1

6 Report to the ~~HOLWG~~ High Order Language Working Group (HOLWG)

11 14 Jan[redacted]77

EXECUTIVE SUMMARY

12 87p.

LANGUAGE EVALUATION COORDINATING COMMITTEE

Prepared by:

10

Serafino/Amoroso ~~(Chairman)~~	CENTACS ✓ Fort Monmouth, NJ
~~and~~	
Peter Wegner (~~Technical A~~visor)	Brown University Providence, RI

~~and with assis~~tance from:

Derek/Morris	CENTACS, Fort Monmouth, NJ
Douglas/White	RADC, Rome, NY
Warren/Loper	NELC, San Diego, CA
Lloyd Campbell	BRL, Aberdeen, MD
Calvin Showalter	NAVAIR, Washington, DC

410 122

Table of Contents

0. Executive Summary

The objectives of the language evaluation coordinating committee are to evaluate, summarize, and structure the findings of the language evaluation reports. In this executive summary, we present the essentials of these findings. An expanded version of this summary is found in Section 4.

Among all the languages considered none was found that satisfies the requirements so well that it could be adopted as <u>the</u> Common Language.

Each feature or capability mentioned in the requirements document can essentially be found in some existing language, hence, some minimal collection of languages exists which collectively would contain all these features and capabilities. However, there are important embedded computer system applications that could make good use of all the major requirements. In fact, most of the requirements would be useful in any embedded computer system applications.

All evaluators felt that the development of a single language satisfying the requirements was a desirable goal.

It is clearly possible to design a language by brute force containing all the technical features and capabilities of the requirements. Problems arise, however, when one adds the general requirements such as simplicity, uniformity, reliability, design integrity, implementation efficiency, etc.

The concensus of the evaluators was that it would be possible to produce a language within the current state of the art meeting essentially all the requirements.

Some evaluators felt that certain requirements should be modified. However, it was felt almost unanimously that the development of a language meeting essentially all of the requirements was both feasible and desirable. The precise degree of trade-off between potentially conflicting requirements (such as simplicity and generality) can be determined only after a substantial amount of additional work on the design of the language.

Almost all the evaluators felt that the process of designing a language to satisfy all the requirements should start from some carefully chosen base language.

Working from a base language would reduce the amount of work required and would reduce the opportunities for making errors. There was no consensus as to which base language to use, but every evaluator indicated that some of the languages considered were more suitable for this role than others. There was unanimous agreement among evaluators concerning the unsuitability of certain languages to serve as base languages.

Even though almost all felt that a modification effort should start from a base language, all felt that a design team must have the freedom to make any changes to a base language they feel is warranted. Hence, the specification for the design effort should be carefully written to avoid any artificial or undesirable restrictions on a design team. For example, upward compatibility with a base language should not be a restriction.

Without exception, the following languages were found by the evaluators to be inappropriate to serve as base languages for a development of the Common Language: FORTRAN, COBOL, TACPOL, CMS-II, JOVIAL J73, JOVIAL J3B, SIMULA 67, ALGOL 60, and CORAL 66.

This should not be interpreted as a statement concerning the technical merits of these languages. Some of the languages on this list are among the most widely used of the languages considered. It is only a statement concerning their suitability to serve as bases for a language modification relative to all the other languages considered. The reasons for rejecting these languages are varied and are discussed in some detail in Section 3 of this report.

The languages that remain are: PASCAL, ALGOL 68, PL/I, LIS, EUCLID, CS-4, PDL/2, RTL-2, LTR, PEARL, SPL/I, HAL/S, ECL, and MORAL.

We believe the following recommendation is consistent with almost all the evaluations.

Proposals should be solicited from appropriate language designers for modification efforts using any of the languages PASCAL, PL/I, or ALGOL 68 as base languages from which to start. These efforts should be directed toward the production of a language that satisfies the DoD set of language requirements for embedded computer applications.

The languages that have not been found to be inappropriate to serve as base languages are HAL/S, PEARL, SPL/I, PDL/2, LTR, RTL/2, MORAL, EUCLID, LIS, CS-4, and ECL. Most of these are modifications of one of the languages PASCAL, PL/I, or ALGOL 68 for an application area close to that with which we are concerned. Many of these languages have features that satisfy certain important DoD language requirements in especially interesting ways. Hence, many of these languages are relevant design experiences that should be considered. The design teams should feel free to make as much use as is deemed appropriate of any of these languages.

At some appropriate time some choice should be made among these design efforts to determine which are most worthy of being continued to completion.

1. INTRODUCTION

1.1 Background

The high costs for software for systems developed within the Department of Defense is receiving increased attention from the highest levels of management. The major part of these costs is for software for what are called "embedded computer systems." Such systems would include tactical weapon systems, command and control systems, avionics systems, etc.

As part of the overall process of investigating the costs of software, in January 1975 a High Order Language Working Group (HOLWG) was chartered by the Department of Defense with representatives from the three services. The purpose of this group is to investigate the requirements and specifications for programming languages for embedded computer applications and to recommend the adoption or implementation of the necessary language or languages to achieve an appropriate degree of commonality of programming language usage in the services.

The first task undertaken by the HOLWG was to formulate a set of requirements for a language, or a set of languages, for these applications. This task which involved the user, research, and development organizations in the services, and the general research and industrial communities, resulted in Jan.1976 in a document informally called "TINMAN." This document which involved several iterations was believed at the time to be the final set of DoD language requirements for embedded computer applications.

The HOLWG then initiated a number of studies to investigate how closely certain existing standard languages came to satisfying these requirements. Besides the language studies sponsored by the services through the HOLWG, several other evaluations of other existing languages against the TINMAN set of requirements were also volunteered. Some of these by organizations outside the United States.

The present report has been prepared by a subcommittee of the HOLWG. Its purpose is to report to the HOLWG a consolidation of the evaluation studies. Based on the results reported from these studies the subcommittee has attempted to resolve differences, identify consensus positions, and to determine the basic findings of these studies.

1.2 Languages and Contractors

The task of evaluating languages against the TINMAN requirements was carried out by six contractors, two chosen by each service. Softech and CSC were chosen by the Army. Intermetrics and RLG were chosen by the Navy. IBM and SAI were chosen by the Air Force. The 23 evaluated languages and the contractors who evaluated them are indicated in figure 1. The languages above the line represent the initial set of languages chosen for evaluation, while the languages below the line represent languages added after the evaluation process was initiated.

	SOFTECH	INTERMETRICS	RLG	CSC	SAI	IBM	OTHER
FORTRAN		X				X	
COBOL		X				X	
PL/I		X				X*	
TACPOL	X	X	X				
HAL/S						X	
CMS-2			X	X			
CS-4		X	X	X	X		
J-3B	X			X			
J-73		X		X			
ALGOL 60					X		
CORAL 66			X	X			X
ALGOL 68	X				X		
SIMULA 67	X						
PASCAL	X		X				

LIS					X		X
LTR							X
RTL/2							X
PEARL				X			X
SPL/I				X			
EUCLID				X			
MORAL							X
ECL							X
PDL/2							X

Figure 1.

*The IBM evaluation of PL/I was not done under the Air Force sponsored contract.

2. The Requirements

In order to make this document more self contained we shall briefly describe the set of DoD language requirements used for the evaluations. The brief description of each requirement given below is necessarily incomplete but does give an idea of the nature of each requirement. From Jan. 1976 when the TINMAN document first appeared until the present time, many comments and critiques have been prepared. These together with a workshop on the TINMAN requirements held at Cornell in late Sept. and the intensive use of the requirements in the language evaluations, has led to a better understanding of how the requirements should be formulated, and resulted in a new requirements document called the IRONMAN which appeared in Jan. 1977. In this document the DoD requirements have been organized in a very different fashion from that found in TINMAN, but the requirements are sufficiently similar in substance that the recommendations given here are not affected.

2.0 Organization of General and Specific Requirements

There are two levels of DoD requirements which we shall refer to as general and specific. General requirements are global language characteristics related to the overall design objectives for the language while specific requirements are concerned with specific language features. The general requirements may be summarized as follows:

Simplicity: avoid unnecessary generality or complexity
Reliability: properties to aid in program safety and error detection
Readability: readability is more important than writability
Maintainability: emphasize modularity, clarity, documentation, few defaults
Efficiency: no sacrifice of run time efficiency for generality
Implementability: state of the art features with known implementation
Machine Independence: well defined interface to object machines
Portability: adaptable to different object machines and different applications
Definition: unambiguous, complete, understandable definition

The above objectives reflect the fact that the costs of the operations and maintenance part of the software life-cycle for embedded computer applications are generally considerably greater than the costs of program development, and the fact that embedded computer applications are often subject to stringent real-time constraints.

These general requirements serve to motivate the choice of specific requirements, and in many instances constrain the way that a specific requirement should be realized or implemented.

The 98 specific TINMAN language requirements are grouped into the following 13 categories:

A. Data and Types (7 requirements)
B. Operations (11)
C. Expressions and Parameters (9)
D. Variables, Literals and Constants (6)
E. Definition Facilities (8)
F. Scopes and Libraries (7)
G. Control Structures (8)
H. Syntax and Comment Conventions (10)
I. Defaults, Conditional Compilation and Language Restrictions (7
J. Efficient Object Representations (5)
K. Program Environment (5)
L. Translators (9)
M. Language Definition, Standards and Control (6)

The above organization of language characteristics is somewhat different from the organization of language descriptions in programming language manuals and language definitions. The IRONMAN requirements specification has reorganized the requirements so that they correspond more closely to the order of presentation of language features in language definition documents, but has not substantially altered the overall requirements for the common language. In order to bring the reader up to date concerning requirements, the 13 categories of IRONMAN requirements are listed below:

1. General Requirements (8)
2. Syntax and Comment Conventions (9)
3. Data Types (3)
4. Expressions (7)
5. Constants, Variables, and Declarations (7)
6. Control Structures (5)
7. Functions and Procedures (9)
8. Input-Output (5)
9. Parallel Processing (6)
10. Exception Handling (6)
11. Machine Dependent Specifications (6)
12. Library, Separate Compilation, Generic Definitions (4)
13. Standards, Translation and Support (7)

The 98 individual specific TINMAN requirements are briefly characterized below.

2.1 Data and Types

The requirements in the "Data and Types" category may be summarized as follows:

A1. Date types determinable at compile time and unalterable at run time.
A2. Integer, fixed, float, Boolean, character, array and record types.
A3. Precision specs for floating point arithmetic and variables.
A4. Exact fixed point numbers with user specified range and fractional part.
A5. Character sets with user defined collating sequence.
A6. Arrays with static lower bound and dynamic upper bound.
A7. Variant records fully discriminated at run time.

A1 is a general requirement on data types. A2 specifies the set of required data types. The remaining requirements specify in greater detail the characteristics of required data types. Requirement A7 is intended as a substitute for the union data types.

2.2 Operations

The TINMAN requirements on operations may be summarized as follows:

B1. Assignment and reference operations for data types
B2. Equivalence operator for all data types
B3. Relational operations for numeric and enumeration types
B4. Arithmetic operations +, -, *, /, ÷ , ↑ , unary minus
B5. Truncation and rounding of least significant digits
B6. Boolean operators and, or, not, xor, short circuit mode
B7. Direct assignment for comformable composite data types
B8. No implicit type conversion
B9. No conversion required for numeric ranges, range checking optional
B10. I/O operations for files, channels, terminals
B11. Power set operations (logical operations on Boolean vectors).

B1 and B2 specify operations applicable to all data types. B8 specifies a restriction on conversion between types. The remaining requirements indicate specific types required by the language and properties of some of these types.

2.3 Expressions and Parameters

The TINMAN requirements on expressions and parameters may be summarized as follows:

C1. Side effects evaluated left to right
C2. Readable expressions with few levels of operator precedence
C3. Expressions permitted whenever constants and variables allowed
C4. Constant expressions evaluated before run time
C5. Consistent rules for parameters of procedures, arrays, declarations, etc.
C6. Type agreement of formal and actual parameters
C7. Classes of formal parameters
C8. Optional parameter attributes in procedure declaration
C9. Procedures with variable number of parameters

C1 - C4 are concerned with properties of expressions.

C5 - C9 are concerned with parameters of procedures and arrays.

2.4 Variables, Literals, and Constants

The TINMAN requirements on variables, literals and parameters may be summarized as follows:

D1. Identifiers with constant values may be defined
D2. Constants will have some value in programs and data
D3. Declared variables may be initialized. No default initial values
D4. Range and step size for fixed point variables must be specified
D5. Arrays and records may have components of any type
D6. Pointer variables must be as safe as other variables

D1 and D2 specify properties of literals and constants.

D3 - D6 specify certain properties of variables.

2.5 Definition Facilities

The TINMAN requirements on definition facilities may be summarized as follows:

E1. Users will be able to define new data types
E2. Defined types will behave like built-in types
E3. There will be no default declarations
E4. Operations will be extendable to new data types
E5. Type definitions do not automatically inherit operations
E6. New types may be defined by enumeration, Cartesian product, discriminated union, power set
E7. Type definition by free union and subsetting is not desired
E8. Type initialization and finalization procedures are definable

2.6 Scopes and Libraries

F1. Distinction between scope of allocation and scope of access
F2. Access to identifiers can be limited both at their point of definition and point of call
F3. Scope of identifiers will be determined at compile time
F4. Libraries will be supported and easily accessible
F5. Libraries will not exclude routines written in other languages
F6. Libraries and compools will be indistinguishable
F7. Standard library definitions for machine dependent interfaces

F1 - F3 are concerned with scopes and rules for accessing identifiers while F4 - F7 are concerned with the interface between the language and libraries.

2.7 Control Structures

The TINMAN requirements on control structures may be summarized as follows:

G1. Structured control mechanisms, parallel processing, exception and interrupt handling
G2. Go-to only within most local access scope
G3. Fully partitioned if-then-else, case statement, Zahn's device
G4. Iterative control with local control variable
G5. Recursive and non-recursive routines
G6. Parallel processes, synchronization, critical regions
G7. User defined parameterized exception handling
G8. Real and simulated time, relative priorities, synchronization

G1 lists the desired control mechanisms. G2 - G5 indicate desired conventional control structures. G6, G7, G8 respectively indicate requirements for parallel processing, exception handling and real-time.

2.8 Syntax and Comment Conventions

H1. Free format, statement delimiter, easily parsed
H2. No modification of source language syntax
H3. Language definable in 64-character ASCII set
H4. Formation rules for identifiers and literals
H5. No continuation of lexical units across lines
H6. Keywords will be few, reserved, informative, not confusable with identifiers
H7. Uniform readable comment convention
H8. No unmatched parenthesis are permitted
H9. No language imposed distinction between function calls and data selection
H10. Symbols in same context cannot have different meaning

2.9 Defaults, Conditional Compilation, Language Restriction

I1. No undefined defaults which affect result of computation
I2. Defaults which optimize implementation of language features are encouraged

I3. Compile time variables which specify object computer environment
I4. Conditional compilation
I5. Simple base language which allows efficient definition of complete language
I6. Translator restrictions should be part of language definition
I7. Object machine restrictions should not be part of language definitions

2.10 Efficient Object Representation

J1. No run-time costs for unused generality
J2. Language design should allow safe optimizations
J3. Encapsulated access to hardware facilities and machine code
J4. Object representation of data structures can be specified
J5. Programmer can specify routine calls to be open or closed

2.11 Program Environment

K1. Language will not require an operating system
K2. Language will support integration of independent modules
K3. Linkers, loaders, debuggers, and other systems software available
K4. Documentation, editing, testing and other support software available
K5. Optional assertions, debugging specs, measurement probes

2.12 Translators

L1. No supersets. Features not permitted are forbidden
L2. No subset implementations will be allowed
L3. User control of optimization and compile time costs
L4. Translators for a variety of object machines
L5. Translator is not required to run on object machine
L6. Syntax checking but not error correction by translator
L7. Error diagnostics specified as part of language definition
L8. Internal translator structure not part of language standard
L9. Translators will be written in the source language

2.13 Language Definition Standards and Control

M1. Individual features must be state of the art
M2. Unambiguous and clear language definition
M3. Tutorial and reference documents, defined by abstract comput-
M4. Configurations control to ensure translators conform to standard
M5. Support agent responsible for maintaining language and support software
M6. Standards and support agents for libraries

3. Language Evaluations

3.0 Summary of Languages and Evaluations

Of the 23 languages considered, three were recommended as base languages, eleven were recommended as relevant to the design effort or deserving of further consideration as potential base languages and nine were regarded as not acceptable to serve as base languages. The 23 languages are listed below.

A: Recommended Languages (These languages each represent a different synthesis of large amount of previous experience, and constitute the nucleus of a family of derived languages).

1. PL/I: Includes concepts from FORTRAN, ALGOL 60 and COBOL
2. PASCAL: A successor of ALGOL 60 emphasizing simplicity
3. ALGOL 68: A successor of ALGOL 60 emphasizing generality

B: Languages which are Relevant and Deserving of Further Consideration

4. HAL/S: PL/I based, NASA Language, strongly typed, real-time
5. PEARL: PL/I-based, German process control language
6. SPL/I: PASCAL-based NRL real-time signal processing language
7. PDL/2: PASCAL with parallel processing, independent module facilities, Texas Instr.
8. LTR: PASCAL-based official French common language
9. CS-4: PASCAL-based real-time with extension facilities, Intermetrics
10. LIS: PASCAL-based French system implementation language
11. EUCLID: PASCAL-based experimental language emphasizing verification

12. ECL: Extensible language with good support environment, Harvard University

13. MORAL: New British language for embedded computer applications

14. RTL/2: Real-time British language developed at ICI

C: Languages Not Acceptable

15. FORTRAN: Developed by IBM in 1954-58

16. COBOL: Business data processing language developed in 1959-61

17. ALGOL 60: Block structure language developed in 1957-60

18. TACPOL: Army language developed in the late 1960's

19. CMS-2: Navy language developed in 1966-69

20. SIMULA 67: Simulation language developed in Norway

21. JOVIAL J3B: Air Force language developed in 1972

22. JOVIAL J73: Air Force language developed in 1969-73

23. CORAL 66: British common language for real-time applications

3.1 PL/I Evaluation:

Evaluators

PL/I was evaluated by IBM and Intermetrics.

Language Description

PL/I was developed in the period 1962-66 by an IBM-sponsored design group. It includes concepts and features from FORTRAN, ALGOL 60 and COBOL, taking over its expression syntax from FORTRAN, its statement syntax, block structure and type declarations from ALGOL 60 and its data description facilities from COBOL. It is rich in data types, data attributes, control structures and input-output features. However, it is not strongly typed and lacks extensibility, parallel processing, synchronization and real-time features.

Overall Evaluation

IBM strongly recommended PL/I as a potential base language, and presented a fairly detailed discussion of how the language could be modified to meet what they interpret as essentially the TINMAN requirements. Intermetrics felt that the language should be rejected because it lacks too many of the DoD requirements. It was designed for a different purpose (expressive power rather than readability, reliability and modifiability) and it was developed before many of the modern language features required by TINMAN were properly understood.

PL/I is the most controversial of the languages considered. However, since the use of PL/I as a base language is recommended by an important and credible group of designers, we believe that this group should have an opportunity to further explore its approach to the development of a common DoD language. Moreover, there is a great deal of experience in the development of subsets and derivatives of PL/I and some of these derivatives such as HAL/S and PEARL may well be suitable design experience.

Positive Features

PL/I is more widely used by an order of magnitude than any of the other proposed base languages.

There is a great deal of experience in subsetting and modification of PL/I.

A great deal of care and effort has been lavished on the language definition. The defining document BASIS 1 has recently been adopted as an American PL/I standard by ANSI. The availability of this standard may make it easier to define the modified language. The work on language definition has resulted in a better understanding of the language which will be helpful in language modification efforts. The tools and support software for PL/I may be useful in developing tools and support software for a modified language.

IBM has already considered in some detail how PL/I should be modified to meet the requirements of TINMAN.

PL/I has an advantage over PASCAL and ALGOL 68 in the area of external procedures and common data.

Negative Features

PL/I was designed with different objectives from those stated in TINMAN. In particular, PL/I was designed to promote power of expression and ease of writing programs, while the DoD requirements emphasize readability, modifiability, maintainability, security, and efficiency. Achievement of the latter kinds of objectives would require large changes in the language specification.

PL/I was developed too early to incorporate advances of design technology of the mid and late 1960's in the areas of data types, control structures, extensibility, modularity, programming methodology, etc. Such developments have not been reflected in the recently adopted standard.

The PL/I language design suffers from a lack of design integrity since it has not been entirely successful in integrating the wide variety of language concepts and features from FORTRAN, ALGOL 60, COBOL, and other sources into a single homogeneous language.

PL/I provides too much freedom of expression for the user, attempting to assign meanings to programs in cases which should be treated as errors. The current emphasis is to introduce restrictions that enforce good programming methodology.

PL/I is a complex language for which the consequences of addition and deletion cannot be easily foreseen.

There is some concern that it would be difficult to evaluate a language design starting from PL/I as a base because the complexity of the base language and the magnitude of the language changes would make it very difficult for a design evaluation team to evaluate the proposed new language.

3.2 PASCAL Evaluation:

Evaluators

PASCAL was evaluated by Softech and RLG.

Language Description

PASCAL was developed in the late 1960's by Niklaus Wirth as a successor to ALGOL 60 emphasizing simplicity (as opposed to the generality of ALGOL 68). It provides richer declarative facilities than ALGOL 60, including records (structures), files, pointers, scalar types (enumeration types), sets and programmer-defined types, but is otherwise as simple as possible. For example, it excludes parameters called by name, own variables and arrays with dynamic bounds. It has strong (compile time checkable) typing and control structures designed to encourage good programming style.

Overall Evaluation

Softech recommends PASCAL because relatively few features need to be deleted, and because those features which it does have are well-designed and very much in the spirit of the DoD requirements. RLG also feels that PASCAL is suitable as a base language.

Positive Features

PASCAL is a strongly typed well designed language with extensibility and control structure facilities very much in the spirit of the DoD requirements.

It is very popular as a starting point for designing new languages. It has served as a starting point for larger general purpose languages, such as CS-4 and LIS, for signal processing languages, such as SPL/I, for real-time languages, such as PDL/2, and LTR, and for experimental languages for concurrent programming (concurrent PASCAL) and program verification (EUCLID).

It has a formal axiomatic definition which has been widely used as a basis for program verification.

It is a simple language which can be easily understood both by programming language designers and by applications programmers.

Negative Features

The experience with PASCAL is largely in a research environment. It has not been widely used for large production programs. However, extensions of PASCAL for use in embedded computer environments have been developed in a number of places (see PDL/2).

Because the language is small, a lot of features will have to be added, such as parallel processing, real-time, error handling, and precision facilities. In adding these features there ary many opportunities for errors and loss of design integrity.

It does not permit independent program and data modules.

There are some slight problems with existing language features such as the fact that dimensions of array parameters must currently be bound at procedure declaration time. These problems can easily be fixed up.

3.3 ALGOL 68 Evaluation

Evaluators

ALGOL 68 was evaluated by Softech and SAI

Language Description

ALGOL 68 was developed during the period 1963-69 by the IFIP working group 2.1 (WG2.1) as the "official" successor to ALGOL 60. It has the block structure of ALGOL 60 with a much richer set of data types (called modes in ALGOL 68). It has an infinite number of modes obtainable from eight basic modes by mode constructors such as ref for specifying pointers, proc for specifying procedures, struct for specifying record structures, and long, short for specifying numerical precision. There are complex but consistent rules for implicit mode conversion (coercion). There is a fairly rich set of input-output (transput) operations, a well designed set of facilities for parallel processing, but no real-time facilities. The language has a "standard prelude" which defines convenient language extensions, a "library prelude" for defining library facilities and a "system prelude" for defining the operating system environment. The language definition uses an expressive but difficult-to-master syntactic mechanism called W-grammars to express program structure.

Overall Evaluation

ALGOL 68 is recommended as a potential base language by both Softech and SAI because its design integrity, consistency and clean definition make it more appropriate as a starting point for language design than a base language designed in an ad hoc manner. It is the language most consistently recommended by its evaluators among all the candidate base languages. However, we talked to many people who were skeptical of using ALGOL 68 as a base language because it is not well understood or widely used in the U.S.A. and because there are apparently difficulties in implementing the language.

Positive Features

ALGOL 68 is designed in an exceptionally clean manner. Because language features are "orthogonal" (independent), it is easy to determine the consequences of adding and deleting features.

ALGOL 68 is a large language which already contains many of the more difficult required features. Features not in ALGOL 68 would have to be added to many of the other candidate languages too, but can be added more cleanly in ALGOL 68.

SAI states that one immediately sees how to add abstract data types, sets, fixed point numbers, or other data types, while reducing the generality of the go-to, eliminating flex arrays and certain operations, etc.

The language definition reflects the cleanness of the language design and is helpful in determining the effect of language modifications.

The control structures of ALGOL 68 encourage writing of clean programs.

The strong typing aids in program reliability.

The reference concept with explicit local generators and heap generators appears to be well suited to the dynamic storage allocation philosophy of the DoD language, but will require some additional language-level specification to explicitly distinguish all stack and heap storage.

Negative Features

ALGOL 68 appears to be difficult to implement. Apart from the British implementation at the Royal Radar Establishment (RRE), there does not seem to be a widely used implementation of the language. It is not clear whether the implementation problems are intrinsic or simply due to a lack of resources.

ALGOL 68 is less widely used than competitor languages like PASCAL. It is not clear whether the language is intrincically too "difficult" for applications programmers or whether the lack of user enthusiasm is due to a lack of implementaticns and poor expository manuals.

The current language definition is difficult for the uninitiated but is said to be an advantage rather than a hindrance to its use as a base language once the forbidding terminology has been mastered.

The complex coercion rules make it difficult for the programmer to determine the effect of certain program constructs and will have to be replaced by explicit mode conversion functions in order to meet the TINMAN requirements. It seems a pity to throw away this language capability, since it is not clear that mode conversion in a cleanly designed, strongly typed language leads to unreliability.

3.4 HAL/S Evaluation

Evaluators

HAL/S has been evaluated by IBM.

Language Description

HAL/S is a PL/I-style language developed by Intermetrics for NASA in the early 1970's and designed especially for use on airborne computers and other embedded systems in connection with the space shuttle program. Its data types include arrays, structures and pointers. Non-recursive procedures may be declared but not passed as procedure parameters. It requires strong type specifications for both declared identifiers and procedure parameters, has a wide range of control structures for structured programming, multiprogramming and error handling, and provides independently compilable program and data modules which facilitate the construction on large programs. It has a real-time processing facility with a flexible facility for scheduled processes in a process queue. Execution and storage allocation for scheduled processes is left to the operating system. The language is carefully designed so that storage requirements for all programs and tasks can be determined at process initiation time. The operating system must decide when initiating a process whether there is enough free memory for that process to operate.

Overall Evaluation

IBM has recommended HAL/S as a base language. HAL/S does not provide recursive procedures, multi-dimensional arrays with dynamic bounds and other forms of dynamic storage allocation in order to achieve efficient runtime code while providing multiprogramming, error handling and real-time facilities.

Positive Features

There is experience in using HAL/S for embedded computer applications.

The language is reasonably clean in spite of the fact that it contains pointers, multiprogramming, error handling, real-time control, independent modules and other difficult to handle features.

The language has proved sufficiently successful to be under active consideration for a number of new military and civil embedded computer applications. It is being considered by the Air Force as the on-board language for the interim upper-stage (IVS), by Collins Radio as the language for the AP-101 on-board guidance and navigation computer and by Boeing as a standard language for civil avionics applications. The language manuals and language specification are clear and adequate.

Negative Features

Substantial modifications to HAL/S would be required in the area of language extensibility. There are no user defined types nor any kind of encapsulation mechanism. It has no recursive procedures or other dynamic storage allocation features. Finally, a number of features of HAL/S used to solve parallel processing and real-time problems may be in need of updating. They were designed in the late 1960's and should be re-examined in the light of more recent research.

3.5 PEARL Evaluation

Evaluators

PEARL was evaluated by CSC and was also evaluated by GMD in Bonn, Germany. It was also evaluated by John Williams.

Language Description

The Process and Experiment Automation Real-Time Language (PEARL) is a PL/I style language which was developed in Germany in the period 1968-73 by a working group of experts from industry and universities. Implementations are available or are underway on eight different process control computers. PEARL is a procedure oriented language designed for reliability, efficiency, portability and machine independence in real-time and process control applications. PEARL has standard data types as well as clock and duration data types, but no mode definition facilities. Procedures are not regarded as data types and must be declared reentrant if they are to be interruptable and executable by several tasks simultaneously. A PEARL program can consist of a series of separately compilable modules each having a system division with information about the system configuration, device interconnection, process signals and terminals and a problem description with module, task and procedure declarations. Tasks may have four states referred to as running, runnable, suspended or dormant, may be synchronized by semaphore variables, may be scheduled using clock and duration data types and may be interrupted by programmed or real-time interrupts.

Overall Evaluation

PEARL is comparable to HAL/S in being a PL/I-based procedure oriented language with parallel processing and real-time facilities. Its portability and machine independence is impressive and it certainly deserves consideration.

Positive Features

It contains well-developed real-time process control and parallel processing facilities.

There has been some experience both in language implementation and in using the language for process control applications.

The language-system interface is specified in a system division and appears to have been well worked out.

Negative Features

PEARL, like PL/I, appears to lack a clear concept of data type. It is not strongly typed and lacks data-type definition facilities and extensibility features.

PEARL does not have pointers, recursive procedures, a fixed point data type or adequate exception handling features.

It lacks a well defined interface to routines compiled in assembly language or other higher level languages.

PEARL is still an experimental language. It has not been used as an operational language for any really large embedded computer applications and has not been adopted as a standard language by government or industry.

3.6 SPL/I Evaluation

Evaluators

SPL/I was evaluated by CSC. The discussion below is based in part on discussions with Richard Harrington of NRL.

Language Description

SPL/I was developed in the period 1972-76 by the Naval Research Laboratory (NRL) in cooperation with Intermetrics. It was developed originally as a Signal Processing Language (SPL) for processing real-time analog inputs and performing fast Fourier transforms and other digital filtering operations for analyzing and processing these inputs. However, the language is general-purpose with block structure, strong typing, standard primitive types, arrays, structures, pointers, procedures with input, output and input-output parameters, no implicit mode conversions, etc. The language has arrays with dynamic bounds, recursive procedures and permits truly dynamic storage allocation but allows optimization when dynamic storage allocation is not required. Procedures can be separately compiled and can contain global data. Multiprocessing with synchronization can be performed using process, signal and resource data types. Real-time applications can be handled using a machine-independent clock facility. An error-handling escape mechanism is available.

SPL/I was initially used for underwater signal processing with the PROTEUS computer, but it is now available on the Navy AN/UYK-20 and AYK-14 computers, and is being considered for non-Navy embedded computer applications.

Overall Evaluation

SPL/I is comparable to HAL/S. It was developed slightly later than HAL/S. It has the advantage of being able to make use of the design experience of HAL/S and the disadvantage that there is less operational experience with SPL/I than with HAL/S. It is more ambitious than HAL/S in handling parallel processing and real-time applications with dynamic rather than static allocation within individual processes, and is in this respect closer to the spirit of the DoD requirements than HAL/S.

Positive Features

SPL/I substantially satisfies the parallel processing and real-time requirements. Moreover, it appears to satisfy the requirements of simplicity, reliability, maintainability and machine independence. Thus, it must certainly be considered by any design team modifying any language to meet the requirements.

Experiments with PROTEUS indicate that object code produced by the Intermetrics SPL/I compiler is only 10% slower than good assembly language code for the same program.

In the opinion of CSC, SPL/I is a cleaner, more readable and simpler language than CS-4.

Negative Features

SPL/I does not include a type definition facility, enumeration types, precision specifications, and other important DoD language requirements. The control structures of SPL/I, including both sequential and parallel, are not the best ones possible and their designs should be re-examined.

3.7 PDL/2 Language Evaluation

Evaluators

PDL/2 was evaluated by Texas Instruments Inc.

Language Description

The Process Design Language PDL/2 was developed by Texas Instruments Inc. in the mid 1970's for the Ballistic Missile Defense Advanced Technology Center (BMD) to meet the needs of BMD real-time software design. It is a PASCAL-based language which extends PASCAL by adding tasking and synchronization primitives, variable length arrays, vector and array operations, assertion statements, independent compilation and common variables, code generation, macro substitution facilities. A compiler for PDL/2 has been implemented on the BMD advanced scientific computer (ASC).

Overall Evaluation

PDL/2 is a direct extension of PASCAL in the direction of embedded computer applications, and should be carefully studied as an example of an attempt to do precisely the kind of design by language modification recommended in this report. PDL/2 should be carefully looked at by a design team modifying PASCAL to meet the requirements.

Positive Features

PDL/2 represents a language modification effort which goes a substantial way towards converting PASCAL into a language meeting the DoD requirements.

Negative Features

Substantial modifications in the areas of precision, exception handling, and other areas are needed to make the language conform with the DoD requirements.

The specific mechanisms used for tasking, synchronization, global data specification, etc. should be carefully analyzed to determine whether they can be significantly improved.

3.8 LTR Language Evaluation

Evaluators

LTR was evaluated by Alan Demers at Cornell. Conversations with Pierre Parayre of the French Naval Programming Center were helpful in developing this evaluation.

Language Description

LTR was designed in France in 1968 for the realization of multitask real-time systems. It is block structured, strongly typed, and has a rich and adequate set of data types and control structures. In addition, it includes the concepts of task, event, resource, system time, delay, interrupt real-time I/O, and primitives for handling these entities. LTR has proved to be well suited for modular programming and structures programming and is almost entirely portable between 16 and 32 bit computers. Since 1973 LTR has been used for programming a number of complex operational systems, and assembly language has been almost completely circumvented.

LTR is the official French common language for embedded computer applications in the Navy, Army and Air Force. It has also been adopted as the common language for civilian air traffic control. It has been used as the language for developing a Naval Technical Data System (NTDS) similar to that for which CMS-2 is being used in the US, and for a battlefield tactical system called ATTILA (similar to TOS).

Overall Evaluation

LTR has been widely used by the French Department of Defense as a language for embedded computer applications, and clearly satisfies many of the DoD requirements for a common language. It is, therefore, very relevant to the common language design effort. According to Parayre, there is an able French language design group at CII who would be eager and willing to develop and submit a preliminary design for the common DoD language. Such a design would be based on great experience with HOLs for embedded computer application.

Positive Features

The language is strongly typed and contains modern notions of control structure and modularity.

LTR appears to be a highly efficient language. LTR has well designed real-time features. The ability to create and destroy processes, process communication and scheduling, interrupt handling, and direct I/O are all present with about as high a level of machine independence as one could expect.

LTR is the only example of an apparently successful common HOL for embedded computer applications. The existence of such a language should increase our confidence in the feasibility of this project. At the same time, we should try to learn from the French experience, either by soliciting preliminary design based on LTR or by involving a member of the French group in some other way to provide inputs concerning technical and managerial experience in introducing a common language.

Negative Features

The equivalence mechanism, quality variables and the free type option for pointers allow strong type checking to be defeated.

The limitation to one-level structures.

The implementation of FOR loops with the test at the bottom.

The language is oriented to 8-byte machines.

The real-time features require a special LTR monitor so that the language is operating system dependent.

3.9 CS-4 Evaluation

Evaluators

CS-4 was evaluated by Intermetrics, CSC, RLG and SAI.

Language Description

CS-4 was designed by Intermetrics for the Navy in the mid 1970's and is not yet implemented because work on its implementation was stopped as part of the "freeze" on DoD new programming language development pending outcome of the HOLWG study. The design of CS-4 was strongly influenced by PASCAL, HAL/s and modern programming language concepts. The language has strong-type checking with few implicit conversions, powerful data structuring tools with full safety, ability to define new abstract types (although not new infix operators), support for parallel processing and exception handling, user control over precision and range of variable values, and adequate control structures. There are adequate separate compilation facilities and good facilities for escape into machine language. CS-4 has no pointers, recursive procedures, machine independent interfaces to hardware components or generic procedures. Although the language is fairly well designed, there are, in the opinion of CSC, clumsy design features in the area of parallel processing and in the wordiness of declarations. The language is large and certain features would have to be deleted.

Overall Evaluation

Intermetrics is very positive concerning the suitability of CS-4 as a base language while RLG and CSC are mildly positive but not enthusiastic. Intermetrics feels that CS-4 is particularly strong in the areas of reliability, maintainability, transportability and language power. CSC and RLG are concerned with the complexity and lack of implementation of CS-4. Since CS-4 represents a large and skillful design effort with similar goals to those of the current DoD effort further refinement of this design to meet the specific DoD requirements should certainly be considered. However, CS-4 is not as well defined a starting point for a new design as an already implemented language, and it may well be that designers might wish to design a CS-4 like language while using some other language as a base language.

Positive Features

The language has made extensive use of modern language design principles and includes many of the language features specified in the DoD language requirements.

The language has been specially designed to satisfy current notions of reliability, maintainability and transportability. However, this claim cannot be convincingly verified, since there is no implementation and no user experience for the language.

Strong typing, extensibility, parallel processing, precision specification and separate compilation are incorporated in the language design. It is only one of the languages considered that meets the requirements in all five of the above areas.

Negative Features

The language is not implemented.

Although the language description is generally clear, there is no really precise language definition.

The language is so large that a clean definition or an implementation is necessary to determine whether claims made for the language are in fact true.

The language lacks pointers and recursive procedures. These features could easily be introduced into the language but would considerably complicate the implementation.

3.10 LIS Evaluation

Evaluators

LIS has been evaluated by SAI and Alan Demers at Cornell.

Language Description

LIS was designed by Jean Ichbiah and the French company CII/GAM as a system implementation language. It is strongly typed, has a rich set of data types and control structures, has good facilities for independently compiled program and data modules, and specifies machine dependent features in separately defined modules called implementation parts. It has coroutines but no parallel processing, exception handling or real-time facilities.

Overall Evaluation

LIS has most of the features of a modern block structure language and is designed to meet criteria of efficiency, reliability and modularity. The experience and ingenuity of the LIS design group is considerable, and it is quite possible that a good language meeting the DoD requirements could be designed starting from LIS as a base. Since the French approach is inevitably likely to be different from that of U.S. design groups, it might be instructive comparing the results of a French design effort with the results of design efforts by U.S. Software houses.

Positive Features

LIS has been designed according to modern notions of efficiency, reliability and modularity.

The goals of system programming languages, such as LIS, overlap to a considerable degree with the goals of languages for embedded computer applications.

The notions of independently compiled program and data modules and of implementation parts are better worked out than in comparable other PASCAL-oriented languages.

It is one of the best designed extensions to PASCAL that is available.

Negative Features

LIS lacks precision specification, fixed point data type, operator extension, generic procedures, parallel processing, real-time facilities and exception handling facilities.

3.11 EUCLID Evaluation

Evaluators

EUCLID has been evaluated by CSC.

Language Description

EUCLID is a PASCAL-based language designed specifically to facilitate program verification. Restrictions are placed on certain language constructs which allow the compiler and verifier to make stronger inferences about language properties, thereby increasing reliability, efficiency and verifiability. These restrictions include explicit control over imported and exported identifiers in procedures and modules, and guarantees that two identifiers in the same scope (including procedure parameters) can never refer to the same or overlapping variables. Extensions of PASCAL include parametrized types, modules which can contain procedures and data components, and assertion specifications. Deletions from PASCAL include input-output, reals, multidimensional arrays, labels, gotos, and procedures as parameters.

Overall Evaluation

EUCLID is a "small" experimental language which meets only very few of the DoD requirements. However, language designers starting from a language in the PASCAL family should seriously consider adopting some of the restrictions and extensions of EUCLID in order to enhance the reliability, efficiency and verifiability of the new language. Although enlargement of EUCLID to meet the DoD requirements would introduce considerable complexity, CSC feels that this approach to meeting the requirements, might if done with great care, result in an acceptable common language.

Positive Features

EUCLID's approach of enhancing reliability by placing restrictions on tricky uses of procedure parameters, identifiers and other language features may well become standard in the design of future languages.

EUCLID appears to be a well-designed language and its approach to the inclusion of parametrized types, modules and assertions deserves to be studies.

EUCLID has a clear language description. This approach to language description, which was pioneered by PASCAL, should be considered as a possible norm for language descriptions produced by language design teams for the common language.

Negative Features

EUCLID is a small experimental language which satisfies only a small number of the DoD requirements.

EUCLID is not intended for the programming of large embedded computer applications. It has no real-time parallel processing or numerical precision facilities.

3.12 ECL Evaluation

Evaluators

A partial evaluation of ECL was received from Stephen Squires of NSA/CSS.

Language Description

ECL was developed by Ben Wegbreit and first described in his doctoral dissertation in June 1970. It consists of a programming language called EL/1 and a system built around this language which allows on-line interactive construction, testing, symbolic debugging and running of programs. It allows extensibility to permit flexibility during program development as well as contractability to permit optimization for production purposes. It is a block structured language with a rich set of data types (called modes) and mode definition facilities. According to the evaluator, almost all language deficiencies, such as precision specification and fixed point variables, can be added to the language by extensions. Parallel processing and exception handling requirements are said to be satisfied by the multipath control and TRAP facilities (these facilities do not appear to be in the December 1974 language manual). The language manual describes an impressive set of supportive software including compile time facilities, built in library routines and support packages for debugging, tracing, testing, metering, etc.

Overall Evaluation

ECL is probably too rich and flexible a language to be adopted as a realistic base for the common language.

However, it has something to teach us in the areas of extensibility and language system interface facilities, and should therefore be studied by language designers using some other language as a base. EL/1 is a large well-designed language, and the possibility of starting from EL/1 and developing a language meeting the DoD requirements by language extension and contraction should not be altogether ruled out.

Positive Features

ECL is a large well designed language which satisfies many of the difficult DoD requirements and can be made to satisfy many of the remaining ones by language extension.

Because the language is well designed it might be possible to develop the required language by placing restrictions on a suitably extended language.

Negative Features

ECL was designed for an experimental interactive programming environment rather than for production programming in embedded computer applications.

ECL is too rich in extension facilities, pattern matching, control structures, non-deterministic algorithms and a number of other areas.

3.13 MORAL Evaluation

Evaluators

MORAL was conducted by Software Sciences Limited (SSL).

Language Description

MORAL was developed in 1975-76 by the Royal Radar Establishment (RRE). Its development is based on "MASCOT," an approach to the design of real-time systems. In the interests of portability it is constrained by the adoption of CORAL 66 as a target language for the implementation of MORAL. MORAL is a block structured strongly typed language with integer, fixed point, floating point, array and record (group) data types. It lacks an explicit boolean data type and precision specifications but has enumeration types and mode definition facilities. The language supports synchronization and real-time facilities but leaves the task of creating new processes to the operating systems. Exception handling is not explicitly supported but its effect can be accomplished by an interrupt mechanism.

Overall Evaluation

MORAL is a clean language designed specifically for real-time embedded computer applications. It handles real-time requirements differently from that envisaged by the TINMAN specification, by placing more responsibility on the operating system. It meets a surprisingly large number of the requirements and should be seriously considered by design teams as a relevant language.

Positive Features

MORAL was designed for the same purpose as the common DoD language.

It meets a surprisingly large number of the requirements.

It is one of the most recently designed languages considered and has made good use of ideas in PASCAL, ALGOL 68, CORAL 66 and other languages.

Negative Features

A prototype translator for the language was completed in September 1976, but the language has not yet been fully tested.

The real-time features of the language are dependent on its operating system.

The language design was constrained by requiring CORAL 66 to be the target language of the associated translator.

3.14 RTL/2 Evaluation

Evaluators

RTL/2 was evaluated by Imperial Chemical Industries (ICI) and by Alan Demers at Cornell.

Language Description

RTL/2 is an ALGOL-based language developed in Britain by ICI (Imperial Chemical Industries). It is strongly typed with standard data types and control structures. A program consists of a number of independently compiled modules each consisting of an environment description and compilable units called bricks. These are procedure bricks which consist of procedure bodies, data bricks which are like compools and stack bricks which can be created during execution to provide work spaces for tasks. Standard I/O facilities and an error recovery system are not part of the language but are supplied as support tools to increase portability. It has recursive procedures but no nesting of procedure declarations. It has no dynamic arrays. It was designed with the same overall objectives as the proposed DoD common language but reflects the state of the art in 1969.

Overall Evaluation

RTL/2 appears to be a soundly constructed, relevant language. Its notion of bricks provide a mechanism for scopes and environments more flexible than conventional block structure and its stack bricks allow task creation and real-time facilities to be built up from lower level primitives.

Positive Features

It has strong typing, standard data types and standard control structures.

Its notion of bricks is an interesting and perhaps useful way of blending block structure and with independent modules.

It provides hooks to standard I/O procedures, error recovery procedures and real-time facilities in an operating system.

There are 150-200 real-time systems programmed in RTL/2 of which half are outside ICI. The applications include process control, factory automations, laboratory systems, communications and on-line banking. Many (including all those in ICI) employ operating systems written in RTL/2.

RTL/2 was developed as a result of a requirements evaluation similar to that of TINMAN. The RTL/2 experience of language development and language use might be useful as an input to the design effort. ICI has offered its help, possibly in association with an American Software Company.

Negative Features

It has no extensibility features, precision features for numerical variables or explicit real-time features.

3.15 FORTRAN Evaluation

Evaluators

FORTRAN was evaluated by Intermetrics and IBM.

Language Description

FORTRAN was developed in the period 1954-58, was essentially the first higher level language, and is still the most widely used language after COBOL. Its subroutine structure and COMMON data blocks was not taken up in block structure languages. However, its lack of data description facilities, control structures, pointer variables and recursive procedures illustrate that there has been some progress in language design since the development of FORTRAN. The FORTRAN evaluation was performed for the 1966 ANSI standard. A revised draft standard for FORTRAN has been prepared and is being considered by ANSI but has not yet been approved. Since it is in principle possible that substantive changes to this draft standard may still be made, it was not considered for purposes of this evaluation.

Overall Evaluation

Both contractors feel that, in spite of the fact that FORTRAN has shown remarkable resilience and longevity, it is not suitable as a base language because its design does not reflect advances in language design during the last twenty years.

Positive Features

Simplicity, efficiency, subroutine structure.

Negative Features

Lack of data types, scope rules, control structures, recursive procedures, pointers, etc.

FORTRAN reflects the language design philosophy of the 1950's and is dominated as a base language by more recently designed programming languages.

3.16 COBOL Evaluation

Evaluators

COBOL was evaluated by Intermetrics and IBM.

Language Description

COBOL was initially developed in the period 1959-61 as a language for business data processing applications, building on previous languages, such as Univac's Flowmatic and IBM's Commercial Translator. COBOL has had significant additions since its initial development. Its data description facilities and its concept of environment, data and procedure divisions are still important today but the algorithm specification part of the language is behind the state of the art. It is currently the most widely used programming language in the U.S.A. but is not intended for embedded computer applications.

Overall Evaluation

Both Intermetrics and IBM agree that COBOL is not suitable as a base language for embedded computer applications, both because it was not designed for this class of applications and because it is dominated as a base language by more recently designed languages.

Positive Features

It seems to be the most widely used higher order language.

It has good I/O and data description facilities and is adequate for specifying simple algorithms.

The distinction between environment, data and procedure divisions is worth while.

Negative Features

It was designed for business data processing rather than embedded computer applications.

Its English language notation makes programs deceptively readable but does not aid reliability or modifiability.

It has only a two level block structure, inadequate data type facilities, inadequate control structure and procedure definition facilities, and too many special cases and implementation dependencies.

The language definition is too large for its expressive power.

3.17 ALGOL 60 Evaluation

Evaluators

ALGOL 60 was evaluated by SAI.

Language Description

ALGOL 60 was developed in the period 1957-60 by the IFIP working group WG 2.1. It pioneered the notion of block structure, data type declarations, access scopes, etc., as well as the development of clean programming language definitions. It has gained wider acceptance as a production programming language in Europe than in the U.S.A. It is carefully designed to be implementable in a stack structured run-time environment. It has no explicit input-output facilities and lacks other important features such as record structures, pointer variables, extension facilities, parallel processing, independent (external) procedures, etc. It has influenced the design of almost all subsequently designed block structure languages, including PL/I, PASCAL and ALGOL 68, and is a "grandfather" of more recently designed block structure languages like CS-4, LIS, and HAL/S.

Overall Evaluation

ALGOL 60 is not directly suitable as a base language, but its influence on the design of the new language will make itself felt since all the recommended base languages are "children" or "grandchildren" of ALGOL 60.

Positive Features

ALGOL 60 is one of the most important programming languages ever developed and has influenced the design of almost all subsequently designed programming languages. Its definition still serves as a model for current programming language definitions.

Negative Features

Development in programming language methodology since the design of ALGOL 60 make this language inappropriate as a base language. PL/I, PASCAL, and ALGOL 68 have, each in a different way built upon the design philosophy of ALGOL 60 and it is suggested that these languages, or even more recently developed derivative languages be used as a base language.

3.18 TACPOL Evaluation

Evaluators

TACPOL was evaluated by Softech, Intermetrics, and RLG.

Language Description

TACPOL is a block structured language developed for the Army for a particular application (TACFIRE) on a particular computer (AN/GYK-12). It is a simple, easy to learn, block structured language with fixed point (but not floating point) numeric data, character and bit data, arrays of up to 3 dimensions, records (called groups), tables and overlay facilities. Although all identifiers require data-type, declarations-type checking can be defeated by certain built-in (intrinsic) operations and overlay operations. Procedures can have value parameters, quantity parameters (by reference but without type checking), parameterless procedures and labels. Control structures include if-then-else and iteration, while and case statements. File handling facilities are provided. Exception handling can be performed by passing label parameters (the TACPOL ON statement is deceptive in that it is really a conditional statement rather than a true interrupt). Parallel treatment of input-output and computation may be specified in special cases but there are no general parallel processing facilities. The above description illustrates that TACPOL has a large number of special language features which were included for reasons of efficiency because the inclusion of corresponding cleanly designed general purpose features was not properly understood. The language may be well suited to the particular application for which it was designed but suffers from restricted expressive power and defects in reliability, portability, and maintainability.

Overall Evaluation

All three language evaluators feel that TACPOL is not suitable as a base language. Its features are a rather small subset of the acceptable features of other recommended languages such as PASCAL (SofTech).

Positive Features

TACPOL is simple and easy to learn.

Negative Features

TACPOL lacks too many of the requirements.

It lacks enumeration types and data definition facilities. There is no floating point data type.

Shortcomings exist in the definition of iterative and conditional control structures, including conditional evaluation and optional ELSE.

Recursion is not well defined. Pointers are not supported. Arrays cannot have dynamic bounds.

Parallel processing is not supported, nor do any time dependent features exist.

Exception handling is inadequate.

There is no facility for describing the object environment or providing an external interface to the object machine.

Certain symbols such as the = symbol are syntactically overloaded.

A considerable amount of defaulting is permitted.

There is inadequate checking for parameters passed by reference. There is no discriminated union facility.

3.19 CMS-2 Evaluation

Evaluators

CMS-2 wes evaluated by RLG and CSC.

Language Description

CMS-2 is a Navy language developed by the Navy in the period 1966-69. It developed from an earlier language CS-1, and has gone through many updates, each requiring upward compatibility with a previous language. CMS-2 now contains most of the structured programming features common to modern HOLs such as the case statement, WHILE and UNTIL iteration, block structuring, etc. and features considered necessary for embedded military systems, such as machine code insertion, compools, direct procedure linkage, direct access to hardware and machine functions. Some of these features make the language very machine dependent and there are currently several different versions for different computers. The evaluated language is CMS-2-Y for the AN/UYK-7.

Overall Evaluation

Both RLG and CSC feel that the language is not acceptable as a base language because it lacks too many of the required features, including scope rules, precision specification, pointers, recursion, extensibility, real time facilities, I/O facilities, machine language interface, etc. Although the language appears to be successful in its present applications, it is not portable to new embedded computer applications on new computers.

Positive Features

The language has been successfully used in embedded computer applications and meets, those TINMAN requirements that are met by other languages currently in use for military systems.

Although the language is large, it is not unwieldly or difficult to learn. However, its design may make it larger than necessary for the expressive power it provides.

Negative Features

Many important required features are missing such as scope rules, precision specification, pointers, recursion, extension facilities, parallel processing.

Lack of high level constructs leads to excessive use of machine language. It is estimated that NTDS is coded 30% in machine language.

The source language is defined in a machine dependent manner and therefore even programs written entirely in the HOL part of CMS-2 will not be portable. For example, array assignment, structure accessing, and the shift operator is machine dependent.

Procedure parameters are not local to procedure definition. Procedure linkage mechanism is not conducive to modularity.

Certain language constructs have unexpected and unnecessary constraints. For example, FIND is restricted to one dimentional arrays. Arrays can contain records but records cannot contain arrays.

The scaling and truncation rules of CMS-2 appear to be unnecessarily complex and can cause undesirable errors in numerical computation.

The CMS-2 language has special purpose features which increase translator complexity and may prevent the translators from producing efficient object code.

CMS-2 provides excessive generality at a number of places where it is not needed while imposing unnecessary restrictions at a number of places where generality is appropriate.

CMS-2 has inconsistencies and ambiguities in its definition. Inconsistencies include scaling rules for constants, interpretation of algebraic notation in MEANS and EQUALS statement, value of control variables on loop terminations. Ambiguities include order of expression evaluation.

The definition of CMS-2 is more complex than necessary. This is particularly evident in its discussion of its error prone scaling rules.

CMS-2 does not meet the requirements of simplicity, efficiency, reliability, maintainability or portability. It is an error prone excessively machine dependent language.

3.20 SIMULA 67 Evaluation

Evaluators

SIMULA 67 was evaluated by SofTech.

Language Description

SIMULA 67 is an extension of ALGOL 60 which contains an important new kind of program module called a class. A SIMULA class just like an ALGOL block consists of a set of declarations followed by a sequence of statements. However, SIMULA separates creation of class data structures from execution of the class body, thereby allowing coroutine control structures and the simulation of objects which have independent data and procedural attributes. The standard SIMULA extension for simulation purposes has interesting simulated time (but no real-time) facilities.

Overall Evaluation

SIMULA has played an important role in the evolution of concepts of modularity but its module structure does not conform to DoD requirements. It is not recommended as a base language since removal of the class feature from SIMULA would result in a weak ALGOL 60-like language which would be a poor starting point for the design of the common language.

Positive Features

SIMULA has played an extremely important role in the development of concepts of modularity.

The SIMULA subclass facility is an important and powerful device for providing extensibility in programming languages.

Negative Features

The requirements of simulation languages appear to differ and in some respects be diametrically opposed to those of real-time languages, since simulation is specifically concerned with modelling behavior by indirect, rather than direct means. Thus, both efficiency and reliability are of less concern in simulation languages than in real-time languages.

SIMULA has considerably advanced the state-of-the-art but contains transitional as opposed to stable modularity and extensibility features.

The SIMULA class concept supports records only indirectly. In order to meet the DoD requirements, it would be necessary to introduce an additional mechanism for directly realizing records thus resulting in two different language mechanisms for realizing essentially the same language feature.

Although classes are an important and ingeneous language concept, they are two general and high level for the common language. In order to meet the DoD requirements classes would have to be removed and replaced by a set of features (such as records and encapsulated data-types) which directly meet individual DoD requirements. It makes no sense considering SIMULA as a base language if the class concept has to be removed.

3.21 JOVIAL J3B Evaluation

Evaluators

JOVIAL J3B was evaluated by SofTech and CSC.

Language Description

JOVIAL J3B was developed by Boeing, the Aerospace Corporation and the Air Force in 1972. It is a small language with no dynamic storage allocation or input-output. It has some nice features, such as local, global, systems and COMPOOL scopes, evaluation of constants at compile time and conditional compilation for if-then-else statements whose condition can be evaluated at compile time. However, it does not have precision specifications, extension facilities, parallel processing, exception handling or real time facilities.

Overall Evaluation

JOVIAL J3B is inappropriate as a base language because it lacks too many of the required features and in addition has numerous arbitrary restrictions and inconsistencies (SofTech evaluation) which would have to be removed or resolved before J3B could serve as a suitable basis.

Positive Features

JOVIAL J3B has a number of nice language features.

Negative Features

Too many of the substantive DoD requirements are not in the language. It does not have fixed point numbers, precision specifications for variables, enumeration types, type definition and extension facilities, recursive procedures, arrays with dynamic bounds, parallel processing, exception handling and real-time facilities.

It has some inconsistencies in areas such a parameter passing so that deletions from the small existing language would be required before extending the language to meet the TINMAN requirements.

3.22 JOVIAL J73 (level 1) Evaluation

Evaluators

JOVIAL J73 (level 1) was evaluated by Intermetrics and CSC.

Language Description

JOVIAL J73 was designed by an Air Force Committee in the period 1969-73. It has integer, floating point, character and bit data types and aggregate data types called tables and blocks. Although all identifiers must be introduced by type declarations, the language is not strongly typed because overlays permit variables of one type to be used as if they had another type and because there is no type checking in the use of table data and in passing procedure parameters. JOVIAL J73 lacks I/O facilities, does not have enumeration types, does not have mode definition facilities and does not have parallel processing exception handling or real-time facilities.

Overall Evaluation

JOVIAL J73 fails to meet an exceptionally large number of the DoD requirements. It is not strongly typed, has no array and record types, fixed point variables, enumeration types, extension facilities, recursive procedures, parallel processing facilities, exception handling facilities, real-time facilities, I/O facilities. Its defining document differs from any of the current implementations. It is not recommended as a base language.

Positive Features

JOVIAL J73 is designed to be simply compilable and to generate efficient code.

Negative Features

JOVIAL J73 is poor with regard to reliability, transportability, and machine independence.

It lacks strong typing, array and record types, fixed point variables, enumeration types, extension facilities, parallel processing, real time and exception handling features, and I/O facilities.

There are problems concerning the definition of J73. Currently, there are several different versions of J73 including a draft standard which differs from any implemented language.

The parameter passing mechanism is inadequate both because it does not enforce type checking and because it imposes arbitrary restrictions on passing table and blocks.

JOVIAL J73 lacks a machine independent interface to the operating system and has no facility for machine code insertions.

It does not specify the order of expressing evaluation and therefore allows optimizations which can change the effect of the program.

3.23 CORAL 66 Evaluation

Evaluators

CORAL 66 was evaluated by CSC and RLG.

Language Description

CORAL 66 was designed in 1966 by the British Royal Radar Establishment (RRE). It was adopted in the 1970's as an inter-service standard for military programming and has been widely used in the British control on automation industry. It has block structured scope, as well as independently compilable segments, global COMMON data, and overlay declarations. Its data types include floating, fixed, and integer (but not Boolean or character), arrays, packed data and table declarations (in place of records). Recursive procedures must be explicitly designated. There are no parallel processing or real time facilities. Error conditions may be handled by label parameters. Code statements which allow escape into machine language are part of the language.

Overall Evaluation

CORAL 66 is too small and early a language to be recommended as a base language. However, its clean ALGOL 60-like design is impressive and its considerable use in Britain for military and industrial applications must be respected.

Positive Features

A clean design which incorporates both block structure and independently compilable program and data segments.

Considerable use in both military and industrial applications.

Negative Features

It was designed too early to incorporate notions such as record structures, reference variables, enumeration types and other ALGOL 68 and PASCAL language features.

It does not contain Boolean variables, precision specifications, mode definition.

It does not contain parallel processing, real-time features or clean exception handling.

4. Discussion, Recommendations, and Conclusions

4.1 Discussion

The Statement-of-Work for the language evaluations called for an analysis of how each of the languages matched each of the requirements found in the TINMAN. As part of the attempt to consolidate all the evaluation inputs, a matrix was constructed of TINMAN requirements vs particular languages. The matrix was filled in by mapping contractor's individual scoring systems into a unified joint score followed by an averaging where more than one evaluation was done for one language. The information provided by this matrix was of little use for the following reasons:

* Only the specific (as opposed to general) language requirements were considered in constructing these numbers (i.e., TINMAN sections A-J).

* There was no attempt to weigh the relative importance of each requirement relative to each of the other requirements. Each Requirement was considered equal.

* If a language did not satisfy some requirement, the difficulty or ease of making the required change was not distinguished in the matrix numbers.

* In certain cases, if a language did not satisfy some requirement it would affect many other requirements as well. This tended to magnify unduely the overall effect of the missing requirement on the score.

* Large languages with lots of features tended to score higher than smaller languages.

* The scoring did not reflect the fact that even though a language might satisfy a particular requirement (perhaps only partially), it might be the case that a better way of satisfying the requirement entailing a significant amount of work might make the language much better.

* Different evaluators, even when they basically agreed on some point, tended to score differently. SofTech was consistently more severe than say IBM which was rather lenient.

* Some of the languages had only one evaluation while others had up to four evaluations.

* Some evaluations were in a form where the information could not be included in this matrix.

The most useful information to the coordinating committee was obtained by detailed discussions with evaluators after their studies had essentially been completed.

4.2 Recommendations

The first question that had to be answered by the language evaluations was: "Does a language already exist that satisfies the DoD set of language requirements as set forth in the TINMAN, so well that it can be adopted as the Common Language with little or no changes required?" With unanimous agreement, none of the evaluators found that such a language now exists.

The consensus of the evaluations was that it would be possible to produce a <u>single</u> language meeting essentially all the DoD language requirements, hence the next efforts should be directed toward the goal of producing a single language.

Each feature or capability mentioned in the requirements document can essentially be found in some existing language, hence some minimal collection of languages can be found that collectively contains all these features and capabilities. However, there are important embedded computer system applications that could make good use of all the major requirements. In fact, most of the requirements would be useful in any embedded computer application. It is clearly possible to design a language by brute force containing all the <u>technical</u> features and capabilities of the requirements. Problems arise, however, when one adds requirements such as simplicity, uniformity, reliability, design integrity, implementation efficiency, etc. Producing a language that satisfies all of the requirements is not a straight-forward task. The consensus was, nonetheless, that such a language can be produced and work toward the development of a single language should proceed.

Some evaluators did feel that some of the requirements will have to be modified. Many of these have already been incorporated in the new IRONMAN document. Others, concerned for example with the precise degree of trade-off between potentially conflicting requirements (such as simplicity and generality) can be determined only after a substantial amount of further work on the design of the language.

Just about all the evaluators felt that work toward the production of a language satisfying the requirements should start from some carefully chosen base language. Working from such a base language would reduce the amount of work requiree and would reduce the opportunities for making errors. There was no consensus as to which base language to use, but every evaluator indicated that some of the languages considered were more suitable for this role than others. Each felt in fact that it would make a substantial difference depending on the base language chosen. There was remarkable agreement concerning the unsuitability of certain languages to serve as base languages.

Even though starting with some base was recommended, all felt that a design team must have the freedom to make any changes they feel are warranted. For example, upward compatibility with the base language should not be required. The specification for a design effort must be carefully written so as not to place any artificial or undesirable restrictions on the design team.

The role of a base language in a design effort is difficult to define, and it appears that any attempt to define it will not be worth the effort. A well-qualified design team working from a good base language will know how to proceed without such a definition.

With a unanimous degree of agreement, the following languages were found by the evaluations to be inappropriate to serve as base languages: FORTRAN, COBOL, TACPOL, CMS-II, JOVIAL J73, JOVIAL J3B, SIMULA 67, ALGOL 60, and CORAL 66.

This should not be interprested as a statement concerning the technical merits of these languages, it is only a statement by the evaluators that better languages exist for the role of base languages for a design effort toward the Common Language. Some of the languages on this list of rejected languages are among the most widely used languages. The detailed reasons for their unsuitability can be found in the evaluation documents (See Sec.5). We have made an attempt to bring this information together in Section 3.

The languages that remain are: PASCAL, PL/I, ALGOL 68, LIS, EUCLID, CS-4, PDL/2, RTL/2, LTR, PEARL, SPL-1, HAL/S, MORAL, and ECL.

The following recommendation appears to be consistent with almost all the evaluations: Proposals should be solicited from appropriate language designers for modification efforts using any of the languages PASCAL, PL/I, or ALGOL 68 as base languages.

Some of the languages, among those listed above that were not found inappropriate to serve as base language, can play the following roles. The languages PL/I, PASCAL, and ALGOL 68 each represent a different synthesis of a large amount of design experience and each constitutes a nucleus of a family of derived languages. The languages EUCLID, PDL/2, LIS, LTR, and CS-4 are in the PASCAL family and to the extent that a design team feels these experiences are useful, they can be used as appropriate. The language EUCLID, although somewhat of a research language (i.e., one still under development) is very carefully designed to support the writing of verifiable well-structured programs and contains many interesting relevant properties that should be helpful to a PASCAL modification effort. CS-4 has also not been completely designed and tested, but the language is relevant and should be considered by such a design effort. The language LIS contains a very interesting treatment of the requirement for separate compilation among other things and should be useful to design teams no matter what base is used. The language LTR is amodern real-time language standard used in France for a wide range of applications.

A similar case can be made for the languages HAL/S and PEARL relative to PL/I, and RTL/2 relative to ALGOL 68. Again, the designers should have the option to use this information to the extent they deem appropriate.

After the designers have developed their ideas to an appropriate degree, an evaluation of these three modification efforts should be made. This should determine which should be further supported to completion.

4.3 Conclusions

The HOLWG language evaluation study is an interesting and significant application of current techniques of software engineering to the problem of programming language design. A programming language may be regarded as having a life cycle with a requirements analysis and design phase, an implementation and debugging phase and an operations and maintenance phase. Programming languages just as large application programs have in the past been designed with insufficient emphasis on requirements analysis. The present study has emphasized requirements analysis and in so doing has considerably increased our understanding of the nature of requirements specification for programming languages. Thus, this study should be helpful not only in the design of the DoD common language but also in the design of other programming languages.

A great deal of work on detailed design and implementation of the common language still remains to be done. However, the language evaluation study has resulted in a clear recommendation to proceed with the design of the common language by modification of a carefully chosen base language. Moreover the language evaluations study has allowed us to systematically evaluate the suitability and relevance of a large number of currently important programmed languages. The information concerning programming languages accumulated as a result of the language evaluation effort should prove useful not only in choosing a base language, but also in determining how a chosen language should be modified to meet the DoD requirements.

5. A GUIDE TO THE SUPPORTING DOCUMENTS

5.1 Language Requirements

D.A. Fisher, "A Common Programming Language for the Department of Defense -- Background and Technical Requirements," Institute for Defense Analyses Paper P-1191, June 1976.

"Department of Defense Requirements for High Order Computer Programming Languages" - "TINMAN" Defense Advanced Research Projects Agency, publication (known as "TINMAN"), June 1976.

"Department of Defense Requirements for High Order Computer Programming Languages" - "IRONMAN" January 1977.

5.2 Documents used as Basis for Evaluation

5.2.1 FORTRAN

"American National Standard (ANS) FORTRAN," ANSI X3.3-1966, American National Standards Institute, Inc., New York, March 196

"Draft Proposed American National Standard FORTRAN," BSR X-3.3 X3J3/76, American National Standards Committee X3J3, March 1976. (also published in SIGPLAN Notices, Vol. II, No.3, March 1976).

5.2.2 COBOL

"American National Standard Programming Language COBOL, "ANSI X3.23-1974, American National Standards Institute, Inc., New York, 1974 (less the Report Writer Module).

5.2.3 PL/I

"Draft Proposed American National Standard Programming Language PL/I; Basis /1-12" Document BSR X353, Feb. 1975.

"Errata for BSR X353" Document X3J1/399, Jan. 1976.

5.2.4 TACPOL

"TACPOL Reference Manual" USA CSCS-TF-4-1, Data Systems Division, Litton Systems, Inc. Jan. 1972.

"CPCEI SPECIFICATION FOR COMPILER/ASSEMBLER FOR FIRE DIRECTION SYSTEM,"
Spec. No. EL_CG-00043082C, Doc. No. 595909, Vol.1 of 2, Appendix 10, Data Systems Division, Litton Systems, Inc. April 1971 (Formal Definition of TACPOL).

5.2.5 HAL/S

"HAL/S Language Specification," Intermetrics, Inc., Cambridge, Mass., April 1973.

5.2.6 CMS-2

"Users Reference Manual for Compiler, Monitor System-2 (CMS-2) for use with the AN/UYK-7 Computer" M-5035, Vol.1 and 2. FCDSSA - San Diego, Cal. 15 Aug. 1975.

"CMS-2Y Programmer's Reference Manual (Preliminary Version)" M50
FCDSSA - San Diego, Cal. 1 Oct. 1976.

5.2.7 CS-4

"CS-4 Language Reference Manual and Operating System Interface." Advanced Software Technology Division, NELC, San Diego, Cal. Oct. 1975.

"CS-4 Primer, Vol.I, Basic Features." Advanced Software Technology Division, NELC, San Diego, California, Feb. 1975.

5.2.8 JOVIAL/J3B

"JOVIAL/J3B Extension 2 Language Specification" Softech, Inc. Waltham, Mass., Document 2044-4.2, July 1975.

5.2.9 JOVIAL /J73

"JOVIAL J73/1 Specification" RADC Document, July 1976.

5.2.10 ALGOL-60

P. Naur (Editor), "The Revised Report on the Algorithmic Language ALGOL-60." Communications ACM, Jan. 1963.

R. Baumann, M. Feliciano, F.L. Bauer, and K. Samuelson, Introduction to ALGOL. Prentice-Hall, Englewood Cliffs, N.J. 196

R.M. DeMorgan, I.D. Hall and B.A. Wichman, "A Supplement to the ALGOL 60 Revised Report," The Computer Journal, Vol. 19, No. 3, 1976.

5.2.11 CORAL 66

"Inter-Establishment Committee on Computer Applications Official Definition of CORAL 66" Ministry of Defense Her Majesty's Stationary Office London, 1970.

5.2.12 ALGOL-68

A.Van Wijngaarden, et.al. (eds) "Revised Report on the Algorithmic Language ALGOL 68. Springer-Verlog, 1976.

A. Van Wijngaarden, et.al., "Revised Report on the Algorithmic Language ALGOL 68" Acta Informatica, Vol. 5, pages 1-236, 1975.

A.S. Tanenbaum, "A Tutorial on ALGOL 68." Computing Surveys, Vol. 8, No. 2, June 1976.

5.2.13 SIMULA 67

O.J. Dahl, B. Myhrhaug, and K. Nygaard, "SIMULA 67, Common Base Language," Norwegian Computing Center Publication No. S-22, 1970.

P. Naur (Ed.), "Revised Report on the Algorithmic Language ALGOL 60," Comm. ACM, January 1963, Vol.6, No.1, pp 1-17.

A. Birtwistle, O. J. Dahl, B. Myhrhaug, and K. Nygaard, "SIMULA BEGIN," Auerbach Publishers Inc., Philadelphia, 1973.

J. Palme, "SIMULA as a Tool For Extensible Program Products, "ACM SIGPLAN Notices," Feb. 1974, Vol.9,No.2, pp.24-40.

NCC System Programming Group, "The Structure of the NCC SIMULA, Compilers and Bench Mark Comparisons with other Major Languages, "Norwegian Computing Center Publication" No. S-43, 1972.

5.2.14 PASCAL

K. Jensen and N.Wirth, "PASCAL User Manual and Report," Springer-Verlog, 1974.

N. Wirth, "An Assessment of the Programming Language PASCAL," IEEE Trans. on Software Engineering, 1, 192-198. (1975).

N. Wirth, "The Design of a PASCAL Compiler," Software - Practice and Experience,,1, 309-333, (1971).

5.2.15 LIS

J.D. Ichbiah, J.P. Rissen, and J.C. Heliary, "The Two-Level Approach to Data Independent Programming in the LIS System Implementation Language." Compagnie Inter. Pour L'Informatique, Louveciennes, France, June 1975.

"The System Implementation Language LIS," CII. Document No. 4549 EL/EN, Jan. 1976.

J.D. Ichbiah, J.C. Heliard, "Plexes in LIS, 1975.

J.D. Ichbiah and P. Cousot, "Visibility and Separate Compilations," 1975.

5.2.16 RTL/2

"RTL/2 Language Specification for RTL/2 Reference: 1 Version: 2", Imperial Chemical Industries Limited, 1964

5.2.17 EUCLID

B.W. Lampson, J.J. Horning, R.L. London, J.G. Mitchell, G.J. Popek, "Report on the Programming Language EUCLID"

5.2.18 MORAL

"A Description of the Programming Language MORAL" Reference No: 22/1132 Software Sciences Limited, Feb. 1976.
"Mascot - A Modular Approach to Software Construction Operation and Test" Technical Note No. 778, Royal Radar Establishment, Procurement Executive, Ministry of Defense, Malvern, Worcs.

5.2.19 PEARL

"PEARL - Subset for Avionic Applications Language Description" ESG Elektronik-System-Gesellschaft M6H, June 197

5.2.20 SPL/I

"Reference Manual for SPL/I" Intermetrics, Inc.

5.2.21 PDL/2

"Process Design Methodology Design Specification, Vol.1, Process Design Language" Sept. 1976, Texas Instr. Inc.

5.2.22 LTR

"LTR Reference Manual" (French) Centre de Programmation de la Marine, Paris December 1976.

"LTR Programmers Manual: Centre de Programmation de la Marine, Paris December 1976.

5.2.23 ECL

"Reference Manual for ECL" Harvard University, 1974.

5.3 Documents produced by Language Evaluators

5.3.1 "HOL Evaluation Project Interim Technical Report," Science Applications, Inc., Software Technology Center, 211 Sutter Street, San Francisco, California 94103.

5.3.2 "Evaluation of CORAL 66"
"Evaluation of PASCAL"
"Evaluation of CS-4"
"Evaluation of TACPOL"
"Evaluation of CMS-2
RLG Associates,Inc.11250 Roger Bacon Dr., Reston, Va. 22090

5.3.3 "Evaluation of ALGOL 68, JOVIAL J3B, PASCAL, SIMULA 67 and TACPOL vs. TINMAN Requirements for a Common High Order Programming Language," Oct. 1976. Softech, Inc., 460 Totten Pond Road, Waltham, Mass. 02154

5.3.4 "Draft Candidate Language Evaluation and Recommendation Report" I.B.M.

5.3.5 B.L. ~~[illegible]~~ Marks, "Some Characteristics for a Common DoD Programming Language and their Relation to PL/I" IBM, United Kingdom Laboratories Limited, Harsley Park, Winchester.

5.3.6 "Draft Version - Candidate Languages Evaluation and Recommendation Report," Dec. 1976, Intermetrics, Inc., 701 Concord Ave., Cambridge, Mass. 02138.

5.3.7 "Candidate Language Evaluation and Recommendation Report" Computer Sciences Corp., 6565 Arlington Boulevard, Falls Church, Va. 22046

5.3.8 L.Campbell, "Comparative Evaluation TINMAN VS. FORTRAN" BRL, APG, Aberdeen, Md. August 1976.

5.3.9 A. Demers, "Evaluation of LIS, LTR, RTL/2" December 1976.

5.3.10 S. Squires, "Draft Evaluation of ECL with Respect to "TINMAN" Dec. 1976.

5.3.11 P.Parayre, "LTR Evaluation towards "TINMAN" Requirements" (From Centre de Programmation de la Marine, Paris).

5.3.12 J.G.P. Barnes, "An Informal Evaluation of RTL/2 against TINMAN" Oct. 1976. (From Imperial Chemical Industries, Ltd.)

5.3.13 "An Assessment of the Programming Language MORAL against TINMAN" Oct. 1976, Software Sciences Ltd., Abbey House, 282-292 Farnborough Rd., Tarnborough Hampshire, England.

5.3.14 "Language Evaluation of PDL/2" Texas Instruments Dec. 1976.

5.3.15 J. Williams, "An Evaluation of PEARL" Cornell Univ., Jan.1977

5.3.16 D. Morris et.al.,"DoD-1 Language Evaluation Matrix" Jan.1977.

5.3.17 J. D. Ichbiah, "A Comparative Evaluation of the LIS Language with Respect to the TINMAN Requirements" Oct. 1976

5.3.18 Thomas Martin, "A Comparative Evaluation of PEARL, the Process and Experiment Automation Realtime Language, against the DOD-TINMAN-Requirements for High Order Computer Programming Languages" June 1976

5.3.19 "An Assessment of the Programming Language CORAL 66 Against The U.S. Department of Defense TINMAN Requirements" Nov. 1976, Software Sciences Ltd., Abbey House, 282-292 Farnborough Rd., Farnborough, Hampshire, England