The DSN Programming System

A. P. Irvine
TDA Engineering Office

The DSN Programming System is described by a heuristic model. Interaction with two elements of that system, anomaly reporting and the MBASIC^{tm} language, is described in detail.

I. Introduction

The DSN Programming System is composed of a body of methodologies, tools, and practices whose major components are the following:

(1) DSN Software Standard Practices.

(2) DSN standard programming languages.

(3) Software implementation aids.

(4) Management aids.

The goal of the system is to produce software:

(1) On time.

(2) Within budget.

(3) Conforming correctly to functional requirements.

(4) Of a low life-cycle cost.

Reference 1 presents a hierarchical model of the DSN Programming System, and Ref. 2 describes each major element in detail. This article will describe information flow generated from activities which provide a heuristic process for the DSN Programming System.

II. The Heuristic Model

Heuristic is defined in Ref. 3 as "pertaining to exploratory methods of problem-solving in which solutions are discovered by evaluation of the progress made toward the final result." Figure 1 shows a heuristic model of the DSN Programming System, with information source elements as well as key component elements of the end-to-end system. Also, the paths of information and feedback between the various key elements are shown.

A. Inputs

The inputs to the model, as shown in Fig. 1, consist of the following:

1. **TDA planning.** The Telecommunications and Data Acquisition (TDA) Planning Office is responsible for generating a DSN Programming System Long-Range Plan which will give, in broad outlines, the general direction of the system.

2. **Technology development.** Programs developed under the direction of the TDA Technology Development Office provide appropriate research and prototypes prior to implementation.

3. **Research from NASA Centers**. Of particular interest is work being done at the Software Quality Laboratory of the Goddard Space Flight Center and work done under the Multi-Purpose User-Oriented Software Technology (MUST) Project, led by the Langley Research Center.

4. **Other research**. Both industry and universities actively engage in research in the software area. A goal of the Programming System is to make use of research from these sources. An example of this technology transfer would be the use of Program Design Language (PDL) (Ref. 4) instead of flow charts for software documentation.

5. **Operations needs**. The operations organization provides feedback which helps to identify needed improvements to the man-machine interface and to software documentation and operating procedures.

6. **Implementation activities**. The process of implementing software helps to identify and focus needs which should indicate future activities for the Programming System. The necessity for implementation tools and the usefulness of certain procedures are examples of this kind of feedback. Other paths of feedback contain information concerning anomaly rates and schedules which are outputs of the implementation process.

7. **Work Authorization Document (WAD)**. The WAD serves as an input of constraints to the Programming System for any given fiscal year.

B. Outputs

The output from the model consists of information which enables the system to direct activities of the components shown in Fig. 1 and listed in the Introduction. All outputs of the system eventually feed into the implementation process.

This article will discuss two subcomponents of the system and how the direction of their activities consists of a heuristic process. The first, anomaly reporting, is a subcomponent of management aids, but is shown as a separate feedback path in Fig. 1 because of its importance to the heuristic model. The second is the MBASIC^{tm} language which is a subcomponent of DSN Software Standard Programming Languages.

1. **Anomaly reporting**. Of all the paths of feedback, that of anomaly reporting is one of the more significant. Anomalies can be reported against any discrepancy noted during verification testing and acceptance testing or subsequent transfer. Anomalies are charged against the subsystem or assembly in which the originator believes the anomaly resides. Reference 5 describes how an anomaly rate lower than industry-reported averages was experienced by the DSN Mark III Data Subsystems Implementation Project (MDS) using the DSN Programming System methodology.

The rate of anomalies incurred is only a quantitative measure for feedback. A qualitative assessment is also necessary to determine from where in the software life-cycle many of the errors originate. There are approximately 1000 reports available which are currently classified as to subsystem and severity. Severity is defined as follows:

Category A:	Critical to the operation of the software. The software will not function as required.
Category B:	Does not meet specifications. The software functions, but may operate in such a manner as to lead to a misunderstanding of the performance.
Category C:	Does not prevent software from operating and satisfying all requirements, but is operationally undesirable.

An effort is being initiated by which causal relationships may be uncovered. This will be done by:

(1) Identifying existing software error taxonomies in use that capture meaningful data.

(2) Auditing historical anomaly reports and assigning each anomaly to the appropriate error type.

(3) Also, auditing historical anomaly reports for the time-wise occurrences in relation to the software life-cycle.

(4) Analyzing the data for trends and significant clusterings of error data.

The outcome of this analysis should point to areas where efforts of the system should be directed.

2. **The MBASIC^{tm} language**. The MBASIC^{tm} language is the DSN standard non-real-time language and is described in Ref. 6. Its initial implementation was as an interpreter. An interpreter essentially consists of an executive routine that, as computation progresses, translates a stored program expressed in some pseudocode or source language into machine code and performs the indicated operations by means of subroutines as they are translated. This method of code execution is much slower than directly executing machine code.

Feedback from costs expended, as reported by users, showed that for Central Processing Unit (CPU) bound jobs in production mode, an interpretive method of execution was very expensive. Based on estimated expenses incurred by the DSN in FY'79, it was determined that the implementation of the MBASIC^{tm} language as a compiler was cost-effective. The speed increase needed to pay for the implementation of the compiler is a five-fold increase in CPU execution times. Any greater increase would not appreciably affect costs. Figure 2 shows the relation between increase in speed and cost-savings.

It has been calculated (Ref. 7) that considerable savings can be realized by the user of higher-order languages. This is true in the implementation of compilers as well as real-time applications programs because compilers exhibit many of the features of a real-time program. Consequently, the PASCAL language was selected as the language of implementation for the MBASIC^{tm} compiler. The University of Wisconsin implementation of this language was chosen as the most suitable compiler for this effort. This compiler was recommended by users at the Langley Research Center.

DEMOBASIC, a subset of the MBASIC^{tm} language, was implemented on the MODCOMP II and was successfully demonstrated during the Configuration Control and Audit Demonstration. That activity terminated in December 1978. Feedback from the demonstration provided the information and incentive to pursue a complete MBASIC^{tm} implementation for the MODCOMP II. DEMOBASIC has been upgraded to match the implementation currently available on the Univac series of computers and is scheduled to be transferred to operations September 15, 1980.

III. Conclusions

Feedback from anomaly reporting indicated that the methodology resulted in a low anomaly rate (Ref. 5) and thereby also provided positive feedback. Further analysis of anomaly reporting will provide more valuable information for future efforts. The need to reduce operating costs prompted the implementation of the MBASIC^{tm} compiler, which was written in a higher order language suggested by another NASA Center.

The DSN Programming System can be considered to be both adaptive and heuristic, incorporating new technology as it is introduced and redirecting emphasis as information relevant to the evaluation of progress toward a fixed goal becomes available.

References

1. Hodgson, W. D., "The DSN Programming System," in *The Deep Space Network Progress Report 42-41*, pp. 4-9, Jet Propulsion Laboratory, Pasadena, Calif., Oct. 15, 1977.
2. Irvine, A. P., "The DSN Programming System," in *The Deep Space Network Progress Report 42-50*, pp. 4-6, Jet Propulsion Laboratory, Pasadena, Calif., Apr. 15, 1979.
3. *Computer Glossary*, The Funk and Wagnalls Library of Computer Science, Funk and Wagnalls, New York.
4. Caine, S. H., and Gordon, E. K., *PDL – A Tool for Software Design*, 1975 NCC, AFIPS Press, Montvale, N.J., 1975.
5. Irvine, A., and McKenzie, M., "Evaluation of the DSN Software Methodology," in *The Deep Space Network Progress Report 42-48*, pp. 72-81, Jet Propulsion Laboratory, Pasadena, Calif., Dec. 15, 1978.
6. *MBASIC^{tm} Manual*, Volumes I and II, Jet Propulsion Laboratory, Pasadena, Calif., Aug. 1975.
7. McKenzie, M., "Cost Evaluation of a DSN High Level Real-Time Language," in *The Deep Space Network Progress Report 42-42*, pp. 214-225, Jet Propulsion Laboratory, Pasadena, Calif., Dec. 15, 1977.

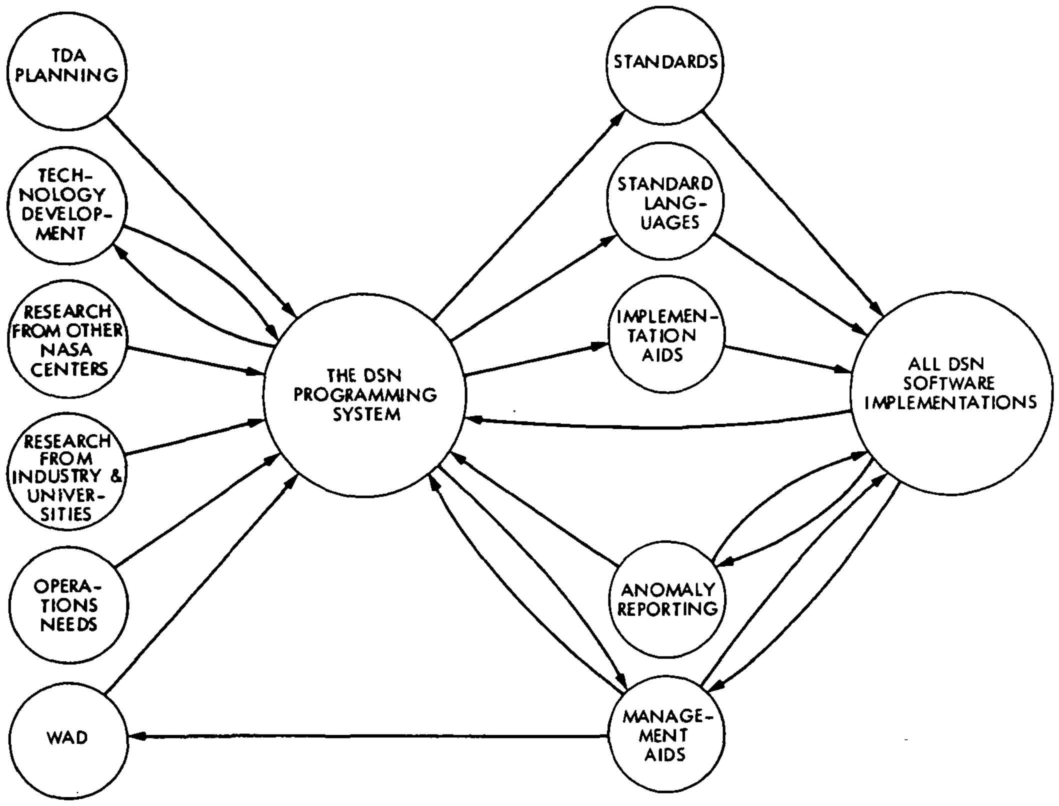

Fig. 1. Heuristic model of DSN Programming System

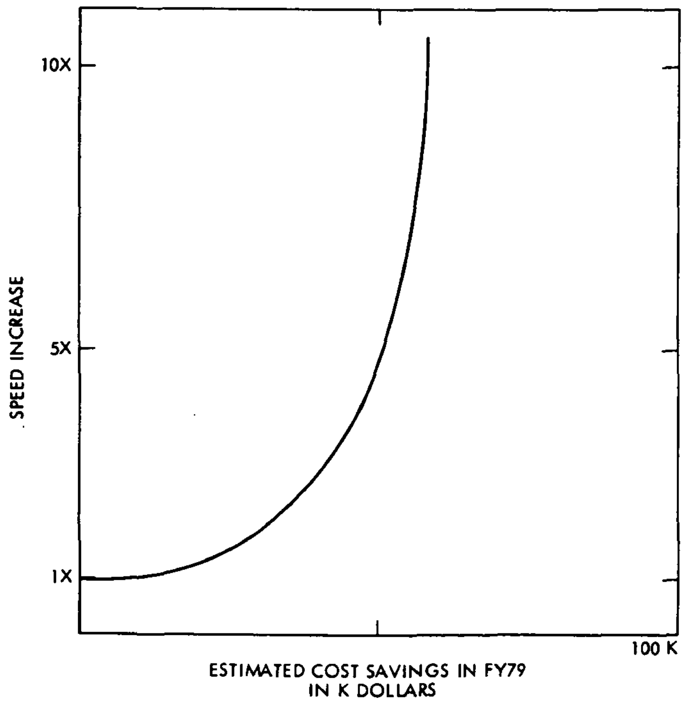

Fig. 2. Speed vs cost