Influence of Identifier Length and Semantics on the Comprehensibility of Source Code

Johannes Hofmeister
University of Heidelberg

Janet Siegmund
University of Passau

Daniel V. Holt
University of Heidelberg

Abstract: Identifiers are essential for the understanding of source code. Programmers can name them arbitrarily, which is a major source for hard to understand code. We investigated how an identifier's length and semantics affect program comprehensibility. In a controlled experiment, we showed that identifier names using proper words lead to a faster defect detection than identifier names using abbreviated words or single letters.

1 Introduction

Software maintenance is one of the major cost factors in software development [5], and programmers spend much of their time comprehending source code [3]. Program comprehension is affected by identifier names, as they can help developers choose an efficient comprehension strategy [2].

Psychological research suggests that identifier names can affect program comprehension, but it remains unclear whether short or long identifier names are beneficial. Some findings seem to favor short identifier names, for example lists of short items fit better into memory than lists of long items [1], and short words are more common in natural language than long words [6].

Other findings support the opposite. Longer words have richer semantics and that allow to regroup concepts as chunks of information, thus relieving working-memory resources [1]. Single letters are easier to spot when embedded in real words [4]. Therefore, program comprehension should benefit from real word identifer names although they might be longer.

Thus, it is unclear whether short or long identifier names are beneficial for program comprehension.

In this paper, we investigate the question how different identifier naming styles affect program comprehension. To research this matter, we evaluated how fast participants detected semantic defects in source code. While syntactic errors are usually caught during compile time, semantic defects, such as faulty calculations or runtime-errors, are caused by syntactically correct code, which does not perform its function according to its specification.

Participants have to use mental execution to find such defects, and if identifier style has an effect on response time, participants should perform differently with different identifier styles.

2 Method

To address our research question, we conducted a web-based experimental study. Via online platforms, such as Twitter and Xing.de, we recruited 72 professional C# developers (Age: 35.3 ± 6.8 years, experience: 14.0 ± 5.8 years, C# experience: 7.8 ± 3.6 years); all were native German speakers.

We designed six snippets of C# code. Each consisted of a commented, self-contained, 15-line long function, and was available in three versions, with their identifier names altered to either whole words (e.g., `customer`), abbreviations of three characters (e.g., `cus`), or single letters (e.g., `a`). Every snippet had a derivate version with a semantic defect, as shown in Listing 1.

```
 1: List<char> a(string b) {
 2:     List<char> c = new List<char> { '{', '}' };
 3:     List<char> d = new List<char>();
 4:     for (int e = 0; e < b.Length; e++) {
 5:         char f = b[e];
 6:         if (c.Contains(f)) {
 7:             c.Add(f);
 8:         }
 9:     }
10:     return d;
11: }
```

Listing 1: Example snippet with single letter identifier names. Identifiers where named alphabetically, but the standard .NET API left intact (e.g. List, String, etc.).

The listings shows a snippet with single letter identifier names. Here, we omitted comments and empty lines to keep the example short. The snippet contains a semantic defect in line 7. The function is supposed to identify all curly-braces within a piece of code, but matching characters are added to the wrong list. The line should read `d.Add(f);`.

We used a repeated-measures design, in which each participant saw all types of snippets (letter, abbreviation, and word identifier names). We instructed participants to find the semantic defects in three different snippets and measured the time they required to succeed. To control for practice or fatigue effects, the order of the snippets was randomized for each participant.

3 Results

We evaluated the differences in response times using planned contrasts, and tested them for significance us-

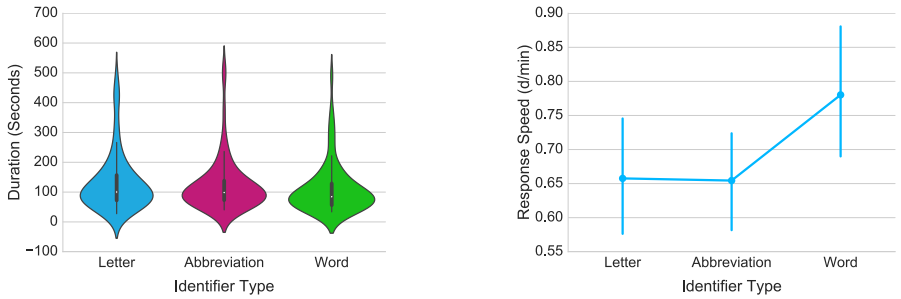

Figure 1: ***Left****: Reaction time, grouped by identifier types. The plots disregard the inter-individual differences which we controlled for in our statistical tests.* ***Right****: Response speeds grouped by identifier type. Participants detected defects 20% faster when code contained identifier names using words instead of abbreviations or single letters. The vertical bars show 95% confidence intervals.*

ing Student's t-test. Our data are described in Table 1, separated for each identifier type. The distributions were skewed to the right, which reduces the power of our significance tests due to slow outliers. To counteract, we transformed our data using an inverse transformation, yielding 'Defect per Minute' rather than 'Minutes per Defect'. The raw data are plotted on the left in Figure 1.

Table 1: Response durations and speeds during the semantic task, split by identifier type. 'Duration' is shown as median (MD) and interquartile range (IQR). Defects per minute are shown as mean (M) and standard deviation (SD).

	Duration		Defects per Minute	
Type	MD	IQR	M	SD
Word	1:25.48	1:12.78	0.78	0.42
Abbreviation	1:38.57	1:05.37	0.65	0.31
Letter	1:40.36	1:24.87	0.66	0.39

We found a statistically significant difference between words and non-words (abbreviations and single letters combined in contrast variable ψ_a, $t_{\psi a}(71) = 2.73; p = .004$). We found no significant difference in the speeds for abbreviations and single letters ($t_{\psi b}(71) = 0.07$). Participants detected semantic defects in the code the fastest when it used normal word identifier names. When code used abbreviations and single letter identifiers, participants were equally fast. However, when participants interacted with code using normal words, they were on average 20% faster ($d_z = 0.32$, Cohen, 1988). This effect is shown in Figure 1.

4 Discussion

Since there was no difference between single letters and abbreviations, we conclude that length alone is not enough to explain how identifier names affect program comprehension, and semantics appear to play an important role. Although the normal-word identifier names are longer than abbreviations, they lead to a faster understanding of the code and detection of the defect. We assumed that short names would relieve working memory; however, the presented comprehension tasks appeared to benefit more from the semantic properties of the normal-word identifier names.

We found that code using normal words can be understood more quickly and eases defect detection. This finding suggests that proper identifer naming should be considered during the initial creation of source code, in order to save maintenance costs later on in a software's life cycle. Although longer names increase typing effort in comparison with abbreviations or single letters, auto completion and the benefit from semantics during program comprehension render this difference negligible.

Acknowledgements

This work has been supported by the DFG grant SI 2045/2-1.

References

[1] A. D. Baddeley, N. Thomson, and M. Buchanan. Word length and the structure of short-term memory. *Journal of Verbal Learning and Verbal Behavior*, 14:575 – 589, 1975.

[2] R. Brooks. Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, 18:543 – 554, 1983.

[3] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining Mental Models: A Study of Developer Work Habits. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 492–501, New York, NY, USA, 2006. ACM.

[4] G. M. Reicher. Perceptual recognition as a function of meaningfulness of stimulus material. *Journal of Experimental Psychology*, 81:275–280, Aug. 1969.

[5] A. von Mayrhauser and A. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, 1995.

[6] B. S. Weekes. Differential Effects of Number of Letters on Word and Nonword Naming Latency. *The Quarterly Journal of Experimental Psychology Section A*, 50:439–456, May 1997.