source code authorship analysis is presented, which is a modified and expanded version of the taxonomy developed by Zheng et al (2003).

Table 1: Taxonomy for Source Code Authorship Analysis

1 Problem	
Category	**Description**
Authorship Identification	Aims to determine whether a piece of code was written by a certain author.
Authorship Characterization	Based on the programming style and techniques used determines some characteristics of the programmer of a piece of code, such as cultural educational background and language familiarity.
Plagiarism Detection	This method attempts to find similarities among multiple sets of source code files. It is used to detect plagiarism, which can be defined as the use of another person's work without proper acknowledgement.
Author intent determination	We need to know whether a piece of code which caused a malfunction was written having this as its goal or was the result of an accidental error.
Author discrimination	Determines whether some pieces of code were written by a single author or by some number of authors.
2 Approach	
Category	**Description**
Manual Analysis	This approach involves examination and analysis of a piece of code by an expert. It can be used to draw conclusions about the authors' characteristics such as educational background, and technical skill.
Similarity Calculation	Uses a set of numeric metric values or token strings (Verco, and Wise, 1996) to represent each program. Based on these values programs are being compared in order to produce a measure that quantifies how close these programs are (Jones, 2001).
Statistical Analysis	Uses statistical techniques such as discriminant analysis in order to investigate differences between authors of programs and to discriminate authors effectively.
Machine Learning	Uses methods such as Case Base Reasoning and Neural networks to predict the author of a piece of code based on a set of metrics.

6. CONCLUSIONS

It seems that source code authorship analysis is an important area of practice in computer security, computer law, and academia as well as an exciting area of research. The experiments that have been performed support the theory that it is possible to find a set of metrics that can be used to classify programmers correctly. Within a closed environment, and with a limited number of programmers, it is possible to identify authorship of a program by examining some finite set of metrics. As part of this development in the field there is the necessity for more formally defined methods and metrics specifically used in this area. Further work will be to enrich the set of metrics in order to improve classification accuracy. An example could be introducing object oriented metrics when examining authorship in C++ or Java. Also by employing other machine learning techniques or statistical methods such as Bayesian techniques, we could produce better results.

7. REFERENCES

Aarmodt, A., and Plaza, E., 1994, *Case-Based Reasoning: Foundational issues, Methodical Variations and System Approaches*. AI Communications, vol 7(1).

Bosch, R., and Smith, J., 1998, *Separating hyperplanes and the authorship of the disputed federalist papers*, American Mathematical Monthly, 105(7):601-608, 1998.

Berghell, H., L., and Sallach, D., L., 1984, *Measurements of Program Similarity in Identical Task Environments*, SIGPLAN Notices 19(8), pp. 65-75.

Clough, P., 2000, *Plagiarism in natural and programming languages: an overview of current tools and technologies*, Department of Computer Science, University of Sheffield.

Diederich, J., Kindermann, J., Leopold, E., and Paass, G., 2000, *Authorship attribution with Support Vector Machines*, Applied Intelligence (Submitted).

Ding, H., Samadzadeh, M., H., 2003, *Extraction of Java program fingerprints for software authorship identification*, The Journal of Systems and Software, article under press.

Dunsmore, 1984, *Software metrics: an overview of an evolving methodology*, Information Processing and Management 20, pp. (183-192).

Elliot, W., and. Valenza, R., 1991, *Was the Earl of Oxford The True Shakespeare?*, Notes and Queries, 38:501-506.

Foster, D., 2001, *Author Unknown: On the Trail of Anonymous*, Henry Holt, New York.

Faidhi, J., A., and Robinson, S., K., 1987, *An Approach for Detecting Program Similarity within a University Programming Environment*, Computers and Education 11(1), pp. 11-19.

Grier, S., 1981, *A Tool that Detects Plagiarism in Pascal Programs*, Twelfth SIGCSE Technical Symposium, St Louis, Missouri, pp. 15–20 (February 26-27, 1981) (SIGCSE Bulletin Vol. 13, No. 1, February 1981).

Grune, D., 1991, *Concise Report on Algorithms in Sim*, (Report distributed with Sim software).

Gray, A., Sallis, P., and MacDonell, S., 1997, *Software forensics: Extending authorship analysis techniques to computer programs*, in Proc. 3rd Biannual Conf. Int. Assoc. of Forensic Linguists (IAFL'97), pages 1-8.

Gray, A., Sallis, P., and MacDonell, S., 1998, *Identified (integrated dictionary-based extraction of non-language-dependent token information for forensic identification, examination, and discrimination): A dictionary-based system for extracting source code metrics for software forensics*. In *Proceedings of SE:E&P'98 (Software Engineering: Education and Practice Conference)*, pages 252–259. IEEE Computer Society Press.

Halstead, M., H., 1977, *Elements of software science*, North Holland, New York.

Jankowitz, H. T., 1988, *Detecting Plagiarism in Student Pascal Programs*, Computer Journal, 31(1).

Jones, E., L., 2001, *Metrics Based Plagiarism Monitoring*, in Proc. Consortium for Computing in Small Colleges

Kilgour, R. I., Gray, A.R., Sallis, P. J., and MacDonell, S. G., 1997. *A Fuzzy Logic Approach to Computer Software Source Code Authorship Analysis*, Accepted for The Fourth International Conference *on* Neural Information Processing -- The Annual Conference of the Asian Pacific Neural Network Assembly (ICONIP'97). Dunedin. New Zealand

Krsul, I., and Spafford, E. H., 1995, *Authorship analysis: Identifying the author of a program*, In Proc. 8th National Information Systems Security Conference, pages 514-524, National Institute of Standards and Technology.

Krsul, I., and Spafford, E. H., 1996, *Authorship analysis: Identifying the author of a program*, Technical Report TR-96-052, 1996

Longstaff, T. A., and Schultz, 1993, E. E., *Beyond Preliminary Analysis of the WANK and OILZ Worms: A Case Study of Malicious Cod*e, Computers and Security, 12:61-77.

McCabe, T. J., 1976, *A complexity measure*, IEEE Transactions on Software Engineering, SE-2 (4), pp(308-320).

MacDonell, S.G., Gray, A.R., MacLennan, G., Sallis, P.J., 1999.*Software forensics for discriminating between program authors using case- based reasoning, feed forward neural networks, and multiple discriminant analysis*. In: Proceedings of the 6th International Conference on Neural Information, vol. 1, Dunedin, New Zealand, pp. 66–71.

Mair, C., Kadoda, G. Lefey, M., Phalp, K., Schofield , C., Shepperd, M., Webster, S., 2000, *An investigation of machine learning based prediction systems* The Journal of Systems and Software 53 23-29.

Mosteller, F., and Wallace, D., 1964, *Inference and Disputed Authorship: The Federalist*, Addison-Wesley, Reading, Mass.

Oman, P., and Cook, C., *Programming style authorship analysis*. In Seventeenth Annual ACM Science Conference Proceedings, pages 320–326. ACM, 1989.

Ottenstein, L., M., *Quantitative estimates of debugging requirements*, 1979, IEEE Transactions of Software Engineering, Vol. SE-5, pp(504-514).

Prechelt, L., Malpohl, G., Philippsen, M., *Finding Plagiarisms among a Set of Programs with JPlag*, Journal of Universal Computer Science, vol. 8, no. 11 (2002), 1016-1038

Sallis P., Aakjaer, A., and MacDonell, S., 1996, Software Forensics: Old Methods for a New Science. *Proceedings of SE:E&P'96 (Software Engineering: Education and Practice)*. Dunedin, New Zealand, IEEE Computer Society Press, 367-371.

SAS on line docs http://www.sasdocs.utoledo.edu/ last accessed 12/1/2004

Schank, R., 1982,. *Dynamic Memory: A theory of reminding and learning in computers and people.* Cambridge University Press.

Spafford, E. H., 1989, *The Internet Worm Program: An Analysis*," Computer Communications Review, 19(1): 17-49.

Shepperd, M. J., and Schofield, C., 1997, *Estimating software project effort using analogies*, IEEE Transactions on Software Engineering, 23(11), 736-743.

Spafford, E. H., and Weeber, S. A., 1993, *Software forensics: tracking code to its authors*, Computers and Security, 12:585-595.

Verco, K. K., and Wise, M. J., 1996, *Software for detecting suspected plagiarism: Comparing structure and attribute-counting systems*, In John Rosenberg, editor, Proc. of 1st Australian Conference on Computer Science Education, Sydney, ACM.

Vel, O., Anderson, A., Corney, M., and Mohay, G., 2001, *Mining E-mail Content for Author Identification Forensics*, SIGMOD Record, 30(4): 55-64.

Whale, G., 1990, *Identification of Program Similarity in Large Populations*, *The Computer Journal* **33**(2), pp. 140–146.

Wise, M., J., 1992, *Detection of Similarities in Student Programs: YAP'ing may be Preferable to Plagueing*, Proceedings, Twenty Third SCGCSE Technical Symposium, Kansas City, USA, 268-271.

Wise, M. J., 1996, *Improved Detection of Similarities in Computer Program and other Texts*, Twenty-Seventh SIGCSE Technical Symposium, Philadelphia, U.S.A., pp. 130-134.

Zheng, R., Qin, Y., Huang, Z., and Chen H., 2003, Authorship Analysis in Cybercrime Investigation Springer-Verlag Heidelberg, ISSN: 0302-9743, Volume 2665.