SHAW81

Shaw, M.; Almes, G. T.; Newcomer, J.M.; Reid, B. and Wulf, W.A.; A Comparison of Programming Languages for Software Engineering; Software Practice & Experience 11, 1 (January 1981) pp. 1-52

Abstraction is not only useful in describing data and programs, it can also be used in the comparison of programming languages. An abstraction of a language, termed 'language core', is a subset of the language that captures the essential properties of the language and the intent of the language designers. By comparing the language cores instead of every feature of the languages, we can focus our comparison on the features that capture the design philosophies and not be distracted by the less important aspects of the languages. The authors devise the 'language core' technique based on some closely-related premises about programming languages. One of them is that the designers of each language has a strong image of how programs are, or ought to be written. This image is strongly and consistently reflected throughout the language and is a major factor in determining how the language will actually be used. The 'language core' will then capture this image and enables one to discuss the extent to which each language really does support its image. steps. 'Cores' are firstly defined for those languages under investigation. Each definition includes the designers' philosophy, the syntactic properaties and the semantic properties of the language. Having done that, we shall compare the language philosophies. The issues to consider are the designers' image of the problem domain (whether it be numeric computations, data processing or some other applications), selection vs. synthesis (whether power is achieved by having a menu of options or fewer options which can be orthogonally composed), and the degree of permissiveness (the degree to which each enforces, encourages, permits or prevents various programming practices). The next step is to compare the effects the languages have on programming loops, procedures or small programs. It includes issues on uniformity, expressiveness, clarity and size of a language. Other issues considered are the abstraction facilities, aliasing consideration, data typing, non-scalar data organizations, data definition and representation, flow of control and efficiency concerns. Finally we shall compare the effects the languages have on prgramming 'in the large', that is, the composition of program modules into whole systems. It addresses on how well the language support functional and data-oriented decomposition and abstraction, the independence of isolated program segments, assembly of a system from components, independent compilation of modules, checking of module linkages and system maintenance and enhancement. Four programming languages (Fortran, Cobol, Jovial and Ironman, the predecessor of Ada) are compared in this paper using the 'language core' technique. Fortran is dominated by its orientation to scientific, numeric computing. It takes the selection approach to data, but a synthetic approach to control constructs. The syntax is 'flat', with few nestings of syntactic constructs. A statement is conceptually a single line of text and grouping of statements is only possible by using the subprogram facility (SUBROUTINE and FUNCTIONS). Data abstraction is poor; the only way for sharing data among subprograms is the COMMON block. The design of Cobol was motivated by the need to write non-numeric file-oriented programs. It is also preoccupied by readability (by non-programmers) that it mimics English as much as possible. One of its striking features is the separation of algorithms, data definitions and machine characteristics. This forces the programmers to structure their data efficiently, and allows portability in moving from one machine to another. It derives its power from a large collection of operations (36 verbs and 262 keywords), especially those record-oriented operations. Jovial is designed to implement large modular command and control systems. It achieves its goal by having low-level control, low-level data representation and the sharing of definitions among the various modules of large systems. Its ITEM declaration allows one to declare a variable of a specific size. It supports separate compilation of modules and is ideal for system programming. Ironman is designed for the application to embedded systems and their maintainability. It is intended to have the simplicity of language, rich data structuring, good data and type encapsulation, efficient accessibility to underlying hardware and conscious restraint. It prohibits aliasing, enforces equivalent data typing and strives for syntactic and semantic uniformity. To conclude, this paper has presented and illustrated how programming languages should be compared using their underlying design philosophies. The to language-comparison methodology.