1994

NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER
THE UNIVERSITY OF ALABAMA

EVALUATION OF THE EFFICIENCY AND RELIABILITY OF SOFTWARE GENERATED BY CODE GENERATORS

Prepared By: Barbara Schreur, Ph.D.

Academic Rank: Associate Professor

Institution and Department: Texas A&M University - Kingsville

Department of Electrical Engineering and Computer Science

NASA/MSFC:

Laboratory: Astrionics
Division: Software
Branch: Systems Engineering

MSFC Colleague: Kenneth S. Williamson

Introduction

Traditional software development follows a cycle wherein the project phases of requirements definition, analysis, design, system construction and system testing are performed sequentially. In software engineering, as in manufacturing, the end product is improved when there is overlap between the phases. This is because feedback from those executing later phases may improve the work of those executing earlier phases. Thus leading to a more coherent finished product. CASE Tools which automate the entire software development cycle encourage the team engineering approach since all that is necessary is that developing files be shared among the various groups.

There are numerous studies, [1, 2] which show that CASE Tools greatly facilitate software development. As a result of these advantages, an increasing amount of software development is done with CASE Tools. As more software engineers become proficient with these tools, their experience and feedback lead to further development with the tools themselves. In fact, new versions of software development programs appear with a similar frequency to those of operating systems and office application software.

What has not been widely studied, however, is the reliability and efficiency of the actual code produced by the CASE Tools. This investigation considers these matters.

Method

Three segments of code generated by MATRIXx ,one of many commercially available CASE Tools, where chosen for analysis . ETOFLIGHT is a portion of the Earth to Orbit Flight software and ECLSS and PFMC are modules for Environmental Control and Life Support System and, Pump Fan Motor Control, respectively. The selection criteria [3] where that:

1. the segments be produced using a code generator,
2. the language available be the C language, and
3. there be no more than 2000 lines of code,.

The original code for ETOFLIGHT was produced in the C language while ECLSS and PFMC where in ADA. MATRIX X was used to regenerate the ECLSS and PFMC code in C from the design diagrams. That this was readily done without code conversion illustrates one of the advantages of CASE Tools. To be honest, the procedure did require a service call to Matrix X as well as considerable transfer of files over the network, but that is simply part of the learning curve.

The software was analyzed with the aid of the McCabe Tool, a commercially available software package. This software preprocesses the code and provides the following:

1. a graphical representation of the relationships between the various modules of the program.

   The graphical representation, the Battlemap, shows the relationship between the modules and gives the cyclomatic, v(g), and essential complexities, ev(g). The cyclomatic complexity measures a model's decision structure while the essential complexity is a measure of the unstructured elements of the module, eg. a jump out of a loop.

2. flow charts of each of the coded modules.

   The flow charts for each coded module also contain a code by which the corresponding section of code can be identified. Each module contains three flow charts. I primarily used the cyclomatic and essential complexity charts.

3. several metrics for each module.

   The metrics [4,5] that I obtained from the tool where

   a) v(g) numerically,
   b) ev(g) numerically,
   c) the number of lines of code,
   d) the number of branches in the code,
   e) the number of lines containing code, comments, or code and comments, and
   f) Halstead metrics consisting of
      i) program length - the number of occurrence of operators and operands
      ii) program difficulty - a measure of how hard it is to follow the code
      iii) error estimate - an estimate of the number of errors in the code
      iv) program volume - minimum number of bits required
      v) intelligent content - a complexity based on the algorithm used without language dependence
      vi) programming time - estimate of the time needed to produce code
      vii) program level - how difficult it is to understand the code
      viii) programming effort - an estimate of the effort necessary to produce the code

Observations

Before utilizing the McCabe Tool, the C code was examined manually. Three points became apparent. First, a switch statement with one choice was generated where one would normally expect a simple if statement.

```
2117                ;
2118  C99           switch( ITSK ){
```

```
2119  C100              case 1  :  SUBSYSTEM01(); break;
2120  C101              default :  break;
2121  C102            }
2122                ;
```

The relative efficiency of switch versus if statements is compiler dependent but the CASE Tools make the decision for you. Thus it does not optimize for efficiency.

Secondly, Matrix X terminated the last case in a nested switch structure without a break, relying on a fall through to the terminating break of the switch structures. Standard practice is to terminate every case with a break unless a fall through is explicitly desired. Modification of the program by adding a new choice to the switch structure could lead to an unexpected fall through. This would not be a problem if the program where regenerated by the CASE Tool but extreme care would have to be taken with manual change to the C code.

```
1929  C8              switch( TASK_STATE[NTSK] ){
1930                  case IDLE :

1932  C9                  switch( TCB[NTSK].TASK_TYPE ){
1933                      case PERIODIC :
1934  C10                     if( TCB[NTSK].START == 0 ){
1935  C11                       READY_COUNT++;

                  ...
1938                        }else{
1939  C12                     TCB[NTSK].START  = TCB[NTSK].START - 1;
1940  C13                   }
1941                      break;
1943                  case ENABLED_PERIODIC :

                ...
1977                  case TRIGGERED_SAF :
1978  C31               if( TCB[NTSK].OUTPUT == 0 ){
1979  C32                 BUS_OFFSET[NTSK] = 15 - BUS_OFFSET[NTSK];

                  ...
1985                    }
1986  C35               }
1987  C36 C37         }
1988              break;

1990              case RUNNING :
```

Lastly, in one module, 1 was the condition in five if statements as shown by the example below.

```
3877  H29         if( 1 ){
3878  H30           fprintf( fp, "YTIME    %5ld%5ld%5d%11s\n",
3879                    YCOUNT-1+IUCNT, ICOL, IIMG, "(1P3E25.17)" );
3880            }else{
3881  H31           fprintf( fp, "YTIME    %5ld%5ld%5d%10s\n",
3882                    YCOUNT-1+IUCNT, ICOL, IIMG, "(1P5E15.7)" );
3883  H32       }
```

For efficiency, these sections of code should be reduced to the statements contained in the true branch, thus reducing the complexity and testing procedures.

An analysis of the flow charts generated by the McCabe Tool showed that three of the subroutines in the one program are identical except for variables used and the order in which some statements are executed. A single subroutine could have been used in their place. This approach, using global parameters, may execute faster than using a single subroutine with passed parameters but the single subroutine makes more efficient use of memory. This is a choice which should be made in the analysis and design phases but the subroutine repetition may well be inadvertent. Future versions of CASE Tools might point out such instances.

Of the two McCabe metrics considered, the cyclomatic complexity gives a good indication of the complexity of a module. The essential complexity is in some cases higher than necessary. In those modules that contain return statements in the middle of a structure, the code cannot be represented as fully structured statements since this represents a jump outside a structure. The essential complexity can be reduced if this jump is performed at a later time. This increases the overall complexity of the program. Statements, such as returns, should be represented by a symbol and not add to the essential complexity of structured programs.

All eight of the calculated Halstead metrics are strongly correlated. Thus, there is no reason to calculate all eight.

Conclusion

The code produced by the code generator was more uniform over the three modules than that that would be produced by hand. The efficiency, though, would benefit from hand optimization.

Future Work

- Comparison of machine generated code and manually generated code.
- Compare to code produced by the next version of the code generator
- Determine if compiler dependence for the most commonly used structure types.

References

1. Weitz, Lori, "Code Generators Gain Versatility", Software, vol. 12, no. 11, August 1992, pp. 38-44.

2. Keyes, Jessica, "Team Tools Targeting Development Process", Software, vol. 12, no. 15, November 1992, pp. 45-56.

3. Schreur, Barbara, "Evaluation of the Efficiency and Fault Density of Software Generated by Code Generators", NASA CR-193862, November 1993, pp. XL-XL3.

4. McCabe, Thomas, "A Complexity Measure", IEEE Transactions on Software Engineering, vol. SE-2, no. 4, December 1976, pp 308-320..

5 .Halstead, Maurice, "Elements of Software Science", Elsevier, New York, 1977.

6. Keyes, Jessica, "Gather a Baseline to Access Case Impact", Software, vol. 10, no.10, August 1990, pp. 30-43.