# On Planning an Evaluation of the Impact of Identifier Names on the Readability and Maintainability of Programs

Mircea Lungu, Jan Kurš
*{lungu,kurs}@iam.unibe.ch*
Software Composition Group
University of Bern
Switzerland

***Abstract*—One of the long running debates between programmers is whether `camelCaseIdentifiers` are better than `underscore_identifiers`. This is ultimately a matter of programming language culture and personal taste, and to our best knowledge none of the camps has won the argument yet. It is our intuition that a solution exists which is superior to both the previous ones from the point of view of usability: the solution we name sentence case identifiers allows phrases as nams for program entities such as classes or methods. In this paper we propose a study in which to evaluate the impact of sentence case identifiers in practice.**

***Index Terms*—user evaluation, programming languages, empirical software engineering.**

## I. Introduction

There are two ways of conveying domain information in a program: through the names of the program entities and through comments in the source code. Since the comments tend to become out of sync, it follows that the naming of entities is a critical factor when it comes to the readability and future maintainability of a piece of software.

In most of the currently popular programming languages the method names and the class names are the ones conveying this semantic information [3] and different programming languages impose different constraints on these names in order to make the parsing of these programs easier. One almost universal such constraint is that names of programming entities must exclude spaces.

Let us consider Smalltalk, a classical OO programming language with a simple syntax and powerful semantics. Traditionally, in the language, a unary message send (calling a method without parameters on an object) has the following syntax (1):

```
anObject aMessage.
```

where both the object that is the receiver of the message and the message are identifiers, that is sequences of alphanumeric characters excluding spaces. For complex identifiers programmers use the camelcase convention. In fact, Smalltalk programmers are notorious for using long identifier names and code snippets like the following are not uncommon (2):

```
HierarchicalGraph
    importFromDownlodedStackOverflowData.
```

The length of the identifier makes it awkward to read[1].

We say that by allowing spaces in the identifier names we enable *sentence case* or *phrase case* – a way of writing identifiers that look like natural language phrases. We hypothesize that this would increase the readability and maintainability of source code.

To allow sentence case in Smalltalk, we slightly modify the grammar of the language by introducing a separator character in all the places in the grammar where two identifiers could be adjacent. With such a syntax change we can allow spaces in the name of the identifiers, or what we call sentence case identifiers.

Let us modify the syntax for the message send (1) and introduce a required comma between the object and the message that it receives (1'):

```
anObject, aMessage.
```

This will allow the example (2) to become:

```
Hierarchical Graph,
    import from downloaded StackOverflow data.
```

We find this second example to be more elegant. It has the added benefit of the editor being able to check the spelling of individual words.

## II. Research Questions and User Evaluation

Not only that we believe that example (2) is more elegant, but we believe that a language which would be using sentence case would would have a series of benefits: it would encourage programmers to think more about their identifiers (H1), will look friendlier to newcomers (H2), and will ultimately lead to an increased readability and maintainability of programs (H3). These are our intuitions, but they would need to be verified.

[1]And sometimes results in a useless comment being added to the method such as *"create example from the downloaded StackOverflow data"*.

However, the approach is not without drawbacks: the extra "ink" required for adding commas everywhere between possible colliding identifiers might be tiring (H4) and it will be received with suspicion by old *Smalltalkers* (H5). Also, the sentence case identifiers idea might not scale to other languages with other grammars.

We are working on implementing the necessary language modifications which would allow us to run user evaluations and learn whether our intuitions are correct. Particularly we plan to answer the hypotheses (H1-H5).

Since most of these hypotheses can be validated through user evaluations, we plan to have a working prototype running by the time the USER workshop will be conducted and to be able to discuss experiment designs with the workshop participants and, if the context is favorable, involve the other participants in small experiments.

If evidence supports our intuition, this knowledge would be useful and guidance for the designers of the future languages and IDEs. Indeed, the solution could be designed either at the programming language level, or at the IDE level. One could also imagine a compatibility layer on top of existing code that would unCamelcase-ise it and then camelcase it back.

## III. Related Work

A very good introduction and discussion on naming in programming is delivered by Liblit et al. [3] who observe that the public identifiers have longer and more informative names, and that the grammar of natural languages influences the choice of identifier names.

Binkley et al. conducted a study in which they evaluated the impact of camel case and underscore conventions on source code readability. They found differences between experts and beginners [1].

Sharif and Maletic performed an empirical study to compare the same two identifier styles using eye tracking software. They found that subjects recognize identifiers with underscore style more easily [4].

Applescript is a popular scripting language which has a certain natural language feel and philosophy [2]. As far as we are aware there are no user studies published about its design.

## IV. Conclusion

We have proposed an empirical evaluation of our proposed modification to a classical programming language. This modification consists in enabling *phrases* instead of *identifiers*. We believe this would make a language easier to understand for beginners, and would encourage the experienced developers to higher quality code. However, since these are only assumptions, we plan to design a series of user evaluations to learn whether our hypotheses will hold. As far as we are aware, this is quite a novel approach to language design but we believe a powerful one.

## Appendix: A Slightly Larger Example

In order to illustrate our idea with a longer example, we must introduce a few other details of the Smalltalk syntax. When a message sent to an object takes multiple parameters, the parameters are interleaved between the keywords of the message as in (3)

```
anObject fooWith: aString andBarWith: aNumber
```

One added benefit of our approach over traditional Smalltalk syntax, is the situation in which multiple messages are chained. The snippet (4):

```
anObject foo bar
```

sends foo to anObject and to the resulting object it sends bar. When the messages take arguments we need to surround all of them with parentheses, so there is no ambiguity whether we have a single message with three parameters or three chained messages with one parameter, as in the example (5):

```
(((anObject foo:1) bar: 'two') baz: #three) qux
```

In our implementation the previous can simply be written as:

```
anObject, foo: 1, bar: 'two', baz: #three, qux.
```

With this rule, consider now the following Smalltalk code:

```
| shortPaper |
shortPaper ←ShortPaper createNewNamed: 'Planning an Evaluation'

shortPaper authors
    addWithFirstName: 'Mircea' andLastName: 'Lungu';
    addWithFirstName: 'Jan' andLastName: 'Kurs';

shortPaper
    addAffiliation: 'University of Bern';
    addAbstract: (FileInputStream createFromFile: 'abstract.txt')
        getStreamContents.

(((Net getProtocol: #HTTP) createRequest: #Post) for: shortPaper)
    send.
```

The same example written with sentence case identifiers would look like this:

```
| short paper |
short paper←Short Paper, create new named: 'Planning an Evaluation'

short paper, authors,
    add with first name: 'Mircea' and last name: 'Lungu';
    add with first name: 'Jan' and last name: 'Kurs'.

short paper,
    add affiliation: 'University of Bern';
    add abstract: (File Input Stream, create from file: 'abtract.txt') get
        stream contents.

Net, get protocol: #HTTP, create request: #Post, for: short paper, send.
```

## REFERENCES


[1] D. Binkley, M. Davis, D. Lawrie, and C. Morrell. To camelcase or underscore. In *Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on*, pages 158 –167, may 2009.

[2] W. R. Cook. Applescript. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, HOPL III, pages 1–1–1–21, New York, NY, USA, 2007. ACM.

[3] B. Liblit, A. Begel, and E. Sweetser. Cognitive perspectives on the role of naming in computer programs. In *Proceedings of the 18th Annual Psychology of Programming Workshop*. Psychology of Programming Interest Group, September 2006.

[4] B. Sharif and J. I. Maletic. An eye tracking study on camelcase and under_score identifier styles. *International Conference on Program Comprehension*, 0:196–205, 2010.