Empir Software Eng (2013) 18:219–276
DOI 10.1007/s10664-012-9201-4

The impact of identifier style on effort and comprehension

Dave Binkley · Marcia Davis · Dawn Lawrie · Jonathan I. Maletic · Christopher Morrell · Bonita Sharif

Published online: 3 May 2012

Editors: Giulio Antoniol and Keith Brian Gallagher

Abstract A family of studies investigating the impact of program identifier style on human comprehension is presented. Two popular identifier styles are examined, namely camel case and underscore. The underlying hypothesis is that identifier style affects the speed and accuracy of comprehending source code. To investigate this hypothesis, five studies were designed and conducted. The first study, which investigates how well humans read identifiers in the two different styles, focuses on

D. Binkley · D. Lawrie
Department of Computer Science, Loyola University Maryland,
Baltimore, MD 21210-2699, USA

D. Binkley
e-mail: binkley@cs.loyola.edu

D. Lawrie
e-mail: lawrie@cs.loyola.edu

C. Morrell
Department of Mathematics and Statistics, Loyola University Maryland,
Baltimore, MD 21210-2699, USA
e-mail: chm@loyola.edu

M. Davis
Center for Social Organization of Schools,
Johns Hopkins University, Baltimore, MD 21218, USA
e-mail: marcy@jhu.edu

J. I. Maletic
Department of Computer Science,
Kent State University, Kent, OH 44242, USA
e-mail: jmaletic@kent.edu

B. Sharif (✉)
Department of Computer Science and Information Systems,
Youngstown State University, Youngstown, OH 44555, USA
e-mail: bsharif@csis.ysu.edu

low-level readability issues. The remaining four studies build on the first to focus on the semantic implications of identifier style. The studies involve 150 participants with varied demographics from two different universities. A range of experimental methods is used in the studies including timed testing, read aloud, and eye tracking. These methods produce a broad set of measurements and appropriate statistical methods, such as regression models and Generalized Linear Mixed Models (GLMMs), are applied to analyze the results. While unexpected, the results demonstrate that the tasks of reading and comprehending source code is fundamentally different from those of reading and comprehending natural language. Furthermore, as the task becomes similar to reading prose, the results become similar to work on reading natural language text. For more "source focused" tasks, experienced software developers appear to be less affected by identifier style; however, beginners benefit from the use of camel casing with respect to accuracy and effort.

Keywords Program comprehension · Text recognition · Coding standards · Identifier names · Memory · Identifier styles · Eye-tracking study · Code readability

1 Introduction

Program identifier names are at the core of program comprehension. They embody the connection between source code and problem domain. Identifiers have been described as *key beacons* to program plans that support higher-level mental models of understanding (Brooks 1983; Soloway and Ehrlich 1984). Identifiers are particularly useful in a wide range of software engineering tasks (Anquetil and Lethbridge 1998; Ohba and Gondow 2005) and especially in the comprehension of programs (Lawrie et al. 2006, 2007; Liblit et al. 2006; Takang et al. 1996). According to Deißenböck and Pizka (2005), identifiers represent the bulk of program text and typically make up approximately 70 percent of source code.

Modern programming languages allow programmers considerable syntactic freedom in the naming of identifiers. Anarchistic use of this freedom interferes with source-code understanding, especially when trying to comprehend code written by other programmers. Several styles exist for engineering more consistent identifiers (Deißenböck and Pizka 2005; Caprile and Tonella 2000; Simonyi 1999); however, scant empirical work has considered their effect on programmers and program comprehension. Because longer names, with more embedded sub-words, are more informative (Liblit et al. 2006), one important feature that these styles share is a means to combine words into a single syntactically-correct identifier. The two dominant identifier styles are camel case (e.g., employeeName) and underscore (e.g., employee_name).

Many early programming languages, such as Ada, Basic, COBOL, Fortran, and Pascal, are case insensitive. Programmers were encouraged, due to convention and practicality, to use underscores to separate compound identifier names. With the popularization of case-sensitive languages such as C, C++, Java, and Python, the trend has been toward the use of camel-case style identifiers. While the general trend has been a greater preponderance of camel casing, there is little hard data on the amount of code written or being written in either style. Many programmers have a strong personal opinion as to which style is better; however, an empirical study is a more appropriate and scientific basis for supporting the use of either. The central

hypothesis considered herein is that identifier style affects the speed and accuracy of reading and comprehending source code.

If a particular style (e.g., the use of camel case or underscores) significantly increases the speed of code comprehension, use of this style would have a tremendous impact on program understanding, quality, and cost. Research in cognitive psychology on natural language suggests that the use of underscores should increase readability and hence improve comprehension, while camel casing should increase reading difficulty. For example, a study by Epelboim et al. (1997) considered the effect of the type of inter-word filler on word recognition. They found that removing spaces or replacing spaces with Latin letters, Greek letters, or digits had a negative impact on reading. However, shaded boxes had essentially no effect on reading. A shaded box depicts a space similar to an underscore.

Assuming that increased readability leads to increased comprehension, underscores should be preferred over camel casing. However, given that camel casing is currently favored by much of the programming community, perhaps something with regards to source code differs from natural language reading. Alternatively, perhaps programmers can be trained to overcome the reading difficulty and perform appropriately with either style. To properly investigate this issue and answer these questions requires empirical study.

While readability is a lower-level concern, an important goal of this work is to provide a solid foundation for answering more complex questions related to program comprehension. Without fundamental results on the effect of identifier style, future comprehension studies are difficult to interpret and compare because the impact of the underlying code readability is unknown. If reading source code and reading natural language are substantially the same undertakings, then a significant body of foundational research on natural language can be used as a basis for program comprehension studies. However, if reading source code and reading natural language are significantly different undertakings, then a new body of foundational research is needed as a basis for program comprehension studies. The need for similar foundational empirical evidence has been observed, for example, when applying topic modeling techniques developed for natural language to source code (Grant and Cordy 2010).

To provide such a foundation, this article describes five studies that leverage past work in cognitive psychology, on how humans read and comprehend natural language prose, and apply it to typical source-code comprehension activities. The following overview of these studies is designed to show how the five studies interrelate and build upon one another. In all five, the key response variable being investigated is the impact of Style, camel case and underscore, on different measures of subject performance. The following description of each study provides the reader with a road map of the progression of, and connections between, the five studies.

Cloud

The progression of studies begins in Section 4 with two low-level readability investigations: the *Applet-cloud* and *Tracker-cloud* experiments. Both ask subjects to find an identifier from among a set of distractors. The identifiers are presented isolated, in moving or static clouds (thus the name of the study).

This simple setting helps to isolate Style's impact on low-level readability. It also provides a basis for the other studies.

Where's Waldo Moving from reading isolated identifiers to reading in context, the next study, presented in Section 5, asks subjects to search a code fragment for all occurrences of a particular identifier. One step "above" simple identifier reading, this study investigates whether style has a impact on time or accuracy when examining a snippet of code.

SAT The third study, presented in Section 6, also considers reading in context, but this time in the context of a natural-language paragraph, where the paragraph is modified to incorporate one of the two styles. Rather than the simple searching of the Where's Waldo study, subject understanding is measured using SAT-style comprehension questions. This study aims to examine the differences between general reading comprehension with those that are specific to reading source code. It also provides a bridge connecting studies from cognitive psychology.

Tracker-code The penultimate study, described in Section 7, mirrors the *Where's Waldo* study by again using snippets of source code. However, subjects are asked to read and comprehend a larger more stand-alone code snippet that includes identifiers in both styles. They are then asked to identify the identifiers seen in the code. The study uses eye gaze behavior gathered by an eye tracker to uncover differences in visual effort during code comprehension. It also considers Style's impact on retention.

Read-aloud The final study, presented in Section 8, asks subjects to study a function and then verbally produce a high-level summary of the code. This study builds on the prior studies, which investigate aspects of readability, to consider the impact of readability on comprehension. The final study goes one step further by asking participants to summarize code, which is expected to indicate their level of comprehension. The goal of this study is to determine the effect of identifier style within a typical code comprehension task such as code summarization.

The remainder of the paper is organized as follows. Background material is presented in Section 2, including an overview of the participants. The two research questions and hypotheses are then presented in Section 3. The studies are presented in Sections 4 through 8. Section 9 discusses the threats to validity and is followed by related work in Section 10. A summary discussion of the results in the context of the two research questions is presented in Section 11, and final conclusions are drawn in Section 12.

2 Background

This section provides background material in support of the empirical studies. Current theories of reading prose are covered first followed by an overview of eye-tracking technology. Next, the makeup and demographics of the participants is discussed. Lastly, the statistical techniques used for analyzing the empirical data are introduced.

2.1 Reading Prose

Current theories about the eye movements while humans read, assert that *reading saccades* (i.e., the quick movements of the eye between fixations when one is reading) are based primarily on information about the length of the upcoming word. This information is determined by low-level visual processes that detect spaces to the right (for English) of fixation points (Epelboim et al. 1997).

Epelboim et al. considered the impact of different inter-word fillers on word recognition (Epelboim et al. 1997). Fillers cause *lateral distraction* and are shown to have a detrimental effect on reading speed. Epelboim et al. were motivated by an earlier study by Bouma that showed how letters presented some distance away from a fixation point took longer to recognize when they were flanked by other letters (Bouma 1970). This effect was stronger when the flanking letters shared features with the target letter. The study compared reading using four fillers and seven filler placements within English prose. The three more relevant combinations are normal (e.g., "sponge bob"), un-spaced (e.g., "spongebob"), and shaded-box filled (e.g., "sponge▀bob)".

From a visual perspective, the un-spaced text is similar to camel casing except that camel casing is expected to be easier to read because of the added clues provided by capitalization. The use of a shaded box is most similar to the use of underscores where, again, underscores might be expected to be easier to read because they are more similar to spaces. Overall, Epelboim et al. found that shaded boxes did not have an impact on word recognition time ($p > 0.3$) while all other fillers did ($p < 0.05$). In addition, the impact on reading speed was more dramatic for more difficult text.

This observation combined with the differences between un-spaced and camel casing and the difference between shaded-box and underscores, suggests the replication of the Epelboim work with a focus on the limited variety of prose seen by programmers. For example, viewed simply as a reading task, reading code can be considered easier than reading a similar length paragraph because of the multitude of visual markers (e.g., { and }) and considerable white space. The first experiment is based on this observation.

Additionally, a study conducted by New et al. (New et al. 2006) used information from the English Lexicon Project. It showed that longer words required significantly more time for subjects to determine their validity as actual words (greater lexical decision time) than shorter words. One possible explanation for this phenomenon is that longer words require more fixations (landings of the eye) before they can be recognized and correctly classified. This greater number of fixations slows reading and thus negatively impacts comprehension.

2.2 Eye Tracking

The motivation for using an eye tracker is that visual attention (focus on a particular location) triggers mental processes to comprehend or solve a given task (Just and Carpenter 1980). Thus, visual attention provides a proxy for cognitive effort as the eye tracker captures quantitative data related to location and duration of a human subject's visual effort. Visual effort is measured as the amount of visual attention devoted to a particular area of the screen.

A Tobii 1750 eye tracker (www.tobii.com) was used in two of the experiments (*Tracker-cloud* and *Tracker-code*). It is a video-based remote eye tracker that uses two cameras to capture eye movements. The cameras are built into a 17 inch TFT-LCD display with a screen resolution of 1024 by 768. This eye tracker does not require the subject to wear any form of head gear, which emulates a subject's normal work environment. The frame rate (temporal resolution) at which sampling occurs is 50 Hz, latency is around 25–35 ms, and average accuracy is 0.5 degrees (approximately 15 pixels). The eye tracker compensates for head movement during the study (i.e., the eyes do not have to be focused on the screen all the time). The ClearView analysis software that comes with the eye tracker was configured using a dual monitor extended desktop setting.

The first monitor is used by the moderator to set up and initiate the study. The moderator gets real-time feedback of eye-tracking quality on the first monitor during the study. The second monitor is used by the study subjects to perform the tasks. The Tobii eye tracker records eye-gaze and audio/video recordings of the entire study session from the second monitor.

The two main types of eye-gaze data are eye *fixations* and *saccades* obtained from the eye-tracker's raw data, which includes a time stamp, x and y gaze coordinates, and any error codes. Unlike a short saccade, a fixation is the stabilization of the eye on an object of interest for a period of time. It has been determined that comprehension mainly takes place during fixations and not during saccades (Duchowski 2007). The eye tracker was set to filter fixations within 20 pixels with a duration of at least 40 ms, the standard setting recommended by Tobii for reading tasks. Small font sizes for the eye-tracking studies were avoided because they can lead to inaccurate fixations and saccades due to tracking errors. The font sizes in the studies were determined experimentally in trial runs with two test subjects to be large enough (11-point Courier New) to avoid inaccuracies.

2.3 Participants

The five studies presented in this paper involve three groups of participants outlined in Table 1. The first group includes 135 students from Loyola University Maryland and includes both programmers and non-programmers. Non-programmers were included to gain insight into the impact of prior programming exposure. The programmers in this group were mostly trained in the camel case style. (For clarity, term "programmer" is used herein to refer to "one who programs", while the term "developer" is used to refer to a programmer who is a professional software developer.) The Age (variables used in the experiments are set in sans serif font) of the subjects ranged from 17 to 22. Overall, participants were 54% male while the programmers were 67% male. After their participation, participants were asked if they had a preference for one style over the other (i.e., felt more comfortable

Table 1 Summary of subject groups

Group	University	Size
Group 1	Loyola	135
Group 2	Loyola	19
Group 3	Kent	15

with one over the other). As expected, those without computer science training either preferred underscores (46%) or had no preference (45%). Interestingly, the preferences of those trained using camel casing was 38% preferring underscores.

The second group is a subset of nineteen programmers from the first group who were all in at least their second year of university. Forty-seven percent had between one and two years of training, while the rest had four years of training. This group included 79% males.

The final group includes fifteen programmers from Kent State University who had experience with both identifier styles. Subjects included seven undergraduates in their second year of study, six graduate students, and two faculty members. Most of these subjects were trained in the underscore style. In this group, 40% stated that they preferred camel case, while 47% preferred underscore. The remainder had no preference. The participants were predominantly male with only two females.

2.4 Statistical Techniques

In addition to standard statistical tests such as a Chi squared test, three statistical modeling techniques are used, one for numerical response variables and the other two for categorical response variables. All deal with repeated-measures and missing values and can easily accommodate unbalanced data (Molenberghs and Verbeke 2006; Verbeke and Molenberghs 2001). Accounting for repeated measures is important because data is collected multiple times from each subject. These statistical models allow the identification and examination of important explanatory variables associated with a given response variable.

The first technique, a linear mixed-effects regression model, is used to model numerical response variables as a function of a collection of explanatory variables that may include a number of interaction terms (written V1 * V2). The interaction terms allow the effects of one explanatory variable on the response to differ depending upon the value of another explanatory variable. For example, if Training interacts with Style in a model where Time is the response variable, then the effect of Training on Time depends on Style (i.e., is different when using camel casing than when using underscores). Backward elimination of statistically non-significant terms ($p > 0.05$) yields the final model. Some non-significant variables and interactions are retained to preserve a hierarchically well-formulated model (Morrell et al. 1997).

A second modeling technique is needed when the response is a binary variable as standard mixed-effect models are not appropriate. In this case, Generalized Linear Mixed Models (GLMMs) are used (Molenberghs and Verbeke 2006). Such models account for repeated measures while providing a model for the probability of the binary outcome (e.g., the probability of correctness). The approach fits a logistic model to the probability of being correct for this binary variable as a function of the explanatory variables with a subject-specific random effect that allows for variability among subjects.

In more detail, to model a response variable Correct using a set of J explanatory variables X_j, the model estimates the probability, p, of a correct answer by modeling the log-odds. Using a simple example, suppose that there are two groups. The log-odds uses a comparison of the odds of Correct between the two groups. The odds of Correct for Group 1 is $\frac{p_1}{1-p_1}$ and the odds of Correct for Group 2 is $\frac{p_2}{1-p_2}$. The ratio of

these odds (or *odds ratio*) is $\frac{p_1/(1-p_1)}{p_2/(1-p_2)}$. Thus, if the odds ratio = 1.5, then the odds of being correct in Group 1 is 1.5 times the odds of being correct in Group 2 (i.e., 50% higher odds of being correct).

For a binary outcome variable, logistic regression models describe the logarithm of the odds as a linear function of explanatory variables, $X_1.X_2, ..., X_J$.

$$\text{log-odds}(\text{Correct}|X_1, X_2, ..., X_J) = \beta_0 + \sum_{j=1}^{J} \beta_j X_j + u_i$$

The effect of a unit increase in X_j (while holding the other explanatory variables constant) on the log-odds is given by β_j. It follows that e^{β_j} is the ratio of the odds for a unit increase in X_j (the odds ratio),

$$\frac{odds(\text{Correct}|X_j + 1)}{odds(\text{Correct}|X_j)}.$$

Additionally, u_i is a random effect that is assumed to come from a normal distribution and allows for variability in responses among the subjects. Significant variables in this type of model will be reported with their p-value and their odds ratio ($e^{\hat{\beta}_j}$, the estimate of e^{β_j}).

When the outcome variable is ordinal (an ordered categorical variable), a model must be used that accounts for the order among the categories. The proportional odds model uses the odds $\frac{P(Y \leq j)}{P(Y > j)}$, rather than $\frac{P(Y=1)}{P(Y=0)}$ as done in the binary case. This third technique models the proportional odds as a linear function of explanatory variables, so it also belongs to the family of GLMMs. In the resulting models, when the odds ratio is greater than 1 then an increase in an explanatory variable is associated with *lower* levels of the categorical variable because it leads to an increase in the numerator $P(Y \leq j)$ relative to the denominator $P(Y > j)$.

Finally, when discussing statistical significance, a p-value < 0.05 is considered statistically significant. A p-value that ranges from 0.05 to 0.1 is considered to be marginally significant and will be described as such.

3 Research Questions and Hypotheses

The goal of this research is to better understand the impact of identifier style on program comprehension. It is addressed by investigating two research questions. The first research question considers low-level syntactic readability issues, predicated on previous work in the area of natural language understanding, which investigates the effect that fillers (e.g., Greek letters, digits, or shaded boxes) have on reading time (Epelboim et al. 1997). The second research question considers semantic implications of the style difference on program comprehension. The results related to the first question provide a foundation for those concerning the second. It is important to understand both levels in the comprehension process. For example, low-level differences may not impact higher-level tasks. Alternatively, higher-level tasks may be greatly impacted by low-level differences. These studies endeavor to

understand the entire picture of low-level readability and high-level comprehension tasks. Two research questions are considered:

Research Question 1 Mirroring research on natural-language reading, does identifier style impact readability?

Research Question 2 Assuming a difference in readability, does identifier style affect higher-level comprehension activities?

Five experimental studies are used to answer these questions. The first two studies (The *Cloud* study and *Where's Waldo* study) present three experiments that address Research Question 1. The subsequent three studies (*SAT*, *Tracker-code*, and *Read-aloud*) investigate the higher-level comprehension issues of Research Question 2.

In the description of the studies, the term Style denotes the use of either camel casing or underscores. The term *identifier* is used to refer to a syntactically legal identifier that follows one of these two styles. The term *part* is used to refer to a well-separated section of an identifier. For example, the identifier total_cost has two parts, total and cost. The term *Length* refers to the number of parts (e.g., two in the case of total_cost). Finally, the term *phrase* is used to refer to a sequence of parts that can be used to construct an identifier. For example, the phrase *total cost* can be used to construct the identifiers total_cost and totalCost.

The two research questions are addressed using two pervasive hypotheses evaluated relative to each study. In the first, performance is dependent on the particular study while the second hypothesis addresses efficiency as measured by the time taken to complete a task. In addition to these two hypotheses, some studies introduce additional hypotheses to explore aspects specific to the study. The first pervasive hypothesis informally states that Style *affects performance*.

Performance Hypothesis

H_0 Performance is the same regardless of the Style of the identifier
H_A Performance is affected by identifier Style

This hypothesis will help determine if developers should use a particular style. If, for example, subjects have higher performance when using identifiers constructed with one style, then it would be a mistake for style guides to encourage the alternative style. Choosing a style that leads to an increased performance could have practical benefits in terms of fewer bugs or reduced time to comprehend a new piece of source code.

The second hypothesis informally states that Style *affects efficiency*.

Efficiency Hypothesis

H_0 Efficiency is the same regardless of the Style of the identifier
H_A Efficiency is affected by identifier Style

In this case, a positive finding comes if subjects are able to complete the task in the study faster when identifiers are in one style over the other. All other things being equal, decreased efficiency equates to increased software cost (i.e., software maintenance cost).

4 The Cloud Study

The Cloud study, whose name comes from the use of cloud images, addresses Research Question 1. This section summarizes two previously presented experiments: the *Applet-cloud* (Binkley et al. 2009a) and *Tracker-cloud* (Sharif and Maletic 2010a). Both measure Style's impact on a subject's ability to read phrases. Their presentation includes experimental design, the explanatory variables gathered, the six hypotheses, and finally the results and implications.

4.1 Experimental Design

The design is described by first considering the selection of the phrases used in the experiments followed by the layout of the two experiments. To investigate the impact of Style, a range of phrases from different sources is needed. This range factors in two aspects: length and origin. The choice of lengths was based on a collection of approximately 6.3 million lines of code taken from a cross section of open-source applications. Out of almost 50 million extracted identifiers, 43.8% had only one part. With each additional part, the percentage decreases with 29.5%, 16.4%, and 6.2% having two, three, and four parts. In fact, approximately 90% of identifiers were composed of one, two, or three parts. Because Style is irrelevant for identifiers of length one, the experiment considers phrases of two and three parts.

The origin of a phrase captures whether or not the phrase is likely to be found in source code. Given that some of the subjects had no programming experience, this aspect was used to investigate whether familiarity with a phrase would have any effect on performance. To this end, non-source code phrases were selected from common English phrases (e.g., *river bank*), while the remaining phrases were taken from existing source code (e.g., *get next path*). The source code phrases were drawn by scanning the 50 million extracted identifiers ensuring the selection satisfied the length requirement.

To balance the time required to take part in the study with the need to collect sufficient data from which to draw statistical conclusions, it was decided that eight questions would be asked. For each style, this choice allows each possible combination of length and phrase origin (2-Word Code, 2-Word Non-Code, 3-Word Code, and 3-Word Non-Code). Each subject saw each combination twice: once for each style.

The eight selected phrases are shown in Column 1 of Table 2, which also presents the *distractors* used for each phrase. Distractors are wrong answers designed to help determine if the kinds of errors subjects make can be systematically classified. In each experiment distractors are chosen systematically to investigate their impact on subject performance. In this study, for each question, one distractor modified the beginning of the phrase, one the middle, and one the end. In addition, for the two styles the modifications were made so that the Levenshtein Edit Distance[1] remained

[1]The Levenshtein Edit Distance is the minimum number of operations needed to transform one string into another. An operation is an insertion, deletion, or substitution of a single character.

Table 2 The phrases used in the study's eight questions, categorized by length and phrase origin

Phrase	Distractors		
	Beginning	Middle	End
2-Word Code			
start time	*smart time*	*start mime*	*start tom*
full pathname	*fill pathname*	*full mathname*	*full pathnum*
3-Word Code			
get next path	*got next path*	*get near path*	*get next push*
extend alias table	*expand alias table*	*extend alist table*	*extend alias title*
2-Word Non-Code			
river bank	*riser bank*	*river tank*	*river ban*
drive fast	*drove fast*	*drive last*	*drive fat*
3-Word Non-Code			
read bedtime story	*raid bedtime story*	*read bedsore story*	*read bedtime store*
movie theater ticket	*mouse theater ticket*	*movie thunder ticket*	*movie theater ticker*

The distractors appear to the right of each phrase. In the Applet-cloud experiment, Style was randomly assigned to each pair for each subject. In the Tracker-cloud experiment, a fixed assignment was used with camel casing being used for Phrases *start time*, *extend alias table*, *river bank*, and *movie theater ticket*

constant between the phrase and its distractor for each combination of length and phrase origin.

To support a test for learning effects, consistency across phrase origin was maintained in the distractors. For example, the middle distractor for the 2-Word Code phrase *start time* is *start mime*, which modifies one character (the first letter of the second word). For consistency, the other 2-Word Code phrase, *full pathname*, has a distractor *full mathname*, which also modifies the first letter of the second word. Table 3 summarizes the modifications used with each combination. In addition to edit distance, the table indicates when the *start* of a word is modified, when the *end* of a word is modified, and when the word's length is reduced by one character. These changes are noted because these types of changes may influence the performance of the subjects (something that is investigated by distractor analysis). Finally, Non-Code distractors were required to consist of English words, although the words did not have to create a coherent phrase. Given that some of the distractors are more coherent, there is a potential impact on participants who attempt to comprehend each phrase. Participants were expected to read but not necessarily comprehend the

Table 3 For each combination shown in the first column, the modification criteria for the three distractors is given

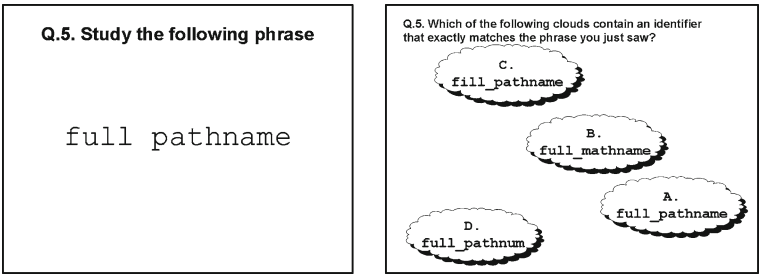
Combination	Distractor change locations		
	Beginning	Middle	End
2-Word Code	1	1 (start)	2 (−1 length)
3-Word Code	1	2 (end)	2
2-Word Non-Code	1	1 (start)	1 (−1 length)
3-Word Non-Code	2	2	1 (end)

Each includes the edit distance for the particular distractor, an indication that the change occurred at the start or end of a word and "−1 length" if the distractor is one character shorter

phrases. While such behavior is unenforceable, it would have shown up in the timing. Finally, the *Applet-cloud* experiment was designed to avoid bias from the distractor selection by using the same identifiers and distractors with both identifier styles. Thus, in this study, over all participants, each distractor was seen approximately the same number of times in the camel case and underscore styles.

The first cloud experiment, referred to as the *Applet-cloud* experiment, is laid out around eight questions, one for each of the eight phrases shown in Table 2. After brief instructions and a practice training question, a subject is presented with the eight study questions. Each question is presented using two screens. The first shows the phrase. The subject is free to study this screen for as long as desired. After pressing "next," the subject is shown the identifier constructed from the phrase and its three distractors and should click on the identifier built from the phrase. Rather than showing these four as a simple list, they are presented inside moving cloud images for two reasons. First the motion increases the reading difficulty. Second, this layout provides a more game-like interface in the hope of encouraging subjects to vest in the activity and complete the task both quickly and accurately. Clouds all move from a starting position in a random direction and at a random speed. For each question, the initial position, angle, and speed of each cloud was randomly determined and then fixed, so that each subject saw the same pattern of motion. Figure 1 shows the clouds as they appeared in the second cloud experiment *Tracker-cloud*.

The order of the eight questions is important to avoid biases and learning effects. The phrases used in the eight questions were arranged into two groups. Each group was balanced by Length and Style and thus consisted of a 2-word identifier using camel casing, a 2-word identifier using underscores, a 3-word identifier using camel casing, and a 3-word identifier using underscores. In addition, within a group, for each length, one of the phrases was a Code phrase and the other Non-Code phrase. Thus, evenly distributing all eight possible combinations used to construct the eight questions. Finally, the presentation order of the questions from within a group was determined using Latin squares to avoid systematic learning biases. This also ensures that initial position, angle, and speed were balanced between the camel case and the underscore identifiers.



phrase stimulus question stimulus

Fig. 1 An example *phrase stimulus* (*left*) that shows the *phrase* to be studied and an example *question stimulus* (*right*) that shows the task description and the four clouds

The experiment was conducted over the Internet using a Java applet, which has several advantages. First, the applet prevents the use of the Web browser's back button and thus provides flow control. Second, the applet was configured to capture how long each subject spends on each question. Finally, the results gathered are already in a digital format, which supports easy manipulation and statistical analysis without data-entry errors.

The second cloud experiment, referred to as the *Tracker-cloud* for short, is laid out around the same eight phrases as the *Applet-cloud* experiment. Replicating the experiment using an eye tracker adds to the empirical evidence as to which Style subjects find visually easier to work with. An eye tracker measures exact gaze location and duration, which supports a fine-grained analysis. Subjects were first presented with instructions and then two sample questions: one camel case and one underscore, illustrating how to answer the questions. Next, eight questions were asked based on the eight phrases. As shown in Fig. 1, each question used two screens where the first presents a phrase and the second shows the four clouds.

The *Tracker-cloud* experiment has four differences when compared to the *Applet-cloud* experiment. First, due to the complexity of running experiments with the eye tracker, the assignment of Style was fixed (using a random assignment) for all subjects. The camel-case Style for *start time*, *river bank*, *extend alias table*, and *movie theater ticket* and the underscore Style for *full pathname*, *drive fast*, *get next path*, and *read bedtime story* were used. From the data collected during the *Applet-cloud* experiment, with one exception, there is no statistical difference in participant performance between corresponding phrases shown in Table 2. The exception is *full pathname* being significantly harder to find than *start time*. In the *Tracker-cloud* experiment, the underscore version of *full pathname* was chosen, essentially handicapping the underscore approach for 2-word code identifiers. The second difference is that due to the complexity of thoroughly testing the data collection program, the clouds remained in a fixed (random) position. Third, to make the task more difficult, the phrase did not accompany the answer and the three distractors on the second screen. Fourth, the *Tracker-cloud* experiment was interactive. Subjects verbally told a moderator when they had studied the phrase sufficiently and which cloud (an alphabetic label was added to each cloud as is shown in Fig. 1) included the correct answer. The subject verbally said "Next" when they were ready to proceed to the next screen. Thus, the setting was more controlled than the *Applet-cloud* experiment where the subject themselves progressed through the questions using a mouse.

Use of a moderator is a necessity given the sophistication of interacting with the eye tracker. For example, the moderator is needed to calibrate the eye tracker for each subject. Additionally, the eye tracker does not deal well with users leaving the "head box". This can occur when a subject rests back in their chair, for example. While this did not occur often during this study, the moderator alerted subjects when data was about to be lost. The alternative is more intrusive as it places the burden of accurate data collection on the subject. Thus, the inclusion of a moderator is preferred.

Finally, the eye-tracker studies were conducted in a dedicated room for the eye-tracking equipment. The subject was seated approximately 60 cm away from the screen. The next step was a five-point calibration of the eye tracker for the subject.

During calibration, a subject focused their eyes on five points that appear on the screen (four for each corner, one for the center). The background color of the calibration was set to white because this was the background of the stimuli used in the study.

4.2 Variables

To account for factors that might affect a response variable and thus, to help ensure that inferences about the effect of Style on a given response variable were not confounded by extraneous factors, 21 explanatory variables were considered. Many of these turn out to have no significant effect in any of the final models. The descriptions of the variables below include only those that have an impact in at least one of the final models. A discussion of the complete set of variables considered and the models they were initially used in is given in a companion technical report (Binkley et al. 2011).

The variables collected during the cloud experiments include three response variables and six relevant explanatory variables. The first response variable, Correctness, reflects the subject identifying the cloud containing the correct identifier. The second, Find Time, represents the time taken to select this cloud. The third response variable, Visual Effort, measures six quantities related to eye movements during the *question stimulus* and is only considered in experiments involving the eye tracker. The first three of the six eye movement quantities are fixation counts, FC, and include FC(correct), which counts fixations on the correct cloud, FC(distractors), which counts fixations on the distractor clouds, and FC(all), which is the sum of FC(correct) and FC(distractors). The other three measure the *average fixation duration* (AFD). AFD(correct) is the average length of time of all fixations on the correct cloud and AFD(distractors) is the average length of time of all fixations on the distractor clouds. AFD(all) is the average length of time of all fixations on any of the clouds. Fixation rates, fixation counts, and average fixation duration for the *phrase stimulus* were also considered, but not found significant.

The primary explanatory variable, Style has two values *camel case* and *underscore*. The remaining explanatory variables come from three sources: the questions, the subject's performance, and the demographics. The two variables related to each question are the Length of the phrase (*two* or *three*) and the Phrase Origin (*Code* or *Non-Code*). The first of the two performance related variables, Reading Time, measures how long the subject spent reading the original phrase on the *phrase stimulus* screen before proceeding to the *question stimulus* to search for it in a cloud. The second, Time on Demographics, measures how long the subject took filling in the demographic information and was only collected in the *Applet-cloud* experiment.

Finally, demographic variables help uncover patterns that arise from the subjects' background. Only Training, which captures the amount of computer science training the subject had received, has a significant effect in at least one cloud model. There are four categories for this variable: "No Training," "Less than a year," "Between one and two years," and "More than two years." In some of the models the last category is divided into two parts based on work experience as given by the binary variable Experience. It has the value *expert programmer* when the subject had more than two years of work experience and at least five years studying computer science, and the value *beginner programmer* otherwise.

4.3 Experimental Hypotheses

Six hypotheses are used to study Research Question 1. The first two, introduced in Section 3, consider subject performance using Correctness and subject efficiency using Find Time. The study also considers two additional hypotheses that deal with Training and two that deal with Visual Effort. In the following description, each hypothesis is formally stated and then explained.

The first additional hypothesis informally states that *training can overcome errors attributed to* Style.

Hypothesis Cloud 1

H_0 The effect of Style on Correctness is independent of Training.
H_A The effect of Style on Correctness lessens due to Training.

Because Hypothesis Cloud 1 involves the influence of one variable on another, support for the hypothesis will be evident in the statistical significance of an interaction between Style and Training. If the interaction is present then the log-odds of Training on Correctness will be different for underscore and camel case. For example, when asked, most non-programmers held the opinion that camel casing would be harder to visually process and thus lead to more errors. Rejecting the null hypothesis for Hypothesis Cloud 1 implies that training can be used to mitigate the impact of such reading errors.

The second hypothesis mirrors the first for the time taken.

Hypothesis Cloud 2

H_0 The effect of Style on Find Time is independent of Training.
(that is the slope of Style on Find Time will be the same for the different levels of Training.)
H_A The effect of Style on Find Time lessens due to Training.

Rejection of the null hypothesis H_0 for Hypothesis Cloud 2 indicates that an engineer can be trained to work quickly and effectively to read identifiers regardless of Style. Visually, this kind of interaction appears as a difference in slopes (looking ahead this is illustrated in Fig. 6).

Rejecting the null hypothesis would indicate that the lines representing *camel casing* and *underscore* in the interaction plot have different slopes. To appreciate the broader implication of Hypotheses Cloud 1 and Cloud 2, assume that there is support for the Performance and Efficiency Hypotheses that predicts better performance when using underscores. While this observation would be an interesting result, it still does not completely address the identifier Style question. For example, it is possible that training could overcome this difference without having negative side-effects. In this example, positive results for Hypotheses Cloud 1 and Cloud 2 would indicate that training with camel casing could be used to compensate for natural ability.

The last two hypotheses only apply to the six eye-tracking measures, collectively referred to as Visual Effort. The third hypothesis informally states that Style *affects* Visual Effort.

Hypothesis Cloud 3

H_0 Visual Effort is the same regardless of Style.
H_A Visual Effort is affected by Style.

Similar to the Performance Hypothesis, this hypothesis will help determine if engineers should be required to use a particular Style. Choosing a Style that leads to a decrease in Visual Effort could have practical benefits in terms of fewer bugs or an easier time comprehending a new piece of source code.

The fourth hypothesis informally states that *training can overcome the* Visual Effort *differences attributed to* Style.

Hypothesis Cloud 4

H_0 The effect of Style on Visual Effort is independent of Experience.
H_A The effect of Style on Visual Effort lessens with Experience.

Similar to Hypothesis Cloud 1, Hypothesis Cloud 4 involves the influence of one variable on another and thus support for the hypothesis will be evident in the statistical significance of an interaction between Style and any of the measures of Visual Effort.

4.4 Distractor Analysis

This section presents highlights from the distractor (wrong answer) analysis of the data from the two cloud experiments. A complete distractors analysis is given in the companion technical report (Binkley et al. 2011). For the *Applet-cloud* experiment, distractor analysis was done for the camel case and underscore versions of each question separately. In all, sixteen questions were considered. Each subject responded to eight of them. The two most difficult questions were the camel case and underscore versions of the phrase *extend alias table*, although there were no significant differences among subjects with or without computer science training on these two items. Considering the distractors, a χ^2 test reveals no difference between camel casing and underscore phrase changes as a function of distractor change locations (as given in Table 3).

In the *Tracker-cloud* experiment, the item *movie theater ticket* produced the most fixations on distractors, followed closely by *extend alias table.* In terms of time, the distractor mouseTheaterTicket had the highest average fixation time. The two items producing the most fixations on distractors were three-word camel-case distractors. Finally, the greatest fixation number and fixation time for the distractors were always camel case distractors. Camel casing's dominance supports the intuition of non-programmers concerning its demand for greater visual effort.

4.5 Statistical Models

Statistical models for the response variables Correctness, Find Time, and Visual Effort are presented herein. Models are always constructed in pairs. The first, or simple, model of a pair includes only one explanatory variable, Style, while the second, or complex model, includes other explanatory variables. While the inclusion of additional explanatory variables tends to improve the model, it can have one of

two effects on Style. In some cases, the influence of Style decreases because other explanatory variables better explain (some of) the variation in the response variable. However, in other cases, the influence of Style increases because it only needs to explain a subset of the variation for which it is well suited. In this study, the two models for Correctness are not meaningful for the *Tracker-cloud* study because only one subject gave a wrong answer. The Visual Effort models are only meaningful for data gathered using the eye tracker and so response variables related to Visual Effort are modeled only for the *eye tracker* experiment. In total, 26 statistical models are considered.

4.5.1 Applet-Cloud Experiment Models

For the *Applet-cloud* experiment, four models were constructed. The first two have the response variable Correctness, which is a binary variable (having the values "yes" and "no"), and thus *Generalized Linear Mixed Models* (GLMMs) were fit to the data. The second two models have the response variable Find Time, which is not a binary variable, so linear mixed-effects regression models were fit to the data.

Overall, the percent Correctness for the underscore Style is 0.84 (Std Dev = 0.36), while the average for the camel case Style is 0.89 (Std Dev = 0.32). In the simple model, the parameter estimate for Style is statistically significant (odds ratio = 1.515, $p = 0.0250$). The model indicates that camel casing has a larger probability (equivalently higher odds) of Correctness when compared to underscores. In particular, the odds of being correct are 51.5% higher for camel case.

To account for other factors that might influence Correctness, additional explanatory variables were considered. Those found relevant were described in Section 4.2, while the complete set can be found in a companion technical report (Binkley et al. 2011). In addition, the variable Find Time and the interactions of Style with each of the explanatory variables were included. After the elimination of non-significant terms, the following terms remained in the final model: Style, Time on Demographics, Reading Time, Length, and Phrase Origin. The parameter estimates given in Column 2 of Table 4 are the estimates of β_j for each model variable. After accounting for the other terms, the model indicates that the use of camel case leads to a larger probability of Correctness than underscore (odds ratio = 1.617, $p = 0.0127$). Comparing these results with the simple model illustrates a case in which the presence of other explanatory variables has improved the estimate for Style. In this case its p-value drops from 0.0250 to 0.0127 indicating that 1.617 is a better estimate of Style's coefficient than 1.51 reported by the simple model.

Table 4 Applet-cloud GLMM model for Correctness

Variable	Estimate	p-value	Odds ratio
Intercept	2.785	<0.0001	
Style	0.480	0.0127	1.617
Time on demographics	0.073	<0.0001	1.075
Reading time	−0.095	0.0073	0.910
Length	−0.820	<0.0001	0.441
Phrase origin	−0.472	0.0142	0.624

Note that the Odds ratio is e^{Estimate}

The complex model also indicates that a longer Reading Time leads to a lower probability of Correctness (odds ratio = 0.910, $p = 0.0073$) and a longer Time on Demographics leads to a higher probability of Correctness (odds ratio = 1.075, $p < 0.0001$). In addition, a lower probability of Correctness comes with having greater Length (odds ratio = 0.441 $p = 0.0142$) or when Phrase Origin is Code (odds ratio = 0.624, $p < 0.0001$). These results indicate that poorer readers (identified by increased Reading Time) are less likely to be correct regardless of Style. Given that Time on Demographics is also associated with a subject's reading speed and this variable has the opposite sign of Reading Time, it is likely that this variable has a dampening effect on an over-estimate created by Reading Time. Such dampening effects are common in correlated variables.

Next, Find Time is considered where the average underscore time is 3.1 seconds (Std Dev = 1.7), whereas the average camel case time is 3.5 s (Std Dev = 2.1). As with Correctness, initially only Style is considered. It is statistically significant ($p < 0.0001$). The estimate indicates that, on average, camel case takes 0.42 s (13.5%) longer than underscores. To account for other factors that might influence this time, the second model includes the additional explanatory variables, Correctness, and the interactions of Style and Correctness with each of the other variables. After backward elimination, the final model includes the five explanatory variables shown in Table 5.

In this model, interactions require all but the effect of Length to be considered together. There are two effects to consider. First, the parameter estimate (slope) of Training is 0.106 for underscore and −0.104 for camel case. This slope difference suggests that those with more training were quicker on camel case than those with less training and that those with more training were slower on underscore than those with less training. However, this effect is only just statistically significant. The final two interactions both involve Time on Demographics and thus must be interpreted together. The effect on Find Time from smallest influence (shallowest slope) to largest influence (steepest slope) comes from correct answers to problems using underscore, correct answers to problems using camel case, wrong answers to problems using underscore, and finally, wrong answers to problems using camel case. Considering Style's involvement, for both correct and incorrect answers, and assuming that Time on Demographics acts as a proxy for reading ability, the camel case Style slows weak readers down more than strong readers. Finally, the effect of Length is independent (it is not involved in any interactions). The effect of a unit increase in Length is to increase Find Time by 0.98 s. Thus, each additional part increases identifier recognition time by about a second.

Table 5 Applet-cloud model for Find Time

Variable	Estimate	*p*-value
Intercept	−3.375	<0.0001
Style	−0.126	0.7253
Correctness	1.164	0.0039
Training	0.106	0.1317
Time on demographics	0.143	<0.0001
Length	0.981	<0.0001
Style * training	−0.210	0.0342
Style * time on demographics	0.033	0.0013
Correctness * time on demographics	−0.040	0.0126

Table 6 Tracker-cloud model for Find Time

Variable	Estimate	p-value
Intercept	4.607	<0.0001
Phrase origin	0.749	0.0170
Length	−0.950	<0.0001
Style	1.816	0.0032
Style * length	−1.768	0.0050

4.5.2 Tracker-Cloud Experiment Models

In all, twenty-four models are constructed from the eye-tracker data. This section considers the sixteen in which Style is significant. The first two consider Find Time while the remainder consider the six different relevant measures of Visual Effort. As with the *Applet-cloud* experiment data, half of these consider only the impact of Style (i.e., the simple model). The other half consider all the explanatory variables and their interactions.

In this experiment the average Find Time for the underscore Style is 4.5 s (Std Dev = 1.8), while the average camel case time is 5.4 s (Std Dev = 2.9). In the simple model for Find Time, Style is significant ($p = 0.0143$). The estimate indicates that on average, camel casing takes subjects 0.93 s longer than underscores. The complex model initially included all the other explanatory variables and their interactions with Style. After backward elimination, the final model includes the three explanatory variables shown in Table 6. This model confirms the significance of Style. Because of the interaction with Length, Style cannot be discussed separately. This interaction indicates that, while there is almost no difference when Length is two, camel casing takes 1.8 s longer for the three-word phrases. Furthermore, phrases extracted from source code take 0.75 s longer than the non-source code phrases.

The remaining twelve models have Visual Effort as their response variable. Style is significant in three of six simple models, and Table 7 contains descriptive statistics for the response variables in these models. These models predict three average fixation durations: AFD(correct) where camel casing takes 0.013 s longer ($p = 0.0205$), AFD(distractors) where camel casing takes 0.022 s longer ($p = 0.0357$), and AFD(all) where camel casing takes 0.051 s longer ($p = 0.0269$). As indicated by the weakness of the statistical association, none of these show a large influence.

Turning to the complex models, for the above three response variables, elimination removes all the explanatory variables except Style. Thus, differences in the values of AFD(correct), AFD(distractors), and AFD(all) can be explained by the single explanatory variable Style. The other three models, for FC(correct), FC(distractors),

Table 7 Descriptive statistics by Style for Visual Effort

Variable	Underscore		Camel case	
	Mean	Std. dev	Mean	Std. dev
AFD (correct)	0.16	0.080	0.18	0.86
AFD (distractors)	0.38	0.23	0.43	0.29
AFD (all)	0.15	0.061	0.16	0.074
FC (correct)	7.92	3.92	8.67	6.14
FC (distractors)	10.47	5.27	11.23	7.82
FC (all)	18.45	8.02	20.28	13.31

Table 8 Tracker-cloud models for FC(correct), FC(distractors), and FC(all)

Variable	Estimate	*p*-value
FC(correct)		
Intercept	6.908	<0.0001
Phrase origin	2.183	0.0015
Length	−0.167	0.8604
Style	2.733	0.0047
Style * length	−3.967	0.0037
FC(distractors)		
Intercept	11.367	<0.0001
Length	−1.800	0.1591
Style	3.400	0.0086
Style * length	−5.267	0.0041
FC(all)		
Intercept	17.683	<0.0001
Phrase origin	3.500	0.0172
Length	−1.966	<0.0001
Style	6.133	0.2074
Style * length	−8.600	0.0037

and FC(all), are shown in Table 8. Phrase Origin is present in two of the three models where phrases extracted from source code require an additional 2.1 fixations for FC(correct) and 3.5 for FC(all). The higher number of fixations indicates longer processing time needed for source-code phrases because they might not be part of everyday vocabulary. In all three models, as with the Find Time models, there is an interaction between Style and Length. The interaction of Style * Length indicates that while there is almost no difference when Length is two, camel casing takes 3.9 more fixations for the correct phrase when Length is three. While it is not surprising to see that it takes more fixations to read a longer phrase, this effect is even more prevalent when camel casing is used.

4.6 Discussion

In considering the impact of the statistical models on the six cloud hypotheses, no *head-to-head* comparisons are made; thus, differences in their setup is not an issue. Instead, the two complement each other. This section considers each hypothesis in turn.

Performance Hypothesis [2]—Correctness *is affected by identifier* Style

Support for the hypothesis is found in both Correctness models where the use of camel case increases Correctness. Because Style is significant, the null hypothesis can be rejected.

Efficiency Hypothesis—Find Time *is affected by identifier* Style

Support for the hypothesis is found in all four Find Time models where Style is significant; thus, the null hypothesis can be rejected. All four models find that identifiers written in the camel-case Style take longer to identify.

[2]The Performance and Efficiency Hypotheses are restated in terms of the variable used to study it.

Hypothesis Cloud 1—*The effect of* Style *on* Correctness *lessens due to* Training

The complex model for Correctness is required to address Hypothesis Cloud 1. Because the interaction between Style and Training is not significant in this model, the null hypothesis cannot be rejected. Training has no observed impact on how Style affects Correctness.

Hypothesis Cloud 2—*The effect of* Style *on* Find Time *lessens due to* Training

The two complex models for Find Time are required to address Hypothesis Cloud 2. Support comes from an interaction between Style and Training. In the *Applet-cloud* experiment, this interaction is significant. Those subjects with more training (in the camel-case Style) were quicker on identifiers written in the camel-case Style. Because subjects with more training were also slower than subject without training on identifiers written in the underscore Style, it appears that training in one Style negatively impacts performance for the other. In the *Tracker-cloud* experiment, the interaction is not significant. Because the interaction is significant in the first model, there is evidence against the null hypothesis, but the evidence is not conclusive.

Hypothesis Cloud 3—Style *affects* Visual Effort

Non-programmers have stated that camel casing would be harder to visually process and thus lead to more errors. Results from the *Tracker-cloud* experiment confirm the belief that the visual processing is harder as higher fixation counts accompany camel-case identifiers.

Based on the three models for FC(correct), FC(distractors), and FC(all), there is evidence to reject the null hypothesis indicating that Visual Effort is affected by identifier Style. In particular, long camel-case identifiers require significantly greater visual effort.

An excellent illustration of this greater visual effort comes from looking at eye-tracker gaze plots. As seen on the left of Fig. 2, the gaze pattern for an underscore example exhibits the expected landing point pattern where the eye gazes between the first and second characters of each word. The middle gaze plot shows the 3-part Non-Code identifier mouseTheaterTicket. Three large fixations are seen on the three parts *mouse*, *Theater*, and *Ticket*. The landing points of all three words has shifted one to two letters to the right.

This is evidence of the visual confusion that camel casing is causing as the fixation points are farther into each word than would be expected when reading normally separated words. The gaze plot also shows the longer duration (larger circles) indicating increased mental parsing time needed to process the joined word. The

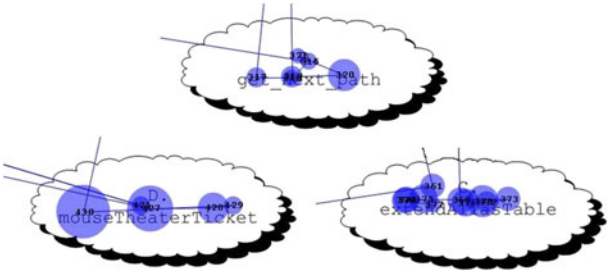


Fig. 2 Gaze plots for an underscore (*top*) and two camel cased 3-word code identifiers. The plots illustrate the visual confusion that camel casing presents to subjects

final gaze plot shows extreme visual confusion as a subject reads extendAliasTable. These gaze plots illustrate how camel casing causes at least two subjects to visually work harder, slowing them down. However, the greater concentration improves their accuracy. Training can help to reduce but not eliminate the time difference.

Hypothesis Cloud 4— *Training can overcome the* Visual Effort *differences attributable to* Style

In the complex models for Visual Effort the interaction between Experience and Style is never significant. Thus, there is insufficient evidence to reject the null hypothesis.

Summary of Results

Unifying these results, from the Correctness models camel casing produces more accurate results. However, this correctness comes at a cost as the camel-case Style significantly increases the time needed to correctly detect the correct identifier. The cause is made apparent in the eye-tracker models where greater visual effort is required. Thus, the first Cloud study provides an affirmative answer to Research Question 1: Style impacts readability.

5 Where's Waldo Study

The cloud study concluded that Style impacts the readability of identifiers. Building on this result, the second study, which also investigates Research Question 1, considers a similar readability question, but in a more realistic setting. Thus it helps to tie the previous results to activities on source code. While no deep comprehension is needed to complete the second task, it does reflect a more common task for a developer.

This study asks subjects to find all occurrences of a particular identifier in a code fragment mimicking the game "Where's Waldo." This task is similar to that performed by programmers scanning source code for a concept or identifier. Although an IDE can easily highlight all uses of a single variable, a programmer reading a function or a class must often keep track of several different variables. The ability to visually discriminate between variables is an underlying task to such reading.

5.1 Experimental Design

The design considers the selection of the four code fragments and the layout used in the study. Two of the fragments were taken from open-source software, one being a C program and the other Java and two from text books, again one C and one Java. There were two requirements on this code. First, a significant proportion of the identifiers must have multiple parts so that Style might play a role. Second, some of the identifiers must have common words. The two open-source fragments were used as-is and appear on the left side of Fig. 3.

The two textbook fragments were modified to ensure that their identifiers met the two requirements. For example, quicksort's partition method was used as one of the code fragments. Rather than pivot, high, and low, the variables currentPivot, currentLow, and currentHigh were used. This code snippet appears in the upper

Search for attackedHealth

```
damageMultiplier = attackerHealth;
damageMultiplier = damageMultiplier / attackedHealth;

if(damageMultiplier > 4)
   damageMultiplier = 4;
else if(damageMultiplier < .25)
   damageMultiplier = 0.25;

totalDamage = ((attackerStrength+attackerHealth) -
                  (attackedDefense + attackedHealth)) / +
                  (Math.random() * 5 + 5);

if(totalDamage < 1)
   totalDamage = 1;

return totalDamage * damageMultiplier;
```

Question 1

Search for currentHigh

```
currentLow = initialLow;
currentHigh = initialHigh;
midpoint = (initialLow + initialHigh) / 2;
currentPivot = data[midpoint];

while (currentLow <= currentHigh)
{
   while ((currentLow < initialHigh)
         && (data[currentLow] < currentPivot))
      ++currentLow;
   while ((currentHigh > initialLow)
         && (data[currentHigh] > currentPivot))
      --currentHigh;

   if (currentLow <= currentHigh)
   {
      Item temporary = data[currentLow];
      data[currentLow] = data[currentHigh];
      data[currentHigh] = temporary;
      ++currentLow;
      --currentHigh;
   }
}
```

Question 2

Search for game_board

```
best_score = -99.9;

for(int i=0; i<moves->count; i++)
{
   position_from_key(game_board, moves->moves[i]);
   swap_sides(game_board);

   if (evaluate_position(game_board, result))
      return -999.0;

   invert_evaluation(result);
   game_score = utility(result);
   if (game_score > best_score)
   {
      moves->best_move_index = i;
      best_score = game_score;
   }
}

return best_score;
```

Question 3

Search for temp_result

```
Input_buffer input = new Input_buffer
      (new Console_input_reader(System.in));
String input_string;

Temperature temperature = new Temperature();
double temp_in, temp_result;

print("Input a temperature in Fahrenheit > ");
input_string = input.read_line();
temp_in = Double.parse_double(input_string);
temperature.set_fahrenheit(temp_in);
temp_result = temperature.get_celsius();
println(temp_in + " F = " + temp_result + " C ");

print("Input a temperature in Celsius > "):
input_string = input.read_line();
temp_in = Double.parse_double(input_string);
temperature.set_celsius(temp_in);
temp_result = temperature.get_fahrenheit();
println(temp_in + " C = " + temp_result + " F ");
```

Question 4

Fig. 3 The four example Waldo questions

right of Fig. 3. The final fragment converts Fahrenheit to Celsius and appears in the lower right of Fig. 3. Finally, a representative identifier for subjects to search for was selected from each fragment. These targets are *attacked health*, *current high*, *game board*, and *temp result* as indicated in Fig. 3.

The *Where's Waldo* study was laid out around four questions. Each question consisted of three screens. The first screen let the subject prepare herself for the exercise. The second screen revealed the identifier for which the subject must search. The final screen consisted of one of the code fragments from Fig. 3. Each line was proceeded by a check box. The subject was instructed to check all lines where the identifier of interest appeared. To eliminate recall issues, the identifier of interest appeared in the upper right corner of the final screen. The subject clicked a button when they believed they had found all the lines containing the target.

As in the *Applet-cloud* experiment, there is a practice exercise before the questions whose order was systematically determined using Latin Squares to avoid biases and learning effects. The four questions were encountered in the same order for each subject; however, the Style in which a question appeared differed by subject. Each subject saw two questions in each style. This study was conducted in the same environment as the *Applet-cloud* experiment using the first group of subjects and was preceded by it. In addition, half of the subjects completed the *SAT* study (in Section 6) before this experiment.

5.2 Variables

The variables collected during the *Where's Waldo* study include two response variables and six relevant explanatory variables. The response variables are similar to the response variables for the *Applet-cloud* experiment, where one concerns correctness and the other time. Correctness is impacted by both *missed lines* where a line includes the identifier of interest but the check box is not checked and *extraneous lines* where the line does not include the identifier of interest but the check box is checked. However, extraneous lines are a rare event (only 4% of the answers included extraneous lines), so correctness is directly measured and more easily interpreted using only the number of missed lines, which is captured by the explanatory variable, Missed Lines. The second response variable, Time Scanning Code, is the amount of time that a subject spent looking for all occurrences of an identifier.

As in the *Applet-cloud* experiment, the primary explanatory variable is Style. The remaining explanatory variables come from three sources: the subject's performance, the questions, and the demographics. There is one variable pertaining to the subject's performance. Correct is a binary variable, which indicates if the subject marked all lines correctly. Its addition to the model allows differences between those giving correct and incorrect answers to be examined separately.

The single question-related variable captures the order in which the subject performed the overall experiments. This variable, Waldo First, indicates whether this experiment or the next experiment, the *SAT* experiment, was presented first to the subject. There is a possibility of learning effects, especially among the novices (non-programmers) who are seeing the treatment for the first time, and of subject fatigue. Half of the subjects completed *Where's Waldo* first, while the others completed the *SAT* experiment first.

The following relate to demographic information. These variables include those used for the *Applet-cloud* experiment (in Section 4.2). In addition, another variable is derived from a question answered after completing the *Applet-cloud* experiment. The subject was asked to assess his or her own behavior on a scale from 1 ("all speed, no precision") to 5 ("all precision, no speed") to ascertain whether he or she favored speed or precision during the *Applet-cloud* experiment, which is referred to as the Speed Question. The Speed Question was used along with the time taken to answer the question as explanatory variables. For the *Where's Waldo* study, the resulting variable, Time Answering Speed Question, is significant and considered to be demographic information.

5.3 Distractor Analysis

Distractor analysis on the *Where's Waldo* data reveals that there are few distractors (about 1.5%) selected, indicating that distractors may have been quite easy to identify and ignore. Only one distractor selected, *attacker health*, was selected disproportionately more when appearing in camel case, which may be due to the fact that for the camel-case style the 'r' and 'H' in attackerHealth 'rH' visually look a bit like the 'd' from the correct answer, attackedHealth. However, in the underscore attacker_health the 'r' and 'h' are kept separate. This difference implies that problems may arise when code is written in camel case due to the proximity of the two middle letters.

5.4 Statistical Models

Only the performance and efficiency hypotheses were considered for this study. Thus, four models were constructed in pairs. The first model of each pair includes only Style, while the second includes other explanatory variables. Neither response variable is binary or ordinal; thus, linear mixed-effects regression models are fit to the data.

For the performance hypothesis, the first two models consider the response variable Missed Lines. Overall, the average number of missed lines when using identifiers written with underscores was 0.72 (Std Dev = 1.31) as compared to only 0.56 (Std Dev = 1.06) when using camel casing. Statistically, the model built using only Style finds this difference significant ($p = 0.0293$) with camel casing leading to an estimated 0.24 fewer missed lines, which is slightly larger than the difference between the raw averages. However, in practical terms, this is only a 1.6% difference.

The complex model for Missed Lines is built using the remaining explanatory variables and their interactions with Style. Elimination of non-significant terms produces the model shown in Table 9. Discussion of its parameter estimates begins with those involving Style, in this case the interaction Style * Correct. This term enables the model to provide different parameter estimates when modeling the number of Missed Lines for subjects who correctly identified all the lines from those who did not. By definition, the number of missed lines is zero when a subject is correct; thus, interactions with Correct always include an essentially horizontal line capturing the case when Correct is true. When Correct is false, the model predicts more missed lines for underscore than for camel case. However, in practical terms, the effect is very small. The slight edge afforded camel-case identifiers may come from the visual difference created by the capital letters. Although as the distractor analysis reveals, very similar camel-case identifiers can cause confusion.

The remaining parameter estimates do not concern Style. Given the lack of interactions, Training can be discussed independently. For each unit increase in Training, there are 0.13 fewer missed lines, which is a 0.87% decrease in the number of missed lines. Thus, overall subjects perform better, but only slightly better with increased Training. Next, from the interaction of Waldo First and Correct, when Correct is false, subjects who completed the Waldo study before the SAT study were likely to have fewer misses. This is most likely an indication of subject fatigue. Finally, none of the variables identifying the question are significant, which indicates that attributes distinguishing the fragments (e.g., such as code density) do not affect the results.

Table 9 Waldo model for Missed Lines

Variable	Estimate	p-value
Intercept	2.3356	<0.0001
Style	−0.4379	0.0019
Correct	−2.0814	<0.0001
Style * correct	0.4509	0.0150
Waldo first	−1.1900	0.0002
Training	−0.1300	0.0310
Correct * Waldo first	1.1951	0.0008

Table 10 Waldo model for Time Scanning Code when subject is correct (ASQ abbreviates Answering Speed Question)

Variable	Estimate	*p*-value
Intercept	6040.67	0.0011
Style	5314.89	0.0105
Waldo first	−3941.78	<0.0001
Time on ASQ	344.64	0.0015
Time on demographics	250.19	<0.0001
Style*Time ASQ	−279.46	0.0012

For the efficiency hypotheses, the second two models consider the response variable Time Scanning Code and are built from the subset of the data with no missed lines to avoid cases where a subject was quick at the expense of correctness. For subjects that did the task correctly, the average number of seconds spent working on the task is 17.2 (Std Dev = 68.5) for the underscore and 15.9 (Std Dev = 61.8) for camel case. The statistical model that includes Style as the only explanatory variable finds that Style is marginally significant ($p = 0.0692$) and indicates that camel case takes 1.2 fewer seconds. Practically, this is about a 7% difference, which becomes more significant with increased frequency of the task.

The complex model for Time Scanning Code is built using the remaining explanatory variables and their interactions with Style. Elimination of non-significant terms produces the model shown in Table 10. Discussion of its parameter estimates begins with those involving Style, in this case the interaction Style*Time Answering Speed Question. This term shows that for the underscore Style more time answering the Speed Question leads to more time spent on the task. However, for camel case, the opposite is true. More time on the Speed Question leads to less time on the task. This result is somewhat counter intuitive. It may be that for the underscore Style, one's personal reading speed has a greater impact. However, for the camel-case Style, the visual cues from the capital letters are more helpful to the slower readers.

The remaining parameter estimates do not concern Style. Both are free from interactions and thus can be discussed independently. First, subjects that performed Waldo First, were 3.9 s (just over 20%) faster, which suggests that fatigue is a factor accounted for by this term. Second, from Time on Demographics, subjects that take more time to read and fill in demographic information are slower at the task. This term helps to account for subject reading speed differences.

5.5 Discussion

Performance Hypothesis—Style *affects the number of* Missed Lines

Support for Performance Hypothesis is found in the first two models where Style is significant; thus, the null hypothesis can be rejected. In both models camel-case identifiers lead to greater accuracy.

Efficiency Hypothesis—Style *affects the* Time Scanning Code

Support for Efficiency Hypothesis is found in the final two models where Style is significant; thus, the null hypothesis can be rejected. In both models camel case leads to lower Time Scanning Code. From the complex model the benefit is greater for slower readers.

Summary of Results

This experiment lends more evidence to the observation that camel casing produces more accurate results. In particular, those subjects that have training are more accurate on this task. In addition, the task can be accomplished more quickly when the identifiers are in the camel-case Style, especially for slower readers. In regard to Research Question 1, the *Where's Waldo* study reinforces the affirmative answer found in the Cloud study: Style impacts readability. Given that this experiment has a more realistic setting, it represents the first step down a path of increasingly complex comprehension tasks.

6 SAT Study

Although the *Where's Waldo* study more closely represents a task carried out by engineers, it does not require significant comprehension to be successful. Turning from readability to Research Question 2, Style's impact on comprehension is now investigated. Three experiments reflect an increasing demand for comprehension. The first considers Style's impact on natural-language reading comprehension using SAT-style questions in which subjects read a passage on a topic and then, as on the SAT exam, answer two comprehension questions. This choice allows the study to use comprehension questions that have been thoroughly vetted: Gates-MacGinitie Reading Tests (MacGinitie et al. 2000) has released passages and related comprehension questions used from 1960 until they were retired in 1970. This experiment attempts to isolate Style's impact on a traditional natural language comprehension problem: a setting almost completely divorced from programming.

6.1 Experimental Design

The design of the *SAT* study is centered around the construction of two passages. After reading each passage the subject answered two multiple-choice comprehension questions. The questions were presented in the same order as used in the original SAT exam. The first passage is a paragraph from 1966 about pulsars. It contains 101 words and was given a Flesh-Kincaid Grade Level (a measure of comprehension difficulty when reading academic English) of 13.0 by Microsoft Word. This indicates a fairly difficult passage. The second passage is a paragraph from 1960 about John Quincy Adams. The original paragraph contains 127 words with a Flesch-Kincaid Grade Level of 9.4. The order of the two passages was randomly chosen to mitigate learning effects.

To introduce Style, two to five word sequences in the text of each passage were rewritten with camel casing or underscores. The words agglomerated were noun and verb phrases and thus provided sensible groupings. The questions retained traditional spacing. For example, the final sentence of the pulsar question is "Such a radio beam, striking the earth with each revolution of the neutron star, can account for the observed radio frequency pulsations" while the Adams passage includes the sentence "All day neighbors traveled the road in front of the Adams farmhouse, retreating from the expected attack." In the study a subject would have seen either

> "Such a_radio_beam, striking_the_earth with each_revolution of the_neutron_star, can_account for the_observed_radio_frequency_pulsations"

and

> "AllDay neighbors traveledTheRoad in front of theAdamsFarmhouse, retreating from theExpectedAttack"

or

> "Such aRadioBeam, strikingTheEarth with eachRevolution of theNeutronStar, canAccount for theObservedRadioFrequencyPulsations"

and

> "All day neighbors traveled the road in front of the_Adams_farmhouse, retreating from the_expected_attack"

Using traditional SAT comprehension questions, this experiment mirrors work in cognitive psychology (e.g., that of Epelboim et al. 1997 where various fillers such as letters replace inter-word spaces). It uses manipulated SAT paragraphs and accompanying comprehension questions to analyze the effect of different inter-word fillers on readability. If the results from cognitive psychology hold for this study, there should be little impact from the insertion of underscores, while camel case should slow the speed at which a subject can read the paragraph.

The SAT experiment was laid out around the two questions. Each question consisted of four screens. The first screen was used for preparation. On the second screen, a paragraph appeared in one of the two styles. On the third screen, the first comprehension question appeared. Subjects were required to select one of four options to answer the question. The remaining three options serve as distractors. Then, the subject clicked on a button to advance to the second question. The fourth screen contained the second multiple-choice question with four possible answers. Again, the subject was instructed to click a button to complete the task.

The first group of subjects in Table 1 (Group 1) completed this experiment, half did so immediately after the *Applet-cloud* experiment and the other half did so after also completing the *Where's Waldo* study. The reason for the two orders was to mitigate the impact of learning effects and fatigue. In the SAT study subjects first performed a practice question and then the two study questions in the same order. The Style in which the paragraph appeared was varied, so approximately half the subjects saw the first question using camel case and the second using underscores, while the other half saw the reverse.

6.2 Variables

The variables collected during the SAT experiment include two response variables and several explanatory variables. The response variables capture correctness and time. The correctness variable Number Correct is a count of the number of multiple choice comprehension questions the subject answered correctly. The value of this variable can be zero, one, or two. The time related variable Paragraph Read Time records the amount of time the subject spent reading the paragraph distorted with underscore or camel cased phrases.

In this experiment, the primary explanatory variable is Style. The remaining explanatory variables came from three sources: the questions, the subject's performance, and the demographic information. There is one variable relating to the passage Passage Id. There are only two passages, so this variable captures all the differences between the passages including the passage difficulty. In addition, there are two performance related explanatory variables Time Answering Speed Question, which is discussed in Section 5.2 and the Speed Question, which is the answer supplied to that question. The remaining variables come from demographic information. Training is discussed in Section 4.2. In addition, the variables Gender, Age, and Years Worked are also included. The variable Years Worked was collected in response to the query "Years of computer science related work experience".

6.3 Distractor Analysis

An item analysis was conducted on the comprehension questions related to each SAT passages to see how individual items functioned. The distractors are the incorrect answers to each of the comprehension questions. There are three distractors per question. First, the number of questions answered correctly for each passage was examined. Subjects answered more questions correctly on the Adams passage (mean = 1.45) than the Pulsar passage (mean = 0.99). A look at the individual questions shows that the second Pulsar question was more difficult than the other three with the correct answer only being selected 33% of the time. The first Adams question was the easiest, being correctly answered 80% of the time. These percentages indicate the range of question difficulty on the test. Although it is clear that some incorrect responses were selected by a disproportionate number of subjects, a subject was not more likely to choose a particular wrong answer because she read the passage in a particular Style.

6.4 Statistical Models

Only the performance and efficiency hypotheses were considered for the SAT study. Thus, four models were constructed in pairs. The first model of a pair includes only Style, while the second includes the additional explanatory variables. For the SAT experiment, the response variables are Number Correct and Paragraph Reading Time. For the response variable Number Correct, there are only three different values, so it is considered an ordinal variable and was modeled using the *Proportional Odds Model*. The response variable Paragraph Reading Time was modeled using linear mixed-effects regression models.

For the response variable Number Correct, the average number of questions answered correctly is 1.22 when reading a paragraph in both the underscore Style (Std Dev = 0.77) and the camel-case Style (Std Dev = 0.75). Given that the means are identical, the statistical model that includes only Style does not show Style as significant when predicting the odds of the number of correct answers ($p = 0.9538$).

The complex proportional odds model for Number Correct is built using the remaining explanatory variables and their interactions with Style. Elimination of non-significant terms produces the model shown in Table 11. Discussion of its parameter estimates begins with those involving Style, which means its three significant interactions. For the interaction Style*Time on Demographics, there is no significant

effect of Time on Demographics on Number Correct for the underscore Style; however, for the camel-case Style, the log-odds effect of Time on Demographics on Number Correct is $0.000525 - 0.06243 = -0.061905$. Thus, for camel case, a millisecond increase in Time on Demographics leads to an odds ratio of 0.940 for Number Correct. Thus, spending more time on the demographic page indicates a lower chance of more correct answers for the camel-case Style. The greater time could indicate that slower readers are having to work harder with the camel-case Style, which impacts their ability to correctly answer comprehension questions. For the interaction Style*Age, the effect (log-odds) of age for underscore is 0.2635 while the effect of Age for camel case is $0.2635 + 0.3157 = 0.5792$, and these effects are significantly different. The odds ratios are 1.302 and 1.785. Therefore, increasing Age, perhaps because of the greater or more varied experience, leads to a greater increase in odds of correctness for camel case than it does for underscore. Finally, for the interaction Style*Training, the effect of Training for underscore is 0.4868 ($p = 0.0559$), which has marginal significance. The odds ratio is 1.63 for underscore. Thus, a unit increase in Training leads to a 63% higher odds of correct answers for underscore. The interaction of Style*Training has a highly significant parameter (-0.9006 with a p-value of 0.0063). This reveals that the effect of Training is significantly different for camel case than for underscore. A unit increase in Training leads to the odds of a correct answers being 34% lower for camel case. This indicates that computer science training is not as helpful to performance on comprehension questions when the paragraph includes camel-cased phrases, perhaps because those with training are more distracted by the unusual format of the words in the paragraph.

The remaining parameter estimates in Table 11 do not concern Style. They include Passage Id, which has a positive parameter estimate of 1.3087, so the cumulative odds ratio is $e^{1.3087} = 3.70$. Therefore, the Adams passage has higher odds for higher values of Number Correct. This observation indicates that the paragraph was easier to comprehend for the subjects and agrees with the lower Flesh-Kincaid Grade Level. Second, Speed Question has a negative parameter estimate (-0.4603), the odds ratio is $e^{-0.4603} = 0.63$. Thus, a subject indicating a greater preference for speed over precision has lower odds for higher values of Number Correct. It is interesting that this personal preference on a game-like experiment carries over to ones performance

Table 11 SAT model for Number Correct

Variable	Estimate	p-value
Intercept0	4.2102	<0.0001
Intercept1	6.5270	<0.0001
Style	−3.0374	<0.0001
Passage Id	1.3087	<0.0001
Paragraph read time	0.000022	0.0122
Speed question	−0.4603	0.0086
Time on demographics	0.000525	0.9823
Training	0.4868	0.0559
Age	0.2635	<0.0001
Style*time on demographics	−0.06243	0.0445
Style*training	−0.9006	0.0063
Style*age	0.3157	<0.0001

in a reading comprehension experiment. Finally, with regards to Paragraph Reading Time, the parameter estimate is 0.000022 per millisecond; therefore, after converting to seconds, the odds ratio is 1.02 per second. Thus, increasing reading time leads to high odds for Number Correct.

The second two models consider the response variable Paragraph Read Time. Overall, the average number of seconds spent reading the paragraph in the underscore Style is 44.4 (Std Dev = 19.7) compared to 47.1 (Std Dev = 21.1) when the paragraph is in the camel-case Style. The statistical model that includes Style as the only explanatory variable finds that Style is marginally statistically significant ($p = 0.0868$). Subjects can read underscore paragraphs 2.7 seconds faster, which is consistent with other reading experiments (Epelboim et al. 1997). From a practical perspective, 2.7 s is roughly a six percent difference in time which, although small, could have significant impact if the task was performed frequently.

The complex model for Paragraph Read Time is built using the remaining explanatory variables and their interactions with Style. Elimination of non-significant terms produces the model shown in Table 12. Discussion of its parameter estimates begins with the two involving Style. First, after accounting for other terms, when

Table 12 SAT model for Paragraph Read Time with interactions broken out

Variable	Number Correct	Passage Id	Estimate	*p*-value
Intercept			75.14	.9936
Style			−2730.04	0.4724
Passage Id		Pulsar	3101.70	0.5620
Passage Id		Adams	0	–
Number correct	0		−5524.78	0.0535
Number correct	1		2385.14	0.2911
Number correct	2		0	–
Style*number correct	0		1401.88	0.7023
Style*number correct	1		−6390.97	0.0354
Style*number correct	2		0	–
Time answering speed question			1410.92	<.0001
Speed question			5377.72	0.0011
Time of demographics			371.01	0.0271
Training			−160.88	0.9246
Gender			6138.57	0.0323
Years worked			2909.25	0.0005
Style*Time answering Speed question			492.34	0.0352
Time answering speed Question*Passage Id		Pulsar	−490.87	0.0368
Time answering speed question*Passage Id		Adams	0	–
Speed question*Passage Id		Pulsar	−3419.91	0.0037
Speed question*Passage Id		Adams	0	–
Training*Passage Id		Pulsar	2703.56	0.0265
Training*Passage Id		Adams	0	–
Years worked*Passage Id		Pulsar	−1945.00	0.0012
Years worked*Passage Id		Adams	0	–

Number Correct is 0 or 1, there is no statistical difference between the underscore and camel-case Style; however, when a subject answers both comprehension questions correctly, the underscore Style takes significantly less time. For the interaction Style*Time Answering Speed Question when Style is underscore, the effect of Time Answering Speed Question on Paragraph Read Time is 1.4 s or about three percent. When Style is camel case, the effect is 1.9 s or about 4 percent. In both cases, as Time Answering Speed Question increases, so does the Paragraph Read Time, which indicates that a subject's personal reading speed partially explains the Paragraph Read Time. Given that the effect is larger for camel case than for underscore, subjects have a greater increase in reading time for the camel-case Style. Both interactions indicate that Paragraph Read Time is lower for the underscore Style.

6.5 Discussion

Performance Hypothesis—Style *affects the* Number Correct

While the simple model for Number Correct does not include Style, it is part of three significant interactions in the complex model. In this model, the variables Age and Training are somewhat correlated ($r = 0.438$) with greater Training accompanying greater Age. Having both increase together leads to a higher chance of correct answers for underscore. Furthermore, for underscores, an increase in either Age or Training leads to higher odds of Number Correct. More importantly, for the camel case Style, the effects of the two variables are opposite, which means one is acting as a dampening factor for an over-estimate by the other. This opposing effect together with slower readers having a lower chance of more correct answers for the camel-case Style supports the rejection of the null hypothesis.

Efficiency Hypothesis—Style *affects the* Paragraph Reading Time

Support for the Efficiency Hypothesis is found in both models where Style is at least marginally significant; thus, the null hypothesis can be rejected. The simple model finds that subjects reading paragraphs written in the camel-case Style take 2.7 s (six percent) longer. When considering the more complex model, restricted to correct answers, camel case leads to longer Paragraph Reading Time than underscore. In addition, for slower readers having a higher Time Answering Speed Question, camel case again leads to longer Paragraph Reading Time.

Summary of Results

The SAT experiment represents a more complex comprehension task than the prior two experiments. Providing an affirmative answer to Research Question 2, Style has a significant impact on comprehension where the use of camel casing reduces performance and also slows subjects down, especially slower readers. This result is the opposite of most of the results from the first two studies where camel casing was favored. To disambiguate these two conflicting results, the next two studies consider problems that require greater comprehension (as in the SAT study) but in a software context (as in the Where's Waldo study and Cloud study experiments).

7 Eye Tracker Code Study

Building on the SAT experiment's reading of natural language, the fourth study (*Tracker-code*) considers reading code. The study's two goals investigate Style's impact on the Visual Effort required while reading code and its impact on short term memory demand. These two goals relate to Research Question 2.

7.1 Experimental Design

The study involves analyzing the C++ code stimulus shown in Fig. 4, which is presented in 11-point Courier New font. The code stimulus includes two functions, main and matrixSum, with six and fifteen identifiers, respectively. Both Styles are present on the same stimulus. While identifiers occurred multiple times, the frequencies are closer to that of the SAT study than the higher repetition found in the readability studies. This study was performed after the *Tracker-cloud* experiment on the same day. The same subjects (Group 3 from Section 2.3) participated.

Subjects were verbally instructed to study and memorize the code so that they could reproduce it and answer questions regarding it. They were allowed to study it for as long as they wanted. While doing so, eye gaze data (fixations) were recorded. After the subjects were done studying the code, the next screen (the identifier-recall stimulus shown in Fig. 5) asked participants to verbally state the identifiers that they

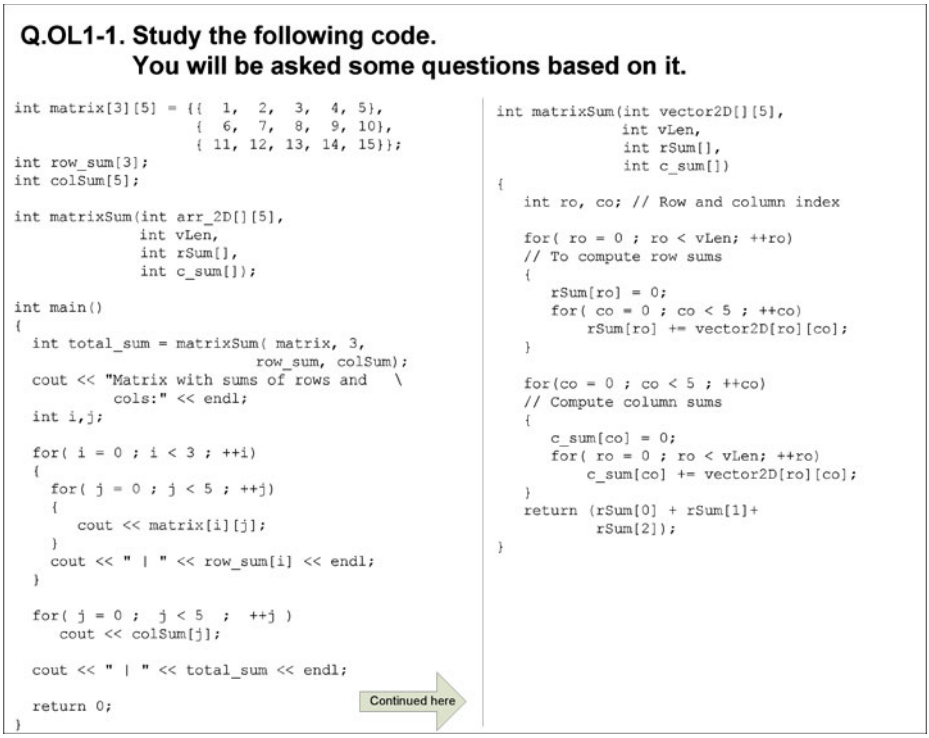


Fig. 4 Code stimulus for the eye tracker code study

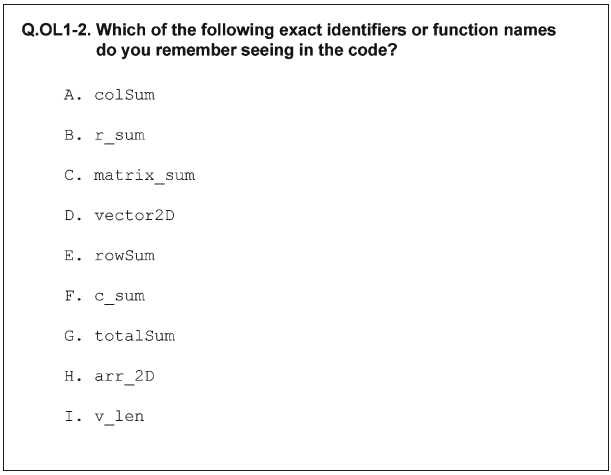


Fig. 5 Identifier-recall stimulus for the eye tracker code study. The correct answers in Fig. 5 are colSum, vector2D, c_sum, and arr_2D

remembered seeing in the code. This screen also included five distractors. The four correct choices included two camel-case and two underscore identfiers. Visually, each identifier on both screens was an area of interest enclosed within a rectangle (not visible to the subject).

7.2 Variables

Two response variables were collected: Recall Correctness and Visual Effort. Recall Correctness refers to the number of identifiers correctly recalled. Visual Effort, as described in Section 4.2, includes eye fixations and their durations within specific areas of interest. In this case, the areas of interest were bounding boxes around the identifiers with some extra padding to compensate for small drifts. The main explanatory variable is Style. The three other explanatory variables considered are Recall Time, the amount of time taken to answer the multiple choice question, Code Reading Time, the number of milliseconds spent reading the code stimulus, and Experience, as defined in Section 4.2.

7.3 Experimental Hypotheses

Four hypotheses are considered. In addition to the Performance Hypothesis, two concern Visual Effort and one deals with Experience and its interaction with Recall Correctness. The Efficiency Hypothesis, which would be based on recall time is not considered because the total recall time for the identifier-recall stimulus includes times for both camel case and underscore identifiers in this experimental setup.

Hypothesis Code 1:

H_0 Visual Effort while reading code or recalling identifiers is the same regardless of the Style of the identifier.

H_A Visual Effort while reading code and recalling identifiers is affected by identifier Style.

This hypothesis considers eye fixations and durations for both the code and identifier-recall stimuli. A style that requires less Visual Effort is more useful than one that requires greater effort.

Hypothesis Code 2:

H_0 The effect of Style on Visual Effort is independent of Experience for reading code or recalling identifiers.
H_A The effect of Style on Visual Effort lessens due to Experience for reading code and recalling identifiers.

This hypothesis considers if Style's influence on Visual Effort diminishes with Experience. Support would be found in an interaction between Style and Visual Effort.

Hypothesis Code 3:

H_0 The effect of Style on Recall Correctness is independent of Experience.
H_A The effect of Style on Recall Correctness lessens due to Experience.

This hypothesis considers if Experience can overcome differences in Recall Correctness caused by Style. Support would be found in an interaction between Style and Recall Correctness.

7.4 Distractor Analysis

A distractor analysis was performed on the eye-tracker data to determine if any of the five incorrect answers were selected more or viewed proportionately longer than the other answers. The distractor that produced the most fixations and greatest gaze duration was matrix_sum, which was also the longest choice among all the identifiers. It could be that the length of this item made it more difficult to process; however, this item did not produce a greater gaze per fixation than the other items. Thus, it may be that the greater length made the item more apparent and thus more likely to be fixated upon, even if only briefly. In all, other than matrix_sum, the distractors were not fixated upon or gazed at more than the correct answers.

7.5 Statistical Models

Statistical models are generated in pairs with the simple model including Style as the only explanatory variable and the complex model initially including additional explanatory variables and their interactions with Style. In total, ten models are generated, two for each of the following: Recall Correctness, the fixation counts on the identifier-recall stimulus, fixation counts on the code stimulus, fixation durations on the identifier-recall stimulus, and fixation durations on the code stimulus.

None of the subjects gave completely correct answers. On average, one camel case and one underscore identifier were recalled correctly. One possible explanation for this rather low recall is the visual confusion that occurs when similar camel case and underscore versions of an identifier (e.g., c_sum and colSum) are present in the same code fragment. The statistical model that includes Style as the only explanatory variable shows that Style does not significantly affect correctly recalling the identifiers ($p = 0.451$).

Table 13 Complex model for Recall Correctness

Variable	Estimate	*p*-value
Intercept	0.875	0.001
Style	0.375	0.251
Experience	−0.964	0.007
Style*experience	1.232	0.013

The complex model for Recall Correctness initially includes the other explanatory variables and their interactions with Style. After elimination of non-significant terms, the resulting final model includes a significant interaction between Style and Experience. The final model is shown in Table 13. and the interaction is pictured in Fig. 6. In this model, two pairwise comparisons are significant: underscores for experts versus beginners ($p = 0.007$) and camel-casing versus underscores for beginners ($p = 0.015$). These differences are clearly visible as large vertical gaps in the interaction plot. However, it is important to realize that, statistically, the two points for experts are not different. In particular it is tempting to observe that camel case for beginners lies between the two expert points. None of these differences is statistical significant. In summary, beginners do better with camel case, and experts do better than beginners when using underscores.

Visual Effort is measured based on the areas of interest defined on the code stimulus and the identifier-recall stimulus. A total of eight models were generated. The simple models use only Style as the explanatory variable. The complex model uses all six variables including Recall Correctness and all the interactions with Recall Correctness and Style. In both the simple and complex models for the code stimulus and the identifier-recall stimulus, Style is not significant.

The design of this experiment was within-subjects (each subject was asked the same question and tested on both styles). This design supports comparison of each individual's performance on the camel-cased and underscored version of similar

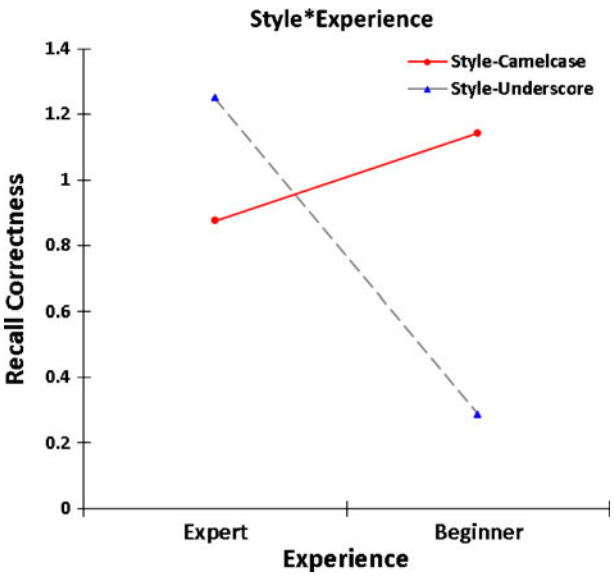


Fig. 6 The interaction of Style and Experience when modeling Recall Correctness

identifiers using the non-parametric Wilcoxon paired test. Two pairs of similar identifiers, row_sum and colSum, and c_sum and rSum are compared for differences in Visual Effort. To normalize for the number of appearances of each identifier, the fixation counts for each are divided by the number of appearances. Statistically, row_sum requires significantly more fixations (5.3 more out of a maximum 23.7 fixations on average) than colSum (1-tailed $p = 0.007$), its average gaze duration was 704 ms (out of a maximum 3,747.2 ms of gaze duration on average) longer than that of colSum (1-tailed $p = 0.005$). No significant differences were found between c_sum and rSum. One possible reason could be that the placement of identifiers might affect Visual Effort needed. The first pair (row_sum and colSum) appeared in the main function, while the second pair (c_sum and rSum) appear in the method matrixSum, which may have accounted for the discrepancy.

7.6 Discussion

Performance Hypothesis—Style *affects the* Recall Correctness

While Style is not significant in the simple model, an interaction with Experience is significant in the complex model. The significance indicates that Experience affects how Style impacts Recall Correctness. In particular, beginners with low experience tend to do better with camel cased identifiers versus underscored ones. On the other hand, experts do better with underscored identifiers compared to beginners. Therefore, there is limited support for rejecting the null hypothesis when Experience is taken into account.

Hypothesis Code 1—Style *affects* Visual Effort *while reading or recalling identifiers*

There is no evidence to reject the null hypothesis in the models for Visual Effort where Style is not significant. The pairwise comparison does find that in one instance the use of underscores requires greater Visual Effort (in both fixation count and gaze duration).

Hypothesis Code 2—*The effect of* Style *on* Visual Effort*, lessens due to* Experience

Because Experience's interaction with Style is not significant in the models for Visual Effort, there is insufficient evidence to reject the null hypothesis.

Hypothesis Code 3— *The effect of* Style *on* Recall Correctness *lessens due to* Experience

Because the interaction between Style and Experience in the complex model for Recall Correctness is significant, the null hypothesis can be rejected. In particular, for beginners Recall Correctness is negatively impacted by the presence of underscores. Thus, the camel-cased style emerges as the better choice. Finally, while experts apparently can compensate for the difference seen in the beginners, such compensation might require effort that could be better spent elsewhere.

Summary of Results

Style continues to play a role in Performance (in this case Recall Correctness). Taking into account Experience, underscores hurt the performance of beginners

(Hypothesis Code 3). From the Visual Effort models, overall, there was no significant difference except in the pairwise comparisons where the underscored identifier row_sum takes significantly more and longer fixations than the camel cased identifier colSum.

Considering Research Question 2, when given a code comprehension task (i.e., studying code well enough to be able to replicate it or answer questions with respect to it), the results for Hypothesis Code 1 and Code 2 indicate that subjects may not actually retain the Style in which they saw the identifiers.

8 Read Aloud Study

Like the prior study, the *Read-aloud* study asks subjects to read a method. However, unlike the prior studies that focus on subtasks of comprehension within source code such as readability and memory recall or comprehension of natural language text, in

```
public double percent_long_words(String input_text)                    ┐
{                                                                      │
   int char_count = 0;                                                 │
   int word_count = 0;                                                 │
   int long_word_count = 0;                                            │ Quadrant 1
                                                                       │
   String modified_input_text = input_text + " ";                      │
   int current_letters_in_word = 0;                                    │
   char current_character;                                             ┘
   for (int k=0; k ¡ modified_input_text.length(); k++)                ┐
   {                                                                   │
      current_character = get_nth_character(modified_input_text, k);   │
      char_count++;                                                    │
      if ((current_character >= 'A') && (current_character <= 'Z') ||  │ Quadrant 2
          (current_character >= 'a') && (current_character <= 'z'))    │
      {                                                                │
         current_letters_in_word++;                                    │
      }                                                                ┘
      else                                                             ┐
      {                                                                │
         if (current_letters_in_word > 0)                              │
         {                                                             │
            word_count++;                                              │
            if (current_letters_in_word > 6)                           │ Quadrant 3
               long_word_count++;                                      │
            current_letters_in_word = 0;                               │
         }                                                             │
      }                                                                │
   }                                                                   ┘
   return 100.0*long_word_count/word_count;                            ┐
}                                                                      │ Quadrant 4
                                                                       ┘
```

Fig. 7 First method in the *Read-aloud* experiment shown in the underscore style. The quadrants were used to assess regions of the code

this fifth study, the task is a true software engineering comprehension task, where subjects are asked to summarize a method in source code. Subjects from Group 2 (see Section 2.3) considered two methods, which were written by the authors to be comprehensible to students after taking two semesters of Java programming. They were engineered to have a purpose that could be discerned by the subjects. These methods were designed to be code analogues to the SAT questions. The first method, appearing in Fig. 7, calculates the percentage of *long* words in its input. The second method detects a full house in a Yahtzee roll and appears in Fig. 8.

8.1 Experimental Design

The *Read-aloud* study was designed to investigate Style's impact on a subject's comprehension, observed by asking the subject to verbally summarize a block of code. The choice of a verbal study may impact the results but is a pragmatic approach to the design because it mirrored a tutorial session that the students were familiar with and alleviated the need to properly vet SAT quality comprehension questions.

```
public boolean fullHouse(int [] yahtzeeRoll)                    } Quadrant 1
{
   int numberOfDice = yahtzeeRoll.length;
   int dieOneValue = yahtzeeRoll[0];
   int dieTwoValue = -1;

   for (int dieIndex=1;                                         } Quadrant 2
            dieIndex < numberOfDice && dieTwoValue == -1;
            dieIndex++)
   {
      if (yahtzeeRoll[dieIndex] != dieOneValue)
      {
         dieTwoValue = yahtzeeRoll[dieIndex];
      }
   }

   int dieOneCount = 0;                                         } Quadrant 3
   int dieTwoCount = 0;

   for (int dieIndex=0; dieIndex < numberOfDice; dieIndex++)
   {
      if (yahtzeeRoll[dieIndex] == dieOneValue)
      {
         dieOneCount++;
      }
      else if (yahtzeeRoll[dieIndex] == dieTwoValue)
      {
         dieTwoCount++;
      }
   }

   return dieOneCount == 2 && dieTwoCount == 3                  } Quadrant 4
            || dieOneCount == 3 && dieTwoCount == 2;
}
```

Fig. 8 Second method in the *Read-aloud* experiment shown in the camel-case style. The quadrants were used to assess regions of the code

The other alternative would be to ask for the response as an essay question, which also has its drawbacks. For example, the subjects were not familiar with the particular mode of response.

The study presented subjects with a page of source code written in Java. Subjects were first given as much time as they needed to familiarize themselves with the code. Before studying the code, they were instructed that "when you are ready, explain in detail what it does. For the level of detail, pretend that you are explaining it to a freshman (first year undergraduate)." To begin, the subjects were given a short example shown in Fig. 9 and after explaining it, a sample explanation was distributed to help the subjects understand what was meant by a "source code summary". The summary is as follows

> The method maximum takes an integer array and returns an int. It first checks if the array is empty. If so, zero is returned. Otherwise, 'max' is assigned the first element of the array. The code then iterates through the rest of the array checking if 'max' is less than the current element. If so, it replaces max with that element. Finally, the method returns the maximum value from the array.

After training, subjects studied and then explained the two methods shown in Figs. 7 and 8. Each subject saw the string-processing method shown in Fig. 7 first and then the Yahtzee method shown in Fig. 8 second. Half of the subjects saw the first method written with identifiers in the camel-case Style while the other half saw the underscore Style. The second method was shown in the opposite style.

The layout of this study has subjects sit at a table with a moderator. An audio recording was made of all interactions. From this recording, timing information was gathered. Audio files containing the summaries were distorted to provide anonymity to the subject. Subjects completed this experiment within three weeks of having participated in *Applet-cloud*, *Where's Waldo*, and *SAT* experiments.

This experimental design does not lend itself to distractor analysis because of the free form nature of the answers. Therefore, no such analysis was performed for this study.

Fig. 9 Example method in the *Read-aloud* experiment

```
public int maximum(int [ ] array)
{
   if (array.length == 0)
   {
      return 0;
   }

   int max = array[0];

   for (int i = 1; i < array.length; i++)
   {
      if (max < array[i])
      {
         max = array[i];
      }
   }

   return max;
}
```

8.2 Variables

The variables collected during the *Read-aloud* experiment include two response variables, Thinking Time and Quality Assessment, and several explanatory variables. Thinking Time captures the amount of time that the subject spent reviewing the code before summarizing it. Quality Assessment captures correctness, which was assessed on a ten point Likert scale independently by two of the authors, using a rubric designed for the task. The assessments were then averaged together using an arithmetic mean. This assessment considered separately four quadrants of each function as indicated in Figs. 7 and 8. The primary explanatory variable is again Style. There are two significant explanatory variables, both relating to the question: Method Id and Quadrant Id. The Method Id captures which of the two methods is considered. The quadrants represent distinct concepts (e.g., initialization) within the code.

8.3 Statistical Models

As in the prior experiments, the models were constructed in pairs. The first model of a pair includes only Style, while the second includes other explanatory variables. The response variables are Quality Assessment and Thinking Time. In total, four models are produced using linear mixed-effects regression.

The first two models consider the response variable Quality Assessment. The average quality assessments was 7.5 (Std Dev = 2.29) for the underscore Style and 7.2 (Std Dev = 2.60) for the camel-case Style. The statistical model that includes Style as the only explanatory variable shows that Style is not statistically significant ($p = 0.3048$).

To account for other factors that might influence Quality Assessment, a complex model including interactions with Style is built. As shown in Table 14, after the elimination of non-significant terms, two terms remained: Method Id and Thinking Time. After accounting for Thinking Time, the string-processing method leads to an assessment whose score is 1.25 (17%) lower, on average, than that for the Yahtzee method. After accounting for Method Id, a ten second increase in Thinking Time leads to a 0.1 decrease in Quality Assessment. This trend reveals that the string-process method was more difficult and that more thoughtful students, as measured by Thinking Time, give slightly higher quality answers; however, at 1%, the difference has little practical significance.

The second two models consider the response variable Thinking Time. Overall, the average number of seconds spent thinking was 148.9 (Std Dev = 64.3) for the underscore Style and 142.1 (Std Dev = 67.9) for the camel-case Style. The statistical model that includes Style as the only explanatory variable shows that Style is not statistically significant ($p = 0.6129$).

Table 14 *Read-aloud* model for Quality Assessment

Variable	Method Id	Estimate	p-value
Intercept		9.5962	<0.0001
Method Id	String-processing	−1.2535	<0.0001
Method Id	Yahtzee	0	–
Thinking time		−0.01108	0.0035

To account for other factors that might influence Thinking Time, all non-demographic explanatory variables described in Section 8.2 and interactions with Style were included to build the model. After the elimination of non-significant terms, one term remained: Quality Assessment, where a unit increase in Quality Assessment for Quadrant 2 leads to a decrease of 8.9 s (6%) in Thinking Time ($p = 0.0148$). Thus, more proficient subjects comprehended the second quadrant, the key part of the method, more quickly.

8.4 Discussion

Performance Hypothesis—Style *affects the* Quality Assessment

Both models for Quality Assessment show no support for the Performance Hypothesis because Style is not significant in either model. Therefore, the null hypothesis cannot be rejected. This study does not find Style having an effect on the comprehension of source code.

Efficiency Hypothesis – Style *affects the* Thinking Time

Both models for Thinking Time show no support for the Efficiency Hypothesis because Style is not significant. Therefore, the null hypothesis cannot be rejected. This experiment does not show any effect of Style on the time it takes a programmer to understand the source code of the four methods.

Summary of Results

In terms of Research Question 2, the lack of significance of Style is somewhat of a surprise considering Style's impact on readability found in the earlier studies and its impact on comprehension as found in the SAT study. It may be that there were too many confounding variables to isolate the effect of Style. For example, comparing means shows there is some difference between Styles, which a different test instrument might capture. Alternatively, a larger subject pool could demonstrate that this difference is not a random effect. Even without finding a significant difference between the two styles, this study highlights that reading and comprehending natural language is fundamentally different from reading and comprehending source code.

9 Threats to Validity

This section presents four main threats to validity: internal, external, construct, and conclusion in separate sub-sections. Each section describes threats individually for each study as well as threats that are common across more than one study.

9.1 Internal Validity

Internal validity refers to the degree to which conclusions can be drawn about the causal effect of the explanatory variables on the response variables. One internal threat applies to all studies. Statistical associations do not imply causation; although,

given the underlying theory, it is reasonable to infer that differences in performance are due to the explanatory variables considered. Other potential threats to internal validity (e.g., history effects, attrition, and subject maturation (Sjøberg et al. 1993)) are non-issues given the short duration of the studies.

Of the threats that apply to only some of the studies, the most common are learning effects. Because of the number of questions involved in *Applet-cloud* and *Tracker-cloud* studies, learning effects may occur when subjects believe that they have detected a pattern in the problems and attempt to exploit it. To help mitigate this concern, the *Applet-cloud* experiment presented questions in a random order. Due to the complexity of the eye tracker, the same was not possible in the *Tracker-cloud* experiment. The *Tracker-cloud* experiment, unlike the *Applet-cloud* experiment, uses a within-subjects design by asking the same subject to find an identifier in both camel case and underscore styles. To mitigate learning effects, different but similar phrases are used for each identifier style within each Code and Non-Code phrase category. Analysis on the different phrases show no statistical difference between the corresponding similar identifiers with the exception of *full pathname* that seemed to be harder than its corresponding camel case version, *startTime*, which hinders the underscore approach for 2-word code identifiers.

The Cloud study consists of two experiments, *Applet-cloud* and *Tracker-cloud*. Specific threats include that different demographics are collected, which may pose a threat to conclusions that combine results from the two studies. Different demographics were captured because of differences in the subject pools. Subjects in the *Applet-cloud* study included both programmers and non-programmers whereas in the *Tracker-cloud* only programmers, some with industrial experience, were used. This lead to the need for slightly different demographic variables. For example, Experience would be false for almost all subjects taking part in the *Applet-cloud* experiment. Thus, rather than Experience, Training is captured. The threat to internal validity here occurs when conclusions involving Training and Experience are drawn from combining models from these two experiments. As noted in Section 4 no head-to-head comparisons are made.

There is also the possibility of learning effects in the *Where's Waldo* and *SAT* studies, but in this case there is more an issue of experience with one part of the experiment influencing the other. Randomization was used to help mitigate this threat. Analysis of the impact of question order suggests the randomization acted more as a control for fatigue than learning. Finally, there are two threats specific to the *Tracker-code* study. First, the position of the identifiers may affect the eye-gaze data. Second, the code shown used both camel-cased and underscored identifiers. The first of these could be mitigated by randomizing the positions, while the second was inherent in the design of the study.

In the case of the studies using an eye-tracker (i.e., *Tracker-code* and *Tracker-cloud*), the use of a moderator can cause additional bias due to the observer or Hawthorne effect. This effect could potentially cause subjects to modify their answers because they are being studied regardless of the treatment factor. The purpose of the moderator was to conduct a proper calibration of each subject and to ensure that the eye tracker does not loose tracking of the eyes. Some people tend to move or rest back in their chairs which causes the eye tracker to not track their eye movements. The eye tracker is very forgiving but the head does need to be within a certain range (head box). The moderator's job is to alert the participant if tracking is lost, which

did not occur often during this study. To avoid putting the burden of accurate data collection on the participant, a moderator was used. It is highly unlikely that the subject changed their answers because the moderator was in the room. There was minimal interaction and virtually no face-to-face contact between the moderator and subjects during the study; with each facing their own monitor screen. The moderator in no way steered the subjects towards one answer or another.

The Hawthorne effect may also be present in the *Read-aloud* study where subjects were expected to explain the source code to the moderator. The moderators attempted to give the same positive non-verbal cues to all subjects regardless of the answer. In addition to mitigate the effect, subjects were asked, at the end of the study, to rate their comfort during the study on a Likert scale. Analysis of this data did not uncover any significant evidence of a Hawthorne effect.

The use of verbal answers in the *Read-aloud* and eye tracker studies (*Tracker-code* and *Tracker-cloud*), pose a social desirability threat, referring to the fact that subjects change their answers to please the experimenter by improving their measured performance. The subjects in the studies did not have any knowledge of the hypotheses in the experiment. It would have been next to impossible for them to guess where to perform better. Hence, this threat is not an issue.

9.2 External Validity

External validity, sometimes referred to as selection validity, is the degree to which the findings can be generalized to other (external) settings. An external threat to all studies uniformly is the assumption that phrases were read left-to-right. It is possible that the results for those who learned to read top-to-bottom or right-to-left might differ; however, the number of subjects in this study that did not learn to read left-to-right is small (no more than 1 or 2).

In addition, for the Cloud study, there is a concern that right-to-left readers might not look at other choices before they state the correct answer, if the correct answer is to the right of the screen. The same threat applies to left-to-right readers if the correct answer is to the left of the screen. To determine if the reading behavior on screen plays a role, the gaze plots for each subject (for the *Tracker-cloud* experiment) were analyzed. It was found that all clouds were examined by subjects to arrive at an answer, thus mitigating the effect.

In the *Applet-cloud* experiment, selection bias exists if the the selected phrases and distractors are not representative; thus, results may not accurately capture the impact of Style on an engineer's performance. Careful selection of phrases mitigates the impact of such bias. In addition, the distractor analysis shows that no type of distractors had any more or less effect than another. It is also possible that taking the code out of context changes the reading process and, thus, the results are not indicative of typical coding situations. Past research on reading natural language suggests that this is not the case (see Section 10). Finally, selection bias is also possible in the selection of subjects. A wide variety of students took part in the study. Based on the demographics data, they represent a representative cross section of Loyola students. Furthermore, the only demographic variable that was statistically significant in any of the models was Training, which was controlled for in the selection of the subjects. However, given that essentially all training was with camel casing, the *Tracker-cloud* experiment mitigates this bias because subjects in that study were primarily trained

on underscores. Finally, the untrained group represents two-thirds of the population, which may mean that the study is under powered to detect all training effects. Thus, for example, Hypothesis Cloud 1, may find support with a larger sample of trained subjects; however, the statistical techniques can accommodate unbalanced data. The drawback of fewer subjects can be an inability to distinguish between random chance and a real difference in the population. For the *Tracker-cloud* experiment, the same threats apply to the phrase selection as discussed in the *Applet-cloud* experiment.

For both *Where's Waldo* and *SAT*, external validity is a concern with regards to questions and subjects. In the *Where's Waldo* experiment, source code was selected that used similar variable names. Although such code exists in the wild, it may have consisted of a disproportional number of identifiers. In addition, the selection of the identifier for which to search may have introduced additional threats to validity. The identifiers were selected to share words with other identifiers in the code in an attempt to make the problem harder. In the *SAT* experiment, the subject matter of the text may have introduced some threats to validity because the text may not represent some average notion of natural language text. These studies have the same threats to validity with regard to the subjects that the *Applet-cloud* experiment has. Selection bias is also possible in the *Read-aloud* study. This selection bias comes from both the questions and the subjects. The questions remain a significant threat as they were generated by the authors. Although they did achieve the goal of making the task tractable for the subjects, they did not lead to great differences in performance. Although the subjects represent a fair cross section of Loyola Computer Science students, these student may not represent the general community.

9.3 Construct Validity

Construct validity assesses the degree to which the variables used in the study accurately measure the concepts they purport to measure. In the *Applet-cloud* experiment, most of the explanatory variables (e.g., Length and Age) can be measured precisely. This threat is only a concern for the variable Phrase Origin where the authors' assessment of a phrase's origin may differ from subject perception because of the assumption that phrases that originate in code will be less familiar to the subjects who were non-programmers. This threat is a concern because it is difficult to predict how familiar a given subject will be with a particular set of words. Choosing domain terms used in source code helps the unfamiliar phrases to be truly unfamiliar and thus helps address this concern. With respect to both eye-tracking studies, because visual attention is related to mental processing of the information (Just and Carpenter 1980), the measures derived from the fixation counts and durations should be valid. Also, six measures for Visual Effort were used to avoid mono-method bias. In the *Where's Waldo* and *SAT* experiments, there was no threat to construct validity because all variables were measured precisely (assuming participants accurately reported their demographic data). In the *Read-aloud* experiment, there were two variables that present concerns. The first is the response variable Quality Assessment where two of the authors rated the responses for each quadrant. It is possible that there was assessor drift as the range of responses became better known and a difference in interpretation of the scale of the assessment. The assessments were done in a single sitting to mitigate the former and extensive discussion was conducted to mitigate the latter.

9.4 Conclusion Validity

A threat to statistical conclusion validity arises when inappropriate statistical tests are used or when violations of statistical assumptions occur. Because all of the studies follow the same statistical modeling techniques, the same threat applies to all the studies presented here. The models applied are appropriate for analyzing unbalanced, repeated-measures data, so that the conclusions drawn from the statistics should be valid. In the *Tracker-code* study, the non-parametric Wilcoxon test was used to compare the results of two identifiers in the pairwise comparisons.

10 Related Work

Related work on identifier names, source code readability, eye-tracking research concerning source code and diagrams, and the psychological aspects of reading is presented in this section. There is a large body of research in the field of cognitive psychology on how people read natural-language prose and how they parse the words and syntax. There is also research on how this information is structured and how to determine the semantics of identifiers. This section covers some of the most closely related works in the context of program comprehension.

10.1 Identifiers Names

Lawrie et al. (2006) conducted a large study on identifier names and showed that actual words rather than abbreviations lead to better comprehension. Butler et al. (2010) study the effect of identifier names on the quality of code. They found that identifiers violating certain guidelines lead to lower code quality (more bugs). Caprile and Tonella (2000) study the restructuring of identifier names and the arrangement of individual words in identifiers. Binkley et al. (2009b) study the effect of identifier length on the recall ability of programmers, showing that longer names reduce correctness and take longer to recall. The results add to these findings, because phrase length significantly interacts with identifier Style to have an effect on performance.

10.2 Eye Tracking

A small number of eye-tracking studies have recently been done on how programmers comprehend UML class diagrams. Yusuf et al. (2007) conducted a study to determine if different class diagram layouts with stereotype information help in solving design tasks. Another study by Guéhéneuc (2006) uses an eye tracker to investigate how designers answer two simple questions about modifying parts of the diagram. They also study the effect of the presence of the Visitor pattern in class diagrams (Jeanmart et al. 2009). Sharif and Maletic applied eye tracking to the effect of layout of UML diagrams on understanding roles in design patterns (Sharif and Maletic 2010b). These works do not investigate how programmers understand code and identifier names. They do not address the issue of naming styles.

Porras and Guéhéneuc (2010) conduct a study using an eye tracker with 24 subjects on the efficiency of three design pattern representations in class diagrams.

Three design pattern comprehension tasks were assessed: identifying pattern composition, design pattern roles, design pattern participation. Results indicate design pattern representations that show stereotypes are more efficient for some design comprehension tasks. Developer's effort is measured using average fixation duration and two derived measures: ratio of fixation count and ratio of fixation time. The derived measures are based on areas of interest and areas of glance. The areas of glance correspond to the distractors in the *Tracker-cloud* study. The average fixation duration (AFD) is also used in the *Tracker-cloud* experiment. The *Tracker-code* study uses the actual gaze duration instead of the average fixation durations because the areas are much smaller and based on psychology research summations are preferred in this context. The measures used in eye tracking studies are dependent on the task: reading text vs. reading a diagram. The task in the studies here was mainly reading. The tasks in Porras and Guéhéneuc (2010) relate to searching and exploring relationships between classes in UML class diagrams. Determining which metric is most appropriate for the different types of studies is still an open issue.

Although many eye tracking studies relate to evaluating user interfaces (Beymer and Russell 2005; Cutrell and Guan 2007; de Kock et al. 2009; Goldberg et al. 2002; Matsuda et al. 2009; Nakamichi et al. 2006), there are a few studies on how programmers read and comprehend source code (Bednarik and Tukiainen 2006, 2008; Crosby and Stelovsky 1990; Uwano et al. 2006). Crosby and Stelovsky (1990) study eye-gaze data of beginner programmers and expert programmers to determine if experience has an effect on viewing patterns. Uwano et al. (2006) study eye viewing patterns of five subjects while they detect defects in source code. Their recent work focuses on multi-document review (Uwano et al. 2008). Bednarik and Tukiainen (2006) study the comprehension of Java programs using eye tracking data on 18 subjects and call for more studies due to important behavior that can be revealed using eye-tracking data. They expand their study on eye tracking by considering pair programmers simultaneously (Sami et al. 2008). In addition, Bednarik and Tukiainen (2008) also investigate debugging behavior of 14 subjects while they debug a program in an IDE setting. The focus of this work is on debugging rather than identifier naming styles and how they may impact debugging.

10.3 Psychological Aspects

In the field of cognitive psychology a number of investigations are relevant. New et al. consider the impact of word length on lexical decision latencies (New et al. 2006). They provide a general understanding of human lexical processing, which is then built upon in this study to focus on computer science. The key finding was that words with a length of 6–9 letters have the highest probability of being processed after a single fixation. Their study is relevant because camel casing essentially produces long words while the use of underscores does not because they are perceived as spaces. Epelboim et al. (1997) conduct a study on the effect fillers have on reading time. Spaces between words are filled with different fillers: Latin and Greek letters, digits, and shaded boxes. They found that the type of fillers had a significant effect of slowing reading speed anywhere between 10–75% depending on the filler. Shaded boxes between words (similar to underscores) had the smallest effect on reading time. Rayner et al. (1998) also show decrease in reading rate by approximately

50% when fillers like x were used between words. The results in the *Applet-cloud* experiment support the above findings because a significant improvement in Find Time for underscores is shown. In light of this study, lexical decision latencies for camel cased identifiers would be quite long because camel casing provides insufficient separation (e.g., the identifier EqualsIgnoreCase is potentially read as a sixteen letter word, which is well into the inhibitory range).

Epelboim et al. observe that some Asian languages (such as Thai or Japanese) are written without spaces and that some Western languages (e.g., German and Dutch) use spaces much more sparingly than the languages for which the popular space-based word recognition theories were developed (e.g., English and French). Thus, it appears that readers, even those initially exposed to space-rich languages, can be trained to use something other than spaces to determine reading saccades. The findings bear out this effect as training appears to help subjects read camel cased identifiers more effectively than those who have had no training.

A number of researchers have investigated how identifiers relate to natural language and describe the solution domain. They develop or model a lexicon of words used in common across programs or for particular projects. Høst et al. analyze Java method implementations and extract the meaning of the verbs used in the method names (Høst and Østvold 2008). The resulting domain-neutral lexicon of verbs represents the common usages across a set of programs. Caprile and Tonella investigated building a standardized lexicon from the words found in identifiers and then producing a grammar for the arrangement of these terms (Caprile and Tonella 2000). Deissenböck and Pizka present a formal model based on bijective mappings between concepts and names for adequate identifier naming (Deißenböck and Pizka 2005). The model is based on the idea that within a given program a concept should always be referred to by the same name. This work sets the stage for the cognitive processes of reading programs.

11 Discussion of Results

The studies described in this paper address the impact of identifier Style on readability and comprehensibility of source code. This section first summarizes the results of the five studies (as highlighted in Table 15) and then discusses a number of implications that arise from the findings. The two research questions are reiterated below for convenience.

Research Question 1: Mirroring research on natural-language reading, does identifier style impact readability?

Research Question 2: Assuming a difference in readability, does identifier style affect higher-level comprehension activities?

The first two experiments (*Applet-cloud* and *Tracker-cloud*) determine if Style impacts readability at the syntactic but not semantic level. The results showed that subjects produce more accurate results using camel-case identifiers but at a cost: the time and visual effort to read camel-case identifiers is greater than those with underscores. The eye tracking data shows that an increase in visual effort for reading camel-case identifiers. These results concur with research done in cognitive psychology, particularly, the results of New et al. (2006), which found that words

Table 15 An overview of the studies conducted

Study	N	Description	Raw data source	Significant explanatory variables	Response variables	Results summary
The Cloud Study						
Applet-cloud (Section 4)	135	Given a phrase, find the correct identifier from a set of distractors.	Mouse clicks	Length Phrase Origin Reading Time Time on Demographics Training	Correctness Find Time	Camel case is more accurate (1) Camel case takes more time (2) Identifier style impacts readability (3)
Tracker-cloud (Section 4)	15	Given a phrase, find the correct identifier from a set of distractors.	Eye gaze, verbal	Length Phrase Origin	Correctness Find Time Visual Effort	Camel case takes more time (2) Identifier style impacts readability (3) Camel case takes more visual effort (4)
Where's Waldo (Section 5)	135	Find all occurrences of an identifier among a set of distractors.	Mouse clicks	Correct Speed Question Time Answering Speed Question Training Waldo First	Correctness Time Scanning Code	Camel case is more accurate (1) Camel case takes less time (2')
SAT (Section 6)	135	Read a passage and answer questions to gauge comprehension of passage content.	Mouse clicks	Age Gender Passage Id Speed Question Time Answering Speed Question Time on Demographics Training Years Worked	Number Correct Paragraph Read Time	Camel case is less accurate (1') Camel case takes more time—especially for slow readers (2)

Table 15 (continued)

Study	N	Description	Raw data source	Significant explanatory variables	Response variables	Results summary
Tracker-code (Section 7)	15	Analyze eye movements on code with both identifier styles and determine identifier recall ability.	Eye gaze, verbal	Experience	Recall Correctness Visual Effort	Camel case takes less visual effort (4') Experience interacts with Style and impacts Correctness (5)
Read-aloud (Section 8)	19	Verbally summarize a block of code.	Verbal	Method Id Quadrant Id	Quality Assessment Thinking Time	No difference in subject performace was found (6)

The main factor (independent variable) is Style with two treatments: camel case and underscore. The column N refers to the number of participants. Each unique result is given a number. Contradictory results are marked with the prime symbol

with greater than six to nine characters require multiple fixations since camel-case identifiers lead to long "words" given the lack of spaces. From these experiments, it can be concluded that identifier Style does have an impact on the readability of identifiers (i.e., Research Question 1)

These two experiments were followed by four more studies attempting to understand if Style impacts comprehension with the assumption that Style impacts readability. That is, Style appears to impact readability in simple tasks not in the context of reading programs. The next three studies examined a variety of more sophisticated comprehension tasks. In the first of these studies (*Where's Waldo*), there is additional evidence that subjects were faster and more accurate in the camel case Style, again supporting a positive answer for Research Question 1. It also gives some indication that Style may have an impact on more complex comprehension tasks. Additionally, training seems to improve accuracy but does little for speed, which is not surprising because training gives subjects more experience with tasks such as reading and scanning code.

The *SAT* study represents a more complex comprehension task. The results concur with the *Where's Waldo* experiment that identifier Style does impact comprehension with respect to time but not accuracy. However, the previous result does provide support in answering positively Research Question 2, as a significant impact on comprehension is occurring. Thus, the impact of Style is more than just mechanical or syntactical.

Interestingly, the result is the opposite of the first two studies where camel casing was favored in accuracy. The results for the SAT study are much more in line with the expectation from similar studies in cognitive psychology (New et al. 2006; Epelboim et al. 1997) on natural language comprehension. Thus, while in a natural-language context underscores provide better readability, in a software context, camel casing seems to provide better readability. The difference between the two tasks is the type of comprehension required (reading source code versus prose). In particular, subjects can take advantage of distinct visual features for doing a recognition task that doesn't help a subject who is actually trying to comprehend the text. In *Where's Waldo* other skills, such as programming background, can enhance effectiveness. In the *SAT* study, the subject must comprehend the passage to be successful at the task. Another difference between the two contexts is the repetition of multi-part words. In the SAT study, subjects saw such words essentially once, while in the first two studies (*Cloud* and *Where's Waldo*), identifiers are repeated frequently. In short, the *SAT* study is in agreement with results for natural language reading but those studies with source code reading contradict natural language reading.

The next two studies consider problems that require a more complex comprehension task (as in the *SAT* study) but in a software context (as in *Where's Waldo* and *Cloud*) and attempt to resolve some of the conflicting results. In the *Tracker-code* experiment, it was found that while there is little difference between the styles for expert programmers, there is a significant improvement in recall correctness for beginner programmers when using camel case. In particular, the visual effort for short identifier names (e.g., rowSum) appears to be greater when using underscores. One possible explanation is that programmers chunk such short phrases into one concept because they are common concepts in the problem or solution domain (e.g., rowSum, xAxis). Thus, the use of underscore gets in the way of understanding these identifiers; an issue that deserves further investigation. Combined, these two results

seem to favor camel-cased identifiers. This study also provides further evidence that reading source code is fundamentally different than reading natural language.

The studies also find some indication that subjects may not retain the Style in which they saw/read the identifiers. This lack of retention could partially explain the difference between reading source code and reading natural language where the complexity of source code overshadows the comprehension impact of Style. Programmers construct a mental model (Brooks 1983; Soloway and Ehrlich 1984) of the code and the details of the syntactic features are less important. That is, the result of comprehension is a program plan with the specifics of the syntactic features filtered out rather than a rote memorization of the exact program text. The first four studies also appear to show that expert programmers exhibit little difference in accuracy between the two styles and that, through training, any difference could most likely be mitigated. This result further supports the impression that the mental model constructed by developers washes out the details of the program text to some degree.

The last study, (*Read-aloud*) was unable to show support for the Performance and Efficiency hypotheses. Given that the experiment is unable to show any effect of Style in the comprehension of source code, it is interesting to consider likely reasons. Four possible explanations are considered. First, there actually is no effect of Style. It could be that at the comprehension-level compounding factors swamp the effects of Style that were observed when subjects performed simpler tasks. The second explanation is that the two methods chosen were not suitable to reveal the difference even though a difference exists. When compared to the *SAT experiment* where the requirements for a suitable paragraph have been refined over several decades, no analogous requirements are known for choosing methods that will reveal differences in comprehension. The third explanation is that the method of ascertaining comprehension was not discerning enough. As in the selection of paragraphs, the particular types of questions asked to measure comprehension in the *SAT experiment* have been refined over several decades. The final explanation is that there were too few subjects. It could be that the differences would be significant if more subjects were evaluated. The study of these possibilities is left to future work. One thing is clear: the software engineering community must be cautious about making assumptions based on natural language comprehension studies.

12 Conclusions and Future Work

This paper presents a family of studies investigating the impact of identifier Style on source code comprehension. A variety of data collection methods such as on-line questionnaires, verbal, and eye-tracking methods are used. Besides the prior work (Binkley et al. 2009a; Sharif and Maletic 2010a), no other studies of the impact of Style on code readability and higher comprehension tasks exist in the literature.

The experimental results are by no means in total agreement. But a number of the studies seem to be pointing towards some general conclusions. Understanding how humans comprehend something as abstract as source code is not a simple problem. The results provide a better understanding of the fundamentals of reading and comprehending source code and thus provide foundation for future research.

Clearly, reading natural language and source code appear to be quite different. Further experiments are necessary to determine the exact differences; however, it

is postulated that the more formal structure and syntax of source code allows programmers to assimilate and comprehend parts of the code quite rapidly independent of Style. In particular, as seen in previous work (Wiedenbeck 1991), beacons and program plans play a large role in comprehension. This type of mental model is less prevalent in understanding natural language prose.

The issues of training and expertise also play a role in Style's impact on reading and comprehension. Expert programmers appear to not be impacted greatly by Style. This would imply that refactoring legacy code to reflect newer identifier styles is most likely unwarranted in the context of readability.

For future work, experiments need to be conducted on a larger more diverse population sample and focus on specific, more realistic, software tasks such as debugging within an IDE setting. These experiments will help determine if Style impacts higher-level comprehension within a realistic setting familiar to developers.

In the end, this work does beg the question: To camelCase or under_score? The accumulated evidence, leads to the conclusion that camel case is the better choice, especially for beginning programmers.

Acknowledgements Special thanks to all the participants as this work would not be possible without your time. Our thanks to David Robbins for assisting in the use of the Tobii eye tracker and Matt Hearn for helping in the preparation and administration of the studies. Finally, thanks to our three reviewers for their thorough and well considered reviews.

References

Anquetil N, Lethbridge T (1998) Extracting concepts from file names; a new file clustering criterion. In: Proceedings of the 20th international conference on software engineering

Bednarik R, Tukiainen M (2006) An eye-tracking methodology for characterizing program comprehension processes. In: Proceedings of symposium on eye tracking research & applications (ETRA), California, USA

Bednarik R, Tukiainen M (2008) Temporal eye-tracking data: evolution of debugging strategies with multiple representations. In: Proceedings of symposium on eye tracking research & applications (ETRA), Savannah, Georgia

Beymer D, Russell D (2005) Webgazeanalyzer: a system for capturing and analyzing web reading behavior using eye gaze. In: Proceedings of CHI '05 extended abstracts on human factors in computing systems, Portland, OR

Binkley D, Davis M, Lawrie D, Morrell C (2009a) To camelcase or under_score. In: 17th IEEE international conference on program comprehension, British Columbia, Canada

Binkley D, Lawrie D, Maex S, Morrell C (2009b) Identifier length and limited programmer memory. Sci Comput Program 74:149–158

Binkley D, Davis M, Lawrie D, Maletic JI, Morrell C, Sharif B (2011) Extended models on the impact of identifier style on effort and comprehension. Technical Report LOY110720, Loyola University in Maryland

Bouma H (1970) Interaction effects in parafoveal letter recognition. Nature 226:177–178

Brooks R (1983) Towards a theory of the comprehension of computer programs. Int J Man-Mach Stud 18:543–554

Butler S, Wermelinger M, Yijun Y, Sharp H (2010) Exploring the influence of identifier names on code quality: an empirical study. In: Proceedings of 14th European conference on software maintenance and reengineering, Madrid, Spain

Caprile B, Tonella P (2000) Restructuring program identifier names. In: IEEE international conference on software maintenance

Crosby M, Stelovsky J (1990) How do we read algorithms? A case study. IEEE Comput 23(1):24–35

Cutrell E, Guan Z (2007) What are you looking for? An eye-tracking study of information usage in web search. In: Proceedings of CHI, San Jose, California

de Kock E, van Biljon J, Pretorius M (2009) Usability evaluation methods: mind the gaps. In: Proceedings of annual research conference of the South African institute of computer scientists and information technologists Vanderbijlpark, Emfuleni, South Africa
Deißenböck F, Pizka M (2005) Concise and consistent naming. In: Proceedings of the 13th international workshop on program comprehension (IWPC 2005), St. Louis, MO, USA
Duchowski A (2007) Eye tracking methodology: theory and practice, 2nd edn. Springer, London
Epelboim J, Booth J, Ashkenazy R, Steinmans ATR (1997) Fillers and spaces in text: the importance of word recognition during reading. Vis Res 37(20):465–472
Goldberg JH, Stimson MJ, Lewenstein M, Scott N, Wichansky AM (2002) Eye tracking in web search tasks: design implications. In: Proceedings of 2002 symposium on eye tracking research & applications (ETRA), New Orleans, Louisiana
Grant S, Cordy JR (2010) Estimating the optimal number of latent concepts in source code analysis. In: 10th IEEE working conference on source code analysis and manipulation (SCAM), Timisoara, Romania
Guéhéneuc Y-G (2006) Taupe: towards understanding program comprehension. In: Proceedings of 16th IBM centers for advanced studies on collaborative research, Canada
Høst E, Østvold B (2008) The programmer's lexicon, volume i: the verbs. In: International working conference on source code analysis and manipulation, Beijing, China
Jeanmart S, Guéhèneuc Y-G, Sahraoui H, Habra N (2009) Impact of the visitor pattern on program comprehension and maintenance. In: Proceedings of 3rd international symposium on empirical software engineering and measurement, Lake Buena Vista, Florida
Just M, Carpenter P (1980) A theory of reading: from eye fixations to comprehension. Psychol Rev 87:329–354
Lawrie D, Morrell C, Feild H, Binkley D (2006) What's in a name? A study of identifiers. In: 14th international conference on program comprehension
Lawrie D, Morrell C, Feild H, Binkley D (2007) Effective identifier names for comprehension and memory. Innovations in Systems and Software Engineering 3(4):303–318
Liblit B, Begel A, Sweetser E (2006) Cognitive perspectives on the role of naming in computer programs. In: 8th annual psychology of programming workshop, Brighton, UK
MacGinitie W, MacGinitie R, Maria K, Dreyer LG, Hughes KE (2000) Gates–MacGinitie reading tests, 4th edn (GRMT-4). Riverside, Itasca, IL
Matsuda Y, Uwano H, Ohira M, Matsumoto K-i (2009) An Analysis of eye movements during browsing multiple search results pages. Springer, Berlin
Molenberghs G, Verbeke G (2006) Models for discrete longitudinal data. Springer, Berlin
Morrell C, Pearson J, Brant L (1997) Linear transformations of linear mixed effects models. Am Stat 51:338–343
Nakamichi N, Shima K, Sakai M, Matsumoto K-i (2006) Detecting low usability web pages using quantitative data of users' behavior. In: Proceedings of 28th international conference on software engineering, Shanghai, China
New B, Ferrand L, Pallier C, Brysbaert M (2006) Reexamining the word length effect in visual word recognition: new evidence from the English Lexicon Project. Psychon Bull Rev 13(1):45–52
Ohba M, Gondow K (2005) Toward mining "concept keywords" from identifiers in large software projects. In: Proceedings of the proceedings of the second international workshop on mining software repositories, St Louis, MO
Porras GC, Guéhéneuc Y-G (2010) An empirical study on the efficiency of different design pattern representations in uml class diagrams. Empirical Software Engineering 15:493–522
Rayner K, Fischer M, Pollatsek A (1998) Unspaced text interferes with both word identification and eye movement control. Vis Res 38(8):1129–1144
Sami P, Roman B, Tatiana G, Vesa T, Markku T (2008) A method to study visual attention aspects of collaboration: eye-tracking pair programmers simultaneously. In: Proceedings of symposium on eye tracking research & applications, Georgia, USA
Sharif B, Maletic J (2010a) An eye tracking study on camelcase and under_score identifier styles. In: 18th IEEE international conference on program comprehension, Braga, Portugal
Sharif B, Maletic J (2010b) An eye tracking study on the effects of layout in understanding the role of design patterns. In: 26th IEEE international conference on software maintenance, Timisoara, Romania
Simonyi C (1999) Hungarian notation. msdn.microsoft.com/en-us/library/aa260976(VS.60).aspx
Sjøberg, D, Hannay, J, Hansen, O, Kampenes, V, Karahasanovic, A, Liborg, N, and Rekdal, A (1993). A survey of controlled experiments in software engineering. IEEE Trans Softw Eng 19(4):733–753

Soloway E, Ehrlich K (1984) Empirical studies of programming knowledge. IEEE Trans Softw Eng 10:595–609
Takang A, Grubb P, Macredie R (1996) The effects of comments and identifier names on program comprehensibility: an experiential study. J Program Lang 4(3):143–167
Uwano H, Nakamura M, Monden A, Matsumoto K (2006) Analyzing individual performance of source code review using reviewers' eye movement. In: Proceedings of 2006 symposium on eye tracking research & applications (ETRA), San Diego, California
Uwano H, Monden A, Matsumoto K (2008) Dresrem 2: an analysis system for multi-document software review using reviewers' eye movements. In: Proceedings of 3rd international conference on software engineering advances (ICSEA), Sliema, Malta
Verbeke G, Molenberghs G (2001) Linear mixed models for longitudinal data, 2nd edn. Springer, New York
Wiedenbeck S (1991) The initial stage of program comprehension. Int J Man-Mach Stud 35:517–540
Yusuf S, Kagdi H, Maletic JI (2007) Assessing the comprehension of uml class diagrams via eye tracking. In: Proceedings of 15th IEEE intl. conf. on program comprehension, Banff Canada

Dave Binkley is a Professor of Computer Science at Loyola University Maryland where he has worked since earning his doctorate from the University of Wisconsin in 1991. From 1993 to 2000, Dr. Binkley was a visiting faculty researcher at the National Institute of Standards and Technology (NIST), where his work included participating in the Unravel program slicer project. While on leave from Loyola in 2000, he worked with Grammatech Inc. on the System Dependence Graph (SDG) based slicer CodeSurfer and in 2008 he joined the researchers at the Crest Centre of Kings' College London to work, among other things, on dependence cluster analysis. Dr. Binkley's resent NSF funded research continues to focus on improving semanticsbase software engineering tools. This work has recently broadened from exclusively considering programming-language semantics to also consider natural-language semantics through the use of Information Retrieval techniques applied to source code and its supporting documents. His recent work also includes a seven school collaborative project aimed at increasing the representation of women and minorities in computer science.

Marcia Davis is an associate research scientist at the Center for Social Organization of Schools and has been with the center since 2006. She holds a doctorate in educational psychology from the University of Maryland and a master's degree in educational statistics. Her research interests include reading comprehension measurement, reading motivation, and school engagement.

Dawn Lawrie is an associate professor at Loyola University Maryland. She received her A.B. from Dartmouth College with High Honors in Computer Science and an M.S. and Ph.D. from the University of Massachusetts, Amherst. Her research interests include information retrieval, natural language processing, and applying techniques from these areas to software engineering problems.

Jonathan I. Maletic is a Professor in the Department of Computer Science at Kent State University in Ohio, U.S.A. His research interests are centered on software evolution and he has authored over 100 refereed publications in the areas of analysis, transformation, comprehension, traceability, and visualization of software. He received the PhD and MS in Computer Science from Wayne State University and the BS in Computer Science from The University of Michigan–Flint.

Christopher Morrell is a professor in the Mathematics and Statistics Department at Loyola University Maryland. He obtained his Ph.D. in statistics from University of Wisconsin-Madison in 1986. Chris also works at the National Institute on Aging in the Laboratory of Cardiovascular Sciences. His research involves statistical models that are used to describe repeated observations from longitudinal studies of aging. He is a Fellow of the American Statistical Association.

Bonita Sharif is an Assistant Professor in the Department of Computer Science and Information Systems at Youngstown State University in Ohio, U.S.A. Her research interests are in empirical software engineering, software visualization to support maintenance of large systems, and eye tracking research related to software engineering. She received her Ph.D. in 2010 and MS in 2003 in Computer Science from Kent State University, U.S.A and BS in Computer Science from Cyprus College, Nicosia Cyprus. Prior to this position, she was an adjunct Assistant Professor at Ohio University.