N94-24445

1993

NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

EVALUATION OF THE EFFICIENCY AND FAULT DENSITY OF SOFTWARE GENERATED BY CODE GENERATORS

Prepared by:	Barbara Schreur, Ph.D.
Academic Rank:	Associate Professor
Institution and Department:	Texas A&I University Department of Electrical Engineering and Computer Science
MSFC Colleague:	Kenneth S. Williamson
NASA/MSFC:	
Office:	Astrionics Laboratory
Division:	Software Division
Branch:	Systems Engineering

Introduction

Flight computers and flight software are used for GN&C (Guidance, Navigation and Control), Engine Controllers and Avionics during missions. The software development requires the generation of a considerable amount of code. The engineers who generate the code make mistakes and the generation of a large body of code with high reliability requires considerable time.

Computer-Aided Software Engineering (CASE) Tools are available which generate code automatically with inputs through graphical interfaces. These tools are referred to as code generators. In theory, code generators could write highly reliable code quickly and inexpensively. The various code generators offer different levels of reliability checking. Some check only the finished product while some allow checking of individual modules and combined sets of modules as well. Considering NASA's requirement for reliability, an in house comparison of the reliability of automatically generated code and of manually generated code is needed.

Furthermore, automatically generated code is reputed to be as efficient as the best manually generated code when executed (2). In house verification is warranted.

Evaluation of CASE Tools

A software project of suitable complexity has yet to be provided for evaluation. When delivered, in the form of hardware and software requirements, this project will lead to a segment of software with

1. a length of at least 2000 lines.
2. a minimum of three levels of hierarchy.
3. one level having a minimum of two routines.
4. minimal complexity.

The plan is to develop the software package using two developers each using a CASE Tool and standard methods (4). Two candidate CASE Tools are ASTER and $MATRIX_X$.

CASE Tools are rigid in how they generate programs. They may, for instance, make extensive use of nested ifs rather than case statements. In some applications, this rigidity may produce inefficient code outright or may not mesh well with the characteristics of the compiler thereby causing inefficient execution. The generated code will be examined for such characteristics and the effects of any such characteristics will be investigated.

The spiral model of the software process is characteristic of CASE Tools. They also allow program changes without using patches because the code is regenerated as an internally consistent whole (1). Additionally, the blocks of code in the CASE Tool libraries are reputedly highly reliable. The principal question is whether a combination of many such blocks retains the high reliability or whether the way they interact is capable of producing faults (2). The generated code will be tested for the existence of faults as the modules are completed, if that is allowed by the CASE Tool. This will be followed by testing of the completed segment.

The metrics selected are those contained in MM 8075.1A (3), which may be tailored. A database will be developed to serve as a collector of the measures. These measures will be provided by metrics generating tools available in the public domain and by tools to be acquired for

this project. The metrics will include the following:

1. Software size: The number of lines of code that must be maintained.
2. Software Staffing: The number of software engineers and immediate supervisor involved in the development.
3. Requirements Stability: The total number of requirements that must be implemented.
4. Development Progress: The number of successfully completed modules.
5. Computer Resource Utilization: Percent utilization of CPU, disk, and I/O channel.
6. Test Case Completion: Percent of successfully completed test cases.
7. Discrepancy Report Open Duration: The time between the report of a problem and the resolution of the problem.
8. Fault Density: The number of open Discrepancy Reports and the total defect density normalized by the software size over time.
9. Test Focus: Percentage of problem reports resolved through software solutions.
10. Software Reliability: Probability that the software works under specified conditions for a specified time.
11. Design complexity: Number of modules that have a complexity greater than a predetermined number.
12. Ada Instantiations: Size and number of generic subprograms developed and the number of times they are used. (For C++, the number of object invocations.)

In addition to the metrics, the effectiveness of the CASE tools will be evaluated using the following criteria:

1. The languages available for code generation.
2. The ability to test modules as they are developed both individually and as part of the system.
3. The language the code generator is written in.
4. The libraries, including icons, that are available.
5. The ability to import code from other files and/or projects.
6. The ability to trace variables through the code and determine the effects they have.
7. The documentation of the software created by the code generator.
8. Check on the ability of the tool to "reverse" engineer a section of code for reusability.

A requirements document and test procedures will be developed for typical flight modules.

The original plan was to begin training on ASTER starting with week five. ASTER has not yet been delivered. When it became apparent that ASTER would not be delivered, training was started on $MATRIX_X$. Training in $MATRIX_X$ is progressing and should be completed by week ten. Draper Labs will conduct a two week training session on ASTER in October, 1993 so training on ASTER cannot begin until then.

Future Analysis

Recommendations for future work include the following:

1. The use of at least three Code Generators using non-trivial complex GN&C source code or the equivalent.
2. Analyzing the source code with respect to McCabe complexity, fault density (per 1000 Lines of Code), and efficiency.

3. Performing Software Verification and Validation (V&V).
4. Recommending V&V Methodology and Work-Arounds for Software Source Code Generators.

Conclusion

The project is ambitious. Training is required with several tools as they become available. This report is a delineation of the project and a substantial portion of the training. It is true that a great deal about CASE Tools and metrics has been learned by this Summer Fellow. Whether this work is continued by this Fellow or another, this report provides the basis for an evaluation of the CASE Tools.

References

1. Billmann, L., Mirab, H. and Winkler, U., "CASCSD-CASE Tools", Measurement and Control, Vol. 25, June 1992, pp. 137-143.

2. Dellen, C. and Liebner, G., "Automated Code Generation from Graphical, Reusable Templates", 10th IEEE/AIAA Digital Avionics Systems Conference Proceedings, IEEE, 1991, pp.299-304.

3. MSFC,"MSFC Software Management and Development Requirements Manual", MM 8075.1A, NASA, August 1993.

4. Williamson, K., "The ASTER Code Generator CASE Tool Evaluation", Internal Report, MSFC, May 12, 1993.