Cognitive Dimensions and End-User Debugging

Position Statement for the Cognitive Dimensions of Notations Workshop

Susan Wiedenbeck
College of IST, Drexel University
susan.wiedenbeck@cis.drexel.edu

Previous Work Using Cognitive Dimensions

My previous work using cognitive dimensions has centered on comparisons of programming paradigms, in particular the object-oriented (OO) and procedural paradigms. The goal of the research of my colleagues and I (Wiedenbeck and Ramalingam, 1999; Wiedenbeck, Ramalingam, Sarasamma, and Corritore, 1999; Corritore and Wiedenbeck, 1999; Burkhardt, Détienne, and Wiedenbeck, 2000) was to identify how aspects of the notation of the OO and procedural paradigms facilitated or failed to facilitate program comprehension. Comparisons across programming paradigms are difficult to do, but nevertheless they are important for understanding how different styles of programming affect the learning of novices and the performance of experts.

We believe that the answer to the question of how paradigms differ with respect to comprehension depends on the notation and the user's comprehension task. The importance of notations in comparing comprehension across programming paradigms is derived from Gilmore and Green (1984) who introduced the "match-mismatch" conjecture. This conjecture suggests that in terms of comprehension there is unlikely to be any universally superior programming notation, but that a given notation may aid comprehension of certain types of information by highlighting that information in some way in the program text. The visibility and parsability of meaningful structures in code has been given the name role expressiveness, one of the cognitive dimensions (Green, 1989). The role expressiveness of a language depends in part on the language itself and in part on the knowledge of the user. The language itself provides or fails to provide perceptual cues which underline important attributes of the program. This perceptual coding can be highly effective, especially if it is redundant. However, the ability to make use of cues in a program depends on the user's experience with programming and the notation, as shown by Davies (Davies, 1990).

We analyzed different comprehension tasks and reasoned about how the role expressiveness of the OO and procedural paradigms affect those tasks. We argued that the OO style highlights information about problem objects and their relationships through the notation of objects, classes, and inheritance. Although the details of the notation differ from one OO language to another, it appeared that all existing OO languages highlight this sort of information. With respect to information about program function, we made a distinction between function at a local level and at a global level. At the local level of an individual class the OO style appeared to highlight function through the class interface in which function declarations are bundled together with the data on which they operate. In the class notation the programmer can easily and quickly see what the basic functionality of a class is, without examining the detail of the class implementation. However, at the global level of overall program function it appeared that function is not highlighted by the notation and may be difficult to derive because the

functionality is dispersed across small interacting component objects. The result of making objects local may be that, as program size grows and more classes and objects are introduced, the programmer has more difficulty getting a high-level view of the program's functionality. With respect to control flow, we argued that, within a single program module, procedural and OO programs would not differ because in both styles local flow of control involves sequence, branching, and iteration. On the other hand, between-module control flow might be clearer in a procedural program because a procedural program is normally based on a hierarchy in which a top-level function calls lower-level functions to carry out smaller parts of the overall task. It is relatively easy to determine where the top is and to understand the calls through successive layers of decomposition. In OO programs there is no top-level, but rather parts of a task are distributed across objects which pass messages to other objects to act on their behalf. We conjectured non-hierarchical interaction of objects may make it more difficult for the novice programmer to understand and form a mental representation of control flow in an OO program. This again is a problem that would grow with program size and larger numbers of interacting objects. With respect to knowledge about data flow, we did not expect to see systematic comprehension differences between procedural and OO style programs. Neither notation appeared to highlight data flow in any special way.

The results of our studies of programmers were somewhat mixed over several experiments. The role expressiveness of the OO notation did generally facilitate comprehension about problem objects and their relationships. We also found that the perceptual saliency of the class interface facilitated the understanding of the class's function. For small novice programs that had only a few classes, the comprehension advantage of the OO paradigm was also reflected in the overall comprehension of the program, but this was not always the case in more complex programs. We found an advantage in comprehension of control flow to the procedural paradigm and no differences between the OO and procedural paradigms in data flow.

Future Work

I am interested in applying cognitive dimensions to end-user languages and environments. End-users are a large and increasingly important segment of programmers. Because of their low experience with programming, having programming languages and tools that supports their cognition in programming is all the more important. It also should be acknowledged explicitly that today complex programming environments surround programming languages and are an integral part of programming. In designing or evaluating programming tools the environment as well as the language should be subject to cognitive dimensions analysis. An area of programming that has been little investigated using the cognitive dimensions framework is end user debugging, such as debugging of spreadsheet formulas or of document templates in a word processing program. I would like to carry out a cognitive dimensions analysis on debugging in an end-user environment with the goal of determining the match between the end users and their tools.

References

Burkhardt, J-M., Détienne, F., and Wiedenbeck, S. Object-oriented program comprehension: effect of expertise, task and phase. *Empirical Software Engineering*, 7, 115-156.

Corritore, C.L. and Wiedenbeck, S. (1999). Mental representations of expert procedural and object-oriented programmers in a software maintenance task. *International Journal of Human-Computer Studies*, 50, 61-83.

Davies, S. P. (1990). The nature and development of programming plans. *International Journal of Man-Machine Studies*, 32, 461-481.

Gilmore, D. J. and Green, T. R. G. (19840. Comprehension and recall of miniature programs. *International Journal of Man-Machine Studies*, 21, 31-48.

Green, T. R. G. 91989). The cognitive dimensions of notations, in: Sutcliffe, A. and Macaulay, L. (Eds.), *People and Computers V* (Cambridge University Press, Cambridge, 443-460.

Wiedenbeck, S., Ramalingam, V., Sarasamma, S., and Corritore, C.L. (1999). A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers*, 11, 255-282.

Wiedenbeck, S. and Ramalingam, V. (1999). Novice comprehension of small programs written in the procedural and object-oriented styles. *International Journal of Human-Computer Studies*, 51, 71-87.