Evaluating Programming Languages: The Power of M

by James Rooney

Many people often make blanket statements comparing one programming language to another, for example: "C is better than COBOL" or "M is dead because C++ is easier." Technical personnel tend to take these statements at face value, perhaps because they are so attached to the idea of "new and improved" that they do not analyze the validity of statements which reinforce our idea of "technological correctness."

This is a small effort to apply objective and measurable criteria to evaluating the pros and cons of programming languages and software development environments. In his 1989 book, *Concepts of Programming Languages*, Robert Sebesta suggests the following criteria for evaluating a programming language. (Please note that I use the term "evaluate." Evaluation is our privilege as individuals. The world and the marketplace will be the ultimate judge of the success or failure of a language or technology. Does anyone remember ALGOL?)

Readability: Overall Simplicity

A language with many elementary components is more difficult to learn than one with a smaller number of components. M, for example, has a very small number of primary language components. Simplicity can also be reflected by a minimum of "multiplicity features," meaning the ability to accomplish a simple operation in more than one way. For example, in C an integer counter can be incremented four ways: count = count + 1; count++; ++count; count += 1.

Contributing to the complexity of a language is "operator overloading." In M the + operator represents (1) integer addition, (2) floating point (real number) addition, and (3) unary canonic representation. While (1) and (2) are common to many languages and readily accepted, (3) represents a feature unique to M.

Orthogonality

By orthogonality, I mean a measure of the ways in which a relatively small number of language components can be combined with other language components to build the control and data structures of a language. Orthogonality is closely related to simplicity. The more orthogonal the design of a language, the fewer exceptions the language rules require. (All who have struggled to learn English will testify to its difficulty in learning due to the myriad of language exceptions.) In M one uses the SET command to both assign values to a variable as well as to save data to disk.

Control Statements - The availability of control statements within a language permits greater use of structured programming. Structured programming permits easier top-to-bottom reading of a program. How often do you HAVE to use a GOTO statement in M?

Data Structures - The presence of facilities to define data types and data structures in a language is another aid to readability, syntax checking, and debugging. Which is easier to read and infer design purpose: done= 1 or done = true?

Syntax Considerations - Variable identifiers (names) with descriptive names are easier to read and understand. Variable length, the character set used to make up variables, and the availability of connector characters (SUMOFCLAIMS vs. SUM_OF_CLAIMS), all improve readability.

Another aid to improved syntax and readability is the use of special or reserved words. The use of words or constructs as elements of program flow speed development and ease code maintenance. Also important are whether or not these words can be used as names for program variables or subprocesses, as this may confuse a person reading the program. For example, in M this

is a valid (but very confusing) statement: DO:do DO(DO) ; invoke a pile of dodo.

Writability - Writability is a measure of the ease in which a language can create programs to accomplish a specific task. A glaring example of poor language selection is using M to solve a system of linear equations. Not only is M not optimized for this task, the programmer would have to develop (and debug) the subroutines herself instead of relying on an existing library.

Simplicity and Orthogonality

As noted by Hoare (*Hints on Programming Language Design*, Proceedings of ACM, 1973), a smaller number of primitive constructs (simplicity) and a consistent set of rules for applying them (orthogonality), is much better than having a larger number of primitives.

Abstraction - In brief, abstraction means complicated structures stated in simple ways by ignoring (or allowing the underlying language or technology to handle) many of the implementing details. A couple of examples in M come to mind: First, the B tree structure of M disk storage is an elegant and efficient way to store sparse, yet related, data. Second, judicious use of indirection can greatly aid in "hiding" irrelevant detail unrelated to the task at hand.

Reliability - Reliability is a measure of how well a program performs to specifications under all conditions.

Type Checking - This is the testing for type compatibility between two variables. While a powerful feature in M, the lack of type checking has led to countless program errors.

Exception/Error Handling - The ability of a program to gracefully intercept and handle runtime errors is a great aid to program reliability and system support. A measure of Visual Basic's lack of this feature is the number of add-on tools vendors are offering to provide this capability.

Aliasing - This refers to the ability of a language to have two or more means of referencing the same address in memory or on disk. It is becoming more accepted that the ability to alias the same location is too dangerous to justify its advantages. In M, while one could use indirection to "alias" a global reference, widespread use of aliases will guarantee unreadable and difficult-to-maintain code.

Readability and Writability - A program written in a language that allows one to naturally express concepts and methods will be easier to read and write.

Cost - The ultimate cost of a programming language or technology is a function of many things including training costs, availability of experienced programmers, how quickly programmers become productive, etc. The cost of writing programs is a function of the "expressiveness" of the language. Of the following, which is more expressive and easier to develop: FORTRAN vs. Assembler, COBOL vs. FORTRAN, or M vs. COBOL?

A friend of mine is the Deputy Director of Software Management for the Dept. of the Army, U.S. DoD. He attributes the lack of acceptance outside DoD for the Ada programming language to the complexity and costs for designing and building systems in Ada (cost of compilation and linking, cost of executing, and cost of maintenance). (Hmm... same problem but the flip side of the coin?)

A program which is interpreted or which requires many runtime checks (like PL/1) will prohibit fast code execution. It will also require increased maintenance. There are many factors to maintenance cost (documentation, change and configuration management, etc.), but the greatest impact on maintenance cost is readability. How quickly can a programmer look at unfamiliar code and determine its purpose and function?

Summary

A final note on programming languages. Most criteria, including readability or writability are neither measurable or scientifically defined. They are, however, useful concepts and provide valuable insight when evaluating a given programming language.

I write this article not to refute or reinforce any position on M made by persons throughout our industry, but rather to offer to the M community and to others watching us, a more formal and objective standard by which one can evaluate a programming language or technology. M

Jim Rooney is Manager for Oleen Healthcare Information Management, Bethesda, MD.