REFERENCES


INITIAL EXPERIENCE WITH A METHODOLOGY FOR CORRECT PROGRAM DESIGN—F. T. Baker

In the forthcoming book, Structured Programming: Theory and Practice, the authors describe three techniques which have been incorporated into a Methodology for achieving correct designs. These are:

1. A view of program correctness as a demonstration of a correspondence between the function of a program design (i.e., the set of ordered pairs corresponding to input states and output states) and the function required by its specification. This approach, when used with stepwise refinement, permits selective and incremental correctness proofs to be carried out, since it incorporates a procedure for verifying the correctness of the expansion of a specification into any one of a basic set of control structures. (The expansion of a specification at any level into a program design can thus be verified, contingent on the correctness of lower-level specifications and their expansions.) Furthermore, proofs can be carried out with varying levels of rigor, ranging from a set of questions the designer may use to validate an expansion to a formal demonstration recorded in a precise manner.

2. A method for incorporating specifications into program designs to support correctness demonstrations when desirable. Each specification (either initial, or those generated in the expansion process) is retained as a comment (logical commentary) directly associated with the control structure which refines it.

3. A design language (Process Design Language) to assist in the design process and to record the history of a design. PDL includes a standard “outer syntax” of essential control and data structures, and encourages development of an “inner syntax” appropriate to each design environment.

Figure 1 is an example of a design for a program which is to save the maximum value occurring in an input sequence. In that design, each of the paired brackets encloses logical commentary which is a functional specification for the control structure which follows it (sequence, ifthenelse or while do). For each of the structures the appropriate proof procedure can be carried out to demonstrate the correspondence between its specification function and its program function. Furthermore, this can be done prior to the completion of the expansion process, or even on selected portions of the design.

The methodology is primarily aimed at the detailed design of a program. It covers the period between the formulation of an effective system design, and the translation from the design language into an implementation language. It was developed to introduce more precision into the design process and to encourage more consistent expression of designs. Whether or not formal correctness demonstrations are carried out, the stress on viewing programs as functions, developed from specifications through a rigorous refinement process, should help achieve the goal.

Experience to date suggests that the methodology is capable of being practiced in the application development environment. We believe that the control structures inherent in PDL are sufficient to support all levels of the design process, from system and module specification down to precise algorithms. The invention and use of logical commentary direct attention to specification and program functions, as they were intended to do. In particular, they encourage the designer to specify and deal with boundary conditions and anomalies which frequently are poorly attended to and which sometimes lead to difficult-to-find errors. Finally, the view that each program should be designed as if it is to be proved correct, means that even if correctness demonstrations are not formally carried out, the program has a greater likelihood of properly embodying its specification.

Experience to date also indicates several areas where more work is needed. The nature of the expansion process, and the desire to record the design history, mean that much copying is done as the design is developed. An interactive support tool appears useful to assist designers in this expansion and recording. The data structures in PDL (stacks, queues, sequences and sets) are useful ones, but better proof techniques to validate operations with them must be developed. There is a notational problem inherent in specifying functions precisely, particularly in nonmathematical envi-