NASA Technical Memorandum 89161

TRADITIONAL VERSUS RULE-BASED PROGRAMMING TECHNIQUES: APPLICATION TO THE CONTROL OF OPTIONAL FLIGHT INFORMATION

WENDELL R. RICKS
KATHY H. ABBOTT

JULY 1987

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

SUMMARY

To the software design community, the concern over the costs associated with a program's execution time and implementation is great. It is always desirable, and sometimes imperative, that the proper programming technique is chosen which minimizes all costs for a given application or type of application.

This paper describes a study that compared the cost-related factors associated with traditional programming techniques to rule-based programming techniques for a specific application. The results of this study favored the traditional approach regarding execution efficiency, but favored the rule-based approach regarding programmer productivity (implementation ease).

Execution efficiency was measured by the number of steps required to isolate hypotheses. A step was defined to be a condition test, function call, or a fetch for information from another body of code. The separate homogeneous rule-base and inference mechanisms of the rule-based program required more steps in the isolation of hypotheses. The best case for the rule-based program was approximately four times less efficient than the traditional program.

The results for programmer productivity were based on the modification ease, verification ease, and the ease of adding explanation capability to each program. These measures were determined by a qualitative summation of the required process for each measure. The separate homogeneous rule-base and inference mechanisms of the rule-based program provided potential for improved programmer productivity.

This study was based on a specific application. The application was both complex and frequently modified, and therefore, tested key features of both programming techniques. Although this study examined a specific application, the results should be widely applicable.

INTRODUCTION

A program that generates correct results, but is too expensive with respect to execution and implementation costs, is a failure (ref. 1). A successful program is one designed to minimize the cost associated with the execution and implementation while maintaining correct results. Since the appropriate programming technique can reduce both execution and implementation costs, decisions concerning the proper programming technique for a specific application are most important. Usually, programmer preference is the factor that determines which technique is used. Most of today's programmers are trained to use traditional programming techniques, and therefore prefer them. However, the successes of expert systems that use rule-based programming techniques have heightened an awareness of a promising new approach. In these efforts, programs which used rule-based programming techniques were easily developed and modified; (ref. 2) thus suggesting that rule-based techniques might serve as a basis for improved programmer productivity in complex and rapidly changing applications.

This paper describes a study that compared rule-based versus traditional techniques for developing a program, the function of which was to control the presentation of optional flight information to the flight crew of an aircraft. The presentation of this optimal flight information depended on multiple combinations of several factors which included, (1) flight phase, (2) control mode settings, (3) signal availability, and (4) switch settings. Although this study examined execution efficiency and programmer productivity based on a specific application, the tradeoffs identified should be widely applicable.

APPROACH

Application.- To determine the tradeoffs between rule-based and traditional programming techniques, an application was necessary so that an evaluation could be performed. A desirable application for this test would be one that required complex decision logic and frequent changes during software development.

An application that fit these requirements was part of a research effort underway in the Flight Management Division at NASA Langley Research Center. This research effort investigated concepts for information display in civil transport cockpits. The flight information of this research effort included both basic information (e.g., indicated airspeed), displayed at all times during the aircraft's operation, and optional information (e.g., reference altitude), displayed only under appropriate conditions.

Presentation of the optional flight information depended on multiple combinations of the flight phase, control mode settings, availability of signals, and switch settings. For example, the reference-altitude pointer was optional information, and was displayed when the following conditions were true: the flight phase was "climb," the control mode setting was "automatic vertical path," the navigation path was "valid," and selector switch 2 was "on."

The rules that determined the display of optional flight information were represented in a decision tree, which was traversed in a data-driven manner. Figure 1 illustrates a portion of that decision tree. The organization of a decision tree allowed common conditions to be grouped near the top of the tree, which facilitated early pruning.

Implementation.- The decision tree for this application was implemented in two programs, one using traditional programming techniques and the other using rule-based programming techniques. Both programs performed the same function, so the end user saw no difference between them. The program using traditional techniques combined the rules and the control structure of the decision tree in nested IF-THEN or IF-THEN-ELSE statements. Figure 2 shows an excerpt from the program code.

The rule-based program, on the other hand, separated the rules from the inference procedures (or inference engine) that manipulated them. Separation of the rules from the inference engine yielded a rule base in which all the rules had a homogeneous syntax.

The rule-based program used frames to organize the homogeneous rules in a decision-tree-like structure (ref. 3). The hypotheses, which for this application were the optional pieces of information, were located in the terminal (or leaf) nodes. The conditions of the rules, which described the conditions that must be true for optional information to be displayed, were located in preceding nodes. The inference engine was a tree traversal algorithm, directed by the successful execution of conditions.

RESULTS AND DISCUSSION

The two prototype programs were evaluated for execution efficiency and implementation ease (programmer productivity). The criterion used to determine the execution efficiency was the total number of steps required to isolate hypotheses that were true. The criteria used to evaluate the implementation ease were (1) ease of modification, (2) ease of verification, and (3) ease of providing explanation capability.

Efficiency.- The objective of evaluating efficiency was to compare the execution cost of each programming technique. It was desirable in this study to obtain a machine-independent measure of execution cost. Therefore, the efficiency of each program was measured by the total number of steps required to isolate hypotheses.

The type of steps used for this measure were (1) function calls, (2) fetches of information, and (3) condition tests. A function call was the transfer of control to different sections of the program. A fetch was defined as the retrieval of information, such as, retrieving rules from a rule base. A condition test was a test for a logical relationship, such as, X > Y. The number of steps were determined by tracing the inference process for different hypotheses.

It was expected that the rule-based program would require more steps to isolate true hypotheses than the traditional program, since the rule-based program had to fetch the rules from the rule base and perform tests to manipulate the tree. A single relationship between the number of steps required for each program was expected (e.g., rule-based steps = 3 times traditional steps). Thus, the objective of this measure was to determine the relationship showing how much more efficient the traditional program was. However, a single relationship among the number of steps was not found due to the traditional program's ability to distinguish mutually exclusive conditions while the rule-based implementation could not.

Mutually exclusive conditions are conditions which cannot be satisfied simultaneously, such as, X = 2 and X = 3. Traditional programs handle mutually exclusive conditions in an IF-THEN-ELSE statement, as shown below:

```
IF X = 2 Then    do action for X = 2
ELSE If X = 3 .....    do action for X = 3
```

The advantage of this capability is that when X is equal to 2, the program will not evaluate the condition X equal to 3. Rule-based programs cannot distinguish sets of mutually-exclusive conditions from those which are not. Therefore, a rule-based program would evaluate the condition for X equal to 3, even after establishing that X was equal to 2.

When no mutually exclusive conditions were evaluated during the isolation of a hypothesis, one relationship between the traditional implementation and the rule-based implementation always resulted. Given J non-mutually exclusive conditions to evaluate, the traditional program required J tests for a total of J steps. The rule-based program required 2J tests, 2J+1 fetches, and 1 function call. Therefore, when no mutually exclusive conditions occurred, the rule-based program always required a total of 4J+2 steps, or approximately 4 times the number of steps required by the traditional program.

However, when mutually exclusive conditions were evaluated, more than one relationship resulted. For example, when there were mutually exclusive conditions and the last condition in the set was true, the relationship between the traditional and rule-based programs remained the same. That is, it required a total of K steps for the traditional program and 4K+2 steps for the rule-based program. However, when the first condition of the K mutually exclusive conditions was true, the traditional program required only one condition test. The rule-based program still required 4K+2 steps.

Adding the capability of efficiently handling mutually exclusive conditions to the rule-based program would therefore, have been beneficial. However, adding this capability to the rule-based program would have been difficult and possibly detrimental to the homogeneity of the rule base. It may have required marking mutually exclusive conditions, which would require the inference engine to always test for them. Adding this capability to the rule-based program would have actually decreased the efficiency by increasing the number of steps required for all cases because of the additional tests.

Therefore, the traditional program was more efficient than the rule-based program. In the best case for the rule-based program, the traditional program was approximately 4 times more efficient. Table 1 provides a summary of the efficiency results. However, one should consider other related information before making a decision based on these results. The development of high-speed symbolic processors may decrease the time needed to execute rule-based programs. This may decrease the impact of the number of steps required. There is also research being performed to develop tools that convert rule-based programs to traditional code. These tools would enable a programmer to obtain the execution efficiency of traditional programs in the final product while using rule-based programming techniques for development.

Modification.- The objective of evaluating modifiability was to determine the degree in which the programming techniques facilitated program modifications. The program's efficiency after a modification was an equally important factor of this criterion. This study determined the modifiability criteria by a qualitative summation of the modification process required by each program.

The traditional program was modified using the same method as employed with the development of most software. Modification in the traditional program required the programmer to search manually through the code and change the code appropriately. In this study, the rules were organized in a data-driven decision tree format. Therefore, all the conditions of a rule for a given display were difficult to locate in the traditional code. Changing, deleting, or adding a condition to a rule, or deleting or adding an entire rule required a very complex search and potentially error-prone manipulation. For example, adding the following rule:

```
     if ATTITUDE-CWS = ENGAGED or VELOCITY-CWS = ENGAGED
and, if FLIGHT PHASE = CLIMB, CRUISE, or DESCENT
and, if NAVIGATION-PATH = VALID
and, if SWITCH5 = ON
     then CROSS TRACK DEVIATION = ON,
```

would have required a very complex search, and the change would have been subject to errors. It would not have been difficult to add a new IF-THEN statement to accommodate the new rule. However, maintaining the efficiency of the already existing rule structure required proper clustering of the conditions in the new rule.

Clustering the above rule would have required a search of the outer-most IF-THEN(-ELSE) statement of the existing code for common conditions with those of the new rule. The programmer would have then placed the remaining conditions of the new rule into an IF-THEN(-ELSE) statement at the location where he found the last match. It is easy to understand how complex and error prone a process like this can be, especially when the decision tree is large.

Determining the overall impact of modifications to the traditional program was also difficult. Interactions among the new rule and the old rules were hard to identify. Erroneous side effects were quite possible with each modification to the rules. In the traditional program, it was the responsibility of the person modifying the program to determine manually all of the side effects. The difficulty was magnified when the number of rules became larger and more complex.

Modifying the rule-based program was easier. This was because the homogeneous rule base, being separate from the inference engine, allowed the rule-based program to access the rules as data. This led to the development of an interactive modification utility for the rule-based program. With this utility, a programmer could add and delete rules interactively. This provided an easier means of code modification than provided with traditional program development. For example, the programmer would have been able to add the new rule given above to the rule-based program by accessing the utility function and providing the new rule in its entirety. The interaction for the above example would be:

```
=> CHANGE-RULEBASE <cr>

For which display? => CROSS TRACK DEVIATION <cr>

(A)dd or (D)elete => A <cr>

Enter each condition of the new rule, followed by a carriage
return.  Terminate rule entry with an "!."

=> ATTITUDE-CWS = ENGAGED OR VELOCITY-CWS = ENGAGED <cr>
=> FLIGHT-PHASE = CLIMB OR FLIGHT-PHASE = CRUISE OR
   FLIGHT-PHASE = DESCENT <cr>
=> NAVIGATION-PATH = VALID <cr>
=> SWITCH5 = ON <cr>
=> !
```

The utility function would have then searched the existing rule base and automatically performed the clustering needed to maintain the efficiency of the decision tree. To say there was no way to provide this interactive modification capability to the traditional program would be false. However, to provide this capability would be very difficult, since it would require a program that could intelligently interpret the entire traditional programming language used. On the other hand, the homogeneous nature of the rule base in the rule-based program made the task easy.

There was still the issue of determining the effects of adding or deleting rules to the rule-based program. In this study, utilities of the rule-based program showed the rules in a textual format when prompted but not the overall impact that a change had on the decision tree. One commercially available program that has this capability is the Automated Reasoning Tool (ART) developed by Inference Corporation. ART displays the clustering of the rule base in a tree-style format, thus giving the developer a visual representation of the impact of an addition or deletion of a rule on the decision tree. This capability is not available with traditional programming techniques.

Therefore, rule-based techniques provided the potential for easier modification with less chance of error. Adding and deleting rules from the traditional code required a programmer to manually perform the tasks that the utility functions of the rule-based program did automatically. Even when a change was completed to the traditional program, there were no automated tools for assessing the impact and side effects. Table 2 provides a summary of the modification results.

Verification.- The objective of software verification is to measure such values as the completeness, accuracy, reliability, and performance of the software. The two major approaches to verification are mathematical verification and verification by software testing. This section will briefly discuss the differences in the two verification approaches and how these approaches differ when applied to traditional and rule-based programs. As with modifiability, this study determined the verifiability by a subjective assessment of the verification process.

In mathematical verification, a formal mathematical proof must insure that the program meets the desired functional and reliability requirements. One way of accomplishing this is to mathematically define the criteria for correct functioning of the program and then prove the program satisfies these criteria (ref. 4). The criteria for a program are usually written as specifications in a natural language. However, when performing mathematical verification, the trend is to use mathematical notation for specifications. This aids in generating more concise and precise specifications. The task of verifying the program is then simplified by proving that a program conforms to its specifications (ref. 5).

Simplifying the specifications can reduce the task of mathematically defining a program's specifications, as the traditional program SIFT does (ref. 4). SIFT formulates its program specifications into a hierarchy. Each consecutive tier of the hierarchy has an easier-to-prove specification than the preceding tier. Rule-based programs could also use this hierarchical

simplification process for the design of the specifications. Neither programming technique was found to have any advantages over the other in the program specifications stage.

The other approach to software verification is verification by software testing. This is the process of comparing statements of intent (specifications) with actuality (the program execution). Three categories commonly used to check adherence to specifications are: static analysis, dynamic analysis, and formal functional analysis (ref. 6). There are many automated tools that integrate these testing methods and yield good results when verifying traditional code (ref. 7).

For rule-based programs, the process of testing that the program is accurate and reliable has two distinct components: (1) checking that the rule base contains all necessary information, and (2) checking that the program can interpret and apply this information correctly (ref. 8). During this process, rule-based programs should be able to employ the testing methods used with traditional code (e.g., static analysis, dynamic analysis, and formal functional analysis). Again, neither programming technique was found to have any advantages over the other in the application of software testing techniques.

However, the separate, homogeneous rule base of rule-based programs may be an advantage in other stages of verification. First, code simplification is an important step in all forms of verification. The separation of the rule base and the inference engine should ease the simplification task (ref. 9). Also, it is already possible with rule-based systems to easily trace the program's reasoning process, set up an interactive mechanism for reviewing and correcting the program's conclusions, and to provide explanation capabilities. The ability to easily add these capabilities to rule-based programs could be helpful in developing automatic testing capabilities for rule-based programs.

Other advantages in verifying rule-based programs may reside in the ability to test the rules before the rule base or inference mechanism are completed. When a rule-based program is being developed, the program builders can run preliminary checks on the knowledge base before the full reasoning mechanism is functioning, and without gathering actual data for test runs (ref. 8). Testing during the knowledge acquisition should prove particularly helpful when working with large rule bases.

In summary, simplification is a major concern in all verification methods. Rule-based programs have separate rule bases and inference engines, which could aid the simplification process. The separate, homogeneous rule base could also be an advantage when developing testing tools and performing preliminary tests. Table 3 provides a summary of the verification results.

Explanation.- Software's ability to explain its actions is a relatively new capability, which originated in rule-based expert system programs. The advantages of explanation capabilities in program development prompted its use as an evaluation criteria. Explanation capabilities are particularly helpful in program debugging and after program modification. Explanations are used when debugging to determine why certain results occur. They also help

determine the effects of a rule modification to other rules in the rule base. It is, therefore, important to have this capability and to be able to implement it easily.

Traditional programming techniques embed the rules into the control structure, which prevents the use of the rules for more than one purpose. To provide explanation capability to the traditional program would have required that the rules be repeated in different program statements. This also meant that each time a modification of the decision tree was necessary, the programmer would have to make the change at every occurrence of the rules. Changing the rules would therefore be more complex, which potentially increases the probability of error. An explanation capability was not implemented in the traditional program of this study.

However, the ease of adding explanation features to the rule-based program yielded two types of explanation capabilities. One type of explanation was to show all the rules for a given hypothesis - command SHOW-RULE. A user could give the command SHOW-RULE at any time to show all the rules for a given hypothesis (i.e., optional display). For example:

```
=> SHOW-RULE <cr>
For which display? => CROSS TRACK DEVIATION DIGITS <cr>
```

would display all the rules that determined the display of cross track deviation in digits.

The other explanation function implemented in the rule-based program was the WHY function. A user could invoke the WHY function to inquire which rule determined a current hypothesis. For example:

```
=> WHY <cr>
For which display? => CROSS TRACK DEVIATION DIGITS <cr>
```

would display the rule that caused cross track deviation in digits to be displayed.

Explaining a (rule-based) program's actions can be as simple as stating the corresponding rule, if the information in the rule adequately shows why action was taken (ref. 10). Therefore, adding explanation capability to the rule-based program was a much simpler task. This again was due to the separate rule base and inference engine in the rule-based program. The homogeneous rule base of the rule-based program made it possible to access the rules and manipulate them as a data. This simplified and reduced the amount of code needed. There remained only one representation of the rules, with different control structures accessing the rules for explanations. Therefore, modifications to the rules need only be done in one location, the rule base.

Adding explanation capabilities to the rule-based program, therefore, was simpler than it would have been for the traditional program. This was due to the difference between embedding and separating the inference mechanism and rule base. Being able to access the rules of the rule-based program and manipulate them as data simplified the code needed for the explanation capability. The traditional program would have needed to repeat the rules,

once for the explanation and once for the reasoning itself. This would increase the amount of code and potentially the difficulty of managing it. See Table 4 for a summary of the explanation results.

CONCLUDING REMARKS

This paper described a study performed to compare traditional programming techniques to rule-based techniques, given a specific application. The application used for this study was controlling the display of optional flight information in a civil transport cockpit. This application required complex decision logic and a frequently modified rule base.

The traditional program was more efficient in execution than the rule-based program. That is, the traditional program required fewer steps to isolate a true hypothesis. The exact relationship in the number of steps between the two programs differed depending on whether the set of conditions tested consisted of mutually exclusive conditions or not. Overall, the rule-based program typically required about four times as many steps as the traditional program. However, high-speed symbolic processors and software tools for converting rule-based programs to traditional code may reduce this disadvantage.

The results show that rule-based programming techniques have the potential for improving the productivity of the programmer or designer who develops a system. In this study, modification of the rule-based program was easier, more efficient, and less error-prone than the traditional program's. The rule-based program's separate, homogeneous rule base and inference engine could aid in the simplification and test-tool development needed during the verification process. It was also easier to implement an explanation capability in the rule-based program.

REFERENCES

1. Schneiderman, B.: Software Psychology. Little, Brown, and Company (Canada) Limited, 1980.

2. Doyle, J.: Expert Systems and the 'Myth' of Symbolic Reasoning. IEEE Transactions On Software Engineering, Vol. SE-11, No. 11, November 1985, pp. 1386-1390.

3. Winston, P.; and Horn, B.: LISP. Second edition. Addison-Wesley Publishing Company, 1984.

4. Melliar-Smith, P. M.; Schwartz, R. L.; C.S. Laboratory; and C.S. and Technology Division: Formal Specification and Mechanical Verification of SIFT: A Fault-Tolerant Flight Control System. Technical Report CSL-123, March 1981.

5. Wulf, W.; Shaw, M.; Hilfinger, P.; C.S. Dept. Carnegie-Mellon University; and Flon C.S. Dept. University of Southern California: Fundamental Structures of Computer Science. Addison-Wesley Publishing Company, 1981.

6. Taylor, R. N.: An Integrated Verification and Testing Environment. SOFTWARE-Practice and Experience, Vol. 13, 1983, pp. 697-713.

7. Senn, E.; Ames, K.; and Smith, K.: Integrated Verification and Testing System (IVTS) for HAL/S Programs. Presented at the (IEEE, ACM, NBS) Softfair '83 Conference, Arlington, Virginia, July 26-28, 1983.

8. Suwa, M.; Scott, A.; and Shortliffe, E.: An Approach to Verifying Completeness and Consistency in a Rule-Based Expert System. Dept. of Computer Science, Stanford University, Stanford, California, Report No. STAN-CS--82-922, June 1982.

9. Alvarez, R.: On Software Aspects of Strategic Defense Systems. Communications of the ACM, April 1986, pp 262-265.

10. Buchanan, B.; and Shortliffe, E.: Rule-Based Expert Systems. Addison-Wesley Publishing Company, 1985.

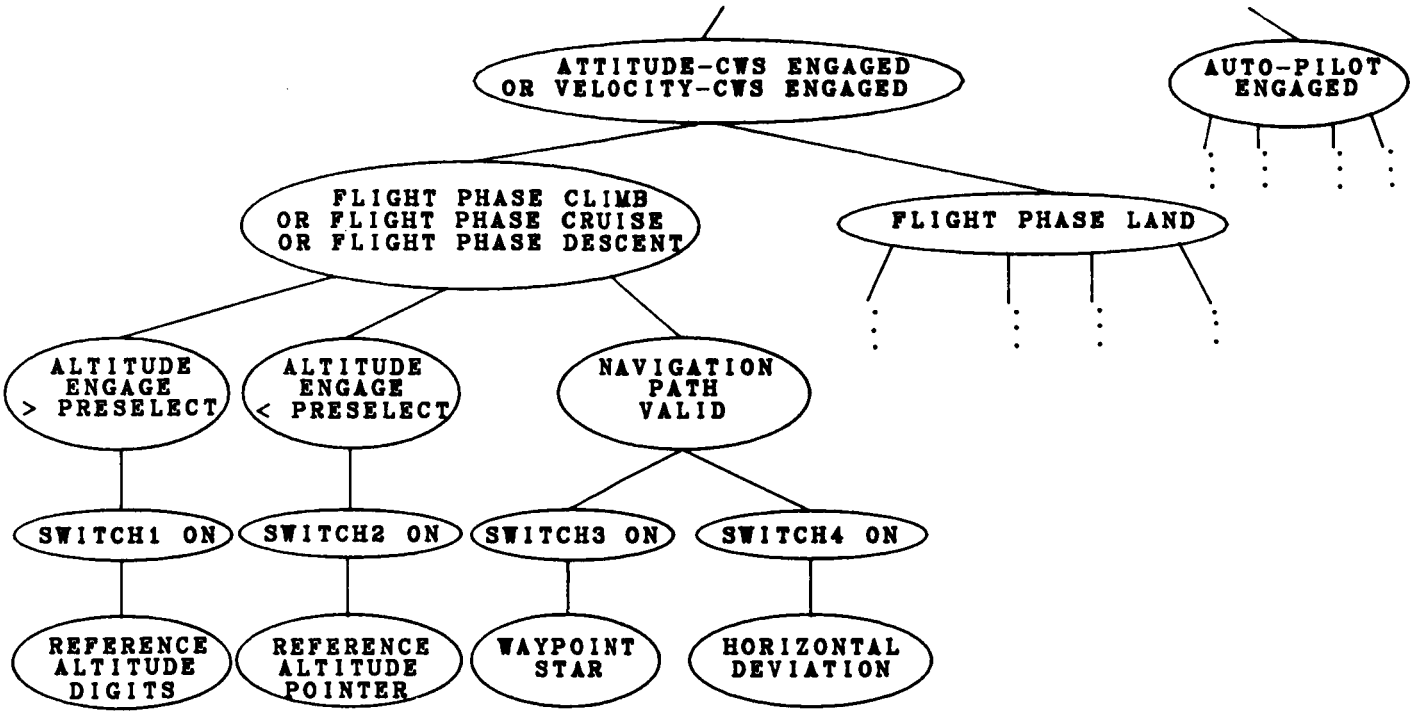

Figure 1.- Portion of Decision Tree.

```
IF ATTITUDE-CWS = ENGAGED OR VELOCITY-CWS = ENGAGED THEN
  IF FLIGHT-PHASE = CLIMB OR FLIGHT-PHASE = CRUISE OR FLIGHT-PHASE = DESCENT
    IF ALTITUDE-ENGAGE > PRESELECT THEN
      IF SWITCH = ON THEN
        REFERENCE-ALTITUDE-DIGITS = ON
    ELSE, IF ALTITUDE-ENGAGE < PRESELECT THEN
      IF SWITCH2 = ON THEN
        REFERENCE-ALTITUDE-POINTER = ON
    IF NAVIGATION-PATH = VALID THEN
      IF SWITCH3 = ON THEN
        WAYPOINT-STAR = ON
      IF SWITCH4 = ON THEN
        HORIZONTAL-DEVIATION = ON
  ELSE, IF FLIGHT-PHASE = LAND THEN
    .
    .
    .
ELSE, IF AUTO-PILOT = ENGAGED THEN
    .
    .
    .
```

Figure 2.- Excerpt from Traditional Representation.

	FETCHES	TREE MANIPULATION	FUNCTION CALLS	MUTUALLY EXCLUSIVENESS
TRADITIONAL	NONE	NONE	NONE	DOES
RULE-BASED	FOR CONDITIONS AND FUNCTION CALLS	TESTS FOR TERMINAL NODES	TRAVERSE RULE BASE	DOES NOT

Table 1.- Summary of the Efficiency Results.

	CHANGES	MAINTAINING EFFICIENCY	RECOGNIZING SIDE EFFECTS
TRADITIONAL	EXTERNAL EDITOR	PROGRAMMER MAINTAINS	PROGRAMMER
RULE-BASED	RUN-TIME	UTILITY FUNCTION MAINTAINS	SOFTWARE AIDS

Table 2.- Summary of the Modification Results.

	RULES	CONTROL STRUCTURE	UTILITIES
TRADITIONAL	WITH CONTROL STATEMENTS	WITH EMBEDDED RULE-BASE	USER INTERACTION
RULE-BASED	WHILE ACQUIRING	SEPARATE	CODE MANIPULATION

Table 3.- Summary of the Verification Results.

	IMPLEMENTATION	CODE MANAGMENT
TRADITIONAL	REQUIRES DUPLICATION OF RULES	INCREASES THE DIFFICULTY
RULE-BASED	ABLE TO USE THE ORIGINAL SET OF RULES	MINIMUM IMPACT

Table 4.- Summary of the Explanation Results.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-89161		

4. Title and Subtitle	5. Report Date
Traditional Versus Rule-Based Programming Techniques: Application to the Control of Optional Flight Information	July 1987
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Wendell R. Ricks Kathy H. Abbott	
	10. Work Unit No. 505-67-41-01

9. Performing Organization Name and Address	11. Contract or Grant No.
NASA Langley Research Center Hampton, VA 23665-5225	
	13. Type of Report and Period Covered Technical Memorandum

12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
National Aeronautics and Space Administration Washington, DC 20546	

15. Supplementary Notes

16. Abstract

To the software design community, the concern over the costs associated with a program's execution time and implementation is great. It is always desirable, and sometimes imperative, that the proper programming technique is chosen which minimizes all costs for a given application or type of application. This paper describes a study that compared the cost-related factors associated with traditional programming techniques to rule-based programming techniques for a specific application. The results of this study favored the traditional approach regarding execution efficiency, but favored the rule-based approach regarding programmer productivity (implementation ease). Although this study examined a specific application, the results should be widely applicable.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
AI Rule-Based Systems Evaluation	Unclassified - Unlimited Subject Category 59

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	14	A02