IDENTIFIED: Software Authorship Analysis with Case-Based Reasoning

Philip Sallis, Stephen G. MacDonell, Grant MacLennan, Andrew Gray, and Richard Kilgour

Department of Information Science
University of Otago, PO Box 56, Dunedin, New Zealand
psallis@commerce.otago.ac.nz

Abstract

Software forensics is the use of authorship analysis techniques to analyse computer programs for a legal or official purpose. This generally consists of plagiarism detection and malicious code analysis. IDENTIFIED is a system that has been designed to assist with the extraction of count-based metrics from source code, and with the development of models of authorship using statistical and machine-learning approaches. Software forensic models can be used for identification, classification, characterisation, and intent analysis. One of the more promising methods for identification is case-based reasoning, where samples of code can be compared to those collected from known authors.

1. SOFTWARE FORENSICS

The frequency and severity of the many forms of computer-based attacks such as viruses and worms, logic bombs, trojan horses, computer fraud, and plagiarism of software code (both object and source) have all become increasingly prevalent and costly for many organisations and individuals involved with information systems. Part of the difficulty experienced in collecting evidence regarding the attack or theft in such situations has been the definition of appropriate measurements to use in models of authorship and the development of appropriate models from these metrics.

Several options for developing such models for identification, discrimination, characterisation, and intent analysis exist, including subjective expert opinion (including fuzzy logic as in [Kilgour, et al., 1997]) and statistical and machine learning models using formally defined metrics. Each offers its own set of advantages and disadvantages for the task of software forensics.

With the difficulties of data collection and the goal of increasing the accessibility of such modelling techniques in mind a system called IDENTIFIED is being developed. It is intended to assist with the task of software forensics which is defined here to be the use of software code authorship analysis for legal or official purposes [Gray, et al., 1997]. IDENTIFIED uses combinations of wildcards and special characters to define count-based metrics, allows for hierarchical meta-metric definitions, automates much of the file handling task, extracts metric values from source code, and assists with the analysis and modelling process. In particular IDENTIFIED will enable the analyst to use several analysis procedures such as case-based reasoning. It is hoped that the availability of such tools will encourage more detailed research into this area of ever-increasing importance.

2. SOFTWARE FORENSICS

Source code is the textual form of a computer program that is written by a computer programmer in a computer programming language. These programming languages can in some respects be treated as a form of language from a linguistic perspective, or more precisely as a series of languages of particular types, but within some common family. In the same manner that written text can be analysed for evidence of authorship (such as [Sallis, 1994]), computer programs can also be examined from a forensics or linguistics viewpoint [Sallis, et al. 1996] for information regarding the program's authorship.

Figure 1 [from Gray et al. 1997] shows two small code fragments that were written in C++ by two separate programmers. Both programs provide the same functionality (calculating the mathematical function *factorial(n)*, normally written as *n!*) from the users' perspective. That is to say, the same inputs will generate the same outputs for each of these programs.

As should be apparent, each programmer has solved the same problem, that of calculating the factorial of an input, in both a different manner (algorithm) and with a different style exhibited in his or her code. These stylistic differences include the use of comments, variable names, use of white space, indentation, and the levels of readability in each function.

These fragments are obviously far too short to make any substantial claims about the feasibility of using source code characteristics to make statements regarding the author(s). However, they do illustrate the fact that programmers writing programs will often do so in a significantly different manner to another programmer, without any instruction to do so. Both of these functions were written in the natural styles of their respective authors.

```
// Factorial takes an integer as an input and returns
// the factorial of the input.
// This routine does not deal with negative values!

int Factorial (int Input)
{
       int Counter;
       int Fact;
       Fact=1;  // Initalises Fact to 1 since factorial 0 is 1
       for (Counter=Input; Counter>1; Counter=Counter-1)
       {
                    Fact=Fact*Counter;
       }
       return Fact;
}
```

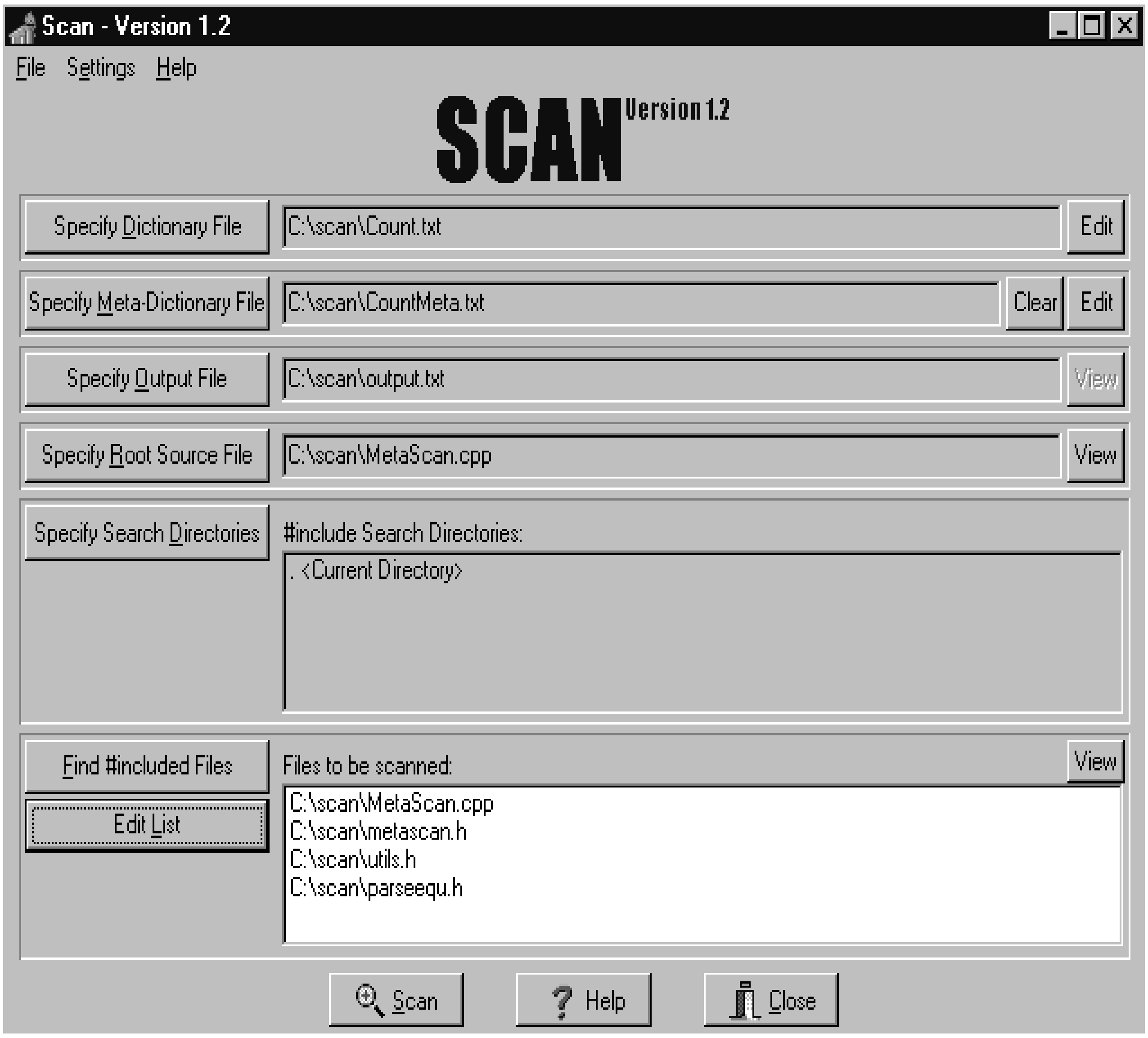
```
int f(int x){
int a, y=1;
if (!x) return 1; else return   x*f(x-1);}
```

Figure 1. Program segments in C++

A large number of candidate metrics for capturing authorship information have been proposed in the literature, but rather than discussing these in detail here we will instead focus on the IDENTIFIED system. For more information regarding the metrics themselves the interested reader should consult [Sallis, et al., 1996], [Kilgour, et al., 1997], and [Gray, et al., 1997]

3. IDENTIFIED

IDENTIFIED (Integrated Dictionary-based Extraction of Non-language-dependent Token Information for Forensic Identification, Examination, and Discrimination) is a, currently, prototype implementation of a dictionary-based metric extraction tool. IDENTIFIED also provides modules for analysing the resultant metric data, including case-based reasoning. Figure 2 shows the Scan Module of IDENTIFIED. The overall structure of IDENTIFIED is as shown in Figure 3.



Specify Dictionary File: C:\scan\Count.txt — Edit

Specify Meta-Dictionary File: C:\scan\CountMeta.txt — Clear — Edit

Specify Output File: C:\scan\output.txt — View

Specify Root Source File: C:\scan\MetaScan.cpp — View

Specify Search Directories — #include Search Directories:
. <Current Directory>

Files to be scanned:
C:\scan\MetaScan.cpp
C:\scan\metascan.h
C:\scan\utils.h
C:\scan\parseequ.h

Figure 2. The Scan Module of IDENTIFIED

The system works in the following manner:

1. The user specifies a dictionary file that using a system of special codes defines count-based metrics to be extracted. For example, a metric may be defined to count the number of lines of code, the number of temporary type variables (indicated as *temp** perhaps), or the number of *while* structures.
2. The user then can optionally specify a meta-dictionary file that defines metrics in terms of the lower-level counts. For example, the ratio of *for* to *while* loops, the proportion of lines that are comments, and the number of closing brackets on the same line as a statement.
3. The user then specifies the base source code file and the directories in which to search for files that this base file depends on, either directly or indirectly. This allows for the user to avoid system header files. Other options such as date ranges may also be used.
4. Once these files have been specified the user can request the system to scan the source code and report the metrics, either on a system level or for each individual program.
5. The analysis modules can now be used to examine the resulting data using descriptive statistics and various visualisation tools
6. Finally. The user can use modelling tools such as case-based reasoning.

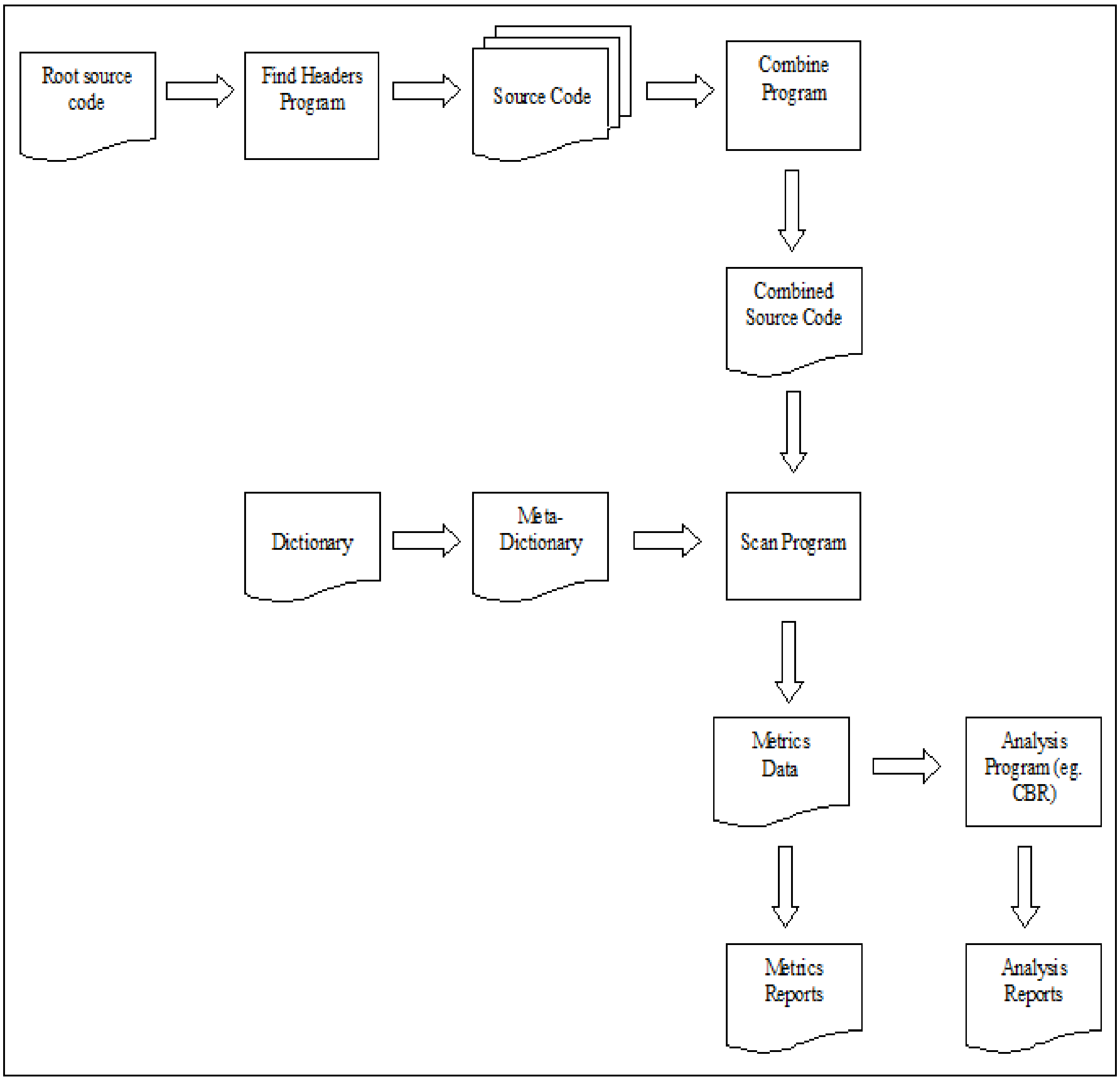


Figure 3. Structure of IDENTIFIED

4. CONCLUSIONS

It appears that software forensics has the potential to become both an important area of practice in computer security, computer law, and academia as well as an exciting new area of research. Systems such as IDENTIFIED will form an important part of such a field – providing fast and accurate data extraction and accessible analysis methods.

REFERENCES

Gray, A.R., Sallis, P.J., and MacDonell, S.G. (1997) Software Forensics: Extending Authorship Analysis Techniques to Computer Programs. Presented at *The Third Biannual Conference of the International Association of Forensic Linguists*, 4-7 September 1997, at Duke University, Durham, North Carolina, USA.

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 in *The Fourth International Conference on Neural Information Processing - The Annual Conference of the Asian Pacific Neural Network Assembly (ICONIP'97)*, 24-28 November 1997, at the University of Otago, Dunedin, New Zealand.

Sallis, P. (1994). Contemporary Computing Methods for the Authorship Characterisation Problem in Computational Linguistics, *New Zealand Journal of Computing*, 5, 85-95.

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.