Effective Identifier Names for Comprehension and Memory

Dawn Lawrie, Christopher Morrell, Henry Feild, David Binkley

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., identifiers that 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? What is the effect of longer identifiers on limited short term memory capacity?

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 and then recall identifiers that appeared in each function. The functions used three different 'levels' of identifiers: single letters, abbreviations, and full words. Responses allow the extent of comprehension associated with the different levels to be studied along with their impact on memory.

The functions used in the study include standard computer science text-book algorithms and 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 using full words and abbreviations. When considered in the light of limited human short-term memory, well chosen abbreviations may be preferable in some situations since identifiers with fewer syllables are easier to remember.

Keywords Program Comprehension, Software Quality, Quality Assessment

D. Lawrie
Computer Science Department, Loyola College, 4501 N. Charles St., Baltimore, MD 21210-2699, USA

C. Morrell
Loyola College

H. Feild
Loyola College

D. Binkley
Loyola College

1 Introduction

Conventional wisdom says that choosing meaningful identifier names improves code quality and thus the ability of future engineers to manipulate the code [9]. Quality is important to all software engineering projects as it effects the bottom line: lower quality leads to higher costs [16]. Quality is particularly important in safety critical applications where the 'cost' of a failure is considerably higher. Software quality is impacted by program features such as code structure and identifier naming. Of particular interest herein is the key role that identifier naming plays in code quality.

Further motivation for studying identifiers comes from a number of prior studies. For example, Rilling and Klemola observe "In computer programs, identifiers represent defined concepts"[15], while Caprile and Tonella point out that "Identifier names are one of the most important sources of information about program entities"[5]. Another motivation for studying identifiers comes from Deissenböck and Pizka who note that "research on the cognitive processes of language and text understanding shows that it is the semantics inherent to words that determine the comprehension process"[7]. Finally, Takang et al. note "The quality of identifier names is an issue that merits closer consideration and exploration in its own right"[19].

As a relatively young field, computer science is still guided by a collection of commonly held, but often unsubstantiated, beliefs. Through empirical study one can ascertain the veracity of such beliefs.

This paper reports on a study aimed at better understand the effect of identifier makeup on a programmer's ability to manipulate code. For example, the ability to comprehend source code or the ability to recall particular identifiers. One hundred twenty-eight participants took part in the study. The number and variety of participants make this study unique. Furthermore, there is sufficient diversity in the subject population to allow the results to broadly explain the impact that identifier construction choices have.

The study confirms the belief that identifiers made up of well formed abbreviations and full words lead to higher quality software; however, excessively long identifiers overload short term memory, negatively impacting program comprehension. Thus, a balance between information content and recall ability must be struck. This suggests careful evaluation of identifier naming conventions. For example, identifier prefixes potentially improve comprehension by providing ancillary information about the identifier. However, they also make the identifier longer and thus have the disadvantage of contributing to the overcrowding of short-term memory.

The paper makes three primary contributions. First, it presents data from a empirical study that involves a considerably larger and more diverse selection of programmers than similar past studies. Second, the paper interprets the results of statistical models describing the collected data using five hypotheses. The two most important of these relate to the impact of identifier quality on programmer comprehension and to programmer recall ability. Finally, combining the two results, the paper discusses their impact on variable naming in programming. For example, the results can be used to inform naming convention choices.

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 Design

This section presents the experimental design. It first provides an overview of the study's five hypotheses and then describes the process used to select and prepare the source code for the study. Next, the experiential setup and potential threats to the validity of the experiment are considered. The final two subsections summarize subject demographics, and finally, the process of readying the data for analysis.

2.1 Overview

Two primary hypotheses are studied:

(1) Full natural-language identifiers and well-constructed abbreviations lead to better source code comprehension than less informative identifiers.
(2) Full natural-language identifiers and well-constructed abbreviations lead to better programmer recall than less informative identifiers.

Three additional hypotheses are also investigated:

(3) Increased work experience and schooling lead to a better ability to comprehend source code.
(4) In support of related studies [14], gender plays a role in confidence but not comprehension.
(5) Shorter meaningful identifiers are easier to remember than longer meaningful identifiers.

To investigate these hypotheses a two part study was conducted. The first part is aimed at ascertaining the effect of identifier quality on comprehension. It investigates the impact of three levels of identifier quality: full words, abbreviations, and single letters. The second part of the study considers the ability of an engineer to recall identifiers and thus investigates the impact of memory and recall on program comprehension [8].

2.2 Source Code Preparation

The first step in constructing the study was to select and then prepare twelve functions. Two kinds of functions were determined to be of interest: algorithms and snippets. Algorithms, taken from text books and course lecture notes, include functions such as binary search and quicksort. Snippets, taken from production code available from the world wide web, include functions for finding the best move in the game go and summing all the debits in an account.

The initial search revealed about fifty candidate functions. From these, six algorithms and six snippets where chosen for inclusion in the study. The selection was based on features of the code. For example, the selected functions ranged in size from eight to thirty-six 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: full-word, single-letter, and abbreviation. The variants differed only in the quality of the identifiers used. First, the full (English) word variant of each function was constructed. The identifier names came from the original programmers if the code had been written with full-word identifiers or were chosen by the authors to be particularly meaningful. Then, the single-letter variants were created by selecting the first letter of each word in each 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 ten identifiers. Five of the ten were the same abbreviations as the authors proposed, three contained less information (*e.g.*, `most_frequent_letter` was abbreviated `mfl` rather than `mfreqlet`), and two 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 cases of disagreement.

To illustrate the difference in the variants, Figure 1 shows the three variants of the algorithm Sieve of Eratosthenes. The top function is the single-letter variant. It is expected that comprehension using this variant will

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;
}
```

Abbreviated 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;
}
```

Fig. 1. The three variants of the algorithm Sieve of Eratosthenes, which finds prime numbers.

be worse than the other variants. The middle function includes abbreviated identifiers (*e.g.*, `isPriNum`). Finally, the bottom function uses full-word identifiers (*e.g.*, `isPrimeNumber`).

2.3 Experimental Setup

This section describes the user interface used to conduct the study. There are three main phases in the interface: first, the collection of demographic information; second, the presentation of and questions concerning the twelve functions; and third, a solicitation of comments from participants. The implementation of the GUI and the different phases are discussed in detail below.

The experiment was setup to be conducted over the web and thus 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 of their browser's 'back' button to view the code multiple times). In addition, an applet simplifies the collection of timing data.

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

The middle part of the experiment involved showing the twelve functions. Three screens were used for each function. Based on previous memory-based experiments, the order of these screens is important [8]: the first shows the subject the treatment, which is followed by a "memory clearing" activity, and then finally the recall activity.

In this experiment, the first screen displayed one of the three variants of the source code. 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 on the left of Figure 2, had two purposes. First, it served as the memory clearing activity. Second, it provided two pieces of information used in the comprehension part of the study. It asked subjects to provide a free-form written description of the purpose of the function and to rate their confidence in their description. (In a few cases the confidence rating was misinterpreted. Participants wrote "I don't know" as the description but gave a confidence of 5, indicating they were very sure they did not know what the code did; however, most treated confidence as intended.) The description provides a qualitative measure of comprehension without leading the subject toward potential answers, while the confidence rating provides a quantitative measure of the subject's understanding. Motivation for including both is found in the following observation of Takang et al.: "the subjective nature of 'quality' suggests that the quantitative studies, which characterize this area, are less appropriate than studies focusing on qualitative measures"[19].

Finally, the third screen, shown on the right of Figure 2, presented the recall activity. It listed six possible identifiers. Subjects were asked to select those identifiers that they recalled appearing on the first screen. The actual number of identifiers from the source varied from one to four, but was held constant for a given function no matter which variant the subject saw. Thus, the list of identifiers was dependent on both the function and the variant viewed by the subject. For each list, there was at least one wrong answer of the correct variant. Other wrong answers included variables used in other variants and names associated with the domain of the function.

To ensure that each participant saw an even distribution of the three different variants in a balanced fashion and to ensure that, for each question, each variant was seen by a similar number of participants, the sequence of variants shown was randomly taken from three possible sequences created using Latin Squares. 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, provides 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 remarked, "Nice survey. Programs are indeed inherently unintelligible especially for the unexperienced eye." Another subject wrote, "It made a difference if I studied the variables in addition to the code fragment. Most times I only studied the code – a few I reviewed the variables and that increased my *confidence* in naming the ones present."

2.4 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 of this bias in terms of the functions used in the study. Given the demographic data, the subjects seem fairly representative of the computer science community at large.

Second, three threats to internal validity, the degree to which conclusions can be drawn about the causal effect of the explanatory variable on the response variable, are considered. First, statistical associations do not imply causation; though, given the experimental setup, one should be able to infer that differences between the question variants are due to the different types of identifiers. Second, attrition effects occur with the loss of participants during the treatment. There are several reasons an individual may have discontinued participation in the study; however, no evidence that the loss is systematically associated with experiential conditions was suggested. Third, learning effects occur where exposure to early questions have an effect on responses to later questions. No evidence of this was found in the participants responses. Finally, other potential threats to internal validity, for example, history effects and subject maturation [2] are non-issues given the short duration of the experiment.

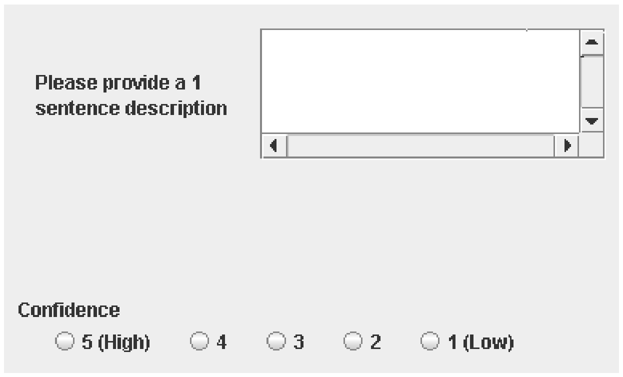
Construct validity assesses the degree to which the variables used in the study accurately measure the concepts they purport to measure. As human assessment of quality is rather subjective, it is possible that some other aspect of the code affected participants' responses. The parallels between the models 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 are appropriate for analyzing unbalanced, repeated-measures data, so that the conclusions drawn from the statistics should be valid.

2.5 Subject Demographics

This section summarizes demographic data on the study's subjects. These include current students and alumni of several colleges and those reached via email sent to various professional groups. In all, 192 people started the survey. Of these 64 filled in only the demographic information. Thus, 128 participants answered at least one question. Eighty of these completed all twelve questions. Participants ranged from students (about 25 percent) to professionals with over forty years of experience. 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.

All studies concerning human subjects must consider drop-outs – subjects who do not complete the study. The drop-out rate for this study, depicted in Figure 3, 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 drop out occurred during the first half of the study. 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 difficulty and the amount of time required to complete the study.



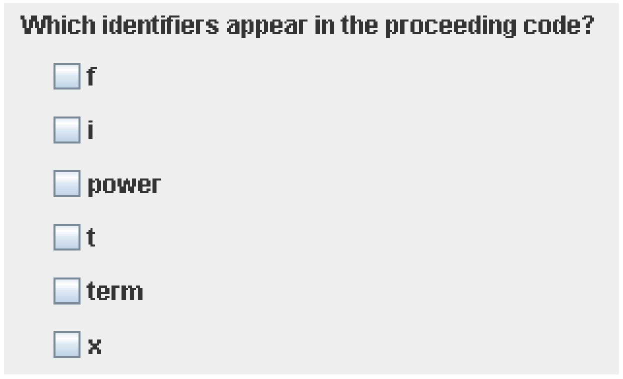


Fig. 2. Screen shots from the data collection applet. The left image shows the second screen where subjects entered a free-form description and rated their confidence in their understanding. The right image shows the third screen where subjects selected the identifiers they thought appeared in the code.

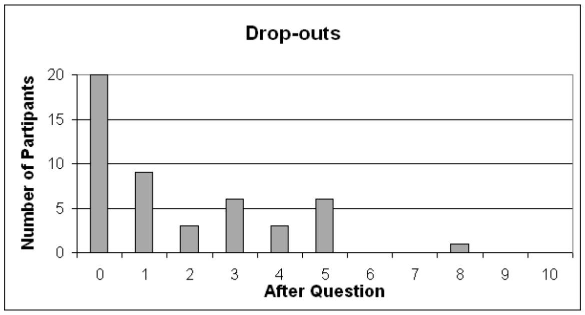


Fig. 3. The drop-out rate – the number of participants that left the study after completing a particular question.

2.6 Data Preparation

The data preparation involved three primary steps. First, two non-numeric values from the demographic information were replaced 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 two subjects who quit the study and then re-started it at a later time. Since they closed their browser, they began with the first question again and thus could answer some questions without first analyzing the code. In these cases, the second responses to the questions were discarded and the remaining responses were merged into a single record.

Long times were observed on all 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.

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

5	correct
4	mostly correct
3	half right
2	mostly incorrect
1	incorrect
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 in random order to avoid any bias by variant. There was direct agreement between the raters on 78% of the responses. In the remaining cases, the average rating was used. To obtain a statistical measure of overall agreement between the two raters, a κ-statistic was computed. The result of 0.71 indicates substantial agreement [10].

Finally, the responses to the third screen were analyzed. The value extracted for use in the subsequent analysis was the percent of correct answers for identifiers that appeared in the code (hereafter referred to as *percent correct in* *source*, *PCIS*). Using a proportion rather than the actual count facilitates comparing questions as the number of correct answers varies by question. It also better supports the evaluation of subject recall ability. Analysis of wrong answers is also interesting as it supports the understanding of the conditions that lead to mistakes. Such an analysis is also conducted.

3 Background

This section provides some background related to the memory aspects of the study and describes the statistical tests used to analyze the data collected.

3.1 Memory

The memory portion of the study was inspired by the work of Jones who investigated the consequences of limited capacity short-term memory on subjects performance in tasks related to the comprehension of short sequences of code [8]. The study presented herein includes a similar focus on the recall of identifiers; however, there are two significant differences between the experimental setup of the two studies. First, "real world" code was used in this study rather than a sequence of assignment statements. Second, the identifiers in this study were inspired by the task, whereas the identifiers in the Jones study, which were comprised of an arbitrary grouping of words, were focused on investigating memory needs. For example, in this study, participants saw the identifiers moves, currentBoard, result, bestScore, and currentScore. In the Jones study, groupings were chosen based on syllable count (a proxy for their memory demand). As an example, participants saw the three syllable identifiers prevented, liberation, and conception.

From a memory perspective, the key difference between the two is that task-inspired identifiers allow for possible ties to long-term memory. Thus, in this study participants could rely on their understanding and recall of the code's purpose in addition to their ability to recall the actual identifiers. In contrast, the Jones study participants had no additional information apart from their memory. Although this makes the results of the Jones study easier to interpret, they are also more distant from the program comprehension process. Together these two studies compliment each other and provide a better picture of the influence that identifier construction and memory requirements have on comprehension.

3.2 Statistics

As the data includes repeated-measures and missing values (*e.g.*, due to participant drop out) linear mixed-effects regression models were used to analyze the data [20]. Such models easily accommodate unbalanced data, and, consequently, are ideal for this analysis. These statistical models allow the identification and examination of important explanatory variables associated with a given response variable.

The construction of a linear mixed-effects regression model starts with a collection of explanatory variables and a number of interaction terms. The interaction terms allow the effects of one explanatory variable to differ depending upon the value of another explanatory 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 hierarchically well-formulated model [13]. Therefore, individual p-values for terms participating in an interaction are not reported.

When interpreting the mixed-effects models described in the next section, graphs are often used to illustrate significant effects in the model. However, when the models have more than two explanatory variables (which most do) it is not pragmatic to graph all the variables. Thus, when plots are constructed, variables not being discussed are set to their variant-specific means or, in the case of categorical variables, a representative value.

In addition to linear mixed-effects regression, when a basic comparison of distributions without considering explanatory variables is needed, Friedman's test is used. This test is often used as an alternative to the parametric repeated measures ANOVA where the assumption of normality is not acceptable. The test is used to detect differences in treatments across multiple test attempts. The procedure involves ranking each block together, then considering the values of ranks by columns.

The study requires many instances of multiple comparisons. For instance, when comparing three variants and twelve questions, thirty-six comparisons 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, Bonferroni's correction is made to the p-values to correct for multiple testing. In essence each p-value is multiplied by the number of comparisons and the adjusted p-value is compared to the standard significance level (0.05) to determine significance.

4 Experimental Results

This section first formalizes the studies five hypotheses. Statistical analysis of the data collected is then used to build mixed effects regression models for *description ratings*, *confidence*, and percent correct in source (*PCIS*). An analysis of wrong answers recalled is presented in Section 4.5. Finally, in Section 4.6, results from the analysis of these models is used to accept or reject each of the five hypotheses.

The five hypotheses consider comprehension, memory, experience and education, gender, and identifier length, and are formalized as follows.

Comprehension Hypothesis

H_0: There is no difference in comprehension among different identifier variants.

H_a: Full word identifiers and abbreviations lead to better source code comprehension.

Memory Hypothesis

H_0: The ability to recall an identifier is the same for all variants.

H_a: It is easier to recall abbreviations and full words than identifiers of the single-letter variant.

Experience and Education Hypothesis

H_0: Work experience and schooling have no impact on comprehension.

H_a: Increased work experience and schooling lead to a better ability to comprehend source code.

Gender Hypotheses

H_{0_1}: Gender plays no role in confidence.

H_{a_1}: Gender plays a role in confidence.

H_{0_2}: Gender plays no role in ability to describe code.

H_{a_2}: Gender plays a role in ability to describe code.

Identifier Length Hypothesis

H_0: The length of meaningful identifiers has no impact on memory.

H_a: Longer meaningful identifiers are harder to remember than shorter ones.

4.1 Initial Comparison

The statistical analysis begins with an initial simple comparison of means for each *variant* for each of the three response variables: *description ratings*, *confidence*, and *PCIS*. Subsequently, mixed effects regression models are considered. The averages for each variant are shown in Table 1.

The analysis was conducted in two (equivalent) ways. First, a mixed-effects model was fit with only variant as a factor. Least squares means are used to perform pairwise comparisons of the three variants with a Bonferroni correction. Second, a two way randomized block design was used with subject as a random factor. This is the simplest repeated measures analysis. Bonferroni multiple comparisons of the three means were conducted.

	Variant			
	single letter	abbre-viation	full word	p-value
Description Rating	3.10	3.72	3.91	$<.0001$
Confidence	3.07	3.55	3.68	$<.0001$
PCIS	0.72	0.81	0.81	$<.0001$

Table 1. Means and p-values for *description rating*, *confidence*, **and** *PCIS* **by variant.**

Both approaches give the same results. For all three variables, full word and abbreviations did not differ significantly but single letter were significantly lower then the other two variants. The analyses for all three response variables provides limited support for the alternative Comprehension and Memory Hypotheses.

4.2 Description Rating Model

The next three subsections consider statistical models for *description rating*, *confidence*, and *PCIS*. Each subsection presents two mixed effect regression models: first a *simple* model and then a *complete* model. In each case, the simple model is first constructed to gain an initial impression as to the influence of *question*, *variant*, and their interaction, hereafter denoted *question*variant*, on one of the three response variables: *description rating*, *confidence*, and *PCIS*. Second, the complete model is generated to assess the effects of four general categories of explanatory variables on the given response variable. The first category includes demographic information such as the *gender*, (language) *comfort*, and *years-worked*. The second category includes question characteristics, which includes *code type* (*snippet* or *algorithm*), *lines of code*, number of identifiers in the code (*identifiers*), and number of identifiers squared (*identifiers*^{2}) – included after a graphical inspection of the data revealed a quadratic shape. The third category includes the time spent analyzing the code, writing the descriptions, and for memory, the time spent recalling identifiers. The fourth category includes answer characteristics, which, for example in the case of description rating, include the variables *confidence* and *length of description*.

In the complete model, the variables from the second category are used in place of *question* to provide a finer level of granularity; consequently, *question* is not included in these models. Finally, the complete model includes the variable *variant* and its interactions with the other variables. Because each model contains significant interactions, *variant* is always discussed in conjunction with other variables.

The simple model for description ratings examines how *question*, *variant*, and their interaction, *question*variant*, effect *description rating*. Mixed effects analysis shows that all three factors are statically significant. Because the in-

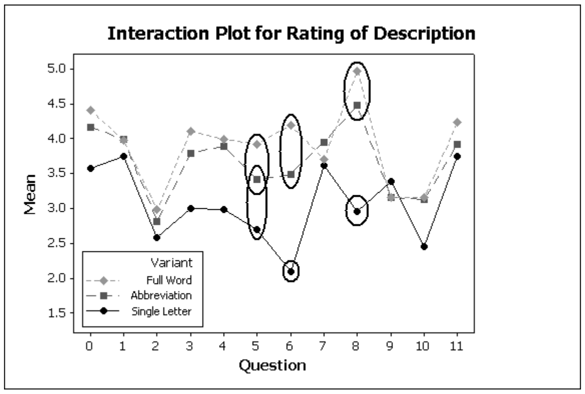


Fig. 4. The interaction between *question* **and** *variant* **when predicting the mean** *description rating***. Significant differences occurred in Questions 5, 6, and 8 where circled points separate significantly different means.**

teraction is significant, only the p-value for the interaction ($<$0.0001) is relevant. The significance of the interaction means that the effect of *variant* on *description rating* differs by *question*.

Given that the interaction is significant, no trends can be discussed for *variant* or *question* independently. Figure 4 depicts a plot of the two variables illustrating the interaction. First, notice that the single-letter line is generally below the other two lines. Interestingly, there is one exception with Question 9, quicksort. It appears that this is such a well-studied algorithm that the structure of the code alone is sufficient to determine its purpose.

The three questions where circles appear in Figure 4 show significant differences (the other questions did not exhibit statistically significant differences). When more than one variant appears in the same circle, it means that there is no statistical difference between those variants. In all three cases, full-word identifiers lead to significantly better description ratings than single-letter identifiers. In two cases, abbreviated identifiers also lead to significantly better description 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 appear to get 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 is generally higher than the abbreviations, it is possible that with a larger sample size the difference between full word and abbreviation would become statistically significant. In conclusion, the simple model provides evidence that the null hypothesis for the Comprehension Hypothesis should be rejected, and, the alternative hypothesis that abbreviations and full words lead to better comprehension than single letters, accepted.

One final interesting observation that comes from this model is that only one of the three questions that showed significant differences was an algorithm, while two were snippets. The one algorithm, Sieve of Eratosthenes (Question 8) determines whether a number is prime. In this function, the identifier `isPrimeNumber` appeared in the full-word variant, which partially explains the higher mean description rating. In the abbreviation variant, the variable was renamed to `isPriNum`, which (along with the structure of the code and other identifiers) appears sufficient for most participants to correctly comprehend the purpose of the code. This led to a mean description rating of about 4.48 (slightly less that the 4.98 when using full-word identifiers). In the single-letter variant, the variable was named `pn`, which did not enable as many subjects to identify the purpose of the code correctly.

The snippet questions should be more representative of the code that an engineer would encounter. 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. These observations indicate that the identifier names for non-algorithms are more important than for algorithms.

After considering the simple model, the complete model, which brings some additional information to light, was constructed. Recall that this model is based on four categories of explanatory variables: demographic information, question characteristics, time, and answer characteristics. From the first category there were significant interactions between *gender* and *variant* (p = 0.0062) and between *comfort* in multiple programming languages (defined below) and *variant* (p = 0.0106). The *gender*variant* interaction, shown in Figure 5, reveals that men produced better descriptions when seeing full-word identifiers, while for women there was no difference between seeing the full word or abbreviation variant. As is also evident in Figure 5, both genders did significantly worse with the single-letter variant and it appears that women have more difficulty with single letters than men. Together, these observations 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 participants rated their comfort with each of the three programming languages on a scale from 1 indicating low comfort to 5 indicating high-comfort. To simplify the presentation these ratings are summarized by two categories: *high comfort,* for subjects indicating a comfort level of 4 or 5 in two of the three languages, and *low comfort,* for all others.

The complete model includes the interaction *comfort*variant*. Visually this interaction, shown in Figure 6, appears as the different slopes in the three lines. Thus, the interaction and the relative values for the two groups indicate that *variant* has a much greater impact on those with less expertise. In particular, uninformative identifiers hurt the inexperienced more than the experienced (p = 0.0032). Figure 6 also reveals that for both groups, there is no sta-

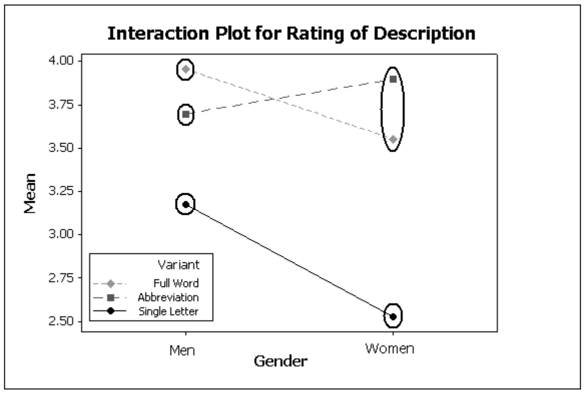


Fig. 5. The interaction between *gender* **and** *variant* **when predicting the mean** *description rating*. **Two variants in the same circle show no significant difference.**

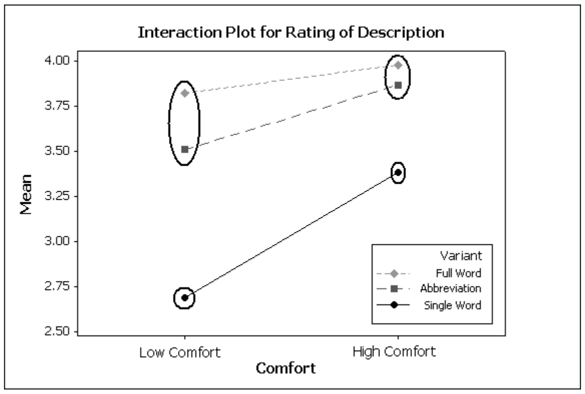


Fig. 6. The interaction between *comfort* **and** *variant* **when predicting the mean** *description rating*. **Two variants in the same circle show no significant difference.**

tistically significant difference between full words and abbreviations, and both are significantly better than single letters.

All four explanatory variables in the second category are significant in the description rating model. *Code type*, having a p-value $= 0.0001$, reveals that algorithms have higher description ratings than snippets – most likely because participants stand a much higher chance of having seen them before, making the code easier to identify and describe. To a lesser extend the algorithms are also easier to describe because they have well known names. The model shows that *description rating* increases with *lines of code* (p-value = 0.0013), indicating that more code improves comprehension, at least over small blocks such as those ranging from 8 to 36 lines. Finally, the variables *identifiers* and *identifiers*^{2} are both significant as well as their interactions with *variant*. The interactions *identi-*

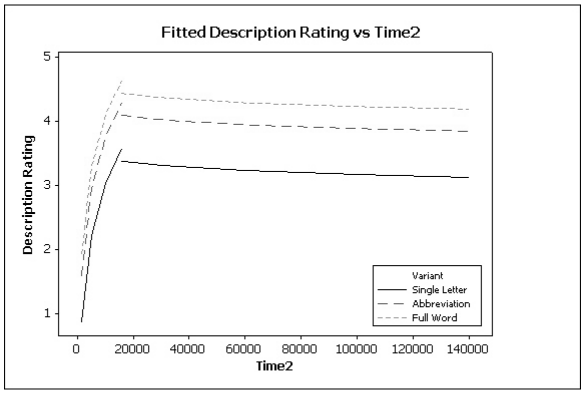


Fig. 7. Graph of each variants *description rating* **against the time spent describing the code.**

*fiers*variant* and *identifiers*^{2}**variant* both had p-values of $<$0.0001. From a plot (not shown) of these variables, the optimal number of identifiers for full words and abbreviations hovers around five. Description ratings of code with greater or fewer identifiers continuously decline as the distance from five grows. The single-letter variant has a different behavior (thus the interaction), with description rating decreasing as the number of variables increases. Given that the single letters, provide little domain information, it is not surprising that its trend is different.

Variables in the third category have to do with time. For description rating, the amount of time spent on the second screen (*time2*) is significant, but the effect is complex. From a plot of the fitted *description ratings* versus *time2*, shown in Figure 7, it can be observed that initially there is a rapid increase in the ratings and then at about 16 seconds, the ratings level off and thereafter tend to gradually decline with increasing time. To account for this pattern, an indicator variable that allows the association between *description rating* and *time2* to be different before and after 16 seconds was added to the model. This variables is 0 before 16 seconds and 1 thereafter.

The model contains the indicator variable, the variable $\log time2$, and the interaction between them as each is statistically significant (p $<$0.0001). Using the parameters from the fitted model, the results show the same sharp increase in ratings followed by a gradual decline. This behavior is accounted for by the observation that early on (during the first 16 seconds) additional time allows subjects who understood the code to write better descriptions. However, as is apparent in their inferior descriptions, some subjects with a weaker understanding took longer.

From the fourth category, there are two significant variables in the complete model. First, the *confidence* subjects have in their understanding of the code has a p-value $<$ 0.0001, which shows that *description rating* increases with *confidence*. This results is not surprising, but rather confirmatory, as both *confidence* and *description rating* are measures of comprehension. The other is the *length* of the description measured as the number of characters it contained. In the model, *description rating* increases with the *length* (p-values = 0.0077); thus, subjects who wrote longer descriptions understood the function better.

In summary, the simple and complete models indicate that subjects wrote the best descriptions for functions with full-word identifiers. They also show 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 five is an optimal number of (domain information carrying) identifiers to be considered at one time. This is consistent with more general research on memory [6] and the slightly dyslexic bias in computer professionals [17]. Finally, an interesting side note is that work experience and schooling are not significant factors for writing correct descriptions.

4.3 Confidence Model

The second pair of models considers the response variable *confidence*, which is an important reflection of comprehension as it reports the subjects' belief in their understanding. The discussion of *confidence* follows the same pattern as that for *description rating* where first a simple and then a complete model are considered. Like the simple model for *description rating*, the simple model for *confidence* examines how *question*, *variant*, and their interaction effect *confidence* as reported by the subjects. All three variables are statistically significant to the model. The p-value for the interaction is 0.0130. Consequently, as shown in Figure 8, the effect of *variant* on *confidence* differs among the questions.

Subjects generally had lower *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 where differences 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* indicating that these two response variables measure similar information, namely comprehension.

The complete model for *confidence* began with almost the same explanatory variables as that for *description rating*. The first three categories (demographic data, question characteristics, and the time spent analyzing the code and writing the descriptions) were the same. In the fourth category, answer characteristics, *description rating* was substituted for *confidence*. Despite starting with largely the same variables, the final model includes different significant variables.

From the first category (demographic information), significant variables include the number of *years worked* ($p = 0.0138$), number of *years in school* ($p < 0.0001$),

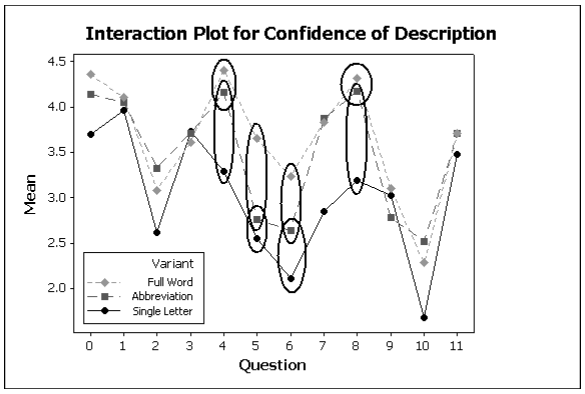


Fig. 8. The interaction between *question* and *variant* when predicting mean *confidence*. Significant differences occurred in Questions 4, 5, 6, and 8 where circled points separate significantly different mean values.

and *gender* ($p = 0.0154$). The data reveals 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 despite showing no performance difference. This pattern has been observed by others [14]. At first glance, it may appear that there are too few female participants to draw such conclusions; in fact, this only means that in order for a statistically significant difference to occur, a very large difference must be observed, making the result noteworthy.

In terms of the second category, question characteristics, two of the four characteristics are significant: *code type* ($p = 0.0004$) and *number of identifiers* ($p < 0.0001$). Algorithms lead to higher *confidence* and more identifiers lead to lower *confidence*. This second conclusion is rather non-intuitive. It is difficult to say why this occurred. Finally, it is interesting to note that *lines of code* is not found to be significant, so encountering more code (in the 8 to 36 line range) does not necessarily improve one's confidence in understanding, although it does improve one's ability to write an accurate description as seen in the previous section.

The third category, concerning time, found that both the time spent on the first screen analyzing the code (*time1*) and time spent on the second screen answering the questions (*time2*) are significant. However, the relation is nonlinear. As with description rating, multiple variables are used to describe each of the times. In general *confidence* increases as *time1* increases from 0 to about 15 seconds. After that point *confidence* slowly decreases as *time1* increases. This indicates that there is an optimal amount of time that one can spend analyzing source code and after that point there are diminishing returns. The variables that represent this are $\log$ *time1* (p-value < 0.0001) and ($\log$

time1)^2 (p-value < 0.0001). The same pattern occurs on the second screen; however, the cut-off time is 16 seconds, rather than 15. Again, some time is necessary, but too much time predicts lower confidence. The significant variables include the indicator variable like the one discussed in the prior section, $\log$ *time2*, and their interaction (all p-values < 0.0001).

In the fourth category, there are two variables 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* is related to years of experience is supported by the data. Also, *gender* plays a role in *confidence*, with women generally having lower *confidence* than men.

4.4 PCIS Model

The final pair of models considers the ability of participants to recall identifiers. Few engineers appreciate how small short-term memory actually is – it has the capacity to hold information related to only a few statements at a time [6]. One important implication of this is that long identifier names may 'crowd-out' needed information. In addition, long names can be misread when they differ little from each other. This section examines a subject's ability to recall specific identifier names.

As before, the analysis includes both simple and complete models. The simple model for memory examines how *question*, *variant*, and their interaction effects *PCIS* (percent correct in source–the percent of identifiers that appeared in the source code and were recalled by a subject). As noted above, *PCIS* ignores the responses to identifiers that did not appear is the source, so the range of possible correct values is between 1 and 4 (of the 6 identifiers). Because *PCIS* ignores wrong choices, these are considered separately below.

In the simple model, all three factors are statistically significant to the model (p-values $<$0.0001). Like the other simple models, the effect of variant on *PCIS*, shown in Figure 9, differs among the questions. Subjects generally have a harder time remembering single-letter identifiers, but this is not a uniform result. For many questions, no significant differences were found among the variants.

In Questions 6 and 7 (both snippets) and 8 (the algorithm Sieve of Eratosthenes) differences were observed. Considering first full-word identifiers and single-letter identifiers, for the snippets, full-word identifiers were easier to recall than single-letter identifiers. The same is not true for the algorithm. Next, considering full-word identifiers and abbreviations, for the snippets there are no difference between the two; however, for the one algorithm, the difference is significant. In general, the two snippets exhibit the expected behavior showing that it is harder to recall single-letter identifiers than the other two variants.

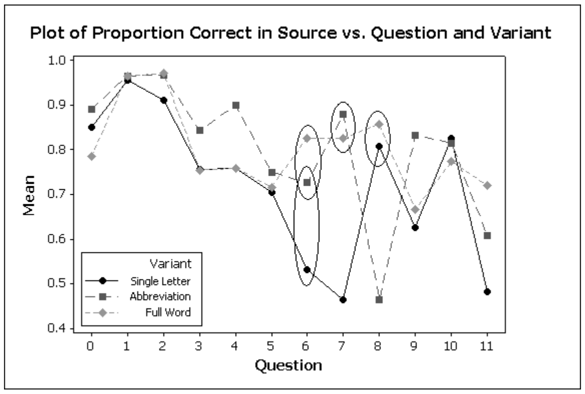


Fig. 9. The interaction between *question* **and** *variant* **when predicting mean** *PCIS***. Significant differences occurred in Questions 6, 7, and 8 where circled points separate significantly different mean values.**

Interestingly, for Question 8 there was no significant difference between full words and single-letter identifiers. It seems counter-intuitive that abbreviations would be harder to recall than the single-letters identifiers. On a closer examination of the data, it was found that a disproportionate number of subjects selected no identifiers when given the abbreviation code, which accounts for the low mean value (0.46). Although several hypotheses were considered to account for this behavior (including an extremely short time viewing the code and a lack of understanding of the code) no trends were evident from the data. Therefore, no explanation can be offered as to why several subjects selected no identifiers for this particular question.

The complete model includes the explanatory variables of the prior complete models for *description ratings* and *confidence* plus four new variables. In the second category, the question characteristics are augmented with the explanatory variable number of *syllables*. The third category now includes the time spent on the third screen (*time3*), and the fourth category includes both *description rating* and *confidence*.

In terms of first category (subject specific information) one variable is found to be significant: *school years* (p = 0.0494) and there is no interaction so the effect is straight forward. For each additional *school year*, *PCIS* increases by 2.4%. This result may come from the fact that with increased schooling, one is more likely to encounter the algorithms presented in the study or perhaps related algorithms and may also have seen code similar to the snippets presented. Therefore, these subjects can make use of ties to long term memory to help recall identifiers [8]. For example, in binary search one usually encounters the variables *right* and *left*, so familiarity with the algorithm helps one remember these identifiers regardless of variant.

The second category on question characteristics includes several significant variables: *syllables* (p <0.0001), *variant*in-source* (p = 0.0004), and *variant*code type* (p = 0.0002). The interpretation for the number of syllables is straight forward; for each additional syllable present in the set of identifiers, on average *PCIS* decreases by 1.4%. This is consistent with an overcrowding of short-term memory thus, other things being equal, shorter identifiers are easier to remember.

The other two variables are involved in two separate interactions. Figure 10 shows a plot of the first interaction *variant*in-source*. *In-source* proves difficult to explain as it is an artifact of the design of the experiment and is not related to other variables such as the number of identifiers present in the code. However, the general trend supports the expected results that full-words or abbreviations are easier to recall than single-letters.

Figure 11 shows a plot of the second interaction, *variant*code type*. For snippets, it is much easier to recall full words and abbreviations than single letters. However, for algorithms, full words are easier to recall than single letters but no difference is observed between abbreviations and single letters. This indicates that recall of single letters is much more difficult for snippets, which provides further evidence for the value of ties to long term memory. For one familiar with an algorithm, even the single letters may gain some advantage from ties to long term memory. For example, p may stand for pivot in quicksort, making p easier to recall. This is not the case for snippets where *a priori* knowledge of the purpose of the code is less likely to provide an indication of which variables are used. It is interesting to note that *lines of code* and *number of identifiers* are absent from this model. This indicates that length of the name (measured by syllables) has a greater impact on memory than the number of identifiers (although there may be some association between the two).

In terms of the third category, both the time spent on the first and third screens have an impact on the ability to recall identifiers. However, the time spent on the second screen is absent from the model. This indicates that subjects spent sufficient time for it to serve its purpose of clearing short-term memory.

In addition to the times themselves, the log of each time is included in the initial model as an explanatory variable because of the shape of the data when graphed. In essence, this shape indicates that after some initial time necessary to read the code or select a few answers, there are diminishing returns for continued study of the code or pondering possible answers. This diminishing returns follows a logarithmic shape.

In more detail, three variables associated with time are significant: *variant**log *time3* (p = 0.0414), (log *time3)*^{2} ($p < 0.0001$), and log *time1* ($p < 0.0001$). When considering *time3* and its interaction with *variant* (the interaction

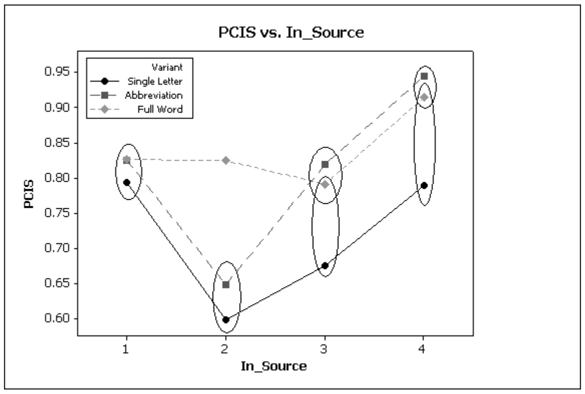


Fig. 10. Mean *PCIS* **values based on** *variant* **and** *in-source*

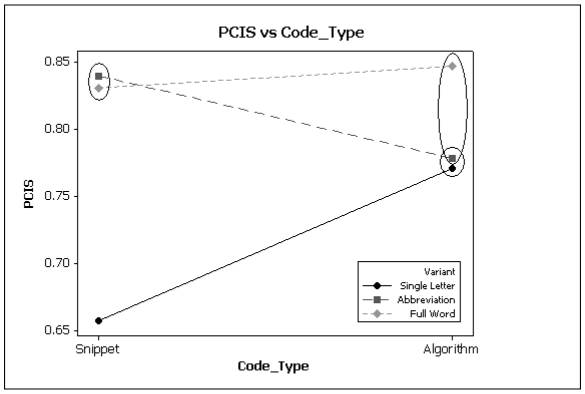


Fig. 11. Mean *PCIS* **values based on** *variant* **and** *code type*

*variant*time3* is shown in Figure 12) the ability to recall full words occurs faster than abbreviations or single letters, and although parallel, one will recall abbreviations better than single letters.

For *time1* there is no interaction with *variant* so the effect of *time1* on *PCIS* is the same for all variants: *PCIS* increases rapidly for the first 20 seconds and then continues to increase more gradually. This is different from other behaviors of time because performance never degrades. The longer a subject studies the code, the more variables are recalled. The plot in Figure 13 shows how subjects remember more abbreviations in a given time period than the other variants.

In the forth category, *confidence* (p = 0.0008) is found to be significant. Like *school years*, there is no interaction so the effect is straightforward. A unit increase in *confidence* brings an average *PCIS* increase of 2.4%. Given that confidence is an indication of comprehension, this provides evidence that the ability to recognize the purpose of the code improves the ability to recall the identifiers. This is

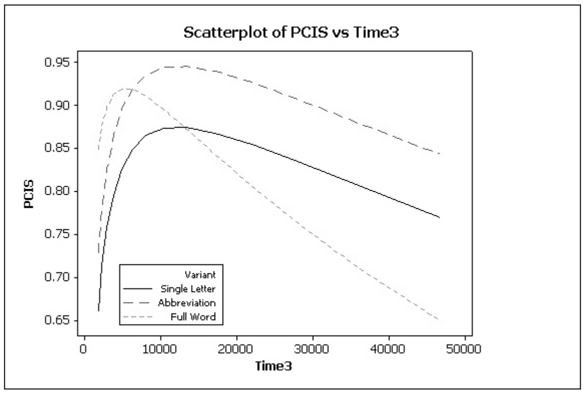


Fig. 12. Mean *PCIS* **values based on** *variant* **and** log *time3*

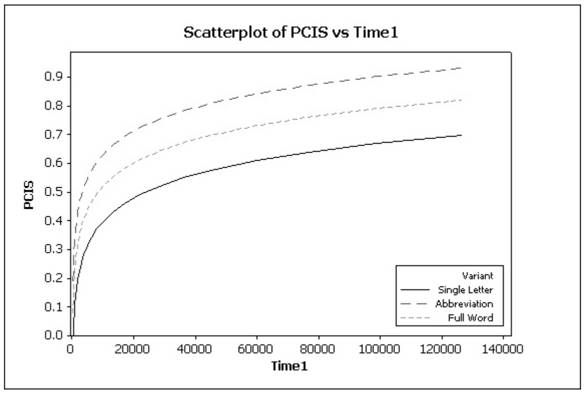


Fig. 13. Mean *PCIS* **values based on** *variant* **and** log *time1*

likely an indicator of subjects exploiting ties to long term memory.

4.5 'Wrong' Answers

The consideration of subject recall concludes by considering wrong answers. The response variable *PCIS* ignores responses to identifiers that did not appear in the source code. However, the kinds of errors programmers make when trying to recall identifiers in the code could impact how one goes about creating high-quality identifiers. For instance, Deissenböck and Pizka hypothesize that having two different identifiers that refer to the same concept or one identifier referring to multiple concepts negatively impacts comprehension [7].

To explore this, wrong answers were divided into three categories based on two boolean factors: was the variant of the identifier the same as the variant of the code and was the concept associated with the identifier the same as the main concept present in the code. For example, with the full-word variant of quicksort, the identifiers data and current_high are both the same variant as the question (full-words), while the identifiers pvt and p are of a different variant. The second factor is true if the concept associated with a given identifier occurs in the code. Thus, again using the full-word variant of quicksort as an example, the concepts associated with the identifiers midpoint and initial_low occur in the code while the concepts associated with the identifiers best_score or bst_scr do not. Combinations of these two factors create four categories, but the category "correct variant where the concept appears in the code" is always a correct answer; wrong answers in this category are accounted for in *PCIS*.

The analysis of the remaining three wrong answer categories considers the mean likelihood that a wrong answer falls into one of the three categories. As shown in Table 2, it is much more likely that a subject selected an identifier of the wrong variant, but having the correct concept, *or* an identifier of the correct variant, but the wrong concept than the category where both dimensions were incorrect.

Selecting an identifier of the wrong variant but whose concept occurred in the code, indicates that subjects have tied the variable to some concept stored in long term memory. This supports Deissenböck and Pizka's hypothesis that two identifiers referring to the same concept can confuse an engineer. For example, a quarter of all the incorrect answers in this category came from confusing n with num, n with number, or num with n. Thus, there is evidence that many subjects knew the concept *number* appeared in the source code but when asked to recall it, did not remember the variant they had seen. Having more than one of these identifiers in a program would likely effect comprehension. The most frequently confused identifier was buf with buffer. Two other frequent confusions were associated with the concept *evaluate* (evaluate with eval and eval with e) and *count* (count with cnt and vice versa).

The second category of wrong answer, where subjects selected a variable of the correct variant but whose concept was not present, indicates that some participants did fixate on the variant. The top three most frequent wrong answers in this category are current_count, curcnt, and fib. The first two occur in code finding the most frequent letter in a string. Thus, the concept is closely related to the concepts in the code. The third identifier was from the Fibonacci code. Understandably, a subject who recognized the purpose of this code might believe that this identifier appeared.

Errors with identifiers in the final category are much less frequent. This category includes both the incorrect variant and a concept not present in the code. The errors in this category are rather erratic; that is, they do not appear to follow any systematic pattern.

Statistically, the percent of wrong answers are compared by creating a variable to represent each unique *subject-question* combination. Because there are three categories for wrong answers, each subject-question combi-

Category	Average % misses
1) Incorrect variant and concept in source	10.6%
2) Correct variant and concept not in source	9.4%
3) Incorrect variant and concept not in source	2.6%

Table 2. Wrong answer means

nation has three repeated values. A randomized block design allows for the repeated measures. Friedman's non-parametric test is used to test the equality of the distributions of percent wrong answers across the categories adjusting for the subject-question combinations. The test shows that the percent wrong is not the same among all three categories ($p < 0.0001$). In addition, using a Bonferroni adjustment to compare the categories pairwise reveals that *correct variant and concept not in source* does not differ from *incorrect variant and concept in source* but that *incorrect variant and concept not in source* is significantly different from the other two.

4.6 Summary of Results

The models discussed in the preceding sub-sections support the five hypotheses presented at the beginning of the section to varying degrees. Each of the hypotheses are now discussed in greater detail.

For the Comprehension Hypothesis, there is evidence to reject the null hypothesis (that there is no difference in comprehension among different identifier variants). Although many questions did not show any differences, the ones that did show full-word identifiers, and in many cases abbreviations, lead to better comprehension as measured by *description rating* and *confidence*. Other evidence in support of the alternative hypothesis comes from the interactions *gender*variant* and *comfort*variant* for *description rating*. For all groups identifiers composed of full words or abbreviations lead to better comprehension than single-letter identifiers. Given this evidence, the null hypothesis is rejected and the alternative hypothesis, that full word identifiers and abbreviations lead to better source code comprehension is accepted.

Next, consider the Memory Hypothesis. The evidence is less clear in this case. In general, there is no clear evidence to reject the null hypothesis (that the ability to recall identifiers is the same for all *variants*). However, for snippets there is a difference in performance. Thus, restricted to snippets, there is evidence to reject the null hypothesis and accept the alternative hypothesis (that it is easier to recall abbreviation and full-word identifiers).

The third hypothesis, Experience and Education, considers how these two aspects impact comprehension. In the complete confidence model, there are no interactions with *variant*; thus the model shows that lower quality identifiers have less of a negative impact on those with greater experience and education. In other words, confidence in comprehending the source code increases with both years of education and years of work experience. Therefore, the null hypothesis (that experience and education have no impact on comprehension) is rejected, and the alternative hypothesis (that increased work experience and schooling lead to better comprehension) is accepted.

Next, when considering the Gender Hypotheses regarding Confidence, females rate their *confidence* lower than males, so null hypothesis can be rejected and the alternative hypothesis that gender plays a role in confidence accepted. However, in terms of ability to describe code, as measured by *description rating*, there is no statistical difference. (Although, men performed best with full-word identifiers, while there was no significant difference between full-words and abbreviations for women.) Therefore, the null hypotheses is not rejected. In summary, although women have less confidence in their answers, they perform at the same level as men, a pattern seen elsewhere [14].

Finally, for the Identifier Length Hypothesis, the explanatory variable *syllables* is significant in the memory model where recall decreased as the number of syllables increased (length is measured as the number of syllables in an identifier). Thus, the null hypothesis (that the length of an identifier has no impact on memory) is rejected, and the alternative hypothesis (that longer identifiers are harder to remember) accepted.

5 Related Work

There is considerable ongoing interest in identifier quality and in particular identifier naming. Although educators tend to stress the importance of meaningful identifier names, such is not universally valued. For example, 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" [18]. However, simply because uninformative 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" [5].

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" [3]. Anquetil and Lethbridge [1] define what it means to have a "reliable naming convention." Deissenböck and Pizka [7] create a formal model for concepts and names, which is used to determine reliable names. Caprile and Tonella [4] 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. [19], and Jones [8] examined how short-term memory impacts the recall of several types of identifiers.

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 [7]. 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 helps maintain consistent naming throughout the lifetime of a software project. 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 without too big a strain on short-term memory.

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 [4].

Others have attempted to determine the informativeness of identifiers including Takang et al. [19]. 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 those with significant experience. 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 subjective 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.

Finally, the study of identifiers by Jones [8] investigated the consequences of a limited human short-term memory capacity on subjects' performance in some representative tasks needed to comprehend short sequences of code. The study had two parts, one was based on the ability to recall identifiers and the other on the ability to follow the path of execution of code given nested if-statements and values of variables. The Jones study found that programmers' performance in processing character sequences varied between different kinds of sequences (*e.g.* words vs. non-words). For instance, frequently used character sequences (*i.e.*, words) are recognized faster and are more readily recalled than rare ones. Also many performance characteristics are slower and more error prone for non-words compared to words. Recognizing known subsequences (*e.g.*, the three that make up ibmciairs) within a longer sequence allows division into more manageable chunks.

6 Conclusion

The study described in this paper shows that better comprehension is achieved when full-word identifiers are used rather than short (virtually) meaningless identifiers. It also shows that in many cases abbreviations are just as useful as the full-words, although this is more true for women than for men.

Gender, work experience, and education also impact confidence, but not ability to correctly describe code. Here men generally report higher confidence as do those with greater training and experience. This has implications at the managerial level where hiring decisions, made in part based on the perceived confidence during an interview, may not necessarily lead to employing the best candidate.

In terms of memory, increased education and increased confidence have a positive effect on subject's ability to recall identifiers. This most likely comes from a greater ability to tie the identifiers to concepts in long term memory. The study also finds that full-word and abbreviated identifiers are generally easier to recall than single-letter identifiers. This is especially true for snippets as compared to algorithms; as snippets are akin to code seen "in the field," the study confirms the belief that engineers should be encouraged to use good recognizable abbreviations or full words for identifiers.

However, longer variables names are harder to remember and it takes more time to do so. This indicates that productivity of software engineers is improved through careful control on identifier length. For example, well chosen abbreviations are an asset; however, extraneous characters that increase the length of an identifier (*e.g.*, the 'its' in the attribute itsHeight of a Person class or the use of Hungarian Notation) have a negative effect on comprehension, in particular from a recall perspective. Thus, the use of naming conventions that involve prefixing or suffixing an identifier should be carefully evaluated to ensure that the added information outweighs the added (recall related) cost.

Finally, from the results of the study, it is clear that whether writing or purchasing software analysis tools, such tools must be able to make use of abbreviated identifier names. For example, consider a tools aimed at aiding programmer comprehension. One technique such a tool might employ would be to automatically associate meanings with abbreviations using machine learning techniques [12] to select key components from the documentation. A second example incorporates into an IDE the tracking and transforming of abbreviations [11]. The tracking feature could provide, perhaps via tool-tip, expansion of abbreviations unknown to an engineer. Transformation would be used to ensure that consistent naming structure is used in a project.

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. 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), 1993.
3. G. Butler, P. Grogono, R. Shinghal, and I. Tjandra. Retrieving information from data flow diagrams. In *Working Conference on Reverse Engineering*, November 1995.
4. B. Caprile and P. Tonella. Nomen est omen: analyzing the language of function identifiers. In *Working Conference on Reverse Engineering*, Altanta, Georgia, USA, October 1999.
5. B. Caprile and P. Tonella. Restructuring program identifier names. In *ICSM*, 2000.
6. N. Cowan. The magical number 4 in short-term memory: a reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24(1), 2001.
7. 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.
8. D. Jones. Memory for a short sequence of assignment statements. *C Vu*, 16(6), December 2004.
9. D. Knuth. *Selected papers on computer languages*. Stanford, California: Center for the Study of Language and Information (CSLI Lecture Notes, no. 139), 2003.
10. J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33, 1977.
11. D. Lawrie, D. Binkley, and H. Feild. Syntactic identifier conciseness and consistency. In *Proceedings of 2006 IEEE Workshop on Source Code Analysis and Manipulation (SCAM'06)*, Phidelphia, USA, September 2006.
12. T. Mitchell. *Machine learning*. WCB McGraw-Hill, 1997.
13. C. Morrell, J. Pearson, and L. Brant. Linear transformation of linear mixed effects models. *The American Statistician*, 51, 1997.
14. P. De Palma. Why women avoid computer science. *Communications of the ACM*, 44(6), 2001.
15. J. Rilling and T. Klemola. Identifying comprehension bottlenecks using program slicing and cognitive complexity metrics. In *Proceedings of the* 11^{th} *IEEE International Workshop on Program Comprehension*, Portland, Oregon, USA, May 2003.
16. H. Saiedan and L. M. Mc Clanahan. Frameworks for quality software process: SEI capability maturity model. *Software Quality Journal*, 5(1):1, 1996.
17. S Silberman. The geek syndrome. *Wired*, 9(12), December 2001.
18. H. Sneed. Object-oriented cobol recycling. In *3rd Working Conference on Reverse Engineering*. IEEE Computer Society., 1996.
19. 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), 1996.
20. G. Verbeke and G. Molenberghs. *Linear mixed models for longitudinal data*. Springer-Verlag, New York, second edition, 2001.