What's in a Name? A Study of Identifiers

Dawn Lawrie Christopher Morrell Henry Feild David Binkley
Loyola College
Baltimore MD
21210-2699, USA
{lawrie, hfeild, binkley}@cs.loyola.edu, chm@loyola.edu

Keywords: Software Quality, Program Identifiers Quality Assessment, Program Comprehension

Abstract

Readers of programs have two main sources of domain information: identifier names and comments. When functions are uncommented, as many are, comprehension is almost exclusively dependent on the identifier names. Assuming that writers of programs want to create quality identifiers (e.g., include relevant domain knowledge) how should they go about it? For example, do the initials of a concept name provide enough information to represent the concept? If not, and a longer identifier is needed, is an abbreviation satisfactory or does the concept need to be captured in an identifier that includes full words?

Results from a study designed to investigate these questions are reported. The study involved over 100 programmers who were asked to describe twelve different functions. The functions used three different "levels" of identifiers: single letters, abbreviations, and full words. Responses allow the level of comprehension associated with the different levels to be studied.

The functions include standard algorithms studied in computer science courses as well as functions extracted from production code. The results show that full word identifiers lead to the best comprehension; however, in many cases, there is no statistical difference between full words and abbreviations.

1 Introduction

Conventional wisdom says that choosing meaningful identifier names improves the ability of the next engineer to comprehend the code [6]. This paper seeks to study the effect that identifier names have on the comprehension of source code. The results help to understand the impact certain choices for identifier construction have on program comprehension.

The primary hypothesis of the study is that full English-word identifiers lead to better source code comprehension. Two other hypotheses are investigated: first, increased work experiences and schooling lead to a better ability to comprehend source code; thus, lowering the value of identifier quality. Second, in support of related studies [10], gender plays a role in confidence but not comprehension.

One motivation for this study comes from a project who's aim is to build tools that allow a programmer to easily assess aspects (*e.g.*, quality) of a large software system. A key focus of these tools is on the identifiers and comments. Although comments tend to be written in a natural language such as English, the same is not necessarily true of identifiers. In order to build tools that exploit high quality identifiers, it is first important to understand the characteristics of identifier quality. In order to ascertain the effect of identifier names on comprehension, this paper reports on a study that investigates the impact of three levels of identifier quality: full words, abbreviations, and single letters. If using full words is a clear advantage over abbreviations, then it will be easier to build tools that extract information from identifiers, for example, through the use of a standard dictionary. However, if abbreviations of full words give an engineer as much information, then tools that consider identifiers must be designed to treat abbreviations.

The study involved over 100 computer scientists including both students and professionals. The size and variety of participants make this study unique. Subjects were asked to describe one of three variants of twelve functions (shown one at a time). The variants differed only in the quality of the identifiers used. Each participant was asked to provide a

free-form written description of the function and a measure of their confidence in understanding the source code. The description provides a qualitative measure of comprehension without leading the subject to possible answers, while the confidence rating provides a quantitative measure of the subject's understanding. In order to access a wide variety of subjects, the study was conducted over the web and email was used to solicit participation. One hundred twenty-eight people participated, ninety-six professionals and thirty-two students.

The remainder of the paper first presents a description of the experimental setup in Section 2, followed by necessary background information in Section 3. This is followed by a discussion of the results and then related work in Sections 4 and 5. Finally, Section 6 summarizes the paper and suggests some places for future work.

2 Experimental Setup

This section describes the experimental design. It first describes the process used to select the twelve functions used in the study. Next the applet used to collect the data and then a summary of the subjects demographic data is presented. Finally, the process of readying the data for analysis is discussed.

2.1 Source Code Selection

The first step in constructing the study was to determine the twelve functions to be used. Two kinds of functions were of interest: algorithms and snippets. The algorithms include "text book" functions such as binary search and quick sort. The snippets were functions taken from production code and included, for example, finding the best move in the game `go` and summing all the debits in an account. Sources for these functions include several algorithm's text books and code available on the world wide web. The initial search turned up about 50 candidate functions. From these, six algorithms and six snippets where chosen for inclusion in the study. The functions ranged in size from 8 to 36 lines of code. Since the focus of the study is on identifiers, comments were omitted from the code the subjects viewed. All information about the purpose of the code came from its structure and its identifiers.

The next task was to create the three variants of each function. First, the full (English) word identifier variant of each function was constructed. The identifier names came from the original programmers if the code had been written with English word identifiers or were chosen by the authors to be particularly meaningful. The single letter variants were created by selecting the first letter of each word in the full-word identifier.

Finally, the abbreviation variants were created based on the full-word variants. Most of the identifiers had common abbreviations (*e.g.*, count → cnt, length → len). Ten of the 63 identifiers (*e.g.*, `current_board`, `target`, and `credit`) had no conventional abbreviation (as known to the authors). In these cases a professional programmer unrelated to the experiment was asked to abbreviate the 10 identifiers. Five of the 10 were the same abbreviations as the authors proposed, three contained less information (*e.g.*, `most_frequent_letter` was abbreviated `mfl` rather than `mfreqlet`). Finally, two abbreviations had minor differences (`scores` was abbreviated as `scrs` rather than `scs` and `credit` was abbreviated `cdt` rather than `cred`. To avoid experimenter bias, the professional programmer's abbreviation were used in the cases of disagreement.

To illustrate the difference in the variants, Figure 1 shows the three variants of the Sieve of Eratosthenes. The top function is the single letter variant. It is expected that comprehension using this variant will be worse than the other variants; thus, this variant is used to provide a baseline. The middle function includes abbreviated identifiers (*e.g.*, `isPriNum`). Finally, the bottom function uses full word identifiers (*e.g.*, `isPrimeNumber`).

2.2 Data Collection

The experiment was administered over the web to allow a geographically diverse group of subjects to take part. A Java applet was used as the user interface to control the viewing of the source code (*e.g.*, to prevent subjects from making use to their browser's "back" button to view the code multiple times). In addition, the applet made the collection of timing data easier. The applet consisted of three main parts: collecting demographic information, presenting the source code and questions for the twelve functions, and collecting final participant feedback.

Subjects began with the a demographics page, which collected the participant's years of computer science schooling, years of computer science related work experience, the title of the last computer science position held, age, and gender. Because the study involved reading code written in the programming languages C, C++, and Java, each subject was also asked to provide their comfort level with each language on a scale of 1 to 5, where 1 indicated low comfort and 5 indicated high comfort.

Each function shown in the middle part of the experiment involved viewing two screens. The first screen displayed the source code of the function. Participants were asked to spend one to two minutes reading the code and not to write anything down regarding the code. The second screen, shown in Figure 2, asked subjects to describe the purpose of the function and to rate their confidence in their understanding of the source code. These two pieces

Single Letter Variant

```
void fXX(bool pn[ ], int l)
{
  int i, f, p;

  pn[0] = false;
  pn[1] = false;
  for (i = 2; i < l; i++)
    pn[i] = true;

  for (p = 2; p < l; p++)
    if (pn[p])
      for (f = p; f * p < l; f++)
        pn[f * p] = false;
}
```

Abbreviatted Variant

```
void fXX(bool isPriNum[ ], int len)
{
  int idx, fac, pri;

  isPriNum[0] = false;
  isPriNum[1] = false;

  for (idx = 2; idx < len; idx++)
    isPriNum[idx] = true;

  for (pri = 2; pri < len; pri++)
    if (isPriNum[pri])
      for (fac = pri; fac * pri < len; fac++)
        isPriNum[fac * pri] = false;
}
```

Full Word Variant

```
void fXX(bool isPrimeNumber[ ], int length)
{
  int index, factor, prime;

  isPrimeNumber[0] = false;
  isPrimeNumber[1] = false;

  for (index = 2; index < length; index++)
    isPrimeNumber[index] = true;

  for (prime = 2; prime < length; prime++)
    if (isPrimeNumber[prime])
      for (factor = prime;
          factor * prime < length; factor++)
        isPrimeNumber[factor * prime]
            = false;
}
```

Figure 1. The three variants of the function that uses the Sieve of Erathosthenes to find prime numbers.

of information were used as the comprehension measures. (As an aside, this confidence rating was misinterpreted by a few subjects, who wrote "I don't know" as the description and gave it a confidence of 5, indicating they were very sure they did not know what the code did; however, most treated their confidence as intended.)

To ensure that each participant saw an even distribution of the three different variants and to ensure that for each question, each variant of the code was seen by a similar number of participants, the sequence of variants shown was randomly taken from three possible sequences. The sequences were created using Latin Squares to ensure that each subject encountered an equal number of each type of question in a balanced fashion. From the data collected, the actual number of responses for each variant was 357, 364, and 366, which indicates that good balance was achieved.

The final screen, seen only by subjects who completed all twelve questions provide space for free-form comments. Participants volunteered such information as their opinion of particular questions, their frustration with the choice of identifier names, and the amount of time that has passed since they last had to read code. One subject observed, "Nice survey. Programs are indeed inherently unintelligible especially for the unexperienced eye."

2.3 Subject Demographics

The subjects who participated in this study include undergraduate and alumni of Loyola College (a convenience sample). In addition, email requests were sent to other institutions' alumni, professional groups, etc. In all 128 participants answered at least one question. Sixty-four others filled in only the demographic information. About 25 percent of the participants can be categorized as students based on the number of years of computer science schooling. The average age of the participants was 30 years with a standard deviation of 11. The average number of years worked was 7.5 with a standard deviation of 8.8. Ten percent of the participants were female. Finally, the average comfort subjects reported for C, C++, and Java on a scale of 1 to 5, were 3.3, 3.4, and 3.6, respectively.

The number of subjects that did not complete the study, known as drop-outs, is depicted in Figure 3. The figure reports the number of participants that stopped after each question. Eighty of the 128 participants or 62.5%, answered all twelve questions. As seen in the figure, all but one person dropped out in the first half of the study.

Given that the functions were shown in same order for all participants, more subjects evaluated the first function than the last one. It is likely that fatigue played a part in dropping out. One subject commented on fatigue multiple times when describing the purpose of the functions. Other factors that may have lead to dropping out include the unexpected

Figure 2. This is a screen shot of the second screen where subjects entered free form text to describe the function they viewed on the previous page. Then a confidence of their understanding of the code was indicated.

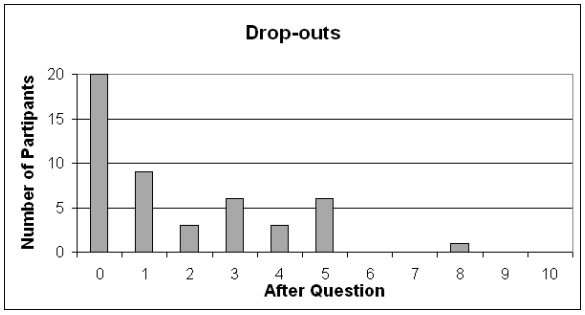

Figure 3. The drop-out rate reported as the number of participants that left the study after completing a particular question.

difficulty and the amount of time required to complete the study.

2.4 Data Preparation

The data preparation involved three primary steps. First, two non-numeric values from the demographic information where replace by approximate numeric values. These two were the age "40+" which was replaced by 45 and the age "old" which was replaced by 60. Both replacements were based on the profession and the number of years of work experience.

Second, the data associated with times that seemed unusually short or long were examined. Most of the extremely short times (less than 10 milli-seconds) came from problems in the interface. In these cases, which numbered 5, the entire question was discarded for that particular subject. Considering short times also uncovered that two subjects who quit the study and then started over at a later time. Since they closed their browser, they began with the first question again and thus, could answer the questions without first analyzing the code. In these cases, the second responses to the functions were discarded and the remaining responses were merged into a single record.

Long times were observed on both screens and appeared to indicate some kind of distraction occurring (*e.g.*, one subject reported that a phone call had been taken). Since these times would adversely effect the statistics, such outliers were treated as missing data, which is common in similar studies. The statistical analysis can readily handle this missing data.

Finally, in addition to cleaning the data, the free-form descriptions were evaluated. Two of the authors independently evaluated each response on a 0 to 5 scale with the following interpretations for each number:

- **5** yes
- **4** mostly yes
- **3** half right
- **2** mostly no
- **1** no
- **0** omitted an answer or reported a problem with viewing the code

For some functions, further directions were agreed on such as for the binary search algorithm, a description was given a 4 if the word binary was omitted from a description that was otherwise correct.

In total, 1087 responses were evaluated. The responses were in random order to avoid any bias by variant. There was agreement between the raters on 78 percent of the responses. To obtain a measure of agreement between the two raters, a κ statistic was computed. The result of 0.71 indicates substantial agreement [7].

3 Background

This section describes the statistical tests used to analyze the study responses. As the data includes repeated-

measures and missing values, due to participants not completing all twelve questions, linear mixed-effects regression models [14] were used to analyze the data. Such models easily accommodate unbalanced data, and, consequently, are ideal for analyzing the study's results. This statistical model allows the examination of important variables that are associated with the various response variables.

The initial model includes explanatory variables and a number of interaction terms. The interaction terms allow the effects of one variable on the response variable to change depending upon the value of another variable. For example, if confidence interacts with gender in a model where rating is the response variable, then the effect of confidence on rating depends on gender (*i.e.*, is different for men and women). Backward elimination of statistically non-significant terms ($p > 0.05$) yields the final model. Note that some non-significant variables and interactions are retained to preserve a hierarchical well-formulated model [9].

The study compares the three variants within each of the twelve questions. With three variants and twelve questions, thirty-six comparison are made. Computing a standard t-value for each comparison and then using the standard critical value increases the overall probability of a type I error. Thus, a Bonferroni's correction is made to the p-values to correct for this multiple testing. In essence each p-value is multiplied by 36 and the adjusted p-value is compared to the standard significance level (0.05) to determine significance.

4 Experimental Results

This section examines the results of the description ratings and the participants confidence in their understanding of the code in the context of the study's hypotheses:

- Full English-word identifiers lead to better source code comprehension
- Increased work experience and schooling lead to a better ability to comprehend source code; thus lowering the value of identifier quality
- Gender plays a role in confidence but not comprehension

A simple comparison of the averages for each variant, as shown Table 4, shows the expected trend for both description ratings and confidence. The single letter variances have considerablely smaller average ratings and confidences, and the full word variants have the best averages. Unfortunately, the existence of interaction between model parameters makes simple statistical statements relating these values difficult.

	description rating			confidence		
variant	sing.	abbr.	full	sing.	abbr.	full
average	3.10	3.72	3.91	3.06	3.55	3.64

Table 1. Mean values calculated for description rating and confidence.

Given the number of explanatory variables that could effect these results, two different linear mixed-effects regression models were used to analyze the results: one simple and one complex. The simple model includes only the effects for question and variant as well as their interaction. It is used to get an initial impression of the data. The more complex model is then used to assess the effects of additional explanatory variables on the response variables.

4.1 Description Ratings

The simple model for description ratings examines how *question*, *variant*, and their interaction, hereafter denoted *question*variant*, affect the description rating. Mixed effects analysis shows that all three variables are important to the description rating. The p-value for the interaction is $<$0.0001. Consequently, the effect of *variant* on the ratings of the descriptions differs among the questions.

Given that the interaction is significant, no trends can be discussed for *variant* or *question* independently. Figure 4 depicts the interaction. First, notice that the single letter line is generally below the other two lines. The only exception is Question 9, quick sort. It may be that this is such a well studied algorithm that the structure of the code was sufficient to determine its purpose.

The three questions where circles appear in Figure 4 show significant differences (the other questions did not exhibit significant differences). When more than one variant occurs in the same circle, it means that there is no statistical difference between those variants. In all three cases, it shows that full word identifiers lead to significantly better description ratings than single letter identifiers. In two cases, abbreviation identifiers lead to significantly better ratings than the single letters. There is never a statistical difference between full words and abbreviations, which means that the subjects who viewed the abbreviations were able to get about as much information out of the identifiers as those that viewed the full word identifiers. However, given that the mean of the full word description ratings were generally higher than the abbreviations, it is possible that with a larger sample size the difference between full word and abbreviation would be statistically significant.

One final interesting observation that comes from this model is that two of the three questions that showed significant differences were snippets, whereas only one was an

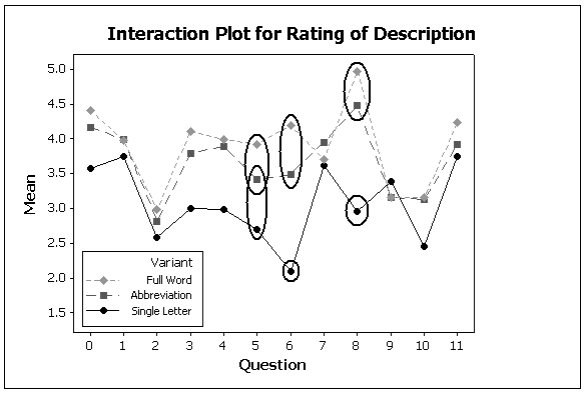

Figure 4. The plot shows the mean ratings based on question and variant, which illustrates the interaction between these two variables. The circles indicate where significant differences occur. If two variants are in the same circle, there is no significant difference between the variants. The plot shows that significant differences occurred in questions 5, 6, and 8. In all other questions where no circles are present, no significant differences were observed between the three variants.

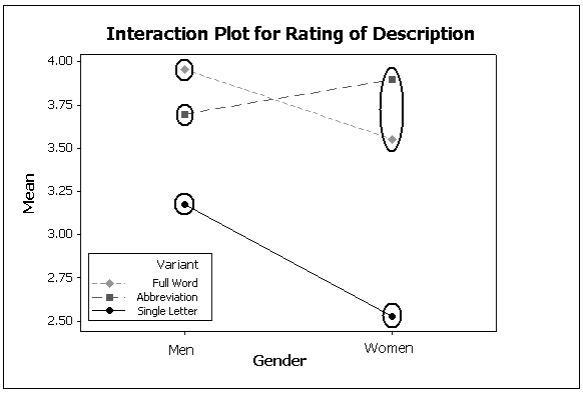

Figure 5. The plot shows the mean ratings based on *gender* and *variant*, which illustrates the interaction between these two variables. The circles indicate where significant differences occur. If two variants are in the same circle, there is no significant difference between the variants.

algorithm. The one algorithm, Question 8, was Sieve of Eratosthenes, which determines whether a number is prime. In this function, the identifier `isPrimeNumber` appeared in the full word variant, which is why it is no surprise that the mean rating was close to 5. In the abbreviation variant, the variable was renamed to `isPriNum`, which along with the structure of the code and other identifiers lead to a mean description rating of about 4.5. In the single letter variant, the variable was named `pn`, which did not enable as many subjects to identify the code correctly. When considering the snippet questions, these should be more similar to the kind of code an engineer would encounter when considering a system. In this case, a third of the snippet questions show significant improvement in description rating when full words are used for identifiers rather than single letters. It can be concluded that the identifier names for non-algorithms is more important than for algorithms.

The second model includes the demographic data collected, the time spent analyzing the code and writing the descriptions, variant, and question characteristics in place of the question number. The model also includes all interactions among these variables. The questions are described using the following attributes: code type (whether snippet or algorithm), number of identifiers in the code (*identifiers*), number of identifiers squared (*identifiers*^{2}), and lines of code.

In terms of demographic information, there were significant interactions between *gender* and *variant* (p = 0.0062) and between *high comfort* in multiple programming languages and *variant* (p = 0.0106). The *gender*variant* interaction shown in Figure 5 reveals that men produced better descriptions for the full word variant, while for women there was no difference between full words and abbreviations. Also, the mean score for men on the single letter variant was 0.75 higher than for women. This may indicate that informative identifier names are more important for women than for men; but that women comprehend more from abbreviations than men do.

In this study, a *high comfort* was equated with a subject indicating a comfort level of 4 or 5 in two of the three programming languages asked about in the demographics section. For the *high-comfort*variant* interaction, participants are divided into groups based on *comfort*. After doing so, it can be seen that variant has a much greater impact on those that have less expertise. With full word identifiers, there is only about a 0.1 difference in the means for the description rating, where the experts have a the slight edge as shown in Figure 6. When considering the difference in means for abbreviations, the gap grows to about 0.3. In the single word variant, the gap is more than 0.6. Although both groups get the most out of full words, this added information is more important to the less experienced programmer. Figure 6 also reveals that for both groups of subjects, there is no significant difference between full words and abbreviations, but both variants are significantly better than the single letter variant.

In terms of question characteristics, all four characteristics are significant variables in the description rating. *Code type*, having p-value < 0.0001, reveals that algorithms had higher description ratings. This could be because algo-

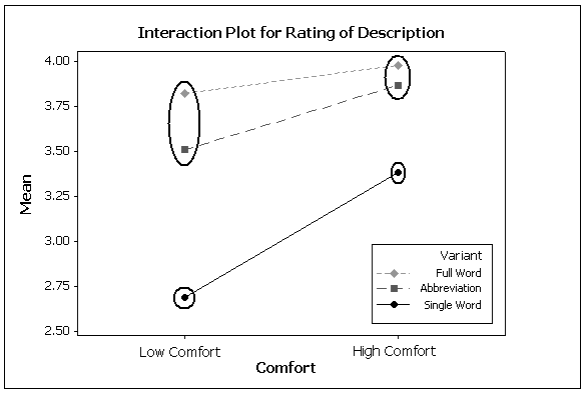

Figure 6. The plot shows the mean ratings based on *comfort* and *variant*, which illustrates the interaction between these two variables. The circles indicate where significant differences occur. If two variants are in the same circle, there is no significant difference between the variants.

rithms have well known names; thus, it is easier for the subjects to describe them, or because most participants had seen them before, the code was easier to identify. The model shows that description rating increases with the lines of code, indicating that more code improves comprehension, at least when lines of code are in the range of 8 to 36. The *lines of code*'s p-value was 0.0013. The variables *identifiers*, and *identifiers*^{2} were also significant as well as their interactions with *variant*. The interactions *identifiers*variant* and *identifiers*^{2}**variant* both had p-values of 0.0001. From a plot showing these effects, the optimal number of identifiers for full words and abbreviations hovers around 5. Function ratings with greater or fewer identifiers continuously decline as the distance from 5 grows. The single letter variant has a different behavior, with description rating decreasing as the number of variables increase. Given that the single letters, provide little domain information, it is not surprising that its trend is different.

In terms of time, the amount of time spent on the second screen is significant, but the effect is complex. When a plot of the *ratings* vs. *time* on the second screen was examined, it was noticed that initially there is a rapid increase in the ratings. At about 16,000 milli-seconds (16 seconds), the ratings level off and thereafter tend to gradually decline with increasing time. To account for this pattern, an indicator variable is defined that is 0 before 16,000 and 1 thereafter. This allows the association between *rating* and *time* to be different before and after 16,000 ms. The model contains this indicator variable, the variable $\ln time$, and the interaction between the indicator variable and $\ln time$.

Each of these terms is statistically significant in the final model. Using the parameters from the fitted model, the results show the same sharp increase in ratings followed by a gradual decline. This behavior can be accounted for by the fact that some subjects simply clicked on their (low) confidence and continued on to the next question. For those subjects that attempted to describe the function, those that understood the code created the description more quickly than those that did not.

There were two other significant variables in this model. One was the *confidence* subjects had in their understanding of the code with a p-value < 0.0001. This showed that description rating increased with *confidence*. This is not a surprising result since both confidence and description rating are measures of comprehension. The other was the *length* of the description, which was calculated from the number of characters in the description. This variable had a p-value of 0.0077 and showed that rating increased with the *length*. This indicates that subjects who wrote longer descriptions understood the function better.

In summary, the data supports the hypothesis that subjects wrote the best descriptions for functions with full word identifiers. It also shows that gender plays a role in the comprehension of code as does programming expertise when it comes to interpreting abbreviations. In addition, it finds that 5 is an optimal number of (domain information carrying) identifiers to be considered at one time. An interesting side note is that work experience and schooling are not significant factors for writing correct descriptions.

4.2 Confidence

Like the simple model for description rating, the simple model for confidence examines how *question*, *variant*, and their interaction affect the confidence reported by the subject. Confidence is an important reflection of comprehension as it reports the subjects belief in their understanding. All three variables are important to the model. The p-value for the interaction is 0.0130. Consequently, the effect of variant on the ratings of the descriptions differs among the questions.

The interaction plot is shown in Figure 7. Subjects generally had less confidence in their comprehension of code with single letters than the other variants, and most often the highest confidence came from code with the full word identifiers. However, significant differences only occurred in four questions. In these questions, the significant difference came between single letters and full words. It is interesting to note that the three questions with significant differences in description rating also had significant differences in confidence, which may indicate that these two response variables were measuring similar information, namely comprehension.

The expanded model for confidence began with the same explanatory variables as that for description rating: the de-

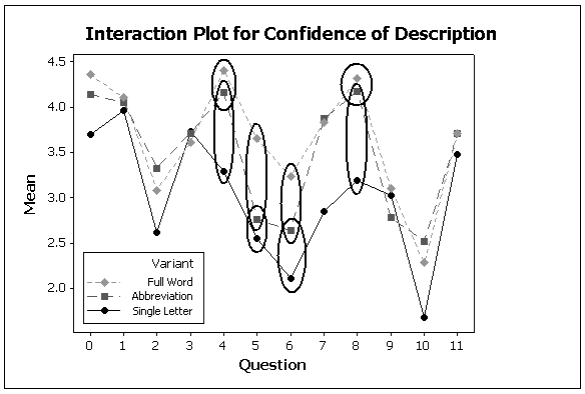

Figure 7. The plot shows mean confidence values based on question and variant, which illustrates the interaction between these two variables. The circles indicate where significant differences occur. If two variants are in the same circle, there is no significant difference between the variants. The plot shows that significant differences occurred in questions 4, 5,6, and 8. In all other questions, no significant differences were observed.

mographic data, the time spent analyzing the code and writing the descriptions, variant, and question characteristics. However, this model found different significant variables.

In terms of demographic information, significant variables are the number of *years worked* ($p = 0.0138$), number of *years in school* ($p < 0.0001$), and *gender* ($p = 0.0154$). The data reveal that confidence increases both with number of *years worked* and number of *years in school*. Neither of these two results is unexpected. When considering gender, females rate their confidence 0.4 lower than males. This pattern has been observed before [10]. At first glance, it may appear that there are too few female participants to draw such conclusions; however, in order for a statistical different to occur, a very large difference must be observed, making the result noteworthy.

In terms of question characteristics, two of the four characteristics used were significant: *code type* ($p = 0.0004$) and *number of identifiers* ($p < 0.0001$). Lines of code was not found to be significant. It is observed that algorithms lead to higher confidence and more identifiers leads to lower confidence. This second conclusion is rather unintuitive. It is difficult to say why this occurred.

In terms of time, the time spent on the first screen analyzing the code and time spent on the second screen answering the questions were both significant. However, the relation is non-linear. Confidence increases as the time spent on the first screen increases from 0 to about 15,000 milli-seconds. After that point confidence slowly decreases as time increases. This indicates that there is an optimal amount of time that one can spend analyzing source code and after than point there are diminishing returns. The same pattern occurs on the second screen; however, the cut-off time is 16,000 milliseconds, rather than 15,000. Again, some time is necessary, but too much time indicates lower confidence.

Two other variable were found to be significant: *description length* ($p = 0.0012$) and *description rating* ($p < 0.0001$). Confidence increases with both increased *description length* and *description rating*. The increase in description length shows that subjects who have more to say are more confident. When considering description rating, it is not surprising that it is a significant factor since confidence is a significant factor in the description rating model.

In summary, the hypothesis that confidence correlates with years of experience is supported by the data. The hypothesis that graduated participants do better than those yet to graduate is supported by the fact that more schooling leads to higher confidence. Also, gender does play a role in confidence, with women generally having lower confidence than men.

4.3 Summary of Results

Although description rating and confidence are highly correlated, the models generated for each provide different information and insights. For example, work experience and years of schooling play a significant role in confidence, but not in description rating. Although gender plays a role in both models, that role is different. In terms of confidence, it is shown that women rate their confidence lower than men. However, in terms of description rating, men perform significantly higher on full words than on abbreviations, where as there is no statistically significant difference observed for women. Finally, subjects tend to have higher ratings and higher confidence in their ability to understand algorithms than the snippets of production code.

4.4 Threats to Validity

In any empirical study, it is important to consider threats to validity (*i.e.*, the degree to which the experiment measures what it claims to measure). There are four types of validity relevant to this research: external validity, internal validity, construct validity, and statistical conclusion validity.

External validity, sometimes referred to as selection validity, is the degree to which the findings can be generalized to other organizations or settings. In this experiment, selection bias is possible as the selected functions and participants may not be representative of those in general; thus, results from the experiment may not apply in the general case. Careful selection of the functions mitigates the impact their selection. Given the demographic data, the subjects seem

fairly representative of the computer science community at large.

Second is the threat to internal validity: the degree to which conclusions can be drawn about the causal effect of the explanatory variable on the response variable. Statistical associations do not imply causation. Though, given the experimental setup that subjects are assigned to questions, one should be able to infer that differences between the question variants are due to the different types of identifiers. One threat comes from the loss of participants in the beginning of the study, known as attention effects. Given that there are several reasons an individual may have discontinued participation in the study, it is thought to be unlikely that the loss is systematically correlated with conditions. Other potential threats to internal validity, for example, history effects and subject maturation [11] are non issues given the short duration of the experiment. Selection effects were addressed by gathering a representative sample of programmers, and by choosing functions that represented various types of code. Finally, it is possible that exposure to early questions had an effect on responses to later questions. No evidence of this was found in the participants responses.

Construct validity assesses the degree to which the variables used in the study accurately measure the concepts they purport to measure. As human assessment of quality is rather subjective, it is possible that some other aspect of the code assessed affected participants' responses. The parallels between the models for the description rating and confidence suggest that this is not a serious concern.

Finally, a threat to statistical conclusion validity arises when inappropriate statistical tests are used or when violations of statistical assumptions occur. The models applied to this data are appropriate for analyzing unbalanced repeated-measures data, so that the conclusions drawn from the statistics should be valid.

5 Related Work

There is ongoing interest in naming identifiers. Although educators tend to stress the importance to meaningful identifier names, this is not universally valued. Sneed observes that "in many legacy systems, procedures and data are named arbitrarily $\cdots$ programmers often choose to name procedures after their girlfriends or favorite sportsmen" [12]. However, simply because un-informative identifiers may exist, does not mean that the code using them is of good quality. Caprile and Tonella state that "identifier names are one of the most important sources of information about program entities" [4].

Given the importance of identifier naming, several research projects have considered the issue of identifier naming conventions. Naming conventions are important because "studies of how people name things (in general not just in code) have shown that the probability of having two people apply the same name to an object is between 7% and 18%, depending on the object" [2]. Anquetil and Lethbridge [1] define what it means to have a "reliable naming convention". Deissenböck and Pizka [5] create a formal model for concepts and names which is used to determine reliable names. Caprile and Tonella [3] use a grammar to define the naming convention and use the grammar to find semantic meaning in the words. In addition to naming conventions, the informativeness of identifiers has been examined by Takang et al. [13].

Anquetil and Lethbridge hypothesized that a "*naming convention* is reliable if there is an equivalence between the name of the software artifacts and the concepts they implement" [1]. This hypothesis was studied through the examination of record definitions. In the legacy code they studied, it was evident that a naming convention existed because records with similar names had similar fields.

Deissenböck and Pizka create a formal model based on bi-jective mappings between concepts and names. The idea is that within a given program a concept should always be referred to by the same name. They introduce an "identifier dictionary" and provide a tool that will help maintain consistent naming throughout the lifetime of a software project [5]. Deissenböck and Pizka argue that naming conventions are needed to enforce consistency and to provide guidelines about the mechanics of turning a concept into a name. With such guidelines, names should contain enough information for an engineer to comprehend the precise concept.

Caprile and Tonella analyze function identifiers by considering their lexical, syntactical, and semantical structure. They break identifiers into word segments and then use a grammar to find semantic meaning in the words. By following a grammar, Caprile and Tonella anticipate improvements in the readability, understandability and, more generally, maintainability of a program [3].

Others have attempted to determine the informativeness of identifiers including Takang et al. [13]. The Takang study compared abbreviated identifiers to full-word identifiers and uncommented code to commented code. The abbreviations were created from the first two letters of the English word (*e.g.*, `CalculateNumericScore` was abbreviated as `CaNuSc`). Given that it may be difficult to recognize the base word in the abbreviation, these abbreviations are more similar to the single letter identifiers used in the study reported in this paper. Subjects for the study consisted of first year students, which means that results may not be applicable to professionals since none participated in the study. To assess understanding of the code, multiple choice test scores and participant's subjective scores were used. The objective test scores showed that commented programs are more understandable than non-commented programs. The sub-

jective scores showed that programs that contain full word identifiers are more understandable than those with abbreviated identifiers; however, nothing can be concluded about more informative abbreviations. Also only a single program was used in the analysis, so it is more difficult to generalize the results to other types of programs in other domains. The study was unable to show any improvement from both full word identifiers and comments.

6 Summary and Future Challenges

The study described in this paper shows that better comprehension is achieved when full word identifiers are used rather than single letter identifiers as measured by description rating and confidence in understanding. It also shows that in many cases abbreviations are as useful as the full word identifiers, although this is more true for women than for men. Gender also impacts confidence, where men generally report higher confidence. In addition, it shows that work experience and education play an important role in an engineer's confidence in understanding a piece of code, but not in his ability to write description of what that code is accomplishing.

Given the results of the study, it is clear that tools assessing identifier quality need to be able to make use of abbreviations. A key next step for such tools is to automatically associate meanings with abbreviations. This might be possible using machine learning techniques [8] to select key components from the documentation based on higher-level concepts.

7 Acknowledgements

The expertise of Jim Glenn and Barbara Vann were invaluable in preparing this study. Special thanks to all the participants as this work would not be possible without their time. This work is supported by National Science Foundation grant CCR-0305330.

References

[1] N. Anquetil and T. Lethbridge. Assessing the relevance of identifier names in a legacy software system. In *Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative Research*, Toronto, Ontario, Canada, November 1998.

[2] G. Butler, P. Grogono, R. Shinghal, and I. Tjandra. Retrieving information from data flow diagrams. In *Working Conference on Reverse Engineering*, pages 84–93, November 1995.

[3] B. Caprile and P. Tonella. Nomen est omen: analyzing the language of function identifiers. In *Working Conference on Reverse Engineering*, pages 112–122, Altanta, Georgia, USA, October 1999.

[4] B. Caprile and P. Tonella. Restructuring program identifier names. In *ICSM*, pages 97–107, 2000.

[5] F. Deißenböck and M. Pizka. Concise and consistent naming. In *Proceedings of the 13th International Workshop on Program Comprehension (IWPC 2005)*, St. Louis, MO, USA, May 2005. IEEE Computer Society.

[6] D. Knuth. *Selected papers on computer languages*. Stanford, California: Center for the Study of Language and Information (CSLI Lecture Notes, no. 139), 2003.

[7] J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33, 1977.

[8] T. Mitchell. *Machine learning*. WCB McGraw-Hill, 1997.

[9] C. Morrell, J. Pearson, and L. Brant. Linear transformation of linear mixed effects models. *The American Statistician*, 51:338–343, 1997.

[10] P. De Palma. Why women avoid computer science. *Communications of the ACM*, 44(6), 2001.

[11] D. Sjøberg, J. Hannay, O. Hansen, V. Kampenes, A. Karahasanovic, N. Liborg, and A. Rekdal. A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering*, 19(4):379–389, 1993.

[12] H. Sneed. Object-oriented cobol recycling. In *3rd Working Conference on Reverse Engineering*, pages 169–178. IEEE Computer Society., 1996.

[13] A. Takang, P. Grubb, and R. Macredie. The effects of comments and identifier names on program comprehensibility: an experiential study. *Journal of Program Languages*, 4(3):143–167, 1996.

[14] G. Verbeke and G. Molenberghs. *Linear mixed models for longitudinal data*. Springer-Verlag, New York, second edition, 2001.