Role of Software Readability on Software Development Cost

Emilio Collar, Jr.
Western Connecticut State University
Ancell School of Business
181 White Street
Danbury, CT 06810
collare@wcsu.edu

Ricardo Valerdi
Massachusetts Institute of Technology
Lean Aerospace Initiative
77 Vassar Street, Bldg #41, Rm #205
Cambridge , MA 02139
rvalerdi@MIT.EDU

ABSTRACT

This paper explores the role of software readability on software development cost. We argue that the up front cost of incorporating software readability pays off handsomely at later stages in the life cycle, especially at the maintenance phase which is where most of the life cycle cost of software is expended. Our analysis of different software development activities shows that software readability has a global effect on software development cost and is independent of software size (i.e., KSLOC). Moreover, we explore the links between software readability and programming domain knowledge.

This paper is organized into the following three sections: overview of software readability, how readability fits into the context of software cost estimation, and how results can lead to the development of a new COCOMO cost driver focused on software readability.

Background on Software Readability

One of the major challenges faced by the information technology industry is the high cost of software development, especially during the maintenance phase. The high cost of software maintenance has been linked to the difficulty of reading and understanding programming code, particularly code written by someone else (Deimel 1985; Pennington 1987; von Mayrhauser and Vans 1995b). A recent study on software readability draws attention to programming code readability, particularly textbase readability, as a contributing factor in programming code comprehension (Collar 2005a; Collar 2005b).

Programming code comprehension is defined as the process by which a programmer makes sense of programming code (Woods and Yang 1996). This paper investigates the effects of propositional organization of programming code on the code's *readability*, i.e., ease of comprehension. In this paper, we propose that programming code is comprehended through its propositional structure. Propositions are built from propositional elements (predicates and arguments) in the code to form a *textbase*, the ordered list of propositions that represents the underlying propositional organization of the code. *Textbase readability*, then, is theorized as the effectiveness of the propositional organization of programming code for supporting textbase comprehension. In this view, what determines code readability is not simply the linear presentation of the code. What we propose to be crucial for programming code comprehension is the code's propositional structure, through which the reader constructs his or her understanding of the semantic relations between propositional elements of the code. Comprehension at the textbase level relies on the effectiveness of the propositional organization in supporting

the reader's ability to construct meaning from the propositions in the text of the programming code (i.e., its textbase).

Role of Readability on Software Life Cycle Costs

The problem of high cost of maintenance has been highlighted by many researchers and practitioners (Boehm 1976; Elshoff and Marcotty 1982; Agresti 1982a; Agresti 1982b; Standish 1984; Robson, Bennett et al. 1991; Shaft and Vessey 1995). The majority of the time and effort expended during maintenance-related activities is code reading (Deimel 1985; Rugaber 2000). Rugaber, for example, has observed that "[a]s currently practiced, program understanding [e.g., for maintenance] consists mainly of code reading" (Rugaber 2000).

Maintenance programmers' tasks are analogous to those of archeologists who "…investigate some situation, trying to understand what they looking at and how it all fits together. To do this, they must be careful to preserve the artifacts they find and respect and understand the cultural forces that produced them" (Hunt and Thomas 2002).

We propose that the effects of software readability on software costs are bipolar. That is, our findings demonstrate that improved readability leads to less time spent reading code. Less time reading code, consequently, results in lower costs throughout each phase of the life cycle. Inversely, reduced readability leads to more time spent reading code resulting in higher cost. Results from a study of software readability for a procedural language (i.e., Visual Basic.NET) are presented to show how programming code can be analyzed in terms of readability. Programming code readability has significant effects on the time needed to comprehend code during the software maintenance phase; the phase that consumes the majority of the cost in the SDLC.

The analysis of programming code structure shows that the *# of repeated arguments* is the most statistically significant predictor of improved software readability. Other relevant parameters studied include *# of new arguments* and *propositional density*. These predictors introduce new questions about the effects of software readability for functional languages (i.e., Java) where the influences of readability may be different. In the end, this work serves as a stepping stone for developing a new COCOMO II parameter that addresses software readability and enables the software cost estimation community quantify code readability effects on cost.

Human Aspects Related to Readability in COCOMO II

Code readability is undeniably related to programming domain knowledge which, in the COCOMO II post-architecture mode, is operationalized in three cost drivers (Boehm, Abts et al. 2000): Platform Experience (PEXP), Applications Experience (AEXP), and programming Language Experience (LEXP). We speculate that these constructs have a moderating effect to the relationship between readability and comprehension.

A fourth cost driver, Required Reusability (RUSE), also plays a role in software readability. Code that is easier to read should be easier to modify. Readability research could intersect RUSE research when we examine the interaction between the readability

and modifiability of code. Improving the readability of code may increase the chances of the code being reused. The tipping point between revising and redeveloping may shift as the readability of existing or available code increases.

References

Agresti, W. W. (1982a). "Measuring program maintainability." Journal of Systems Management **33**(3): 26-29.

Agresti, W. W. (1982b). "Managing program maintenance." Journal of Systems Management **33**(2): 34-37.

Boehm, B., C. Abts, et al. (2000). Software Cost Estimation with COCOMO II. New York, Prentice Hall.

Boehm, B. W. (1976). "Software Engineering." IEEE Transactions on Software Engineering **C-25**(12).

Chall, J. S. and E. Dale (1995a). Readability Revisited: The New Dale-Chall Readability Formula. Cambridge, MA, Brookline Books.

Collar, J., E (2005a). An Investigation of Programming Code Readability Based on a Cognitive Readability Model - Volume I: Manuscript. Leeds School of Business. Boulder, CO, University of Colorado at Boulder**:** 403.

Collar, J., E (2005b). An Investigation of Programming Code Readability Based on a Cognitive Readability Model - Volume II: Appendices. Leeds School of Business. Boulder, CO, University of Colorado at Boulder**:** 371.

Deimel, L. E. (1985). "The Uses of Program Reading." ACM SIGCSE Bulletin **17**(2): 5-14.

Elshoff, J. L. and M. Marcotty (1982). "Improving computer program readability to aid modification." Communications of the ACM **26**(8): 512-521.

Hunt, A. and D. Thomas (2002). "Software Archeology." IEEE Software **19**(2): 20-22.

Klare, G. R. (1974-1975). "Assessing readability." Reading Research Quarterly **10**: 62-102.

Pennington, N. (1987). "Stimulus structures and mental representations in expert comprehension of computer programs." Cognitive Psychology **19**: 295-341.

Robson, D. J., K. H. Bennett, et al. (1991). "Approaches to program comprehension." The Journal of Systems Software **14**: 79-84.

Rugaber, S. (2000). "The use of domain knowledge in program understanding." Annals of Software Engineering **9**: 143-192.

Shaft, T. and I. Vessey (1995). "Research Report: The relevance of application domain knowledge - the case of computer program comprehension." Information Systems Research **6**(3): 286-299.

Standish, T. A. (1984). "An essay on software reuse." IEEE Transactions on Software Engineering **SE-10**(5): 494-497.

von Mayrhauser, A. and A. M. Vans (1995b). Program Understanding: Models and Experiments. Advances in Computers. M. C. Yovits and M. Zelkowitz. San Diego, CA, Academic Press. **40:** 2-36.

Woods, S. and Q. Yang (1996). The Program Understanding Problem: Analysis and a Heuristic Approach. International Conference on Software Engineering (ICSE), IEEE.