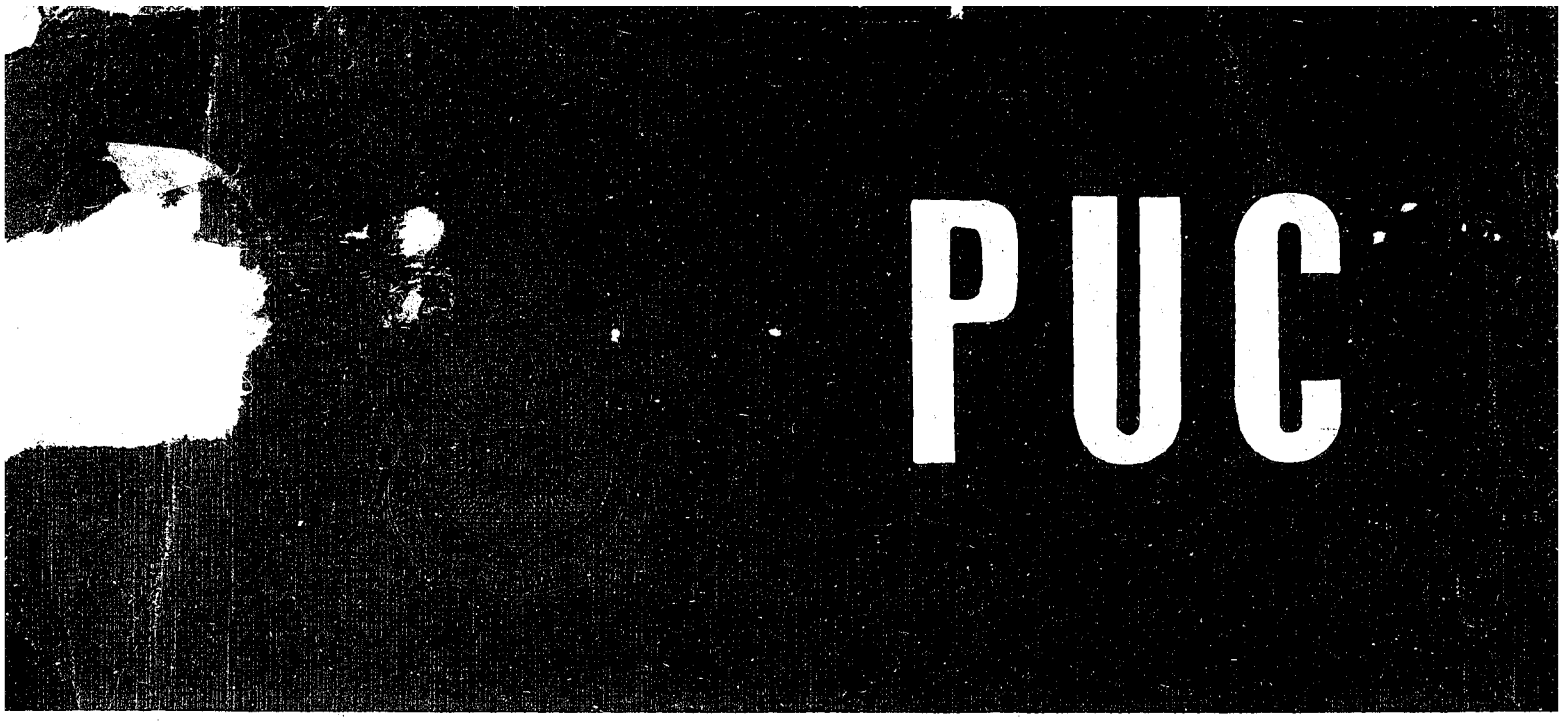


PRE-EDIÇÃO

# SIMPOSIO INTERNACIONAL SOBRE METODOLOGIAS PARA O PROJETO E CONSTRUÇÃO DE SISTEMAS DE SOFTWARE E HARDWARE

PATROCINADOR

CONSELHO NACIONAL DE DESENVOLVIMENTO CIENTIFICO E TECNOLOGICO

28 DE JUNHO A 2 DE JULHO DE 1976

RIO DE JANEIRO
BRASIL

Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 209 — ZC-20
Rio de Janeiro — Brasil

005.106
I61p

# PREPRINTS

# INTERNATIONAL SYMPOSIUM ON METODOLOGIES FOR THE DESIGN AND CONSTRUCTION OF SOFTWARE AND HARDWARE SYSTEMS

SPONSOR : CONSELHO NACIONAL DE DESENVOLVIMENTO CIENTIFICO E TECNOLOGICO

CO-SPONSORS: CANADIAN INTERNATIONAL DEVELOPMENT AGENCY
GESELLSCHAFT FUR MATHEMATIK UND DATENVERARBEITUNG
NATIONAL SCIENCE FOUNDATION
IBM DO BRASIL

ORGANIZING COMMITTEE:

PROFESSORS

C.J.P. LUCENA
W. BRAUER
N. MACHADO
A.L. FURTADO

DATES

JUNE 28 TO JULY 2, 1976

NOTE

THESE PREPRINTS HAVE BEEN EDITED. THE AUTHORS ARE KINDLY INVITED TO REQUEST ANY CORRECTIONS OR CHANGES FOR THE FINAL EDITION.

# A DISCUSSION OF THE CONCEPT OF MODULES IN PROGRAMMING SYSTEMS

by

Donald D. Cowan
Computer Science Department
University of Waterloo
Waterloo, Ontario
Canada

C.J. Lucena
A. von Staa
Informatics Department
Pontifical Catholic University
of Rio de Janeiro
Brasil

Key phrases: Modularity, Programming languages, Programming Systems.

CR Categories: 4.2

This research was supported in part by grants and contracts from:
The National Research Council of Canada,
The Canadian International Development Agency,
IBM (Canada).

## 1. Introduction

The effective and widespread use of methods for the design and construction of programs will require support by automatic or semi-automatic procedures; such a set of procedures has often been called a system's environment for program development. When designing such systems a clear understanding of the semantics of programs and of the synthesis rules for creating such programs is required.

Recent research in software engineering has led to some new concepts in the area of programming language design, a better understanding of program structure and a better characterization of some of the fundamental properties of software systems. The concept of modularity, for instance, has received a very thorough treatment in the works of Parnas [2,3], Dennis [1], Liskov and Zilles [4,5] and others.

This paper and a comprehensive report [14] on this subject are motivated by a desire to contribute further to the understanding of the concept of modularity, since the authors felt the need for a very systematic characterization of the concept of modularity in the process of designing a programming system for software development. Program units called modules will allow the synthesis of complex programming systems from "off-the-shelf" program components and hence programmers can reduce the cost of producing such systems. Furthermore, modules are powerful mechanisms to model the abstractions which occur at the various levels of development of a programming system.

When trying to characterize the concept of a module a number of issues come into play. Some of the important ones are: system and program structure, programming language design, specification language

and techniques and management of programming. Previous research in our opinion has not adequately related these issues to the definition and use of the concept of modularity.

In this paper we start with a reference to Boebert used by Dennis [1] (on the role of linguistic levels in modularity) and the concept of software module specification proposed by Parnas [2,3]. We give some properties as a base for our definition of modularity and then identify some types of modules that can be used in connection with a programming language system. After defining modularity we define linguistic levels and their relation to modules. The paper concludes by discussing some pragmatics related to the implementation of modularity.

Although the subject of modularity lends itself to a formal mathematical treatment, we opted, at this stage of our research, for a systematic, but informal presentation of our ideas.

## 2. Basic Concepts

Discussion of the modularity problem is based on an informal statement by Boebert [1], that we will state here as our basic axiom of modularity.

Axiom. Modularity is a property possessed by certain programming units defined at a given linguistic level of programming.

A linguistic level of programming, in the present context, is a particular notation for the expression of computational algorithms that has a syntax and semantics which distinguishes it from other notations. Different linguistic levels are used to express different phases of the development of a programming system. For example, a set of precompiled programs can be joined together for execution by a linking language; the

a program module because it cannot be further combined with other units to form larger modules.

Property 2 - Syntactic non-interference is the property which allows program units to be invoked and/or declared in a program text, written in a given programming language, without requiring any syntactic changes in the program text in which it is being placed.

The linguistic level defined by ALGOL 60, for instance, violates this property. A situation may occur in ALGOL 60 where a clash of names occurs when two procedures are placed in the program as declarations within the same enclosing procedure. Thus the use of nonlocal references in an ALGOL 60 program unit (procedure) violates property 2.

Property 3 - Semantic context-independence is the property which allows a program to have an invariant meaning independent of the location in which it is declared and/or invoked within an algorithm expressed at a given linguistic level.

Suppose that a program unit, defined at a given linguistic level, is specified by two first-order predicate-calculus formulae that represent respectively its input and output assertions in the Floyd sense [12]. If the program unit's specification, as given by these two assertions, is invariant within the program text where it is invoked and/or declared, then we say that the program unit satisfies property 3.

Property 4 - Data generality is the property which allows a program unit to communicate with other program units of the same linguistic level, using arbitrary data structures.

precompiled programs having been written in high-level languages such as ALGOL 60, PL/1 and COBOL. The precompiled programs also may have been described at an early stage of development through a non-executable specification language such as PSL [7] or the more informal language HIPO [11]. Each of these notations, the linking language, the normal programming language, and the program specification language all represent different linguistic programming levels. In the paper we use the concept of linguistic level to help define modularity but our choice of levels is somewhat unconventional. By associating modularity with a given linguistic level it will allow characterization of a module independent of the linguistic level at which it occurs.

DeRemer and Kron [6], in a recent paper, make implicit use of the concept of linguistic levels; in fact, the difference they describe between LPSs (Languages for Programming in the Small) and MILs (Module Interconnection Languages or Languages for Programming in the Large) supports the axiom of modularity, that modularity is closely related to linguistic level.

One's experience and the literature on software lead to a list of properties which when applied to a program unit at a given linguistic level, defines it as a module. The properties follow:

Property 1 - Composition by nesting is the property which allows program units to be invoked and/or declared inside other program units to an arbitrary depth.

The linguistic level defined by FORTRAN for instance, makes no provision for combining separately written FORTRAN programs; a complete FORTRAN program consisting of main program and subprograms cannot serve as

Satisfying this property means that the program units implements Parnas' hiding principle [2]. Through this principle no program unit (programmer) should have any information about the inner workings of the modules with which it communicates. The hiding principle is, of course, a fundamental property required in off-the-shelf programming.

Property 5 - Definitional completeness is the property which forces a program unit defined at a given linguistic level to carry a semantic definition which is comprehensible to a class of users.

A program unit's definition is an inherent part of the concept of a module. In fact, no off-the-shelf programming is viable unless modules carry a complete operational definition,

A program unit must also be correct in order to be called a module. Properties 3 and 5 imply correctness and so this is not explicitly stated as another property.

## 3. The Concept of Modularity

The concept of modularity is stated here as a notion which is independent of particular language features and programming methods. The reasons for this method of presentation appear in the discussion following the definition. Modularity is defined as follows:

Definition A program unit defined at a certain linguistic level is a module, or a linguistic level allows a program unit to exhibit strong modularity (or simply, modularity) if and only if the program unit satisfies properties 1 through 5.

By defining modules in this abstract manner, a principle is provided for the software engineer in designing and constructing

programming tools. When designing languages for constructing programming systems the software engineer can choose the linguistic levels and methods of constructing and manipulating modules using the properties stated in section 2. In general, each linguistic level will contain features for encapsulating program units into modules, a set of control structures and data types.

Several approaches to program development can be discussed abstractly in terms of our definition of modularity with specific reference to properties 1 though 5. For example, top-down programming, bottom-up programming, incremental program verification, and interchangeability and efficiency can be characterized.

Our definition of linguistic level is somewhat different than the ones presented in the literature [8] on top-down structured programming. Top-down structured programming in its "classical" form is usually practised at a single linguistic level that is considered to support modularity, because of property 1. The use of decreasing linguistic levels allows an interesting extension of the concept of top-down programming, since we are able to support this type of programming both within and between levels. A hint of this extension of top-down programming was contained in [9].

Bottom-up programming is characterized by properties 2 to 5 since we require modules to be independent and well-described entities, which allow for the composition of programs from ready-made (off-the-shelf) program units.

Properties 3 and 5 allow for incremental program verification since each module may be proved correct independent of its environment. In fact, if we examine the program as a whole, expressed at a given linguistic

level, the properties of individual modules can be proved correct independently and then the properties of the modules can be abstracted when the whole program is proved correct. Of course, the same remarks apply to the testing of modules.

Properties 2 to 5 allow a module to be replaced by an equivalent one. Furthermore, property 4 (data generality) allows not only the replacement of a module by one that implements a different algorithm on the same data structures but replacement by one that uses the same or a different algorithm on a different data structure. Optimization, becomes a widely applicable technique.

## 4. Levels of Language for Modular Programming

In previous sections there was no attempt to pre-define the number and form of the linguistic levels required for the complete expression of a programming system. Although it is now generally accepted that programming should take place through a spectrum of inter-communicating linguistic levels, which within levels proceed from various stages of specification to various stages of implementation, it is difficult to establish criteria to specify the number of linguistic levels and their respective characteristics. In previous sections only the conditions for a given linguistic level to support modularity have been stated; in this section a specific model is proposed that fixes the number of linguistic levels and tries to characterize modularity in each of these levels. Of course, this is only one possible model which might be proposed but it seems to work well in explaining the role of various programming constructs in modularity.

There are a number of reasons that support our proposed partitioning into three linguistic levels, which is also the partitioning we propose for our programming system for program development [13]. They are:

a) Some reasonable theoretical arguments in favor of similar three-level models [10].

b) Most of the current language features and programming techniques in the literature can be comfortably classified into the three levels.

Modularity is examined at three linguistic levels that are called respectively systems, program and data-abstraction levels. As mentioned before, the reader will note some similarities between these three levels and the three levels called relational, access path and machine levels, that Earley [10] defines for data structures.

In a working modular system the three linguistic levels (the system-level, the program-level and the data-abstraction level) can be visualized in terms of contemporary program structures. At the system-level, modules (which we will call system-modules) are usually pre-compiled programs connected together by a module-interconnection language. The program-level, supports program-modules, which are usually procedures in the ALGOL sense. The data-abstraction level, supports data abstraction modules which are implementations of abstract data types.

The description in the previous paragraph is an analytical presentation of the levels of modularity. Using a top-down approach to synthesis, the three levels relate to each other in the following way: system-modules are specified together with their interconnections;

program-modules are defined so as to implement system-modules (data abstractions are left unspecified); data abstractions are later implemented and they define the programming system's machine level. It is important to emphasize at this point the differences between the control structures used at the various levels. The module interconnection language at the system-level will probably resemble a graph language, describing the connection between the various nodes (system-modules) and the content of each node. At the program-level the control structures will be the standard control structures found in an ALGOL-like language (IFTHENELSE, DOWHILE, CASE, etc.). The data-abstraction level illustrated will call for a module interconnection language similar to the type used at the system-level.

## 5. Conclusions

We feel that most existing and proposed features for supporting the concept of modularity can be satisfactorily examined through the use of our three-level model. Furthermore, we think that our general concept of modularity and the use of the three-level model of linguistic levels, helps classify and clarify specific language features for modular programming.

Although we are considering the three-level model as the basis for our programming system for program development, no engineering decisions about specific system features are discussed in this paper. By specific system features we mean special notation for module specification and programming mechanisms for module encoding. It can be observed that the higher the linguistic level, the closer the specification and the programming mechanisms will be. Several current proposed language features do satisfy our very strict definition of modularity providing that a few changes are introduced. The changes have mostly to do with supporting the

property of data generality. We acknowledge the need for sharing in the design of programming systems and claim that a weak concept of modularity should also be accepted providing that it is reinforced by some additional specification features. We believe that our approach to the modularity issue contributes to the better understanding of the concept and that in particular it helps in designing systems to support modular programming.

References

[1] Dennis, J., "Modularity" in Advanced Course on Software Engineering, Springer Verlag, 1973.

[2] Parnas, D.L., "On the Criteria to be used in Decomposing Systems into Modules", CACM, vol. 15, No. 12, 1972.

[3] Parnas, D.L., "A Technique for the Specification of Software Modules with Examples", CACM, vol. 15, No. 5, 1972.

[4] Liskov, B.H. and Zilles, S.N., "Programming with Abstract Data Types", Proceedings of ACM SIGPLAN Symposium on Very High Level Languages, SIGPLAN Notices, vol. 9, No. 4, 1974.

[5] Liskov, B.H. and Zilles, S.N., "Specification Techniques for Data Abstractions: IEEE Transactions on Software Engineering, vol. 1, No. 1, 1975.

[6] DeRemer, F., Kron, H., "programming-in-the-Large Versus Programming-in-the-Small", Proceedings of the International Conference on Reliable Software, Los Angeles, 1975.

[7] Teichroew, D., Bastarache, M.J., "PSL User's Manual", ISDOS Working Paper No. 98, Department of Industrial Engineering, The University of Michigan, 1975.

[8] McGowan, C.L., Kelly, J.R., "Top Down Structured Programming Techniques", Petrocelli/Charter, New York, 1975.

[9] Mills, H.D., "OS 360 Job Control Language Programming", Classroom notes.

[10] Earley, J., "Relational Level Data Structures for Programming Languages", Acta Informatica 2, 1973.

[11] IBM Report GC 20-1851-1, "HIPO - A Design Aid and Documentation Technique", 1975.

[12] Floyd, R., "Assigning Meaning to Programs", American Mathematical Society, vol. 19, 1967.

[13] Lucena, C., Cowan, D.D., "Toward a Systems Environment for Computer Assisted Programming", Computer Science Department, CS-76-06, 1976.

[14] Cowan, D.D., Lucena, C., Staa, A.v., "On the Concept of Modules in Programming Systems, Computer Science Department, CS-76-05, 1976.