Asian Journal of Information Technology 6 (1): 38-46, 2007

Statistical Correlation Counts of Program Command Sequence for Establishing Copyright Violations

O.B. Longe and S.C. Chiemeke
Department of Computer Sciences, University of Benin, P.M.B. 1154, Benin City, Nigeria

Abstract: Any consideration that requires proofing that an author's copyright has been violated, is firstly an endeavor to tackle the general problem of showing that there are grounds for suspicion that a program has been copied in whole or in part, or that it is being used illicitly. The further task is to provide evidence of similarity in order to aid the identification of a program, or program modules concealed in another program as an aid to litigation. Effective protection against piracy combines both legal and technical methods. Legal protection encompasses legislation to protect intellectual property by copyright or contract law etc. Technical methods take advantage of software identification techniques such as watermarking, fingerprinting and birthmarks. Premised on a software developed by the authors (Christened *CoVioChecker*) to count program keywords, this paper introduces the concept of statistical correlation counts as a measure for comparing program codes in order to establish copyright infringement. Results obtained using the program similarity measure show that the measure is 1 if all command in one program are the same with that of another and -1 if all commands in one program are absent in the other. As a standard, similarity measures ranging between 0.7 and 1 could serve as a basis for further probing regarding copyright infringements.

Key words: Copyright, correlations, frequencies, graphs, identification, piracy, protection, statistics, validation and violations

INTRODUCTION

Software development from its conceptualization stage to the actual implementation is tasking intellectually, financially and otherwise. Programming management must therefore safeguard the creative efforts and investment represented by proprietary computer software and database development. Knowledge of legal and other measures available is required to design a comprehensive program for protecting these resources. The determination of the most appropriate protective measure should be based on a risk analysis of each situation. Risks associated with the protection of proprietary software and databases vary, depending on the nature of software or database involved, the access allowed and the distribution scheme used. These risks include unauthorised access, malicious attacks and virus attacks, damages associated with usage and system failures, software theft in the form of piracy and counterfeiting. Protective schemes must therefore be assessed in the light of trade-offs associated with implementing, maintaining and enforcing the forms of specific protection available (Jaiyesinmi, 2002).

Among the various threats against which software must be protected, piracy ranks highest in terms of its impact on developments in information technology and the economic and social implications on the society at large. The term software piracy covers such activities as the unauthorised copying of programs, counterfeiting and distributing software illegally. It is important to understand the different software piracy channels, not just to comply with the law but also to protect against bigger economic problems like lost revenue and lost jobs. The survival of the software industry is paramount to the survival of modern business and IT activities (Christian, 1992).

Program characteristics: A program has varying degrees of structure which be inspected and measured as a characteristic of the data sequence. The measurement may not be unique to the data, but data representing a structured program is likely to show definite characteristics, which permit a measure of similarity between two different sets of codes. Since the aim is to seek such characteristics in order to decide whether a program or a program module has been copied and perhaps embedded in a larger program, then it is imperative to check the extent to which these unique characteristic provides evidence to justify suspicion. One of the most unique features of a program is the relative

Corresponding Author: S.C. Chiemeke, Department of Computer Sciences, University of Benin, P.M.B. 1154, Benin City, Nigeria

frequency of commands and reserved words. Others are the repetition of program sequences and their position in the code, program variables and the use of structured styles. The value of a program lies in the operations, which can be out rather than in the appearance, design and arrangement of the code. For instance, visible features such as and names of variables in a high level language can be readily altered or rearranged by a smart pirate without affecting the basic output or operation of the program. When this is done, variables can either be eliminated or translated into different forms when the program is compiled. The need therefore arise to check not only whether an entire program has been copied but also whether a module of a program has been copied into another program. Such a module, when compiled, will be allocated storage addresses which will be a function of the position of the module in the program so that the machine code will look very different on a straight correlation measure.

The intrinsic qualities of program commands for validating infringements: In checking on the similarity between programs, there are various criteria that are independent of the modifications that a pirate may introduce in order to disguise the code. For this reason we seek features of a program which are characteristic of the program. The characteristic needs to relate to features of the program that are difficult to alter and which if possible are represented in the machine code (Derrick, 1990). The program commands (or reserved words) and their distributions throughout the program fall into this category. Firstly, it is difficult for an infringer to tamper with them without modifying the function of the program. Secondly, if they are altered, it takes substantial understanding of the entire program code and the consequences of such code alterations for the infringer to be able to do any meaningful thing with the program. Of course it is as good as re-writing the program from the start. This is the investment of time and energy that the infringer is running from, so he is likely not going to go to the extent of removing commands, mere window dressing of variable names and code re-arrangement may suffice for his trade.

Identifying similarities in program command sequence: Based on the premises in the last section, it can be established that the sequence of commands in a program is independent of the simple façade that might be attempted by an infringer by altering other program features such as variable names, remarks, line numbers, fonts etc. A routine can be written in any language. *CoVioChecker* (an acronym of Copyright Violation Checker) has been designed using visual basic to accommodate different programming language for the purpose of extracting or harvesting program command sequence (Longe, 2004).

Features of *coVioChecker*: The program provides the following capabilities:

- The facility to set up the programming language in which the programs to be compared are written using the Setup Lang Command. Provision has been made for six programming languages viz: VB6, JAVA, C++, C#, DBaseIV and PASCAL.
- It also provides the avenue through which new keywords can be added to the existing keywords in the language database using the ADD New Keyword command. This is to ensure that the program remains relevant and operational as new developments and upgrades comes into the various languages for which provision has been made. Simply put, it maintains the history of the various programs and ensures compatibility and continuity.
- Facility for loading files under examination (text-based) after the proper applicable languages have been selected.

The add keyword and language dialog box: As explained before, it is used to add new keywords in a selected language shown in Fig. 1.

The language selection dialog box: Used to select the language for the programs to be tested. Both original and testing programs must be written in the same language shown in Fig. 2.

If the programs subjected to testing are not written in the language selected and therefore have no keyword(s) in the selected language, the dialog box below will appear. Note also that provision is made only for testing source codes and not already compiled codes (object codes). Testing object codes will yield results such as the one shown in the testing section of the dialog box below.

Extracting program commands using the *coVioChecker*: The command counts for some modules of a stock control program in an agric-produce sales outlet and two other programmes developed (also as projects) to computerize sales records at a supermarket and a filling station are extracted using the *CoVioChecker*. The program listings are shown in the Appendix. Graphs plotted from the command counts obtained from the comparison made by the *CoVioChecker* are illustrated in Fig. 3(a-d), for each of the 16 commands used in the Dbase (IV) programs under

Fig 1: Adding keywords dialogue box

Fig 2: Language selection dialogue box

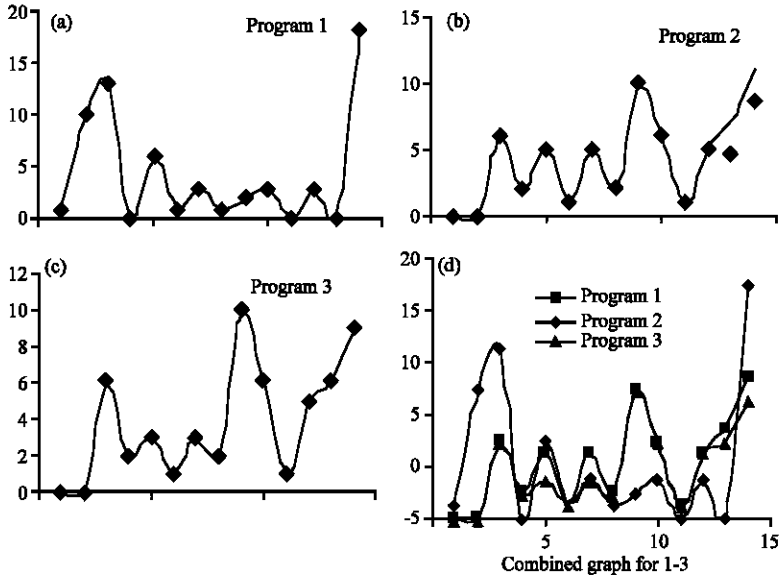

Fig. 3: (a-d) MS-excel plot for the various data derived from the count

Table 1: Commands retrieved using *CoVioChecker*

Command	Program 1	Program 2	Program 3
Append	1	1	1
Case	0	9	9
Do	8	15	15
Do while	2	0	0
Set	5	3	4
Use	1	1	1
Clear	5	3	1
Else	2	1	1
If	10	2	2
Read	6	2	3
Wait	0	0	0
Pack	3	0	0
Locate	1	0	0
Store	2	6	6
Replace	7	0	0
Say	11	17	16

examination. It is observed that some commands occur very frequently in all programs, some are infrequent and some are absent. The various co-ordinates on the graph depict the occurrence of a particular command in the order shown in the Table 1. Displayed below are the results obtained from the *CoVioChecker* for the comparison carried out.

APPENDIX

Program 1: Development of a database package for stock control in an agric-produce sales outlet:

```
THIS PROGRAM IS TO CREATE A NEW STOCK
SET ECHO OFF
SET TALK OFF
SET STATUS OFF
CLOSE ALL
CLEAR
USE STOCK. DBF
DO WHILE. T.
CLEAR
IC = L
DO WHILE (I C<=23)
@1 C 10, SAY REPLICATE (CHR 177),80)
IC = C+1
END DO
STORE TOD. ("11") TO M DATE,
EXP = O
STORE SPACE (15) TO DS
A = ( )
QT = 0
RD = 0
PR = 0
ITN = 0
SET COLO TO W/B
@ 6,20 CLEAR TO 20,65
@5,191021,66 DOUBLE
@ 5,33 SAY "NEW ITEMISTOCK"
@ 7,21 SAY "STOCK NO:" GET IN
READ
IF 1TN = 0
CLEAR
RETURN
END IF
LOCATE FOR STOCK = ITN
IF FOUND ( )
@17,21 SAY "NUMBER ALREADY EXISTS TRY AGAIN
"Y/N" GET A
READ
IF ITN = 0
IF A 'Y'
LOOP
END IF
ELSE
@ 8,21 SAY "STOCK NAME:" GET DS @ 9,21 SAY
"DATE: "GET M DATE @ 10,21 SAY "DESCRIPTION:"
GET QT
@ 11,21 SAY "QUANTITY ON HAND:" GET DR @ 12,21
SAY "QUANTITY SUPPLIED:" GET EXP READ
END IF SET COLO TO W/B ANS= Y
@ 21,20 CLEAR TO 23,45
@ 22,23 SAY "SAVE RECORD{YIN}" GET ANS
PICTURE "! READ
IF ANS = {Y}
APPENDIX BLANK
REPLACE NAME WITH DS
REPLACE DESC WITH QT
REPLACE HAND WITH RD
REPLACE SUPPLIED WITH PR
REPLACE STOCK WITH ITN
REPLACE DATE WITH M DATE
REPLACE SOLD WITH EXP
END IF
@ 17,21 SAY MORE CREATION (Y/N) GET A
READ
1F A "Y". OR. A = "Y"
LOOP
ELSE
CLOSE ALL
RETURN
END
END IF
END DO
```

Program 2: Computerised sales systems for jovenson supermarket auchi:

```
***A PROGRAM TO RUN SUPERMARKET SALES
USING COMPUTER SYSTEMS ***
*** DESIGNED BY OSAMUYI KINGSLEY ****
```

```
*** ND II COMPUTER SCIENCE.****
*** MAT NO. ST/04/345****
SET TALK OFF
SET STATUS OFF
SET SCORE OFF CLEAR
@5,30 SAY "M A I N PROGRAM"
@23,20 SAY "JOVENSON SUPERMARKET AUCHI."
COLOR+/ W
@7,22 SAY "1. ADD NEW INFORMATION"
@8,22 SAY "2. VIEW ALL INFORMATION "
@9,22 SAY "3. VIEW A SINGLE INFORMATION"
@10,22 SAY "4. DELETE A SINGLE INFORMATION"
@11,22 SAY "5. DELETE ALL INFORMATION"
@12,22 SAY "6. SALES CALCULATION"
@13,22 SAY "7. PRINT REPORTS"
@14,22 SAY "8. EXIT PROGRAM" STORE SPACE (1) TO
CHOICE
@18,20 SAY "Choose the Option Needed" GET CHOICE
READ
DO CASE
CASE CHOICE = "1"
DO ADDREC@.PRG
CASE CHOICE = "2"
DO VIEW1.PRG
CASE CHOICE = "3"
DO VIEW2.PRG
CASE CHOICE = "4"
DO DEL1.PRG
CASE CHOICE = "5"
DO DEL2.PRG
CASE CHOICE = "6"
DO SALES@.PRG
CASE CHOICE = "7"
DO PRINT@.PRG
OTHERWISE
DO EXT.PRG
ENDCASE
*** TO ADD NEW INFORMATION***
CLEAR
STORE SPACE (25) TO PNAME
STORE SPACE (0) TO PCODE
STORE 0 TO PPRICE,PQTY,STCKQTY,PNETQTY
STORE {} TO PDATE
@5,10 TO 20,50 DOUBLE COLO G±
@7,11 SAY "ITEM NAME" GET PNAME
@9,11 SAY 'ITEM CODE" GET PCODE
@11,11 SAY "NUMBER SOLD" GET PQTY PICT "9999"
@13,11 SAY "NUMBER IN STOCK" GET STCKQTY
PICT "999999" @15,11 SAY DATE SOLD
USE TEXSTOCK.DBF
GO BOTTOM
APPEND BLANK
REPL NAME WITH PNAME
REPL CODE WITH PCODE
REPL QTY WITH PQTY
REPL QTYINSTO WITH STCKQTY
REPL DTBOUGHT WITH PDATE
CLOSE DATABASES
STORE SPACE (1) TO ANS
CLEAR
@15,20 SAY "Add More Information (Y/N)" GET ANS
READ
IF UPPER(ANS)
DO ADDREC@.PRG
ELSE
DO MAIN@.PRG
ENDIF
```

Program 3: Computerisation of sales records, case study of texaco filling station, agbor:

```
***A PROGRAM TO CONTROL SALES AND
STOCK***
*** THIS PROGRAM IS WRITTEN BY MR. NELSON C.
ENUONYE****
*** ND II COMPUTER SCIENCE.****
SET ECHO OFF
SET STATUS OFF
SET SAFETY ON
SET SCORE OFF
@6,20 TO 16,55 DOUBLE COLO G±
@7,22 Say "1. Add New Records"
@8,22 Say "2. View All Records"
@9,22 Say "3. View a Single Record"
@10,22 Say "4. Delete a Single Record"
@11,22 Say "5. Delete All Records"
@12,22 Say "6. Sales Calculation"
@13,22 Say "7. Print Reports"
@14,22 Say "8. Exit Program" STORE SPACE (1) TO
CHOICE
@18,20 SAY Choose the Option Needed" GET CHOICE
READ
DO CASE
CASE CHOICE = "1"
DO ADDREC@.PRG
CASE CHOICE = "2"
DO VIEW1.PRG
CASE CHOICE = "3"
DO VIEW2.PRG
CASE CHOICE = "4"
DO DEL1.PRG
CASE CHOICE = "5"
DO DEL2.PRG
CASE CHOICE = "6"
DO SALES@.PRG
```

```
OTHERWISE
DO EXT.PRG
ENDCASE
*** TO ADD NEW RECORDS***
STORE SPACE (25) TO PNAME
STORE SPACE (0) TO PCODE
STORE 0 TO PPRICE,PQTY,STCKQTY,PNETQTY
STORE {} TO PDATE
@5,10 TO 20,50 DOUBLE COLO G±
@7,11 SAY "PRODUCT NAME" GET PNAME
@9,11 SAY 'PRODUCT CODE" GET PCODE
@11,11 SAY "QUANTITY SOLD" GET PQTY PICT
"9999"
@13,11 SAY "QUANTITY IN STOCK" GET STCKQTY
PICT "999999" @15,11 SAY DATE SOLD
@15,21 SAY DATE() READ
USE TEXSTOCK.DBF
GO BOTTOM
APPEND BLANK
REPL NAME WITH PNAME
REPL CODE WITH PCODE
REPL QTYINSTO WITH STCKQTY
REPL DTBOUGHT WITH PDATE
CLOSE DATABASES
STORE SPACE (1) TO ANS
CLEAR
@15,20 SAY "Add More Records (Y/N)" GET ANS READ
IF UPPER(ANS)
DO ADDREC@.PRG
ELSE
DO MAIN@.PRG
ENDIF
```

Fig. 4: Count for program 1

DISCUSSION

Apart from the popularity of some commands which may cause a degree of similarity to be seen, especially when two programs cater for a similar purpose since the purpose will often determine the choice of commands used program 1 is obviously different from that of programs 2 and 3 shown in Fig. 4. This is because even though the need for the 3 programs seems similar, program 1 is actually written in a style and with command combinations different to that of 2 and 3. The plot for 2 and 3 are very similar. From an examination of the program command list and the program listing in the appendix, it will be seen that a lot of window dressing has been done to disguise variables and other structures in program 3 copied from program 2 shown in Fig. 5 and 6. As has been said earlier, in checking on the similarity between programs, the criterion must be as far as possible be independent of the modifications that an infringer might introduce in order to disguise the code. Hence the basis of our comparisons is the reserved words which if altered will mean going through the rigour of re-writing the whole program.

Looking at the combined Fig. 3(d), the proof of similarity is the overlapping between graphs 2 and 3. The portion of the graph that does not overlap depicts points of commands at which some alterations made are very obvious. Sometimes, the eye is less good at discerning the relative differences observable in the plots. According to Derrick (1990), for programs suspected to be similar from the analysis so far, plotting the frequency of a command in one program as the

Fig. 5: Count for program 2

Fig. 6: Count for program 3

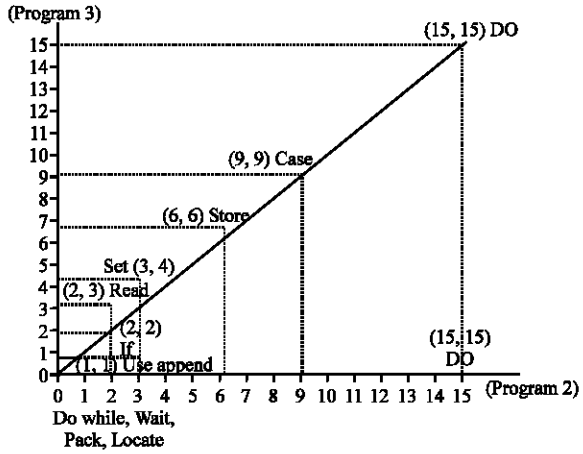

Fig. 7: Comparison Graph for programs 2 and 3

abscissa and the frequency in the other as the ordinate may better facilitate a comparison.

In the event that the programs were very identical then the plotted points would nearly all fall on a line at

45 degrees to the axis. An example of this is shown in Fig. 7 where comparisons are made for programs 2 and 3. The point off the 45-degree line represents the alterations made to the copied program.

Mathematical technique for correlation counts: The assessment of frequency distribution can be quantified mathematically using a correlation count. Taking, for each command, the mean of the frequency count in each program and subtracting the difference between the counts would obtain a simple measure for similarity. Expressed mathematically, the measure for similarity for one command (which for the moment is dependent on program length) is:

$$\frac{f_1 + f_2}{2} - |f_1 - f_2|$$

where f_1 is number of repetitions of a command in the first set of codes (setcode1) and f_2 for Second set of codes (setcode 2). For the nth command this is more simply written as:

$$\frac{S_n - 2D_n}{2}$$

where S_n is the sum of the repetitions of command *n* in the program and D_n is the difference.

A measure for the whole command sequence is obtained by summing over all commands and in order to normalize the measure to be independent of the length of the program we divide by the number of commands present in each program, i.e.,

$$\text{Program similarity measure} = \frac{\sum_{n=1}^{N} S_n - 2D_n}{\sum_{n=1}^{N} S_n}$$

where N is total number of commands in the program. This measure is 1 if the command frequencies in each program are the same and -1 if all commands in one program are absent in the other. The measure of similarity for program 2 and 3 is 0.96.

Vijay and Vijay (2006) posited the statistical formula for correlation is as:

$$\frac{\sum_{n=1}^{N}(f_{1n} - f_{1m})(f_{2n} - f_{2m})}{\left\{\left[\sum_{n=1}^{N}(f_{1n} - f_{1m})^2\right]\left[\sum_{n=1}^{N}(f_{2n} - f_{2m})^2\right]\right\}^{1/2}}$$

f_m is mean number of repetitions over all commands.

The correlation function by this formula will always be between 0 and 1 since it is not possible to have a negative number of commands. This technique can be used to effectively validate the presence of one program or program module in another program that has been accused of infringing on the right of an author. A contribution to the magnitude of the correlation function of frequency distribution will occur by virtue of the popularity of certain commands, which will ensure their use in any program in reasonable numbers.

This contribution will appear to imply a greater degree of correlation by virtue of the characteristics of the medium than in fact occurs due to similarity of program function. It implies that the threshold of correlation at which similarity of programs is diagnosed must be increased, or alternatively the corresponding measure could be reduced by an amount, which compensates for the commonality problem. The frequency distribution is therefore more useful when comparing two programs since it is independent of the order of routines.

CONCLUSION

After copyrights have been infringed, it may be expensive or even difficult to obtain a remedy in courtroom proceedings. For example, a charge of copyright infringement may fail if the court as a defence accepts reverse engineering. For these reasons, technical defences (known in legal circles as self-help) will continue to be important for any software developer who is concerned about litigation against infringing parties. It is encouraging to note that in the advanced countries, software protection is becoming a major area of study, research and professionalism. This should spread also to the third world countries so that for everyplace and everywhere, professionals can develop technical schemes that best suit their particular needs.

Software piracy is likely to continue so long as it continues to be easy, delivers immediate tangible or intangible rewards to the pirate and is socially acceptable. The goal of this paper is to develop measures by which proof of infringement can be established. A combination of the *CoVioChecker* and statistical techniques for correlation counts will aid in establishing grounds for initiating legal proceedings against any infringer.

REFERENCES

Christian, C., 2002. Watermarking, Tamper proofing and Obfuscation-Tools for Software Protection. IEEE Transactions on Software Engineering, Vol. 28.

Cornish, W., 1990. Intellectual Property: Patents, Copyrights, Trade Marks and Allied Rights. (2nd Edn.), London: Sweet and Maxwell Limited.

Derrick, G., 1990. Data identification and authentication from fingerprints, Computer Law and Security Report, Vol. 6.2.

Jaiyesimi, S., 2002. Protecting proprietary software and databases. Special Topics in Mathematics and Computer Science, A Technical Report of Occasional Lecture Series. Federal University of Technology, Akure, Nigeria.

Longe, O.B., 2004. Proprietary software protection and copyright issues in contemporary information technology. Unpublished master of technology thesis submitted to the department of computer science, manuscript.

Vijay, G. and B. Vijay, 2006. Introduction to statistical methods. India: Prentice-Hall.