

# Cognitive Perspectives on the Role of Naming in Computer Programs

Ben Liblit<sup>1</sup>, Andrew Beigel<sup>2</sup>, and Eve Sweetser<sup>3</sup>

<sup>1</sup> University of Wisconsin, Madison WI 53706, USA, <liblit@cs.wisc.edu>

<sup>2</sup> Microsoft Research, Redmond WA 98052, USA, <andrew.beigel@microsoft.com>

<sup>3</sup> University of California, Berkeley CA 94720, USA, <sweetser@berkeley.edu>

**Abstract.** Programming a computer is a complex, cognitively rich process. This paper examines ways in which human cognition is reflected in the text of computer programs. We concentrate on *naming*: the assignment of identifying labels to programmatic constructs. Naming is arbitrary, yet programmers do not select names arbitrarily. Rather, programmers choose and use names in regular, systematic ways that reflect deep cognitive and linguistic influences. This, in turn, allows names to carry semantic cues that aid in program understanding and support the larger software development process.

## 1 Introduction

Programming languages are designed to be precise, mathematical, unambiguous, and interpretable by machines. Natural languages, on the other hand, evolved over long periods of time to be interpreted by humans using a rich suite of cognitive processes. It would be wrong, however, to dismiss programming languages as wholly *unnatural*. Programs are written by humans and do not proceed directly from keyboard to compiler; throughout development, humans read and modify the code that they and others have produced. Code that cannot be understood by a human is of little value, and human programmers are well aware that the code artifacts they produce must be readable as well as runnable. Thus, human cognition is reflected in the text of computer programs. To the extent that source code is expressed textually, code reflects linguistic cognition especially strongly. The purpose of this paper is to examine the impact of human cognition, and especially human language, on the text that forms computer software.

Among the many facets of software construction, this paper closely examines *naming*: the act of assigning identifying labels to programmatic constructs. This intensive use of invented words is a major deviation from natural language dialog, in which participants share a large, mutually understood, but relatively fixed lexicon. To a compiler, the choice of names is devoid of semantic meaning. Compilers have no understanding of natural language, so “blue” and “Sgu9Asd1M” are equally opaque, arbitrary sequences of letters. Provided that the programmer uses consistent spelling and capitalization, any name is as good as any other. Yet precisely because names are arbitrary, programmers have great freedom to select names that promote code understanding. We will examine several ways in which this is done, relating standard programming practice to modern theories of human language and cognition.<sup>4</sup>

<sup>4</sup> For a much more in-depth treatise on identifiers, see “The New C Standard” [14].



























