Studying Work Language As An Aid In Evolutionary Design Process

Barbara Katzenberg, Peter Piela

EDRC 05-59-92

Studying work language as an aid in evolutionary design processes: The naming problem

Forthcoming in
Proceedings of the Participatory Design Conference, 1992

Barbara Katzenberg
School of Education
Stanford University
katzenberg@cmu.edu

Peter Piela
Engineering Design Research Center
Carnegie Mellon University
piela@cmu.edu

Abstract

The naming problem for computer interfaces is one of choosing verbal labels to refer to meanings in a way that people recognize them. Naming in interfaces has been extensively studied by psychological and human factors researchers, however the studies have focused on the properties of names rather than examining how names are interpreted by people in different situations. We employ the pragmatic Principles of Contrast and Conventionality (Clark, 1987, 1990) as a framework for defining what it means for a name to be good, and propose a method for making naming decisions based on linguistic and ethnographic analysis. We present two case studies drawn from a project in which a collaborative group of users and developers have been developing a new technology for equational simulation.

Introduction: Psychological and human factors research on naming

The naming problem is one of choosing verbal labels to refer to meanings in a way that people recognize them. Naming in computer systems has been extensively studied by psychologists and human factors researchers. (cf. Black & Moran, 1982; Carrol, 1985; Grudin & Barnard, 1984; Landauer, et. al, 1983). The major thrust of this work has been to isolate properties of names that lead to better performance, operationalized in terms of the time it takes a novice user to learn a given set of names so that they can be used with minimal errors. Typically, the researcher devises a task that requires a small number of operations and compares the subjects' performances when the operations are named according to the properties of interest.

The Grudin and Barnard studies are representative of this work (1984). In their studies, each experiment involves presenting a novice computer user with a simplified text editing environment in which the set of all command names (the nameset) were derived or chosen using the same scheme.

Some of these namesets were chosen by the researchers, and others derived using a rule. Specific names were chosen to have a precise semantic relationship with the commands they labeled (e.g., "delete" to name the operation that removed characters and words from a line of text). Pseudowords were chosen to be pronounceable but meaningless (e.g., "ragole") and unrelated names were chosen because they had meanings, but ones unrelated to the underlying text-editing commands (e.g., "parole"). The names in the abbreviation and consonant string namesets were derived; the former derived using a rule for abbreviating the specific names (e.g., "dlt" for "delete"), and the latter derived by the rule of any three consonants not included in specific names (e.g., "fnm" for the delete command).

The subjects' performance was gauged on a simple text-editing task in terms of their speed, errors, use of a Help feature, and memory for command names over three separate sessions. The major finding of these studies was

that the specific names like "delete" led to better performance. However, as pointed out by Landauer and Galotti (1984, p. 427), while the effect was significant, no objective procedure was offered for finding specific words, so the work could not be applied directly to designing namesets. Our work is aimed at addressing this deficiency.

We approach the naming problem from a linguistic and ethnographic perspective. From the linguistic perspective, we are taking a *pragmatic* rather than a *semantic* approach to determining which names are good. The semantic approach is focused on the aspects of language where meaning is invariant across situations. It is based on the assumption that one meaning can have only one name. The pragmatic approach is focused on the situation of use, and while retaining the premise of one name for one meaning, this is presumed to stay constant only within one language use situation. For example, one meaning can have different names among groups that have their own *dialects*. (Clark, 1990 p. 421); and conversely the same name can be used by different groups to denote different meanings, or even within one group to denote different meanings in different situations.

Our experience with computer systems is that more than one group is involved in their development and use, so not only must dialect differences be considered, but so must the differential interpretation of names that are shared among groups. Therefore naming is not a matter of finding *the* specific name, (as suggested by the psychological approach) but rather it is one of finding a set of possible names and choosing the most appropriate among them. Our approach to finding appropriate names is through ethnographic study with an analytic focus on work language.

The remainder of the paper is organized as follows. First, we introduce the principles of Contrast and Conventionality, which are pragmatic constraints on how people acquire and use language, and discuss their relevance to the naming problem. In the second section, we review one experiment in which a method was tested for finding conventional names that could be applied to computer systems. Third, we introduce work language analysis as a way of determining what is conventional, and propose a method for applying it to the naming problem. Fourth we present case studies to demonstrate how we applied this method to two naming decisions. Finally we will summarize and outline areas for further study.

The Principles of Contrast and Conventionality

The concept of pragmatics is that successful communication depends upon assumptions made by language users about what others know and intend. Clark posits two principles that people implicitly employ in making such judgments—the Principles of Contrast and Conventionality. The Principle of Contrast states that any difference in *form* in language marks a difference in meaning. Its corollary is the Principle of Conventionality, which states that for certain meanings, there is a conventional form that speakers expect to be used in the language community (Clark, 1987 p.2). By these principles, using a name that contrasts with the conventional one is taken as an intent to convey a meaning that is different from the conventional meaning. However, what is considered conventional can vary across situations according to dialect, level of formality, or such denotational characteristics as specificity or category level.

Using Contrast and Conventionality as a framework, good names at minimum must not violate the users' expectations. They should either denote conventional meanings with the conventional form, or they should advertise the arrival of new meanings via the display of a new form. However, we propose a definition of "goodness" with two additional features. First, we propose that a name be considered good when it is the most general conventional name available that still conveys the necessary meaning. By general, we mean that it is interpretable by as wide a community of users as is necessary. In addition we propose an additional restriction that the name chosen have a unique meaning across the set of situations in which people will be using the names.

To make this framework useful, however, a way of establishing when a name is conventional, when it is it is being used uniquely, and when it is general is needed.

Using language conventions for computer system names

One study aimed at finding names that suited potential users' expectations was performed by Landauer, Galotti, and Hartwell (1983). The researchers gave typists who had never used computers a paper manuscript marked with proofreader marks and asked them to prepare a list of brief instructions for someone else who was going to retype the document. The purpose was to have these typists generate a list of names that could be used to label commands in a text-editing computer system. They called these "natural" names, but from this point we will call the names generated "conventional" names, since we believe a similar meaning was intended

In a follow-up study, the quality of these names was tested by giving subjects from a similar population of secretarial and high school students text to edit on a stripped-down UNIX text-editor (ED). In one experimental condition, subjects performed editing tasks in a version of ED labeled with the most popular conventional names (e.g., "omit") from the first study, and their performance was compared to subjects whose version of the editor had the existing

ED nameset (e.g., "delete"). No significant difference was found.

The lack of improvement was no surprise, because we believe the researchers did not in fact find conventional names for the computer system's operations. The initial hypothesis for the study was given that "the words an actual user would employ to describe the actions to be taken to perform the editing task in its non computerized form would make initial learning easier" for the computerized form. We add the last, implied phrase because it shows that this hypothesis would not violate Contrast and Conventionality only if the chosen names denoted the same meaning to the subjects for both the manual and computerized tasks. However, we conjecture that there are major differences between the typing of a document from scratch given a previous version and a set of editing instructions, and directly editing existing text in a computer system. In the former, for example, "omit" is a passive action and involves leaving out the next section to be typed, whereas "delete" in ED requires active manipulation of the text that is not wanted. The difference is not just one of dialect, but also one of denotation.

Studying work language as a means of establishing conventionality

Landauer et al. concluded that in their experiment, the use of conventional names did not matter; we believe, however, that the experiment did not actually test this, and that conventionality is still worth pursuing as a basis for naming. In this section we propose a method for generating such names based on analyzing what people say when they are engaged in the actual activities that require the interpretation of names in the computer system. We first begin by reviewing one paper in which the use of language in work situations has been studied.

In *Work Language and Information Technology*, Holmqvist and Andersen (1987) study the relationship between work language and the work being done. The primary goal of their work was through empirical study to describe a set of criteria that a theory must meet in order to provide a systematic basis for studying work language. They recorded language in a car repair shop and a large accounting department and interpreted their transcripts in light of ethnographic data about the roles and relationships of the speakers, the way work was accomplished in the work sites, and the jobs individual speakers were working on. They drew two conclusions that have been directly relevant to our work. First, they concluded that understanding work language requires detailed knowledge about the work taking place. Second, they make an important distinction between language used *within* actual work situations, and language about work that takes place outside of work situations. They claim that since computers must support people within work situations, it is the former language that should be used as a basis for assigning names. However, they do not study the naming problem in computers directly.

Our naming study took place as a part of an extended research project, known as ASCEND (Advanced System for Computations in Engineering Design), in which a collaborative group have been developing and evaluating a new technology for equational simulation (Piela, 1989; Piela, et al. 1991; Piela, et al. 1992b). Equational simulation is the name for a set of techniques used to analyze design alternatives when the design can be described as an explicit set of mathematical relations, and it is used in domains such as engineering and economic forecasting. From the start, the focus in the project has been on learning from people's use of the technology as a means of determining what form and function the technology should have. To do this, the developers in the project have relied on a small set of users who have chosen to work with a series of functioning ASCEND prototypes.

At any one time throughout the six years of the project, there have been 5-10 active users: some of whom are graduate students or faculty from a number of Carnegie Mellon University departments and who have used it for their own research goals; and some of whom are engineers in industry with an interest in exploring the technology. The primary developer is the second author, Piela, who began the work as a thesis project with his advisor Arthur Westerberg. Others who have contributed to the development effort have been undergraduate programmers, members of the Design department conversant in graphic design and human factors issues, and more recently, the first author, a social science graduate student.

In an earlier paper (Piela, et al , 1992a) we categorized three primary sources of data we have used for learning from users and their work: conversations within work situations; observation of people's work with the system; and the products of people's work—solved problems. When analyzed, these data have provided ideas for changes that can be discussed among users and developers, and when appropriate, folded back into the system or into improving the underlying theory. The current study is intended to extend this methodology to the naming problem. For this study we also sought conversations within work situations, however, there is a difference. In the larger study we listened to such conversations primarily as a conduit to the intentions of the speaker, and judged the adequacy of the data based on whether the conversations emerged from what we called "real work situations."

In this case, however, we were also interested in the language itself, and wanted to be able to make a convincing assignment between speakers' words and their intended meanings. So, we judged the adequacy of our data based on whether we had sufficient basis for making such an assignment. This basis came from two places: from ethnographic fieldnotes kept by the first author to provide information about the conversation's circumstances; and from collaborative examination of tapes and transcripts.

People involved in this collaborative examination were: the first author—unfamiliar with the domain of equational simulation—who was looking primarily for patterns of language usage and discrepancies of usage among different speakers and situations; the second author who brought system, domain, and project historical knowledge; and sometimes the speakers themselves, who offered what they remembered about what they were doing at the time.

We found our most valuable source of data to be people's *explanations* to one another about how work with the system should proceed. Explanations are valuable because they reveal what individual speakers consider to be relevant to explain, and what words are considered adequate to explain it for their listeners. Since explanations often occur across groups (between experienced and new users, or between experienced users from different domains) we expect speakers to choose a sufficiently general, conventional word that covers the necessary facts. Viewed as a collection, such explanations provide a way of extending the pool of judgments about what is necessary to convey meaning to users beyond the ones made initially in naming by the developer.

Although we don't have an operational definition for explanation, we call an utterance an explanation when the speaker makes reference to aspects of actions or objects as they are in more than the local situation. Examples are: descriptions of the speakers' or listeners' activities that refer to general categories of object, but not instances of objects (e.g., "I tend to edit *the file*" but not local ones like "I should edit *this file*") and declarative statements about the system that refer across situations (e.g., "It's got a relational database" but not local ones like "It got stuck here").

To a lesser degree, we looked for the use of names in language intended primarily to refer the local situation, e.g., "It got stuck here." Although such language was rarely a direct source of candidate names, it provided evidence of how people used or worked-around system names when explanation was not their goal.

The conversations cited in this paper were recorded, either on audiotape or on videotape, and then transcribed. Since our interest was primarily in the words people used, we transcribed speech word-for-word, but did not represent interactional details such as overlaps in speech. Work with these transcripts led us to question certain system names as problematic. We then assembled collections of transcripts pieces that either used the name of interest, or which we interpreted as referring to the same action or object. Such collection-making was iterative, since reviewing the collections often led to a recasting of the question.

What follows are two examples of the outcomes of these analyses. The first is our discovery of a system action for which users had a different name than the one by which it was named in the system. The second example demonstrates a problem that occurred when one name is used to convey different meanings in different situations.

Case 1: Two names for one meaning

When the study began, the existing version of ASCEND displayed the name **instantiate** for an operation that converted the users' written code into a collection of equations. It was drawn from the developers' study of the existing literature in computer science and information science, and it "sounded like the right word" to them.

Although some users were unfamiliar with the name, the activity of **instantiating** apparently looked close to a conventional meaning—something they called **compile**. Note the following explanations from a session when one user is demonstrating modeling in ASCEND to a new user. In the first, he answers a question about how the system works by linking the known word, **compile**, to the system name, instantiate. In the second, he directs the user to select the proper command from a menu and then provides a simple explanation for what the command does.

1) User 1: ...what, how the vectors are related, and then just link it to this file when, is it compiled or interpreted?

   User 2: It gets compiled every time you use it. It doesn't store compiled versions, or what we say is instantiations. You instantiate a model every time, so it's meaningless to have this instantiated.

2) User 2: Go to Create (the menuheader). Instantiate.

   User 1: And that actually starts the damned thing?

   User 2: It compiles it.

We interpret User 2's explanation, "what we say is **instantiations**" as his informing User 1 that ASCEND has a dialect difference he needed to be aware of. He considers no additional explanation necessary beyond showing User 1 that he considers them to be equivalent. This suggested to us that **compile** might be a better name than **instantiate**. To investigate this hypothesis, we returned to our transcripts to find other uses of **compile**, **instantiate**, and other words that conveyed the instantiation activity.

We found uses of both **compile** and **instantiate** in the transcripts; however, as the name that was visible in the system we would have expected the use of **instantiate** to be the most common. We did find two other words frequently used to refer to the activity of instantiation, however: **create** and **build**.

User 3: We created a simulation here called f1 that's a flowsheet.

> Developer 1: Don't be afraid to build something and throw it away.

However, both words were used in other situations by both speakers to refer to other ASCEND activities. Therefore, while **create** and **build** met the criterion of conventionality, they did not meet the restriction of having a unique meaning within ASCEND situations. So, the choice was between the developers' original name, **instantiate** and the way it was expressed by some users, **compile.**

The developers also knew the name **compile,** but at the time when they named the new meaning they found it sufficiently different from **compile** to justify the difference. What they had to decide now was which was more important—that this difference in meaning be communicated to users; or whether the users' conventional name was sufficiently close in meaning for the purpose. If the former, the difference in meaning had to be made more salient or else—as happened above—it would be treated by users as a difference in dialect, a bad result. The developers decided, however, that **compile** was sufficiently close, and the change was made. To date, we have seen no evidence that the change was a bad one, and some that it was a good one: the locus of explanation in the data we have collected since that time is no longer the name itself and instead focuses on what it means to perform the activity in an ASCEND situation.

Case 2: Two meanings for one name

For the reader to understand this case study, which is focused on different meanings for the name **model,** some background in the domain of equational simulation may be helpful. In equational simulation, the user formulates a problem as a system of equations and then submits this formulation to a numerical solving algorithm to compute chosen variable values. Within most existing systems, e.g., GAMS (Brooke, et al., 1988), these formulations are augmented with information about how they are to be solved, thus creating an executable definition which is commonly referred to as a **model.** By executable, we mean that the **model** can be supplied to a computer program which produces answers without any user intervention. In GAMS practice, users formulate **models** to represent their problems and then solve them.

ASCEND differs from systems such as GAMS because the problem descriptions are kept entirely separate from the solving procedure. Problem descriptions are formulated as **types,** which define the prototypical structure of a system of equations. In order to actually solve the problem, an **instance** of the **type** must first be created. Although ASCEND uses three categories of **types,** two (elementary and atomic) are essentially building blocks for the third kind, which is called a **model.** In ASCEND practice, users formulate **models** to represent their problems, create **instances** of them, and then solve the **instances.**

When the first author joined the ASCEND project in the summer of 1991, she noticed the ubiquity of the name **model** in people's speech, and in the system itself. Although in explanation people differentiated between **models** and their **instances,** in the local references of both developers' and users', **model,** or the individual **model** name, was often extended to include the **instance** being solved, as it would have been in GAMS:

> User 4: (demonstrating a problem to his advisor) This is the way the model comes up now, and it does solve.
>
> Developer 1: (referring to actions on the instance of the model "stream") What you were doing- you would refine the stream, you would bring the stream into the Solver and resolve it, and I think that is a fine way to think about it.

This extension of **model** was even employed in the interface in a command named "initialize model" that actually operated on the instance of a model.

On the other hand, except in explanations where the explicit topic was the relationship between **types** and **instances,** the name **type** was rarely used in speech. Even then, and even in careful explanations by developers, **type** was often replaced by such terms as "prototype" "concept" or "definition." There were also few cases of the use of the name **type** in the system interface, even where this meaning was intended. For example, a toolkit in the interface where all categories of **type** were stored was called the Model Library. The original name of the toolkit was the Type Library, but it had been changed in response to the perceived negative reactions by some users. As Piela explained:

> What we found was, that engineers, and most of the users tended to like the idea of- model was something they could understand and seemed to indicate notions of prototype. And introducing the word type seemed to alienate them...it was some extra computeiese that made them a little uncomfortable.

Thus, the name **model** had been extended in two ways in the system interface: to refer to the general category of stored **type** definitions; and to refer to the **instance** of the **model.** This, however, was not something that was talked about by either users or developers, and apparently was not recognized. It was through the first author's attempts to get definitions for different ASCEND objects and actions, and the subsequent collaborative analysis of the data that we eventually arrived at the conclusion that there was in fact more than one meaning for the name **model** in ASCEND situations.

So far in this example, we have attempted to show that there are conflicting meanings of the name **model** that are available to users. As such, this appears to conflict with the restriction we proposed that the name chosen have a unique meaning across the set of situations in which people will be using the names. Next, we want to show that these conflicting meanings, in fact, caused confusion for some users.

An essential fact about simulation in the ASCEND paradigm is that **types**, including **models**, are separate from their **instances**. But, not all users had drawn this conclusion. For example, one experienced user when asked his opinion about a "Remove Model" command that had been recently taken out of the system, volunteered that he was never sure what effect removing a **model** would have on the **instance** of that **model**. Another user—who had worked with ASCEND for over a year—expressed surprise when, having removed all **types** from the Model Library where they were stored, he found that an **instance** of a **model** created earlier was still "hanging around" in the system. Another time, not realizing that he could have more than one **instance** of the same **model**, he made manipulations to one that were intended for another, with a great cost of time and effort. Since both of these users were familiar with the existing simulation technology, we conjecture that they expected ASCEND **models** to share characteristics of, for example, GAMS **models**.

Our analysis convinced us that **model** was naming both the ASCEND **type**, and a still-present conventional meaning, and that it was causing problems. Thus some renaming was required. The first step we chose was to return **type** and **instance** to a much more explicit role in the interface. **Type** may have been alienating, but in its newness, it did properly denote that there was a new meaning to be learned, thus allowing the users to appeal to Contrast. Masquerading the newness with the conventional name **model** turned out to be a disservice to the users.

In the current version of ASCEND, all references to **models** are to the narrower meaning. Operations that act on **types** are explicitly denoted as such, and operations that act on **instances** are also explicitly named. The goal has been to make prominent the new meanings that ASCEND embodies, so they can be accepted or rejected for what they were are, and consistent interface naming is one strategy for achieving that goal. Of course, the recasting of **model** in the interface has not changed people's conventional use of **model** in non-explanation situations. We expect, however, that if people continue to work with ASCEND or ASCEND-like technology that the names **type** and **instance** will become a larger part of their work language.

Conclusion

In this paper we have described a framework for naming in computer systems based on the Principles of Contrast and Conventionality. We argue that the goodness of names cannot be evaluated through examining properties of the names themselves, but only through determining how names are interpreted by system users. We have proposed three ways in which names can be good: 1) When they adhere to Contrast and Conventionality—by affiliating conventional meanings with conventional names or by affiliating new meanings with new names; 2) When they use the name that is most general, i.e., is conventional for as many users as possible; and 3) when the name has a unique meaning for the set of situations in which it is employed.

We presented two individual naming decisions that were analyzed according to these guidelines. To determine what was conventional, we recorded the work language of users and developers while they were focusing on local tasks to be accomplished and when they were explaining their work, and supplemented these with ethnographic field notes that described the situations in which the conversations occurred. We then analyzed this data collaboratively in order to make convincing assignments between meanings and conventions. To determine what was general, we compared the conventional language of different users and of the developers in the study. To determine uniqueness, we looked for counterexamples, i.e., situations in which we could demonstrate that a name had more than one meaning and that the difference was significant to users.

We would like to explore whether these guidelines could be recast in a more productive way, or whether there are other guidelines that could further limit and thus simplify the task of choosing names. Also, this study offers no guidance about how new names for new meanings should be generated. We suspect there are pragmatic guidelines that could be developed which could aid in devising names that suggest some of their intended meaning to users. These are some of the many naming issues we would like to examine in future work.

References

Black, J.B., and Moran, T.P. (1981). Learning and remembering command names. *Proceedings of Human Factors in Computer Systems* (Gaithersburg), 8-11, New York: ACM.

Brooke, A., Kendrick, D. and Meeraus, A., (1988). *GAMS. A User's Guide*. Redwood City, CA: The Scientific Press.

Carroll, J.M., (1985). *What's in a Name? An Essay in the Psychology of Reference*, New York:W.H. Freeman and Company.

Clark, E.V. (1987) The Principle of Contrast: A constraint on acquisition. In B. MacWhinney (Ed.) Mechanisms of Language Acquisition, pp. 1-33, Hillsdale, N. J.: Lawrence Erlbaum.

Clark, E.V., (1990). On the pragmatics of contrast. *Journal of Child Language*, 17 417-431.

Grudin, J. and Barnard, P. (1984) The cognitive demands of learning and representing command names for text editing. *Human Factors*, 26 (4), 407-422.

Holmqvist, B., and Andersen, P.B., (1987) Work language and information technology. *Journal of Pragmatics* 11, 327-357.

Landauer, T.K., Galotti, K.M., and Hartwell, S. (1983). Natural command names and initial learning: a study of text-editing terms. *Communications of the ACM*, 26(7) 495-503.

Landauer, T.K., and Galotti, K.M. (1984) What makes a difference when? Comments on Grudin and Barnard. *Human Factors*, 26(4), 423-429.

Piela, P.C., (1989). *ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis.* Unpublished Ph.D. thesis.

Piela, P., Epperly, T., Westerberg, K., and Westerberg, A. (1991). ASCEND: An OBject-Oriented Computer Environment for Modeling and Analysis: The Modeling Language. Computers and Chemical Engineering, 15(1), 53-72.

Piela, P.C., Katzenberg, B., and McKelvey, R.D. (1992a). Integrating the user into research on engineering design systems. *Research in Engineering Design.*, 3 211-221.

Piela, P., McKelvey, R.D., and Westerberg, A.W. (1992b). An Introduction to ASCEND: Its Language and Interactive Environment, *Proceedings of the 25th Hawaii International Conference on System Sciences, in Koloa, Hawaii.*