Communicability of Natural Language in Software Representations

Pankaj Kamthan
Concordia University, Canada

INTRODUCTION

In software engineering, separating problem and solution-level concerns and analyzing each of them in an abstract manner are established principles (Ghezzi, Jazayeri, & Mandrioli, 2003). A software representation, for instance, a model or a specification, is a product of such an analysis. These software representations can vary across a formality spectrum: informal (natural language), semi-formal (mathematics-based syntax), or formal (mathematics-based syntax and semantics).

As software representations become pervasive in software process environments, the issue of their communicative efficacy arises. Our interest here is in software representations that make use of natural language and their communicability to their stakeholders in doing so. In this article, we take the position that if one cannot communicate well in a natural language, then one cannot communicate via other, more formal, means.

The rest of the article is organized as follows. We first outline the background necessary for later discussion. This is followed by the proposal for a framework for communicability software representations that are created early in the software process and the role of natural language in them. We then illustrate that in software representations expressed in certain specific nonnatural languages. Next, challenges and directions for future research are outlined and, finally, concluding remarks are given.

BACKGROUND

The use of natural language in software is ubiquitous. In spite of its well-known shortcomings that are primarily related to the potential for ambiguity or limitations of automatic verifiability, surveys have shown (Berry & Kamsties, 2005) that the use of natural language (such as English) continues to play an important role in software representations. Agile software process environments such as Extreme Programming (XP) (Beck & Andres, 2005) tend to accentuate the use of "lightweight" models such as user stories (for input to software requirements and test cases) (Alexander & Maiden, 2004) that often depend exclusively on the use of natural language. In Literate Programming, natural language prose is used in documenting (explaining) the source code as if it were the work of literature to make it more readable to both humans as well as to machines.

In recent years, there has been an increasing emphasis on the quality of software representations that are created "early" from the point of view of control and prevention of problems that can propagate into later stages. The significance of expressing software requirements in natural language that minimizes ambiguity is emphasized in Berry and Kamsties (2005) whereas a model to study their quality is presented in (Fabbrini, Fusani, Gervasi, Gnesi, & Ruggieri, 1998). However, these efforts do not systematically address the issue of communicability in software representations.

USE OF NATURAL LANGUAGE IN SOFTWARE REPRESENTATIONS

Our understanding of communicability of a software representation is based on the following interrelated hypotheses:

- **Hypothesis 1.** Readability is a prerequisite to communicability, which in turn is a prelude to comprehensibility. The basis for this hypothesis is that if a user has difficulty accessing or deciphering a certain message, then that user will not be able to understand it in part or in its entirety either.
- **Hypothesis 2.** The comprehension of a given software representation in a nonnatural language takes place only when it is first internally translated into the user's natural language of choice. The basis for this hypothesis is that the mode of internalization of some knowledge is the conversion of explicit knowledge into the tacit knowledge. The natural language acts as a "proxy" in this internalization of knowledge inherent in a semiformal/formal software representation.

Using these as the basis, the discussion of software representations that follows rests on the framework given in Table 1.

Table 1 provides necessary but not sufficient conditions for communicability. We now discuss the elements of the framework in detail.

Table 1. A high-level view of communicability of a software representation making use of natural language

Entity	Natural Language Use in Software Representation	
Pragmatic goal	Communicability	Feasibility
External quality attributes	Readability, other	
Internal quality attributes	Secondary notation (labeling, typography)	

Communicability and Semiotics

Semiotics (Nöth, 1990) is concerned with the use of symbols to convey knowledge. From a semiotics' perspective, a representation can be viewed on three interrelated levels: syntactic, semantic, and pragmatic. Our concern here is the pragmatic level, which is the practical knowledge needed to use a language for communicative purposes.

Readability

For our purposes, readability (legibility or clarity) is the ease with which a stakeholder can interpret a piece of text character by character. In general, readability applies to both textual statements and nontextual constructs (say, a histogram), but we shall restrict ourselves to the former.

Secondary Notation

The *secondary notation* (Petre, 1995) is one of the cognitive dimensions and, in our context, defined as the stylistic use of the primary notation (that is, the normative syntax) for perceptual cues to clarify information or to give hints to the stakeholder. The nonmutually exclusive natural language secondary elements that affect the communicability of artifacts are labeling and typography.

- **Labeling.** Labels are comprised of names and, in general, names are not useful by themselves (Laitinen, 1996). For example, names such as `iIndex` for an integer variable reflect their type rather than their purpose or role. Use of the terminology of the application domain in text labels and metaphors (Boyd, 1999) makes it particularly easier for nontechnical stakeholders or users new to language extension to become familiar with the representation. Furthermore, these labels will be more readable and reduce possibilities of misinterpretations if they follow a *natural naming* scheme. Natural naming (Keller, 1990) is a technique initially used in source code contexts that encourages the use of names that consist of one or more full words of the natural language for program elements in preference to acronyms or abbreviations. For example, `QueueManager` is a combination of two real-world metaphors placed into a natural naming scheme.
- **Typography.** The positioning (layout, justification, space between words and lines) of text impacts readability in any document context. Of special concern in our case are the choice and the sequence of characters in the use of text that can affect readability. For example, the characters in a name like `O0lI1` are hard to distinguish and therefore difficult to read. The choice of fonts used for labeling depends on a variety of factors (serif/sans serif, kerning, font size, and so forth) that are important for legibility. Color can be used as an emphasis indicator and for discriminability in text. For example, by associating different colors with text, a stakeholder can be informed of the semantic similarity and differences between operations and attributes in two object classes.

Feasibility

By acknowledging that there are time, effort, and budgetary constraints on producing a software representation, we include the notion of feasibility as an all-encompassing factor to make the framework practical. There are well-known techniques such as Analytical Hierarchy Process (AHP) and Quality Function Deployment (QFD) for carrying out feasibility analysis, and further discussion of this aspect is beyond the scope of this article. Any feasibility analysis, however, also needs to be in agreement with the organizational emphasis on decision support for software engineering in general.

Use of Natural Language in UML

The Unified Modeling Language (UML) (Booch, Jacobson, & Rumbaugh, 2005) is a standard language for modeling the structure and behavior of object-oriented software. The issue of communicability of UML models has been addressed in (Kamthan, 2006).