MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Laboratory for Computer Science

Computation Structures Group Memo 166

Practical Benefits of Research in Programming Methodology

by

Barbara Liskov

(Published in the Proceedings of the National Computer Conference, June 1978, pp 666-667.)

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under contract N00014-75-C-0661, and in part by the National Science Foundation under grant MCS 74-21892 A01.

August 1978

PRACTICAL BENEFITS OF RESEARCH IN PROGRAMMING METHODOLOGY—Barbara Liskov

In the past few years, considerable progress has been made in the area of programming methodology. Although all research in this area is interrelated, two main research directions can be distinguished. One direction is the study of software system structure, in particular study of desirable kinds of modules and module interconnections. The other direction is the study of the process of developing correct software having such desirable structure.

Study of software system structure has been particularly effective within the framework of research on programming languages, especially the languages CLU[1] and Alphard.[2] This language work has succeeded in identifying new kinds of modules, and has provided precise rules governing the implementation and use of such modules. Each kind of module supports a kind of abstraction found to be useful in constructing software. The most important new kind of module is that supporting a *data abstraction*; however, there are other kinds of modules, and in addition rules governing the interaction of modules. These latter rules constrain and reduce the interconnections among modules.

Earlier work in structured programming[3] and stepwise refinement[4] made evident the advantages of top down development of software. Recent work has elaborated the top down development process, taking into account the work on software system structure discussed above, and clarifying the way that top down development proceeds. The most important contribution has been the recognition of the role played by *program specifications*. A program specification is a description of the behavior of a module, the behavior that will be depended on by any user, and that must be provided by any implementation. At any stage of design, the goal is to identify lower level abstractions useful in implementing the current level. As these abstractions are identified, their behavior is specified. The specification is given in advance of the implementation, and it provides a complete description of the interface of the module that will later implement the abstraction. The presence of the specification permits the question of how to implement the lower level modules to be deferred until a later stage of design.

As programming methodology has become better understood, there has been increasing interest in defining formal methods to support it. In particular, there has been much recent research in formal specification techniques,[5,6] which permit the specifications discussed above to be expressed in a formal language, i.e., one with a well-defined and unambiguous syntax and semantics. The advantages of such formal specifications are twofold: they are more precise and concise than informal specifications, and therefore may serve better the role of interface descriptions described above, and, in addition, given formal specifications, a formal proof that a module's implementation satisfies its specification is possible. However, formal specifications are more difficult to write than informal ones, and our current understanding of specification and verification techniques is insufficient to permit all useful abstractions to be described.

It is my belief that the present and near future construction of software systems can best be helped by popularizing the methodology. This can be done in a way that relies neither on a particular language, nor on the as yet incompletely understood specification and verification techniques. Instead, the following two ideas must be made clear to programmers:

1. What constitutes good modularity.
2. What constitutes good design practice.

Rules about good modularity are best explained by developing conventions, or better yet preprocessors, for existing languages in actual use; such conventions would permit a limited use of data abstractions and would prohibit current bad practices (such as non-local use of data). By expressing the rules in this way, they are explained in terms the programmers can understand, and furthermore, a tool is provided that helps in the development of a well-structured system. Good design practice then consists of top down decomposition into the kinds of modules that the programmers already understand, with emphasis on the role of (informal) specifications in the process, and especially on the necessity of specifications being given in advance of implementation. To aid programmers in understanding what specifications are, it is helpful to establish a specification standard which describes the kind of information that should be included in the specification, and gives a format for expressing that information. However, it is too early to require that specifications be given in a formal language. Formal methods in system design are not yet ready for practical use, but I believe use of the methods in an informal way can have considerable practical benefit.

REFERENCES

1. Liskov, B., A. Snyder, R. Atkinson, and C. Schaffert, "Abstraction Mechanisms in CLU," *Comm. of the ACM 20, 8*, August 1977, pp. 564-576.
2. Wulf, W., R. London, and M. Shaw, "An Introduction to the Construction and Verification of Alphard Programs," *IEEE Trans. on Software Engineering SE-2*, 1976, pp. 253-264.
3. Dijkstra, E. W., "Notes on Structured Programming," *Structured Programming, A.P.I.C. Studies in Data Processing 8*, Academic Press, New York, 1972, pp. 1-81.
4. Wirth, N., "Program Development by Stepwise Refinement," *Comm. of the ACM 14, 4*, April 1971, pp. 221-227.
5. Guttag, J., E. Horowitz, and D. Musser, *Abstract Data Types and Software Validation*, Report ISI/RR-76-48, Information Sciences Institute, University of Southern California, Marina del Rey, August 1976.
6. Liskov, B. and V. Berzins, *An Appraisal of Program Specifications*, Computation Structures Group Memo 141-1, Laboratory for Computer Science, M.I.T., Cambridge, Mass., April 1977.