A method of software evaluation: the case of programming language translators

H. C. Lucas, Jr.* and L. Presser†

This paper presents a method for the specification and evaluation of software modules. The proposed strategy involves the use of existing documentation and synthetic or benchmark programs to evaluate a set of key characteristics that represent the module under consideration. The method is discussed and illustrated by considering the evaluation of programming language translators in detail. An example demonstrates how the method is applied to the evaluation of the popular WATFIV FORTRAN compiler.

(Received August 1972)

The evaluation and selection of computer systems is an important aspect of modern technical management. At the present time, software represents a major portion of the cost of computer systems. Therefore, systematic approaches to the specification, evaluation and selection of software are of great practical importance.

The problem is emphasised by the recent 'unbundling' of the computer industry. Originally, all hardware and software was obtained from the computer manufacturer at a single, 'bundled' price. Today, different manufacturers have different practices, but IBM with well over half the world's computer market has separate pricing. A large number of independent software companies offer software packages which compete with those available from the hardware manufacturer. For the decision maker, this increased flexibility means that more alternatives will have to be considered in the configuration of a system. Indeed, the sophisticated and aggressive manager may effect large savings through a sound evaluation procedure.

In this paper a method is presented for the evaluation (specification) of programming language translators which include compilers, interpreters, and assemblers. The method illustrated here for the case of translators may be applied to the evaluation of other software modules.

The evaluation of programming language translators is carried out by first identifying for evaluation a set of fourteen key translator characteristics. The weight and applicability of each characteristic depends on the evaluator's environment. Next, techniques for the evaluation of these characteristics are discussed. Finally, as an example, the method is applied to the evaluation of the WATFIV FORTRAN compiler.

Translator characteristics

For purposes of evaluation and specification it is appropriate to consider a programming language translator as composed of a set of independent characteristics, the aggregate of which defines the desired translator (Presser, 1970). These characteristics are not listed in the order of importance; their weight will in general be unique to each situation.

Set of characteristics

1. *Cost*

Specifies the cost of the translator in dollars, the terms for purchase or lease, and the provisions for update and modifications.

2. *Source language*

A translator must accept and correctly translate any syntactically legal source language program. The process of defining the syntax of a source language, that is, the total set of valid programs, is under control (e.g. Backus Naur Form). However, the translator may only have been designed to accept a subset of the full language specifications.

3. *Translation environment*

It is necessary to specify the minimum environment required by the translator, or the available environment if it is in excess of what is considered the minimum requirements. The specification should consist of a description of the hardware environment (e.g. CPU, memory space, I/O units) as well as of the software environment (e.g. library, utility, and storage management routines). Also, a delineation of the interfaces between the translator and the rest of the system should be included.

4. *Translation time*

How much time does the translator require in the translation of programs? Is the translator I/O or compute bound?

5. *Execution environment*

Comments similar to those appearing under 3 above apply here.

6. *Execution time*

Comments similar to those appearing under 4 above apply here.

7. *Diagnostic information and error recovery*

There are two aspects of this characteristic: the first is diagnostic information provided and degree of error recovery. The effectiveness of a language and translator system should be measured in terms of the amount of time and effort required from the time a programmer starts coding (assuming he understands his problem well) to the time he obtains a working program. Thus, it is important to determine the degree to which a translator aids a programmer in the isolation and correction of errors.

The second aspect is general information. Any summary data and its quality should be evaluated including: location of variables, constants and literals; which variables were referenced and where, etc.

8. *Documentation*

Four types of documentation are important: first, structural/functional documentation describes the logical and modular composition of the translator as well as its interfaces with the

*Graduate School of Business, Stanford University, Stanford, California 94305, USA.
†Department of Electrical Engineering, University of California, Santa Barbara, California 93106, USA. (This work was supported in part by the National Science Foundation, Grant GJ-31949.)

rest of the system; more detailed documentation should include the actual code interspersed with comments. Third, installation information explaining how to install and tailor the translator to a particular environment should be provided. Finally, documentation describing maintenance procedures is useful.

9. *Translator writing system organisation*
A translator writing system is the aggregate of tools that facilitate the writing of translators for various languages and different computers (Presser, 1972). The importance of this characteristic depends on the environment and use planned for the translator. What is involved in making modifications to the translator such as moving it to a different computer or translating a different source language? If such changes are contemplated, consideration of this characteristic at translator selection time could result in major savings.

10. *Translator mode*
The translator may operate in diverse environments (modes) such as batch or time-sharing modes.

11. *Execution mode*
Comments similar to those appearing under 10 apply here.

12. *Conversion and function evaluation*
Most programming languages include automatic conversion and evaluation of functions (e.g. SIN); the accuracy of these evaluations is important.

13. *Effect of translator on language*
To what extent does the translator, itself, influence the original source language definition? For example, in some cases it may be possible to effect a much better translator implementation if changes to the source language are permitted (McAfee and Presser, 1972). This characteristic would only be applicable if a new language was being designed.

14. *Monitoring artifact*
It is useful to obtain measurements on the translation process and on the execution of translated programs (Lucas, forthcoming). It is necessary to specify here the type of measurements desired and an upper limit to the cost of obtaining these measurements. The cost is specified as a percentage of the time and resources (storage space in particular) required for measurement. Experience shows that the introduction of software measurement artifact for the collection of a large class of interesting data is inexpensive if considered during the design of the translator; this is particularly true if the translator is well organised (Presser and Melkanoff, 1969).

Evaluation techniques

Methods
Two basic tools are available to evaluate the performance of computer systems or their component modules. Modelling, whether analytic and/or through simulation, allows the prediction of system performance. Monitoring, through hardware and/or software techniques, permits the measurement of actual performance. A number of block-level models of programming language translators have been developed (Presser, 1972); however, these are not suitable for detailed evaluation of translator performance. The coding of the detailed simulation of a translator would require as much effort as the actual translator implementation, particularly so when translator writing system tools are available. Therefore, the recommended methodology for the evaluation of implemented translators is based on benchmarks or synthetic modules and monitoring (Lucas, 1971a). Benchmarks and synthetic modules are examined in some detail here. For a careful treatment of monitoring techniques the reader is referred to Presser and Melkanoff (1969) and Dumont and Presser (forthcoming).

A benchmark program represents an existing program which has been in use in the installation. The program is physically executed on the computer which is to be involved in the evaluation. A synthetic module is used in the same way as a benchmark program; however, it does not necessarily represent an existing program. Instead, a synthetic module is written to model the characteristics of the anticipated job stream. The major advantage of synthetic modules over benchmarks is the flexibility they provide. A series of synthetic modules can be parameterised so that the user has a number of different options available for representing his job load. For example, a module dealing with matrix manipulation should make it possible to select the different operations performed and to input parameters defining the size of the matrices involved in the calculations (Lucas, 1972). For translator evaluation, the use of synthetic modules may be required if there is no history from which to select benchmarks.

Use
To use a series of benchmarks or synthetic modules, it is necessary for the evaluator to have some idea of either his present or anticipated job stream. Appropriate benchmarks or synthetic modules can be selected which reflect that job stream. Once a job load has been estimated, a series of benchmarks or synthetic modules can be linked together to form programs. At the next level, the different programs are merged into a series of experiments in which the jobs are executed and the results interpreted. For example, to model a 40 per cent I/O bound and 60 per cent CPU bound system, the evaluator could configure synthetic modules or benchmarks representing exactly this proportion of activity, or he could use four exclusively I/O bound and six CPU bound jobs in one experiment. For some of the criteria for translator evaluation, timing information will be the only data to be evaluated. However, for a thorough profile of the translator, the evaluator will have to examine listings, error diagnostics and other information produced by the runs and some of this information must be rated subjectively.

Problem areas
There are a number of problems with the technique suggested above for evaluating translator performance. In many instances, determining the job load is a difficult undertaking for several reasons. First, particularly in scientific or engineering computing centres, the load varies widely from day to day. While the average load may not tax the available equipment, peak loads create large bottlenecks in processing which are difficult to model. A large number of runs is needed to faithfully model the anticipated job mix. When the purpose of the evaluation is to select a translator for a system development project, the type of code which will be translated may not be obvious at the stage when translators are being evaluated. (For an interesting discussion of some experimental results on FORTRAN usage, the reader is referred to Knuth, 1971.)

It is not clear exactly how different modules should be linked together to form jobs. If the modules are too small, a large amount of time may be spent in linkage between modules so that the runs become unrepresentative. Using the approach suggested here also requires access to the translator and to the computers on which it might be used. A large number of runs and corresponding computer time may also be required to undertake a thorough evaluation. Finally, the analysis of the results is difficult. For some of the simple indicators such as execution time and memory space utilisation, there is little problem. However, other indicators such as quality of documentation or degree of error recovery require careful consideration.

Table 1 Translator characteristics and evaluation techniques

TRANSLATOR CHARACTERISTIC	BENCHMARK/SYNTHETIC MODULES	EVALUATION	SPECIFICATION DOCUMENTATION
1. Cost	—	Is the translator, in the desired form, and its documentation available for the amount quoted?	Cost
2. Source language definition and acceptability	A series of syntactically correct program modules representing all source language elements	Are the programs translated and executed correctly?	Are any restrictions and/or extensions to the source language accurately documented?
3. Translation environment	A series of program modules that exercise the specified translation time minimum environment	Are the programs translated and executed correctly?	Are the stated minimum environment and interfaces accurate?
4. Translation time	A representative set of program modules is translated (and executed)	Is translation time within specified limits?	Any special cases?
5. Execution environment	A series of program modules that when translated exercise the specified execution time minimum environment	Are the programs executed correctly?	Are the stated minimum environment and interfaces accurate?
6. Execution time	A representative set of program modules is translated and executed	Execution time below specified limits?	Any special cases?
7. Diagnostic information and error recovery	A series of program modules with specific errors is translated and executed	Evaluation of diagnostic information and error recovery [refer to (Presser & Benson, submitted for publication) for a possible approach]	Is error handling within specified limits?
8. Documentation	—	Does the documentation satisfy the general objectives stated in the text of this paper?	Is the documentation available?
9. Translator writing system organisation	—	Does the translator organisation permit the type of modifications desired? Is the effort required within specified limits?	What are the inherent system limitations?
10. Translator mode	A series of program modules is translated (and executed)	Does the translator operate correctly in the desired mode(s)?	Any special cases?
11. Execution mode	A series of program modules is translated and executed	Does the object code execute correctly in the desired mode(s)?	Any special cases?
12. Conversion and function evaluation	A series of program modules is translated (and executed)	Are converted and evaluated values within specified limits?	Any special cases?
13. Effect of translator on language	—	To what extent is it permissible to modify the source language?	—
14. Monitoring artifact	A series of program modules is translated and executed, with and without the artifact	Are the desired measurements clearly output? Is the monitoring cost within specified limits?	Are the monitoring facilities and their use clearly documented?

Translator evaluation

Translator attributes

It is not possible to evaluate directly or simply some of the criteria for translator performance discussed in the second section of the paper. **Table 1** lists the various criteria and describes how synthetic modules or benchmarks are used for evaluation along with an examination of documentation. The second column of the table describes the modules which are

executed for each particular criterion. The third column describes how the results of the tests are evaluated and the final column discusses the role of the documentation.

For example, the second criterion, the acceptability of specified source language, can be evaluated using a series of syntactically correct modules which represent all of the language elements of interest. The output of this process is evaluated on whether or not the programs were translated and executed correctly. The documentation is also examined to determine if it properly specifies any limits of the source language. The methods suggested in the table are sufficient for the evaluation of a single translator or for comparing several alternative translators for the same language.

Module requirements

In reviewing Table 1 it can be seen that the evaluation process requires a number of different benchmarks or synthetic modules and calls for the formulation of several different types of experiments. Two major types of modules are required as shown in **Table 2.** The first of these is a 'standard series' of modules which reflects the anticipated job mix. It contains a representation of the important elements of the anticipated job stream. Different subsets of this series are selected for experiments. For example, one subset can be used to evaluate function conversion and evaluation. Other criteria are evaluated with this series by selecting modules to represent the proportion of input and output activities; the proportion of compilation versus execution, etc. This representation can either be accomplished in the selection and linking of modules, or in the types and numbers of modules incorporated into a series of experiments.

Table 2 Synthetic/benchmark module series for translator evaluation.

Series 1
Modules reflecting anticipated job mix
Series 2
(*a*) All language elements
(*b*) Syntax errors in all language elements
(*c*) Logical errors

The second series of modules is composed of three subseries used for evaluating particular translator criteria. The first of these subseries incorporates every statement type specified for the translator. The modules in this subseries should also execute in some meaningful way so that both the syntactic acceptability of the subset and the correctness of the code generated can be evaluated. The second subseries contains statements with incorrect syntax for each statement type to evaluate the handling of individual errors and their cumulative effect on translation. This subseries contains a number of modules in which the type and severity of errors are varied. The last subseries includes modules with logical errors to determine how errors are handled at run time.

Using these two major series of modules, experiments can be formulated to evaluate translators on many of the criteria described earlier as shown in **Table 3.** The amount of effort devoted to the evaluation can be balanced against the expected benefits. Simple modules which are roughly representative of the anticipated job mix have been used where time or resources for evaluation are quite limited (Lucas, 1971b). Also, if two translators are being compared which are known to be very similar and to differ only on minor points, some of the criteria may not be evaluated to reduce the amount of effort involved.

An example

A hypothetical example has been constructed to illustrate the proposed method for translator evaluation.

Setting

An engineering computer centre is used primarily by different personnel for engineering computations both for research and design work. The processing load is almost entirely in FORTRAN; there is minimal Input/Output activity and there are a large number of compilations as many short jobs are debugged. The computer in use is an IBM 360 model 75 running under Operating System (OS) 360.

Table 3 Translator characteristics and evaluation modules

1. Cost	*
2. Source language definition and acceptability	2a
3. Translation environment	1
4. Translation time	1
5. Execution environment	1
6. Execution time	1
7. Diagnostic information and error recovery	2b, 2c
8. Documentation	*
9. Translator writing system organization	*
10. Translator mode	1
11. Execution mode	1
12. Conversion and function evaluation	1
13. Effect of translator on language	*
14. Monitoring artifact	1

*Not suitable for evaluation with modules; see Table 1

The management of the computer centre is interested in evaluating WATFIV[1], an in-core compiler whose design emphasises detailed diagnostics. The major management questions are: Will it execute successfully on the present computer? How does it compare with two other FORTRAN compilers available at the centre: FORTRAN G[2] and FORTRAN H[3]?

Approach

The proposed evaluation method can be used to answer these management questions. While the evaluator does not have specific data available on the job mix, it appears to be similar to the one reported in Knuth (1971). In that work, static analysis and dynamic monitors were used to analyse the structure of a number of FORTRAN programs; the most important statements in terms of compilation and amount of time in execution were listed. In summary, it was reported that the most popular constructs were: ASSIGNMENT, IF, GO TO, and DO.

To represent such a job mix, a group of synthetic modules was programmed. The first series consists of several modules linked together to represent exactly the proportion of statements observed by Knuth. Since a dynamic monitor was not available to give execution frequencies, the synthetic program was designed to execute each statement the same number of times as the frequency of execution time reported in Knuth (1971). These tests matched the static counts exactly and approximated the dynamic behaviour. A separate synthetic module was coded to evaluate function conversion. This module printed the sine, cosine, tangent, and cotangent for a sample of arguments. Common and natural logarithms were also evaluated in the same manner.

[1]The WATFIV compiler (Version 1, level 2, Aug. 1970), is available from the University of Waterloo Faculty of Mathematics, Waterloo, Ontario.
[2]Level 18.
[3]September 69 release.

Table 4 Evaluation of WATFIV

1. *Cost*

WATFIV: $500 (Includes a copy of the translator, future updates, user's guide and installation guide.)

G: No additional cost to IBM 360 centres. (Includes an object copy of the translator, future updates, user's manual, installation information, and program logic manual.)

H: No additional cost to IBM 360 centres. (Includes an object copy of the translator, future updates, user's manual, installation information, and program logic manual.)

2. *Source language*

WATFIV supports the FORTRAN language defined in IBM document C28-65-15-7; in addition the following facilities are supported.

Free form I/O
Character variables
Multiple assignment statements
Expressions in output list
Initialisation of blank COMMON
Initialisation of COMMON in other than block data subprograms
Implied DO's in data statements
Subscripts in right-hand side of statement function definitions
Logical complex, or character value subscripts
Multiple statements per card
Comments after statements

3. *Translation environment*

Hardware. The minimum translation time requirements of WATFIV are an IBM System 360 with:

128K bytes of main memory
Universal instruction set
One printer with at least 132 print positions
One direct access storage device
One tape drive.

Software. A detailed list of all OS 360 Data Management and Supervisor Macros used by WATFIV is available; however, it is not presented here for the sake of conciseness.

4,6. *Translation and execution times*

As described under item 14 below, the WATFIV compiler outputs translation and execution times. Since such information is not supplied by the G and H compilers it is difficult to make a fair comparison without careful study (monitoring) of these systems. WATFIV performs translation and execution in one step. The G and H compilers require one step for translation and one for execution. Operating System 360 outputs the times spent in each step of a job. Thus, listed below are the total times spent in translation and execution as read from the printed step times. The adjusted results are the difference between the synthetic series step times and the step times of a program consisting solely of an END statement. The adjusted figures represent a rough estimate of translator performance less operating system overhead. When interpreting the results tabulated here, the reader should be well aware of the fact that the Standard timing mechanisms in the System 360 are rather poor for measurement purposes (Dumont and Presser, 1972).

	STEP TIME IN SECONDS	
	WATFIV	G
Synthetic series 1	2·57[2]	3·44 + 1·29 = 4·73
END card program	0·35[3]	0·40 + 0·66 = 1·06
Adjusted results	1·28 + 0·85 ≃ 2·22	3·04 + 0·63 = 3·67
		H(H2)[1]
Synthetic series 1		5·07 + 1·12 = 6·19
END card program		0·77 + 0·55 = 1·32
Adjusted results		4·30 + 0·57 = 4·87

5. *Execution environment*

Similar to 3.

7. *Diagnostic information and error recovery*[4]

	WATFIV	G	H(HO)
ASSIGNMENT	66%	55%	47%
IF	73%	58%	69%
GO TO	72%	51%	42%
DO	65%	12%	45%

8. *Documentation available to user*

	WATFIV	G	H
Structural/functional	No	Yes	Yes
Listings	Yes	Yes	Yes
Installation	Yes	Yes	Yes
Maintenance	No	No	No

9. *Translator writing system organisation*

The WATFIV translator is intended for a well defined FORTRAN/360 combination. It is coded in 360 assembly language and this characteristic does not apply.

10. *Translator mode*

WATFIV	G	H
Batch (in-core)	Batch	Batch

11. *Execution mode*

WATFIV	G	H
Batch	Batch	Batch

12. *Conversion and function evaluation*

Selected trigonometric functions agreed to at least five decimal places between all three compilers. Selected common and natural logarithms agreed to at least four decimal places between all three compilers. All results agreed to four decimal places with a book of tables (Allen, 1947).

13. *Effect of translator on language*

Since FORTRAN is well defined, no changes to it are permitted and this characteristic does not apply.

14. *Monitoring artifact*

The WATFIV translator outputs the time that it took to translate a program, as well as the time that the translated program spent in execution. Information about the amount of memory used by a program during execution is also output. Such information is not supplied by the FORTRAN G and H compilers.

[1]The FORTRAN H compiler allows one of three levels of optimisation to be specified: H0, H1, H2; the highest level is H2.
[2]According to the WATFIV output 1.31 secs was devoted to compilation and 0.86 secs to execution.
[3]According to the WATFIV output 0.03 secs was devoted to compilation and 0.01 secs to execution.
[4]The higher the percentage, the better the diagnostics. For details on the rating scheme, refer to the text and Presser and Benson (submitted for publication).

The second series of modules did not include a program to exercise all language constructs. It is known that WATFIV does include the IBM FORTRAN specifications from experience, documentation, and because this was a major design goal of the compiler. This second series consisted of a set of submodules that were used to evaluate syntactic error detection and correction.

The approach employed for the evaluation of diagnostics is that discussed in Presser and Benson (submitted for publication). In essence, it consists of defining a weighted error range and assigning to each translator a total score based on its performance; the higher the score the better the performance. The level of the error range utilised varies from detection and correction of specific errors at the positive extreme to incorrect error information at the negative extreme. Four synthetic error submodules were coded. Each submodule was utilised to evaluate the response of the translator to errors in each of the four FORTRAN statement types previously mentioned. The overall quality of the diagnostic messages was also taken into consideration.

Finally for those translator characteristics shown in Table 3 which were not amenable to evaluation using synthetic modules, relevant documentation was examined. All runs were made on an IBM 360/75.

Results

The results of the evaluation are shown in **Table 4.** The WATFIV translator is relatively inexpensive and is, in essence, compatible with IBM's G and H FORTRAN translators. (WATFIV provides extensions to the FORTRAN language not supported by IBM; these might have to be prohibited from use by management to maintain compatibility.) The translator requires a translation and execution environment that is amply covered by the hypothetical facilities.

The various times measured are listed in Table 4. In evaluating these times it must be taken into account that different translators are designed with emphasis on different characteristics. For example, WATFIV is supposed to emphasise diagnostics while the IBM FORTRAN H is intended to emphasise the optimisation of object code. The evaluation of diagnostics indicates that WATFIV indeed performs a much better diagnostic task than either the G or H compilers. To do a fair evaluation of the object code (i.e. execution time) generated by the FORTRAN H compiler it would be necessary to run a more elaborate job mix and to obtain more accurate execution times. Also, since the WATFIV translator is resident in main memory during execution of object code, the WATFIV system requires more main memory space than the G and H translators.

The results relating to the remaining characteristics can be read directly from Table 4, answering management's earlier questions. First, the WATFIV compiler will run on the system. Second, it compares well with the presently available FORTRAN compilers. WATFIV offers superior diagnostics and fast compile times but slower execution times. Thus, it is well suited to an environment which includes heavy debugging.

Summary

The proper evaluation and selection of software is of great importance to those technical managers responsible for the software acquisition process. A method has been presented that permits a sound approach to the problem of evaluating software modules. In essence, the method consists of identifying those characteristics that determine the behaviour of the software module under consideration. Once these characteristics have been isolated they are evaluated through synthetic or benchmark programs that model the anticipated job mix. The results obtained are examined to arrive at an overall evaluation of the software module. Throughout the process the available documentation is examined in detail.

To illustrate the method, the evaluation of programming language translators was treated in detail. Fourteen key translator characteristics were identified and modules for their evaluation described. Finally, the technique was employed to evaluate the WATFIV compiler.

References

ALLEN, E. S. (ed.) (1947). *Six Place Tables*. McGraw-Hill Book Co., Inc. New York.

DUMONT, D., and PRESSER, L. System Monitoring, in *Infotech's* 1972 *State of the Art Report on Operating Systems*, Infotech Information Limited, United Kingdom (forthcoming).

KNUTH, D. (1971). An Empirical Study of FORTRAN Programs, *Software Practice and Experience*, Vol. 1, pp. 105-133.

LUCAS, H. C., Jr. (1971a). Performance Evaluation and Monitoring, *Computing Surveys*, Vol. 3, No. 3, pp. 79-91.

LUCAS, H. C., Jr. (1971b). The Evaluation of a Time Shared Computer Using a Synthetic Job, *La Primera Conferencia de Computacion para Latinoamerica*, Mexico City, Mexico, August 1971, pp. 573-582.

LUCAS, H. C., Jr. Performance Evaluation and Project Management, in *Command and Control Software Technology for* 1975-1985. Naval Electronics Laboratory Center, San Diego, California (forthcoming).

LUCAS, H. C., Jr. (1972). Synthetic Program Specifications for Performance Evaluation, *Association for Computing Machinery* 1972 *Annual Conference*, Boston, Massachusetts, pp. 1041-1057.

MCAFEE, J., and PRESSER, L. (1972). An Algorithm for the Design of Simple Precedence Grammars, *JACM*, Vol. 19, No. 3, pp. 385-395.

PRESSER, L. (1970). On the Specification of Programming Language Translators, IEEE Sixth Regional Conference, Seattle, Washington, May 1970.

PRESSER, L. (1972). Translation of Programming Languages, in *Computer Science*, A. Cardenas, L. Presser, M. Marin (eds.), John Wiley & Sons, Inc., New York, pp. 365-408.

PRESSER, L., and BENSON, J. *Evaluation of Compiler Diagnostics*, (submitted for publication).

PRESSER, L., and MELKANOFF, M. A. (1969). Software Measurements and Their Influence upon Machine Language Design, Spring Joint *Computer Conference*, Boston, Mass., May 1969, pp. 733-737.