Natural Naming in Software Development: Feedback from Practitioners

Kari Laitinen

VTT Electronics / Embedded Software
P.O. Box 1100, 90571 Oulu, Finland

Abstract. During a five-year period several groups of software developers have been educated on using natural naming in software development. Generally, natural naming means avoiding abbreviations. In programming it means that program elements such as variables, tables, constants, and functions should be named using whole natural words and grammatical rules of a natural language. To assess the usefulness of natural naming and the importance of naming in general, we have requested the opinions of 52 software developers who have participated in naming courses or to whom a naming handbook has been introduced. The subjects had to judge the relevancy of 25 statements related to naming. The results of the inquiry indicate that most software developers, and especially the experienced ones, consider that natural naming facilitates their work.

1 Introduction

People need to write, read, and understand different kinds of documents in software development work. Documents are equally important in the development of information systems as well as in the development of less-traditional software systems, such as telecommunications systems and embedded computer systems. Software development can, in fact, be regarded as a documentation process during which more and more elaborate documents emerge as the work proceeds [11]. Typically, some sorts of requirements descriptions are produced first. They are followed by design documents and implementation documents from which an executable software system can be generated. Some software documents may be written according to some development methods (e.g. [1, 20]). Implementation documents can be source programs or other descriptions which can be processed with computer tools (e.g. compilers or application generators).

Software development is partly a learning and communication process [3]. Documents are important in communication and thereby the understandability of software documents affects the efficiency of software development work. Usually, the least understandable software documents are the implementation documents which need to describe all the details of the system being developed and which are, in most cases, source programs.

The use of different kinds of abbreviations (e.g. acronyms and shortened words) is common in software documentation and in the world of computers in general. The use of abbreviations has been, and still is, especially popular in source programs as names for different program elements such as variables, constants, and procedures. One reason for this is that, unlike modern tools, early software development tools allowed only short names. Our concern is, however, that the overuse of abbreviations makes source programs and other documents difficult to understand, which is harmful in software development and maintenance.

As source programs are the most contaminated with abbreviations and they are developed and maintained in many software development organizations, we will study the problems related to abbreviations in the context of source programs. To avoid abbreviations, a principle called natural naming has been proposed [8, 9, 10]. Natural naming means that all names in source programs should be constructed using, preferably several, natural words of a natural language while respecting the grammatical rules of the natural language. The natural names should also describe the functionality of the program. By using natural names it is possible to bring source programs symbolically closer to other types of software documents which contain written words of a natural language. Because natural naming can be applied to many types of software documents, the idea is an important issue in software documentation.

Intuitively, a natural name like "customer_number" is more understandable than an abbreviated name like "cnumbr" or "cn". Also, naturally named programs seem to be much more understandable than the same programs written with abbreviated names (see Fig. 1). It has not, however, been fully proven that natural naming is always an appropriate principle in software documentation. We need more experience and evidence about natural naming. In this paper we will present and analyze practitioners' opinions about natural naming. During the last five years we have given several courses on natural naming and also delivered naming handbooks in software development organizations. The practitioners' opinions have been collected by asking them to answer a questionnaire about natural naming.

We believe that asking software developers' opinions on the use of natural naming is a relevant research method. We will justify this belief in the second section in which we discuss related work which deals with naming. In the third section we will explain what we have taught to software developers and how they were questioned. In the fourth section we analyze the feedback received from the people involved.

2 Related Work

Empirical understandability tests have been carried out to find out how people understand source programs. Tests related to naming are reported in [2, 15, 16, 17, 18]. The effect of naming has usually been tested by presenting a badly named source program to a group of students and the same program with more informative names to another group of students. The performance of the student groups has been measured by asking questions about the programs or by asking the students to modify the programs. The reported understandability tests have not, however, always produced

```
#define   C0001       13
#define   C0002       0
#define   C0003       1

/*---------------------------------------*/

f0001 (  char  s0001 [],
         int  *i0001  )

/*---------------------------------------*/
{
    int   i0002,   i0003 ;

    *i0001  =  C0002  ;

    i0003  =  strlen ( s0001 ) ;

    if ( i0003  >  C0001 )
    {
        *i0001  =  C0003 ;
    }
    else
    {
       for (i0002=0; i0002<i0003; i0002++)
       {
          if(( s0001[ i0002] < '0') ||
             ( s0001[ i0002] > '9') )
          {
             *i0001  =  C0003 ;
          }
       }
    }
}
```

Version (a): Numerical names

```
#define   CNUMMAX   13
#define   VALID     0
#define   NVALID    1

/*-------------------------------------*/

isvalid (  char  cnumbr [],
           int  *rcode  )

/*-------------------------------------*/
{
    int   i,   len ;

    *rcode  =  VALID  ;

    len  =  strlen ( cnumbr ) ;

    if ( len  >  CNUMMAX )
    {
        *rcode  =  NVALID ;
    }
    else
    {
       for  ( i=0 ; i<len ; i++ )
       {
          if (( cnumbr[ i] < '0') ||
             ( cnumbr[ i] > '9') )
          {
             *rcode  =  NVALID ;
          }
       }
    }
}
```

Version (b): Abbreviated names

```
#define   MAXIMUM_CUSTOMER_NUMBER_LENGTH        13
#define   CUSTOMER_NUMBER_IS_VALID               0
#define   CUSTOMER_NUMBER_IS_NOT_VALID           1

/*-------------------------------------------------------------------------*/

check_customer_number_validity (  char  possibly_valid_customer_number [],

                                  int  *success_code  )

/*-------------------------------------------------------------------------*/
{
    int   customer_number_index,   customer_number_length ;

    *success_code  =  CUSTOMER_NUMBER_IS_VALID  ;

    customer_number_length  =  strlen ( possibly_valid_customer_number ) ;

    if ( customer_number_length  >  MAXIMUM_CUSTOMER_NUMBER_LENGTH )
    {
        *success_code  =  CUSTOMER_NUMBER_IS_NOT_VALID ;
    }
    else
    {
       for  (  customer_number_index  =  0  ;
               customer_number_index  <  customer_number_length ;
               customer_number_index  ++    )
       {
          if(( possibly_valid_customer_number[ customer_number_index] < '0') ||
             ( possibly_valid_customer_number[ customer_number_index] > '9') )
          {
             *success_code  =  CUSTOMER_NUMBER_IS_NOT_VALID ;
          }
       }
    }
}
```

Version (c): Natural Names

Fig. 1. Differently written versions of the same source program

statistically significant results, although the performance of the subjects has usually been better with source programs containing clearer names.

It is hard to test how names affect understandability, because it is difficult to judge how much meaning there is in a name. Different people may interpret the same names in a different manner [13]. In the context of the mentioned understandability tests the term "mnemonic name" is used, whereas we speak about natural names. Weissman [18] has used one to three words long natural names in his tests, but Curtis et al. [2], for instance, have used mnemonic names in Fortran programs. Because Fortran has traditionally had the six-character restriction in name lengths, Curtis et al. could not use very long natural names. That may be one reason why they did not find any differences in performance when different kinds of names were used. We would also like to point out that the term "mnemonic" is not very accurate. It has been used to denote instructions of assembler languages (e.g. [6]). In these cases, "mnemonic" means abbreviations such as MOV, STA, and LDA. To make a distinction between mnemonic names and natural names, let us study the following names which could all represent the same variable:

```
(1)     n
(2)     nbytes
(3)     bytes
(4)     byte_count
(5)     number_of_bytes
(6)     number_of_bytes_in_buffer
```

Since we recommend that a natural name should contain more than one word, only names (4), (5), and (6) above can be considered natural, whereas all of them excluding name (1), could be considered mnemonic. By studying the examples above, we can also notice that natural names can usually be constructed in many ways.

Because understandability tests have not produced statistically significant data, we can say that the effect of naming is difficult to measure. We can find support to this statement in other scientific fields. Natural naming is using a natural language to describe how programs work. Therefore, studying naming in software documentation is related to linguistics. Linguists admit that natural languages are complex and they are not yet fully understood [4]. The complexity of natural languages can thus be one reason why the effects of natural naming are so hard to measure in the context of software documentation. Because natural languages change all the time, even the concept "natural word" is vague. Thereby the definition for natural naming is vague. New words emerge in natural languages and even some abbreviations can be considered belonging to natural languages. Philosophers have also studied meanings of natural words and other symbols. Famous philosophical studies related to languages have been done by Wittgenstein. He was, however, dissatisfied with his work, possibly because he did not find any clear and conclusive theories to explain languages and how they relate to the real world [19].

Despite the fact that naming seems to be a hard research subject the following facts appear to support the use of natural naming in software documentation:

- The use of abbreviations has been criticized in other contexts of technical documentation [5, 12].
- Natural names are generally used in graphic-textual descriptions of software development methods (e.g. [1, 20]). We can assume that natural naming is one reason why graphic-textual descriptions are considered useful in software development.
- Some software development methods (e.g. [14, 20]) recommend the use of so-called pseudo coding which means describing programs with a language that is somewhere in between a natural language and a programming language. The use of natural naming brings source programs closer to natural language.

To summarize the discussion above, we can say that we already have evidence about the usefulness of natural naming, but more evidence is needed. For this reason, we have surveyed the opinions of people who are engaged in practical software development work. Our research approach can be justified by taking into account the fact that practical software development differs a great deal from studying short examples of programs in a classroom. The source programs of practical software systems may, for example, contain about one thousand different names, whereas the number of names in the program examples used in understandability tests can be counted in a few tens. Supposing that the mentioned understandability tests had produced statistically significant data in support of natural naming, we could still not be completely sure that natural naming would be useful in practical software development work, because the experiments were done during a short period of time and with students. Supposing also that experiments in a classroom would never produce any significant data, it could still be possible that natural naming would be useful in practical work [17]. Because it is hard to do controlled experiments in which we could compare two different groups building the same real software system using different naming styles, we have to rely on the opinions and intuitions of people.

3 Practical Arrangements

3.1 Introduction of a Handbook for Natural Naming

All software developers who participated in this study had been given a naming handbook. The main ideas of the naming handbook are published in [9]. The first version of the naming handbook was introduced about five years ago, and new versions have emerged afterwards. All versions of the handbook include the following:

- a definition for the principle of natural naming;
- a high-level classification of names needed in programs: function names, constant names, and data names;
- rules for constructing different types of names (e.g. function names should have at least two words, and an imperative verb should be used at the beginning of a function name);

- instructions to use so-called name refining words to separate related names;
- name tables that provide low-level classifications of different types of names, suggest certain words to be used, and give some examples of appropriate names; and
- examples of naturally named programs.

3.2 Preparation of Courses on Natural Naming

A typical course on natural naming is a half day session, combined with other instructions on programming style. The courses given to the respondents of our survey involved the introduction of the ideas presented in the naming handbook. The following additional issues were highlighted on every course:

- The use of natural naming was justified by explaining its potential benefits and disadvantages.
- The use of abbreviations was strongly discouraged. Programs were compared with other types of writings in which abbreviations are less common (see Fig. 2).
- Natural naming was considered the easiest way to establish standard naming practices, since, compared to maintaining a list of acceptable abbreviations, nothing needs to be maintained when a pure form of natural naming is applied.

A typical naming course also involved public discussion on specific naming problems in the organization where the participants worked.

> In newspapers we use whole English words and very few abbreviations. If newspapers were written like programs they would look like the text below.
>
> In n_paper we use whl Engl wrd and very few abbr. If n_paper were writ like progr they wld lk like this txt.

Fig. 2. A slide used on a naming course

3.3 Naming Questionnaires

The naming-related inquiries were arranged so that all the responding subjects had at least one year to get accustomed to using the natural naming approach. All subjects were familiar with a naming handbook, and some of them had attended a naming course. It should be noted that we did not arrange the courses or develop the naming handbook in order to be able to arrange the inquiries afterwards.

The primary hypothesis for the naming inquiries can be formulated as "Natural naming facilitates the work of software developers". The inquiry form contained 25 statements which were either for or against this primary hypothesis. Each individual statement on the form can be considered an elementary hypothesis for this study (see Table 1). The subjects had to judge the relevancy of each statement by answering

"completely disagree", "partially disagree", "no opinion", "partially agree", or "completely agree".

Fifty two software developers filled in and returned the inquiry form. Twelve persons to whom the inquiry form had been sent did not return it. One subject reported being too busy, and perhaps some of the non-reacting respondents did not consider the subject important. The missing responses have not been noted in the statistical calculations in the appendices, although it could have been possible to count them as having "no opinion" on all the statements.

3.4 The Responding Groups

Nearly all of the subjects who responded to the inquiries were using the C programming language in their work, and all had tools that allowed the use of long natural names in programs. All respondents spoke Finnish, but most of them used English to document their programs. At least half of the respondents had a master's level degree from a university. 27 of the respondents work in two telecommunications companies, and the remaining 25 respondents work in a research institute. The experience of the subjects ranges from 2 to 20 years. The subjects represent several application domains: telecommunications systems, real-time embedded systems, various PC and workstation-based software engineering and testing tools, and systems involving artificial intelligence.

4 Analysis of the Responses

Table 1 lists the statements and summarizes the responses. The statements are in the same order as they were presented to the subjects. The third column shows the distribution of the answers in percentages We have used the numeric scale of one to five in order to make a statistical analysis of the responses. According to the scale, 1 means "completely disagree", 2 means "partially disagree", 3 means "no opinion", 4 means "partially agree", and 5 means "completely agree". Some of the statements of the questionnaire are against the use of natural naming. These are marked with a minus (-) sign in Table 1. Correspondingly, plus (+) signs denote those statements which support natural naming.

The rightmost column of Table 1 contains statistical data which has been calculated using the numeric scale. The t-test was used to find out whether the responses can be considered statistically significant. The t-values have been calculated by comparing the responses given to each statement with the responses of an equally large imaginary group which was normally distributed. The respondents of the imaginary comparison group had no opinion on any statement. In this invented comparison group, the distribution over the alternatives from 1 to 5 was 10%, 20%, 40%, 20%, and 10%, respectively. The t-values which are marked with an asterisk (*) indicate statistical significance. When the t-value is more than 2, the likelihood that the mean response does not correspond with reality is less than 5%.

Table 1. Summary of the responses given to the naming questionnaire

	STATEMENT	RESPONSES (%)					+/-	Mean response, standard deviation, and t-value.		
		1	2	3	4	5				
1	The time required to write long names slows down software development.	38	40	3	17	0	-	2.0	1.1	4.65*
2	More and more often, I find myself thinking about appropriate wording for a name needed in a program.	9	9	25	42	13	+	3.4	1.1	1.81
3	In practice, there emerge difficulties when natural names are used.	17	21	17	37	5	-	2.9	1.2	0.33
4	The use of naming guidelines limits the freedom of software development work.	42	40	3	9	3	-	1.9	1.1	4.93*
5	Discussion related to choosing suitable names has increased among my colleagues.	17	21	30	23	7	+	2.8	1.2	0.76
6	Natural naming does not contribute to how easily we can locate the place in a program that we are searching for.	29	35	13	11	9	-	2.4	1.3	2.61*
7	The understandability of the programs written by my colleagues has not improved after the introduction of the naming guidelines.	12	16	52	14	4	-	2.8	1.0	0.89
8	One needs several months to get accustomed to using natural naming in programming.	26	32	17	15	7	-	2.4	1.3	2.37*
9	Because there is such a hurry in projects there is no time to think the understandability of names.	34	40	5	15	3	-	2.1	1.2	3.83*
10	It is difficult to change a naming style one has once adopted.	28	32	9	23	5	-	2.4	1.3	2.34*
11	Commonly used abbreviations, such as i, j, ptr, tbl, and msg should be accepted without exception.	2	15	13	44	25	-	3.8	1.1	3.49*
12	Generally, too little attention is paid on naming.	0	11	11	47	29	+	3.9	0.9	4.60*
13	The natural naming course / the naming handbook really changed my attitudes towards naming.	7	13	30	42	5	+	3.3	1.0	1.18
14	Other programming style factors, such as indentation, uniform use of braces, and uniform order of function arguments, contribute more to the understandability of programs than naming.	0	41	33	25	0	-	2.8	0.8	0.85
15	Nowadays, I always try to use natural naming.	2	13	11	49	23	+	3.8	1.0	3.69*
16	Clearly, during the past couple of years, the names in my colleagues' programs have become longer.	4	4	60	14	16	+	3.4	1.0	1.69
17	Compared to the use of abbreviations or single letters, the use of natural names makes the thinking process of software developers easier.	5	2	9	55	26	+	4.0	1.0	4.63*

Table 1 (continued). Summary of the responses given to the naming questionnaire

	STATEMENT	RESPONSES (%) 1	2	3	4	5	+/-	Mean response, standard deviation, and t-value.		
18	Trying to invent suitable names is a means for analysing the problem at hand.	2	23	15	32	26	+	3.6	1.2	2.63*
19	It is useful if one can remember the names in programs.	0	4	16	37	41	+	4.2	0.9	5.81*
20	Abbreviated names are easier to remember than natural names.	37	35	16	6	4	-	2.0	1.1	4.32*
21	I am satisfied with my work when I am able to invent descriptive names when writing a program.	6	6	31	45	10	+	3.5	1.0	2.26*
22	It is necessary to pronounce the names in practical work. Natural naming has facilitated the oral communication.	2	9	41	41	5	+	3.4	0.8	2.02*
23	It would be easier to learn information technology and programming, if the program examples used in teaching and literature were naturally named.	3	7	25	36	26	+	3.8	1.1	3.49*
24	With how many fingers do you use the keyboard of your terminal (1 = 2 fingers, 2 = 4 fingers, 3 = 6 fingers, 4 = 8 fingers, and 5 = 10 fingers).	0	25	21	28	25	o	3.5	1.1	
25	Being able to type with 10 fingers speeds up software development.	7	11	21	28	30	o	3.6	1.3	2.71*

4.1 General Observations

All the statistically significant responses, excluding statement 11, indicate some kind of positive attitude towards the use of natural naming. Considering all the responses given, we could not find any indication that the use of natural naming was somehow harmful, which would have subsequently made us wary about recommending this naming approach.

Because of the fact that many public names (e.g. library functions and operating system calls) in large software systems are abbreviated, it is unlikely that someone working in a software development group could always use purely natural names. Therefore, instead of asking whether the subjects *use* natural naming, we asked whether they *try* to use natural naming. All the respondents who reported their agreement with the natural naming approach also gave the most positive responses to all the statements.

Clearly, the majority of the respondents agree that too little attention is paid to naming (S12)[1]. This supports the notion that literature provides too little advice on naming. Textbooks on programming and software engineering usually state that descriptive names should be used. However, no instructions are given on how the names can be made descriptive and informative [10].

[1]These markings refer to the statements in Table 1.

Although the respondents seem to favor the natural naming approach, they also want to use the traditional and most common abbreviations (S11). Using purely natural names and accepting the commonly used abbreviated variable names, such as i, j, tbl, ptr, and msg, is contradictory. Natural names seem to be favored as global and public names.

4.2 Observations Related to Understanding, Communication, and Thinking

The distribution over the five alternatives is the highest in responses to the statements related to the understandability of programs (S7, S14, and S16) and to communication among software developers (S5 and S22). This shows that the respondents had no clear opinion on these matters. It may also indicate that they do not pay attention on how they communicate with their colleagues or whether programs are understandable. We had anticipated that natural naming would facilitate oral communication. However, there is only minimal evidence that favors this anticipation (S22).

The responses do not clearly indicate whether naming, indentation, or some other programming style factor contributes the most to the understandability of programs (S14). This may mean that it is difficult to make a clear distinction between programming style factors, or that programs are always considered rather hard to understand and, therefore, improvements in understandability are difficult to perceive. Only a minority of the respondents had found that the names in their colleagues' programs have clearly become longer, whereas most of the respondents had no opinion (S16).

Although the communicability of programs was found difficult to judge, most of the respondents agreed that if program examples used in teaching and textbooks were naturally named, learning information technology and programming would be easier (S23). Indeed, programs must be complex reading to those who see them for the first time. If we lessen the complexity by using commonly known words instead of abbreviations, it is obvious that a person unfamiliar with programs is able to perceive something familiar when he or she tries to find out what a program does.

Laitinen and Mukari [10] show that judging the relevancy of names is a means for analyzing the problems of an application domain. It is thus relevant to presume that software developers, at least unconsciously, use naming as a thinking tool. The responses support this presumption (S18). Considering the thinking process during programming, the respondents gave answers that support the use of natural naming (S17, S18, S19, and S20). Generally, abbreviated names were considered more difficult to remember than natural names (S20).

4.3 Observations on Practical Matters

The majority of the respondents see no difficulty in using naming guidelines in their work (S4). In practice, however, the use of natural naming can cause some difficulties. For example, long natural names do not fit so easily on a screen or on a piece of paper, and some software development tools are not able to interpret long names. The respondents had to judge whether the benefits of natural naming exceed

the practical inconveniences. Unfortunately, we did not find any significant data about this matter.

Software developers often need to find a certain piece of code in a program module or related pieces of code in several program modules. These kinds of search activities are often carried out during software maintenance. The majority of the respondents consider that the use of natural naming makes the search activities easier (S6). It is rather obvious that searching for natural words is easier than searching for something that symbolically represents a concept of the real world. For example, if a maintenance task is to change the definition and processing of a customer number in a system, it is easier to start searching the names which contain the natural words "customer" or "number," rather than trying to guess what name might represent the customer number.

A complaint sometimes expressed by a participant on a naming course is that natural names are long and too much time is therefore wasted in writing them. However, these are the opinions of a minority, since most of the respondents disagreed with this kind of a statement (S1). The respondents thus seem to consider that the physical writing process is not the activity that takes most of the time needed in implementing a computer program, or they think that the time that is required to write longer names is paid back as the resulting programs can be understood more readily. Although the speed of the physical writing process would not directly affect how quickly programs can be created, software developers do spend a considerable amount of their time operating their computers. Therefore, we also asked how well the respondents are able to type. Although there is great variation in opinions, most of the respondents agree that being able to type with ten fingers indeed speeds up software development (S25).

4.4 Comparing Different Groups of Respondents

As the respondents consisted of different types of people, we made some comparisons between different respondent groups. In the case of individual statements, we did not find very many statistically significant differences between different groups, However, when we compared all the responses of the groups we found some important differences. To compare the general attitude towards natural naming, we first reversed the numeric scale of those statements in Table 1 which are against natural naming, and then compared all the responses of one group to all the responses of another group. Statements 24 and 25 were excluded in these comparisons.

We found out that people with more than 3 years of experience are more enthusiastic about natural naming than less experienced software developers in the same company. In this comparison, the mean response for the less experienced people was 3.4 while the mean for the more experienced people was 3.7. These figures are significantly different with t-value 2.56. When we compared people working in a research institute to those working in commercial companies we found out that people working in companies (mean response 3.6) have a more positive attitude towards

natural naming than people working in a research institute (mean response 3.4). The mean responses were significantly different with t-value 3.36.

The fact that the experienced people in companies have especially positive attitudes towards natural naming is an indication of the usefulness of this approach. The respondents represent typical people in the software industry. They work in large projects, need to co-operate intensively, and also carry out software maintenance. The comparison group, software developers in a research institute, do not usually work in large development groups. Some of them develop software only for scientific purposes or occasionally. Maintenance does not usually belong to their duties.

We also made artificial respondent groups by comparing people who completely agreed or disagreed with some statement to the other people who had a different attitude. Generally, people who agree with statements like 15 tend to be more enthusiastic about natural naming than others. We had presumed that skillful typists, the people who responded with 5 to statement 24, would have a more positive attitude towards natural naming than others, but we did not get statistically significant data that would support this presumption.

5 Concluding Discussion

We summarize the most important findings of this study as follows:

- The natural naming approach can be considered useful in software development. We could not find anything that would prevent us from recommending the use of natural naming in practical work.
- Compared to using abbreviations, the respondents believe that using natural names facilitates their thinking process. Trying to invent descriptive names is obviously an important means for problem analysis in software development.
- Experienced software developers in industrial organizations were more enthusiastic about the natural naming approach than less experienced developers or the software developers in a research institute.
- The understandability of programs is hard to assess since the respondents did not give clear opinions whether natural naming facilitates communication or had improved the understandability of source programs.

On the basis of these findings we can say that software development organizations in particular, but also the research community, should focus more attention on naming and on the use of natural languages in software documentation. Although more or less official naming rules exist in many industrial software development organizations, naming is still often a matter of a programmer's personal taste and style. Organizations should, however, strive to establish naming rules, as accurate as possible, in order to standardize their programming practices. We recommend that natural naming principles be favored in the creation of these rules. Naming rules, among other kinds of programming rules, can be conveniently adopted as part of a quality system for software development [7]. When naming rules belong to a quality system, they can be adjusted according to the standard practices of the quality system.

Software development usually involves writing other types of software documents than source programs. All software documents that describe the same system should

be understandable and they should not be contradictory. Therefore, it is important that the names used in programs correspond with the textual expressions in other types of software documents. Considering their software documentation practices, software development organizations should try to ensure that they use the same terminology in requirements descriptions, design documents, and source programs. One solution is to maintain standard vocabularies for the application domains in which the organization is involved.

Those engaged in research should, in our opinion, pay more attention to naming as well. Practically every software development method and tool involves the use of a natural language in some form or another. It is possible that naming is a difficult research subject because natural languages are hard subjects. However, we feel that naming and the use of natural languages should be taken into account in research related to information systems and other software systems. It is well known that software development is a difficult process to manage. One reason for this may be that natural languages are used too carelessly in the development process.

Acknowledgments

This work has been funded by the Technical Research Centre of Finland (VTT). The author wishes to thank the people who responded to the naming questionnaires and who helped in delivering and collecting the forms. The anonymous referees of the two versions of this paper have also helped with their comments. Special thanks are due to Mr. Douglas Foxvog, Prof. Pentti Kerola, Ms. Minna Mäkäräinen, Dr. Veikko Seppänen, Ms. Eija Tervonen, and Dr. Matti Weckström.

References

1. P. Coad, E. Yourdon: Object-oriented analysis. Englewood Cliffs, New Jersey: Prentice-Hall 1990
2. B. Curtis, S. B. Sheppard, P. Milliman, M. A. Borst, T. Love: Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. IEEE Transactions on Software Engineering 5, 96-104 (1979)
3. B. Curtis, H. Krasner, N. Iscoe: A field study of software design process for large systems. Communications of the ACM 31, 1268-1287 (1988)
4. V. Fromkin, R. Rodman: An introduction to language, fourth edition. New York: Holt, Rinehart, and Winston 1988
5. A. M. Ibrahim: Acronyms observed. IEEE Transactions on Professional Communication 32, 27-28 (1989)
6. Intel: MCS-80/85 family user's manual. Santa Clara, California: Intel 1979

7. ISO 9000-3: Quality management and quality assurance standards - part 3: Guidelines for the application of ISO 9001 to the development, supply, and maintenance of software. Geneva, Switzerland: International Organization for Standardization 1991

8. D. A. Keller: A guide to natural naming. ACM SIGPLAN Notices 25, 5, 95-102 (1990)

9. K. Laitinen, V. Seppänen: Principles for naming program elements, a practical approach to raise informativity of programming. In: Proceedings of InfoJapan'90 international conference, part I. Tokyo: Information Processing Society of Japan 1990, pp. 79-86

10. K. Laitinen, T. Mukari: DNN-Disciplined natural naming, a method for systematic name creation in software development. In: Proceedings of 25th Hawaii international conference on system sciences, Vol. II. Los Alamitos, California: IEEE Computer Society Press 1992, pp. 91-100

11. K. Laitinen: Document classification for software quality systems. ACM SIGSOFT Software Engineering Notes 17, 4, 32-39 (1992)

12. D. Logsdon, T. Logsdon: The curse of the acronym. In: Proceedings of the international professional communications conference. New York: IEEE 1986, pp. 145-152

13. P. R. Newsted: Flowchart-free approach to documentation. Journal of Systems Management 30, 4, 18-21 (1979)

14. M. Page-Jones: The practical guide to structured systems design, second edition. Englewood Cliffs, New Jersey: Prentice Hall 1988

15. S. B. Sheppard, B. Curtis, P. Milliman, T. Love: Modern coding practices and programmer performance. Computer 12, 12, 41-49 (1979)

16. B. Shneiderman: Software psychology, human factors in computer and information systems. Cambridge, Massachusetts: Winthrop Publishers 1980

17. B. E. Teasley: The effects of naming style and expertise on program comprehension. International Journal of Human-Computer Studies 40, 757-770 (1994)

18. L. M. Weissman: A methodology for studying the psychological complexity of computer programs. Ph.D. Thesis. Toronto, Canada: Department of Computer Science, University of Toronto 1974

19. L. Wittgenstein: Philosophical investigations. Oxford, England: Basil Blackwell 1953

20. E. Yourdon: Modern structured analysis. Englewood Cliffs, New Jersey: Prentice-Hall 1989