THE GEM COBOL MONITOR SYSTEM

Antonio Salvadori
Computing and Information Science
University of Guelph
Guelph, Ontario
CANADA N1G 2W1

INTRODUCTION

During the past five years an increasing number of people have been searching for an answer to the question: How do people write, debug and optimise a computer program? Several authors have written numerous papers in the "considered harmful" series not really knowing if what they were admonishing against was actually taking place in the real world environment. Only recently have Knuth and several other authors [1-4] tried to shed some light on this fascinating problem.

It was during the course of a discussion with Professor Uzgalis of UCLA that the present system was begun. All studies had confined themselves to studying how students in a University environment solving university type problems behaved. This seemed unsatisfactory to me since such a population would necessarily consist of amateurs and not professionals, hence, I set out to develop a system which could be used by professionals. The language that I chose to monitor was COBOL which is the most common language used in the data processing industry. This system is now being used in several environments. The analysis from the data collected is presently in press.[5]

GEM STRUCTURE

GEM, a synonym for Guelph Efficiency Monitor, is a preprocessor system which can analyse a COBOL program at any development or running stage. The system has been developed to provide all levels of management with a tool for improving the total efficiency of COBOL programs. It may be used during the development of new programs to monitor their performance or used to optimise the run time of existing programs.

There are four procedures to the system providing the following facilities:

GEM1. STATIC PROFILE. Summarises all COBOL constructs coded in the identification, environment, data and procedure divisions.

GEM2. DIAGNOSTIC PATTERNS. Summarises the COBOL diagnostics generated during program development and keeps a date account of the number of times a program is run.

GEM3. DYNAMIC FREQUENCY PROFILE. Identifies and calculates the frequency of verb and code segment usage in the procedure division at run time as the program is processing test or live data.

GEM4. DYNAMIC TIME PROFILE. Accounts for the C.P.U. time spent in segments of Procedure Division code as the program executes.

One or more of the four modules may be used to:

. Identify which parts of operating programs are frequently used, so that the code may be optimised.

. Check whether certain parts of a program have never been tested on test data or used in live data.

. Accurately describe programs operating on live data for the selection of a benchmarking suite.

. Provide information for the programming training staff on the common errors made during program development, including the use of non-ANSI COBOL verbs.

. Provide the CODASYL committee or anyone interested in language design and implementation with statistics regards language usage.

GEM source code is available in either ANSI COBOL or PL/I. Both systems have been thoroughly tested on a variety of programs in different environments. GEM is currently proving itself to be a useful tool to both programmers and managers.

<u>GEM 1 MODULE</u>

The system diagram for the GEM1 procedure is shown in Figure 1. As

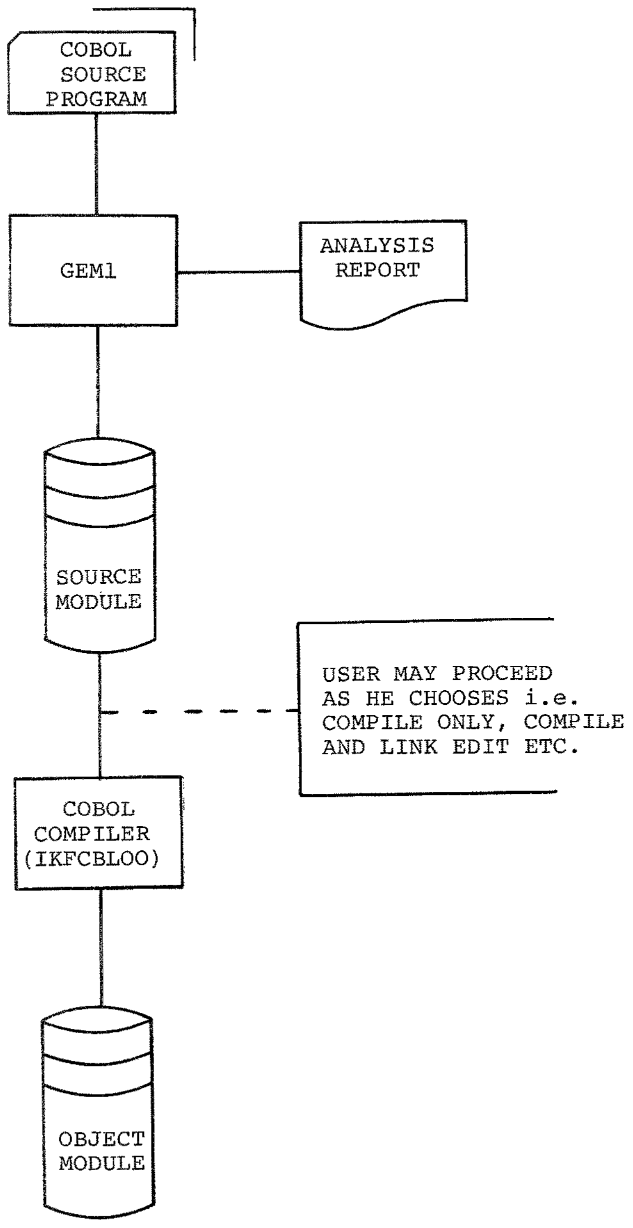
COBOL SOURCE PROGRAM
GEM1
ANALYSIS REPORT
SOURCE MODULE
USER MAY PROCEED AS HE CHOOSES i.e. COMPILE ONLY, COMPILE AND LINK EDIT ETC.
COBOL COMPILER (IKFCBLOO)
OBJECT MODULE

Figure 1. System Diagram for the GEM1 module.

input, the system only requires the user's COBOL source program. GEM1 scans the source code for the relevant statistical information and then submits the unaltered code for processing according to the user's wishes. The code may be compiled and executed in a normal way.

The statistics gathered and printed consist of:

. a COBOL clause and verb count.

. a percentage breakdown of PROCEDURE DIVISION verbs used.

. the number of source records, number of comment cards, indications of non-ANSI standard verbs, etc.

Part of a typical report from GEM1 is shown in Figure 2. This report

```
                         COBOL STATIC STATISTICS

       USER ID :   XXXXXXXX                 PROGRAM ID :   XXXXXXXX

                           PROCEDURE DIVISION

  ACCEPT            0                  PERFORM          0
  ADD               1                * PROCESS          0
  ALTER             0                  READ             1
  CALL              0                  RECEIVE          0
  CANCEL            0                  RELEASE          0
  CLOSE             1                  RETURN           0
  COMPUTE           2                  REWRITE          0
  COPY              0                  SEARCH           0
  DECLARATIVES      0                * SEEK             0
  DELETE            0                  SEND             0
  DISABLE           0                  SET              0
  DISPLAY           0                  SORT             0
  DIVIDE            0                  START            0
  ENABLE            0                  STOP(*GOBACK)    1
* EXAMINE           0                  STRING           0
  EXIT              0                  SUBTRACT         0
  GENERATE          0                  SUPPRESS         0
  GO TO             2                * SUSPEND          0
* HOLD              0                  TERMINATE        0
  IF                1                  UNSTRING         0
  INITIATE          0                  USE              0
* INITIALIZE        0                  WRITE            5
  INSPECT           0                  +                1
  MERGE             0                  -                1
  MOVE             13                  /                0
  MULTIPLY          0                  *                4
* NOTE              0                  **               0
  OPEN              2
```

Figure 2. Part of the report produced by GEM1.

should prove useful for benchmarking, COBOL programmer training, ANSI or in-house standards and various language developers.

GEM 2 MODULE

GEM 2 was motivated by a desire to understand how programs are written and develop from the initial stages to the production phase. A record is kept of each run of the program together with any observable errors which can be automatically gathered. The system diagram for the procedure is shown in Figure 3.

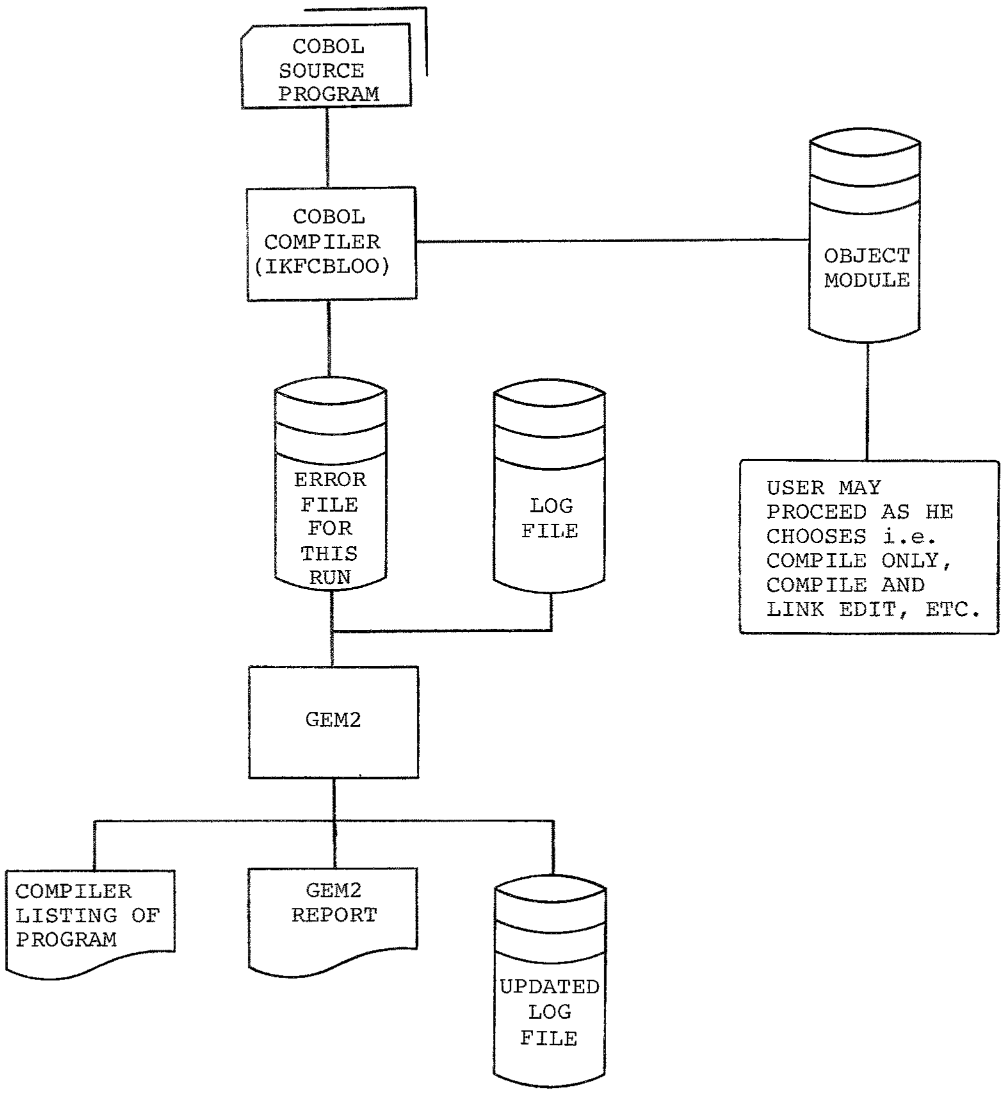
COBOL SOURCE PROGRAM
COBOL COMPILER (IKFCBLOO)
OBJECT MODULE
ERROR FILE FOR THIS RUN
LOG FILE
USER MAY PROCEED AS HE CHOOSES i.e. COMPILE ONLY, COMPILE AND LINK EDIT, ETC.
GEM2
COMPILER LISTING OF PROGRAM
GEM2 REPORT
UPDATED LOG FILE

Figure 3. System diagram for the GEM2 module.

The output from the various levels is scanned for diagnostics and the history record is updated with the new information. These diagnostics are recognised by the manufacturer generated codes. No distinction is made between such codes and user produced results, should the user produce these codes as part of his normal output.

The report generated by GEM2 is shown in Figure 4. Since a record is

```
                          COBOL ERROR STATISTICS

USER ID : XXXXXX          PROGRAM ID : SOCRUPD          DATE : 10/02/75

                                 RUN NUMBER
 ERROR    1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  TOTAL
MESSAGE
 1087W    2  2  0  2  2  1  0  0  4   2   1   2   2   1   0   0    21
 1004E    1  0  1  1  1  0  0  0  0   1   1   1   1   0   0   0     8
 3001E    7  7  7  0  0  0  0  0  2   2   1   7   0   0   0   0    33
 1081W    3  3  1  1  0  0  0  0  1   0   0   3   0   0   0   0    12
 1080W    3  0  0  0  0  0  0  0  1   0   0   3   0   0   0   0     7
 1117E    0  1  0  0  0  0  0  0  0   0   0   0   0   0   0   0     1
 1128W    0  3  0  0  0  0  0  0  0   0   0   0   0   0   0   0     3
 1078W    0  0  2  0  0  0  0  0  0   0   0   0   0   0   0   0     2
 1042E    0  0  0  1  0  0  0  0  0   0   0   0   0   0   0   0     1
 1003W    0  0  0  0  0  1  0  0  0   0   0   0   0   0   0   0     1
 1016E    0  0  0  0  0  0  0  0  1   0   0   0   0   0   0   0     1

 TOTAL   16 16 11  5  3  2  0  0  9   5   3  16   3   1   0   0

   TOTAL ERRORS FOR SOCRUPD IS 90


   START DATE : MON   09/22/75
                                NUMBER OF RUNS

                                   1   2   3   4   5   6
                          DAY

                               ---------------------------
                      MON   1 |********
                      TUE   2 |****
                      WED   3 |************
                      THR   4 |********
                      FRI   5 |
                      SAT   6 |****************
                      SUN   7 |
                      MON   8 |
                      TUE   9 |
                      WED  10 |
                      THR  11 |********
                      FRI  12 |********

                              TOTAL JOBS = 0016
```

Figure 4. Part of the GEM2 report.

kept of the frequency distribution of errors this information should help a programmer diagnose his deficiencies with respect to COBOL and thereby remedy these. Further, when figures are kept on a more global scale true language deficiencies and troublesome points are found which can be remedied in future development and new language design. Supervisors and management should also find the development time information useful for it allows them to assess programming and debugging time accurately and therefore plan later projects more accurately and efficiently. A programmer efficiency index has also been proposed by the author based on this type of information.[6]

GEM 3 MODULE

The PROCEDURE DIVISION of a COBOL programme is divided into paragraphs and sections. Execution of statements within a paragraph is sequential unless a branching statement is encountered in which case, execution resumes at the beginning of the new paragraph to which branching has occurred. If we therefore wish to monitor the execution of statements, it is obvious that the paragraph level subdivision is too coarse and

```
PAR-X
   MOVE _ _ _ .
   ADD _ _ _ .
   MULTIPLY _ _ _.
   IF _ _ _ THEN GOTO PAR-Z.
      ELSE _ _ _ .
   MOVE _ _ _ .
   MOVE _ _ _ .
   PERFORM _ _ _ .
   ADD _ _ _ .
PAR-Y.
   _
   _
   _
   _
PAR-Z.
   _
   _
   _
```

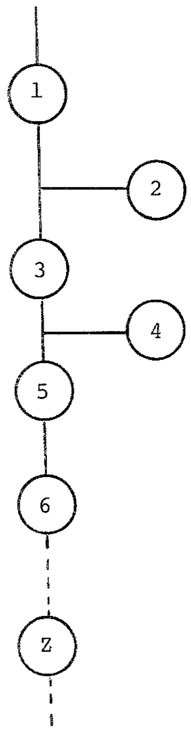

Figure 5. Subdividing a program into basic blocks.

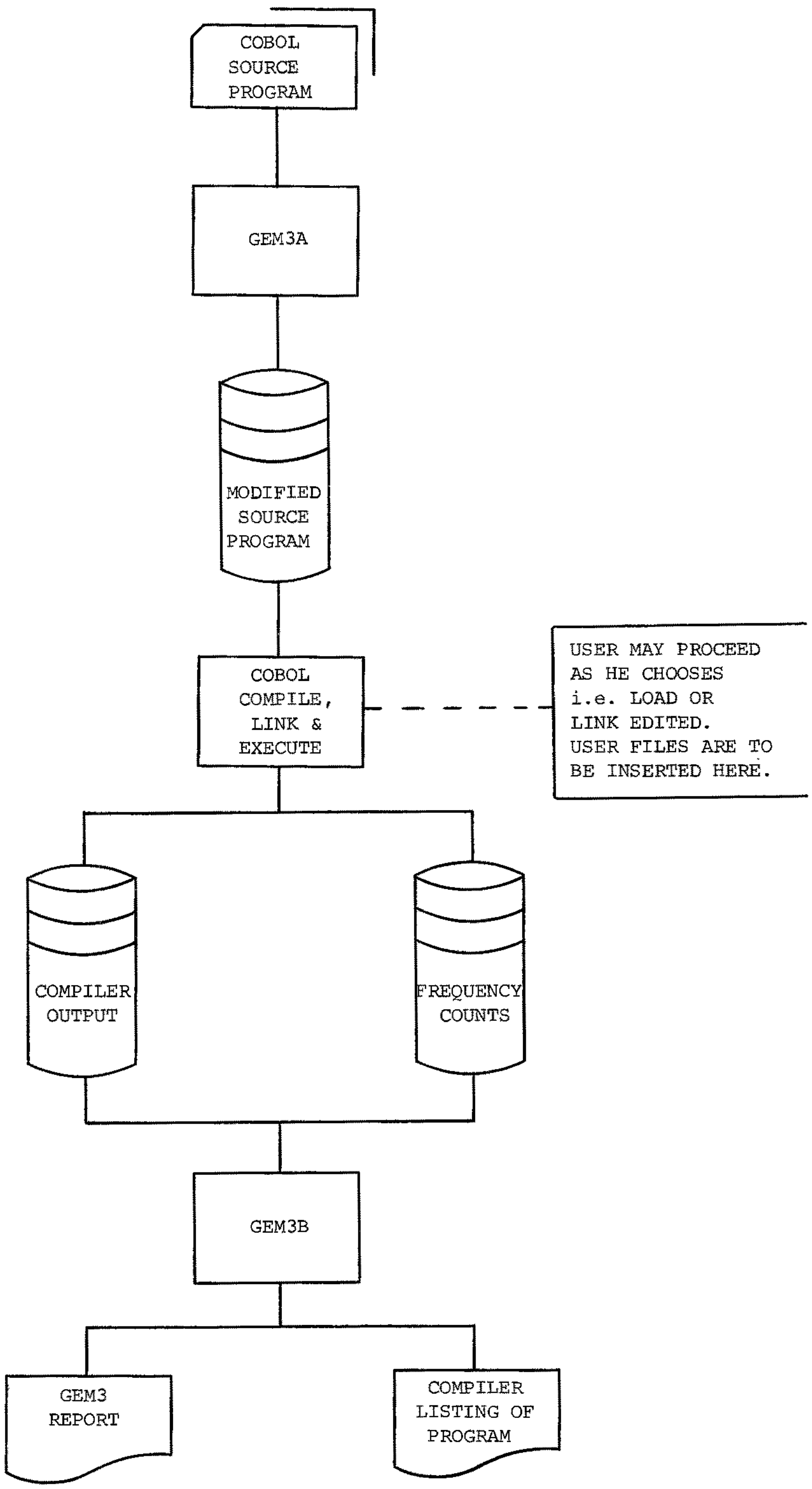
COBOL SOURCE PROGRAM
GEM3A
MODIFIED SOURCE PROGRAM
COBOL COMPILE, LINK & EXECUTE
USER MAY PROCEED AS HE CHOOSES i.e. LOAD OR LINK EDITED. USER FILES ARE TO BE INSERTED HERE.
COMPILER OUTPUT
FREQUENCY COUNTS
GEM3B
GEM3 REPORT
COMPILER LISTING OF PROGRAM

Figure 6. The system diagram for the GEM3 module.

we must subdivide paragraphs into finer segments. Hence we define a basic block as a linear sequence of program instructions having one entry point (the first instruction executed) and one exit point (the last instruction executed). Figure 5 illustrates how a program can be divided into basic blocks. These may be represented as the nodes of the control flow graph.[7]

GEM3 is divided into two parts as shown in the system diagram in Figure 6. GEM3A subdivides the COBOL PROCEDURE DIVISION into segments and inserts monitoring code to keep track of execution time frequency counts. The modified source is then passed to the compiler, linkage editor or loader and executed giving the user his normal output. GEM3B then performs an analysis of the results and produces the reports shown in Figure 7.

```
                                                                        FREQUENCY

     WRITE HEADING-LINE AFTER ADVANCING NEW-PAGE
     MOVE LINE-2 TO HEADER-LINE.
     WRITE HEADING-LINE AFTER ADVANCING 2 LINES.
*BEGIN DYNAMIC
 READ-A-CARD.

*********************** BLOCK NUMBER 00001*******************         17
     READ EMPLOYEE-FILE RECORD; AT END
*********************** BLOCK NUMBER 00002*******************          1
                                   GO TO EOJ.

*********************** BLOCK NUMBER 00003*******************         16
     IF NO-OF-HOURS IS GREATER THAN 40 THEN

*********************** BLOCK NUMBER 00004*******************          1
     COMPUTE GROSS-PAY ROUNDED =
      HOURLY-RATE * 40 + HOURLY-RATE * 2 * (NO-OF-HOURS - 40)
     ELSE
*********************** BLOCK NUMBER 00005*******************         15
          COMPUTE GROSS-PAY ROUNDED =
     HOURLY-RATE * NO-OF-HOURS.
*END DYNAMIC
     ADD GROSS-PAY TO GROSS-COUNT.
     IF COUNT-A > 4 THEN PERFORM PARA-1 THRU PAR1-EXIT.
     IF HOURLY-RATE > 3 THEN GO TO PARA-1
       ELSE GO TO PARA-1, PARA-2 DEPENDING ON THE-FLAG
****************************** COMMENT
     MOVE NO-OF-HOURS TO NO-OF-HOURS.
     MOVE HOURLY-RATE TO HOURLY-RATE.
*BEGIN DYNAMIC
PARA-1.
*********************** BLOCK NUMBER 00006*******************         16
         ADD 1 TO COUNT-A ON SIZE ERROR
*********************** BLOCK NUMBER 00007*******************        NOT
                                    GO TO EOJ.                      TESTED
```

Figure 7. Part of the GEM3 frequency report.

These reports show, among other items, the frequency of execution of each segment of code. The user may use this information in essentially two ways. Firstly, in a debugging or testing environment, he may isolate the areas of code which have never been tested. He may consequently draw up test data to exercise these parts. Second, he may isolate frequently executed parts of code and see if some optimisation can take place. In several cases it was found that by looking at GEM3 results programs could be made up to 20% more efficient by removing certain pieces of code that had become obsolete and by optimising some crucial tests.

The greatest use of GEM3 is at the programmer level. He may directly benefit from its use. However, GEM3 can also be used for very accurate benchmarking purposes.

GEM4 MODULE

This module performs a task similar to GEM3 except that instead of frequency counts CPU timings are given for each section. The inserted code is a call to an assembler routine which disables the input/output interrupts and records the time from the absolute time of day clock of the machine. As a result of this, a specific assembler routine has to be written for the host machine or at best for a line of machines. Professor Gordon at the University of Guelph is presently writing a suite of assembler routines which will allow GEM4 to run on a variety of machines.

In my preliminary study of GEM4 it appears that little or no extra information can be gathered from using GEM4 over GEM3 since the timing is of necessity only marginally accurate due to the routines overhead and also because of the side effects, such as, taking over control of the machine which is undesirable.

SYSTEM OVERHEAD

GEM requires 80k bytes of main memory to execute. Source statements are scanned at the approximate rate of 0.5 seconds per 100 statements on an IBM 370/155 running under MVT.

CASE STUDY

Since the information gathered by GEM is of necessity of a confidential nature, the results must therefore be lumped together to preserve anonimity.

In the study presented here a random sample of novice and experienced programmers were analyzed. Fifty-six programs which they had written were divided into two classes. The first class consisted to programs of an editing nature, i.e. where input data was edited for correctness and an updating file prepared. The second class consisted of programs of an analysis nature, i.e. programs in which data was analysed and reports issued. Figure 9 shows the results of applying GEM2 as the programs were being developed and figure 10 shows the dynamic frequency counts of the verbs used for both groups.

Description of Diagnostic	Percentage of Total
Identifier has not been declared	16.48
Ambiguous reference to identifier	5.76
Undefined procedure or paragraph name	2.44
Warning - this statement cannot be reached	2.24
Paragraph has no statements	2.07
Invalid record name in WRITE statement	1.94
Illegal use of ELSE or OTHERWISE	1.48
Invalid file name in OPEN statement	1.12
Illegal operand in PERFORM statement	0.84
Constant or variable required AFTER advancing	0.68
Variable has too many subscripts	0.51
Procedure or paragraph name already defined	0.36
Variable has too few subscripts	0.29
Residual (other miscellaneous errors)	0.20

Figure 9. The error statistics gathered in the case study for the PROCEDURE DIVISION.

DISCUSSION

As the results show, GEM has proved itself very useful in getting a better understanding of the error-proneness of COBOL. This information has been very useful to the subsequent teaching of the language since students can now be forewarned about the various pitfalls. The language features used by programmers have allowed us to understand the manner in which programs are written. Programmers are greatly influenced by their immediate environment. They tend to write programs using similar language features to those of their colleagues at whatever installation they are at. They tend to follow installation

VERB	EDIT PROGRAM % a	EDIT PROGRAM % b	ANALYSIS PROGRAM % a	ANALYSIS PROGRAM % b
MOVE	26.2	26.2	49.1	42.0
IF	24.7	24.6	16.1	19.4
GOTO	15.0	4.4	10.8	2.3
PERFORM	13.8	15.2	5.6	12.0
ADD	8.1	7.0	3.4	4.0
WRITE	6.5	5.4	10.4	13.4
SET	2.2	7.0	0	0
READ	1.0	2.9	1.2	1.2
EXIT	0.7	2.0	0.4	1.6
OPEN	0.7	0.9	1.1	1.7
CLOSE	0.5	0.6	1.0	1.2
STOP	0.4	2.6	0.8	1.2
SUBTRACT	0.2	0	0.2	0
SEARCH	0.1	1.2	0	0

a: notices b: professionals

Figure 10. The verb usage statistics in the case study.

guidelines closely but not necessarily correctly. For example, at one installation where programmers were supposed to write structured programs without ever using the GO TO statement, it was found that indeed they never used the GO TO statement but were using the PERFORM verb just like the GO TO.

Not enough feedback has yet been received on the use of GEM3 and GEM4 to analyse the results in detail. In a few live test cases some significant improvements were obtained. So far programmers are finding it very useful in program testing and in improving the reliability of a program. However nothing can as yet be said about general code optimisation.

ACKNOWLEDGEMENTS

The author wishes to thank Professors C.K. Capstick and J.D. Gordon for their participation in this work, Peter McMullen for diligently translating his thoughts into code, the University of Guelph and the Department of Supply and Services of the Canadian Government for providing financial assistance.

REFERENCES

1. Knuth, D.E.: An Empirical Study of FORTRAN Programs. Software - Practice and Experience, 1, 105-133(1971).

2. Uzgalis, R., Simon, G., Speckart, W.: Compiler Measures in the Perspective of Program Development, a comparison of the IBM PL/I F-Level Compiler with Cornell's PL/C in a Student Environment. Proc. Sixth Hawaii International Conference on System Science, 104-107(1973).

3. Litecky, C.R., Davies, G.B.: A Study of errors, Error-Proneness, and Error Diagnosis in COBOL. CACM, 19, 1, 33-37(1976).

4. Endres, A.: An Analysis of Errors and Their Causes in System Programs, International Conference on Reliable Software, Los Angeles, 327-336(1975).

5. Gordon, J.D., Capstick, C.K., Salvadori, A.: An Empirical Study of COBOL Programmers, in press INFOR(1976).

6. Salvadori, A., Gordon, J.D., Capstick, C.K.: A System for Evaluating Programmer Performance, Proc. Thirteenth Annual Conference on Computer Personnel Research, Toronto, 100-113(1975).

7. Allen, F.E., Cocke, J.: A Program Data Flow Analysis Procedure. CACM, 19, 3, 137-147(1976).