# Lightweight Programming Experiments without Programmers and Programs: An Example Study on the Effect of Similarity and Number of Object Identifiers on the Readability of Source Code using Natural Texts

Tim Marter Paul Babucke Philipp Lembken Stefan Hanenberg

Department for Computer Science and Business Information Systems
University of Duisburg–Essen, Germany
tim.marter | paul.babucke | philipp.lembken@stud.uni-due.de, stefan.hanenberg@uni-due.de

## Abstract

It is often said that controlled experiments should check the effect of programming languages or styles on programming. But it is also often said that running controlled experiments is a very time consuming and error prone task – that's why a lot of researchers do not run such experiments. Both arguments are plausible, but there is potentially an alternative: lightweight experiments where the effort to run such experiments is low and which (still) fulfill the requirements of controlled experiments with two exceptions: First, these experiments do not use programmers as subjects, and second, the experiments do not contain programming tasks. Instead, such experiments try to find analogies from other domains where the topic to be studied is (still) close enough to the original target domain but where it is easier to find participants and experimental setups. This paper illustrates such a lightweight experiment by introducing a study on the effect of number of identifiers and similarity of identifiers on (code) readability – without using source code and without programmers as subjects. The result of the experiment is comparable to other experimental results which gives a first indicator that it is possible to run such lightweight experiments that approximate the results of full-blown experiments. This paper argues that such lightweight experiments could be useful in the process of experimentation – they cannot and should not supersede full-blown experiments, but they can help in early stages of experimentation.



*Onward!'16*, November 2–4, 2016, Amsterdam, Netherlands
ACM. 978-1-4503-4076-2/16/11...$15.00
http://dx.doi.org/10.1145/2986012.2986020

***Categories and Subject Descriptors*** H.1.2;D.1 [*User/Machine Systems*]: Programming Techniques

***General Terms*** Experimentation, Human Factors

***Keywords*** Experimentation, Programming, Randomized Controlled Trials, Human Factors

## 1. Introduction

Much has been said about possible factors that influence the comprehension of software. In the tradition of software metrics (see [7] for an introduction into that field) things such as size or structure metrics are analyzed in regard to their (possible) influence on the readability. Other approaches refer more to the human-computer-interaction aspect of source code (the book by Shneiderman [21] gives a broader introduction into that field) by applying human-centered controlled experiments (see [15, 26]).

The human-centered approach seems plausible to a number of people, but a number of authors criticize that the number of controlled experiments in the area of programming or programming languages is very low (see for example [12, 20, 22, 24, 25]). For example, Kaijanaho has shown that the number of studies that analyze programming language features using human-centered methods is quite low: if reduced to the approach of randomized controlled trials, Kaijanaho just found 22 studies in the field of programming languages between 1970 and 2012 [16].

There might be reasons why relatively few studies are executed per year: it is often said that the design and execution of controlled trials take a lot of time and is relatively expensive (see for example the often mentioned arguments against experimentation mentioned by Tichy [24]). And this seems to be an intrinsic problem of programming experiments: Non-trivial programming tasks require still a large amount of time and require programmers to participate. Such programmers need to be recruited and then they need to spend a lot of time in an experiment – which causes

not negligible costs. And such costs become even more annoying when an experiment fails. As a consequence, most researchers rather refuse to run experiments and spend more effort in the design and implementation of artifacts (instead of the design and implementation of experiments that evaluate these artifacts).

We think both arguments should be taken into account. First, we agree that randomized controlled trials are required in order to check whether new programming technologies do have an effect. But we also think that it is desirable to reduce the costs of experimentation: if the software science community runs controlled experiments in the same quantity as in the last decades (again, Kaijanaho just found 22 randomized trials in more than 30 years), there is relatively little hope that software science becomes a mature discipline (from the empirical perspective). Hence, the general question is how the number of controlled trials can be increased in a way that the effort for running such trials is much less than what's currently practiced.

We think that it is (to a certain extent) possible to drastically reduce the effort for designing and running such experiments if the problem domain can be mapped to some other, similar domain that is not directly related to programming and where it is not necessary to use programmers as participants in the experiment. In case the problem domain is much smaller (in terms of time required to solve a given experimental task) this reveals new opportunities do run a larger number of small-scaled experiments – experiments that can be executed before running a full-blown experiment.

### 1.1 Motivation: Experimentation with Non-Programmers

Often, there is the claim that programming experiments are too expensive. For example, Tichy mentions (and criticizes) the often heard phrase that "doing an experiment would be incredibly expensive" [24]. By observing an experiment series that one of the authors is involved in (see [5, 6, 8, 10, 11, 13, 19]) we share this impression: designing and running such programming experiments takes a lot of time.

When doing programming experiments, experimenters have to write some code (that requires exhaustive testing), to provide the experiment's participants with some programming environment, to find participants who are capable to write code, to find participants who are familiar with the programming languages used in the experiment, to prepare training sessions, etc. And finally, it is rather hard to design programming experiments that require less than for example an hour (for a single programming task). In case, the experiment requires some within-subject measurements, this implies that it takes even more time to measure a single participant. And finally, in case the experiment assumes that the participants are volunteers, the experiment must not take too long (otherwise the motivation to volunteer is rather low).

Although we think that it is worth the effort to do such kinds of programming experiments, we still think that there are circumstances where it is worth to think about alternative ways – ways where the effort for running an experiment is much reduced.

We were heavily influenced by Binkley et al. [1] who ran an experiment about identifiers where it was not necessary to read or write code but where participants just had to remember a given identifier and click in a game-like environment on identifiers shown to them. This kind of experiment has a number of interesting implications:

- First, the overall time required by each participant is rather low: Compared to traditional programming experiments, it is rather a matter of minutes instead of hours.
- Second, such kind of experiment does not require subjects to be familiar with a given programming language or environment, it is not even necessary for participants to be programmers.[1]
- Third, exhaustive training sessions for subjects are hardly required.

Hence, we conclude from the Binkley study that it is probably easy for such a study to find participants (because not much time and no special skills are required from the participants) and to collect the data (because a single participant can be measured within minutes – i.e., a larger number of subjects can be measured in hours instead of weeks, what is in our experience more likely the case in programming experiments). Of course, there is no guarantee that such lightweight experiments reveal effects comparable to experiments executed in a programming environment. But these experiments give first indicators that it is worth to run such a study in a larger context with programmers.

### 1.2 Example: Study on Code Readability in Terms of Number of Identifiers and Similarities

For the application of lightweight experiments, we had a concrete research question: we were interested in whether the similarity and the number of identifiers plays a role for the readability of source code.

Despite the fact that programmers and programming language designers have the tendency to care less about syntax – Leavens et al. mention that "it is often said that syntax does not matter" [18] – there are authors that analyzed the usability and readability of programs from the syntax perspective; from the syntax of programming languages up to the concrete syntax in programs.

From the programming language perspective, Stefik and Siebert analyzed to what extent different keywords provided by a programming language (such as keywords for loops, conditions, or operators) have an effect on the usability of the programming language [22]. The authors were able to

[1] Although it should be mentioned that Binkley et al. were using programmers with different experience levels in the experiment in order to study whether the experience level itself is an influencing factor.

detect differences in usability of different keywords with the same or at least similar semantics.

From the program perspective, authors such as Lawrie et al. [17], Binkley et al. [1], or Hofmeister [14] asked themselves to what extent the concrete syntax of a program influences the readability. More precisely, they asked whether different kinds of identifiers have an impact on the readability. The results were that full name identifiers make it easier for developers to read and understand the code (in comparison to abbreviations or single letter identifiers) and that camel case identifiers are easier to detect than identifiers written in an underscore style.

Using such studies as a starting point, a number of additional questions arise. Probably the most obvious one is to ask whether the number of identifiers in a program do influence the readability of the source code: It seems plausible that the more identifiers are in the code, the harder it is to read the code. At least, refactorings such as remove parameter [9] (for an unused parameter) assume that additional parameters cause additional effort for developers.

However, by thinking in more detail about the possible relationship between identifiers and readability another question arises: the question, whether the similarity of identifiers do actually influence the readability of code.

```
public float maxDistance(Point pointA, Point pointB,
                         Point pointC, Point pointD) {
  float distance = Math.sqrt(
    (pointD.x - pointA.x)*(pointD.x - pointA.x) +
    (pointD.y - pointA.y)*(pointD.y - pointA.y));
  distance = Math.max(distance, Math.sqrt(
    (pointB.x - pointA.x)*(pointB.x - pointA.x) +
    (pointB.y - pointA.y)*(pointB.y - pointA.y)));
  ...
  return distance;
}
```

**Figure 1.** Source code with similar identifiers

Figure 1 illustrates code that computes the maximum distance from the first passed point to any other point. In the code snippet the passed parameters have similar names: they are just different in the last letter. We think the code is hard to read because there are so many similarities in the code: a wrongly chosen identifier somewhere in the code is (probably) hard to identify.

```
public float maxDistance(Point origin, Point first,
                         Point second, Point third) {
  float distance = Math.sqrt(
    (third.x - origin.x)*(third.x - origin.x) +
    (third.y - origin.y)*(third.y - origin.y));
  distance = Math.max(distance, Math.sqrt(
    (first.x - origin.x)*(first.x - origin.x) +
    (first.y - origin.y)*(first.y - origin.y)));
  ...
  return distance;
}
```

**Figure 2.** Source code with different identifiers

Figure 2 illustrates code with the same semantics but with different identifiers. We think that the second example is easier to read, because the identifier names are easier to distinguish. For example, in case somewhere in the code the identifier `third` is used where the identifier `second` is expected, we assume that this is easier to identify just because the names of both identifiers are so different.

These examples motivated us to study whether the number of identifiers as well as their similarities influence the readability of code. An obvious approach would be to use the code snippets directly in a controlled experiment. Maybe the mentioned debugging task (one wrong identifier) could be a good task for the experiments. But maybe the effect size of the (possible) difference is too small so that only a very large set of participants in the experiment would reveal this difference. Maybe the code needs to be more complex in order to show such a difference. But complexity of code is not easy to determine upfront. Again, the code must not be too complex, otherwise the code's complexity would be the dominant factor that potentially hides all other factors.

At that point, we usually design and run a number of small pilot studies where we give programmers (in our case typically students) a number of tasks. But even for such small pilot studies the effort for designing and running experiments is still relatively high. In case we find indicators that the factor we are studying has an influence, we check what tasks could be appropriate for a real study. In case the pilots do not show such indicators, we typically do not run such a study. Again, this process is not trivial and still very time-consuming. Although the goal of pilot studies is to be small our experience is that even for pilot studies the costs for designing tasks and collecting data is comparable to full-blown experiment. At least, it is not an order of magnitude smaller.

Again, the time for preparing and executing a pilot study is quite high. But our suspicion is that the similarity of identifiers is something that is not specific to code, but rather a general issue in any kind of text. We think that when we read a text with similar words, it is harder to identify differences between the words. Hence, we think that it is possible to find an analogy in text reading. And hence we decided to apply a lightweight study instead of starting with a number of pilot studies with source code and programmers.

### 1.3 Contribution and Structure of this Paper

Our general goal is to describe an approach to run lightweight experiments instead of traditional pilot studies. We do that by describing the design and execution of an experiment. Hence, we think that the paper has two different contributions:

- This paper describes an experiment that compares the effect of number of object identifiers (in the concrete experiment, these are nouns in a natural text) and their similarities on the readability of a natural text (which is for us a representative for source code). We did that by giving 32 subjects six different texts to read – each equal in length –

and asked questions about it. The result of this study was that the number of object identifiers are an influencing factor – no matter whether we asked for characteristics (adjectives or verbs) of a given thing (the identifier) or whether we asked for things that fulfill given characteristics: the more object identifiers are in the text, the more time it took the subjects to respond. With respect to similarity of such identifiers it mattered only if we asked for adjectives or verbs for a given noun where a large number of similar nouns appeared in the text. In such a case the more similar these nouns are the more time it took our participants to respond. Hence, the first contribution of the paper is to answer a research question related to programming – but in a non-programming environment.

- The second contribution of this paper, which is actually the main focus here, is that we describe an experiment that is from our perspective easy to design and to execute – and which does neither require a time consuming procedure in the data selection, nor does it depend on developers participating in the experiment. We answer a given research question (by showing that the number of object identifiers as well as their similarities do matter from the readability perspective) with relatively low expenses in the context of natural texts – but with the risk that by transforming the research question into a different domain, we lose too much context. I.e. the risk is, that although we showed an effect in natural texts, such effect would not show up if we run such an experiment in a programming context.

By making the design of this experiment explicit, we hope that it motivates other people to think about to what extent it is possible to design a relatively cheap experiment – people that currently rather refuse to run controlled trials.

The paper is structured in two parts. The first part of this paper is the ordinary part of performing a controlled trial. We describe for a given research question first the related work in section 2, then, we describe the experiment by introducing first the research questions and then the experimental design in section 3. Then, section 4 describes the analysis of the measured data and the results. After summarizing the experiment and interpreting the results in section 5, we interpret the results from the source code perspective in section 6. Then, we discuss the experiment's threats to validity.

In the second part, we discuss in section 8 to what extent the whole approach – running a lightweight experiment in order to answer a domain-specific, programming related research question – might be applicable to software science in general. Finally, we conclude this work in section 9.

## 2. Related Work on Identifiers

We discuss here workd from three different perspectives: studies on (a) object identifiers, (b) programming language syntax, and (c) on readability (in regard to eye movement).

### 2.1 Identifiers

**Lawrie et al. (2006):** The study by Lawrie et al. [17] analyzed the effect of parameter names on the understanding of code. More precisely, Lawrie et al. gave more than 100 subjects the source code of functions with the length between 8 to 36 lines of code. The experimenters constructed three versions of the functions: functions with parameter names with full names, with parameter names consisting of abbreviations, and with single letter parameter names. The full names were the ones developers originally have chosen, the abbreviations and the single letter names were derived from them. The participants were given a number of different functions (with different parameter names) one after the other. After some reading time, they were asked to write down what the code actually does. The analysis of the resulting data revealed a significant difference between the different kinds of identifiers with respect to the correctness of the answers. The means for the full names were (for almost all functions) always higher than the means for the abbreviations and means for the single letter names.

**Hofmeister (2015):** Hofmeister performed a similar study on 72 professional developers [14]: Subjects needed to find semantic defects and syntax errors in C# code snippets. Each subject received such snippets with full name identifiers, abbreviations, and single letters. The experiment was performed over the web where the subjects were not able to execute the code in order to get feedback. The code snippets given to the subjects were single methods, each consisting of 15 lines of code and each contained between 91 and 94 tokens. Time for fixing errors was measured. Additionally, the number of found errors were measured. The results were comparable to the results by Lawrie et al.: Full name identifiers permitted developers to find and fix more errors per time unit in comparison to abbreviations and single letter names. The difference between full names and non-full names and reveals a significant benefit for full names. Additionally, Hofmeister analyzed the results for the abbreviations and the single letter names and did not find a significant difference. Finally, Hofmeister checked, whether the different treatments had an effect on the different kinds of errors to be fixed. Syntactic errors were faster to find than semantic errors and for syntactic errors no impact of the treatments was measured.

**Binkley et al. (2009):** Binkley et al. [1] studied the possible effect of camel case compared to underscore style for identifiers on 135 subjects. The general idea of the study was to test, to what extent subjects are able to detect equal identifiers by first showing such identifiers to subjects and then let them choose among a given set of identifiers which one was initially shown. Binkley et al. distinguished between identifiers which were common English phrases and identifiers selected from an initially constructed code base. Additionally, the authors varied the length of the words in the identifiers (two word and three word identifiers). After an identi-

fier was shown to the subjects, a new screen appeared where four identifiers were shown to the subjects. One of the shown identifier was the same as the originally shown identifier, three were different. The experiment was conducted over the internet. The result of the study was that camel case increased the correctness of the subjects' responses, but in the general case the response time with camel case was longer as well. However, the study revealed an interesting interaction effect; a longer training influenced the results: 'Subjects with more training were slower on identifiers written in the underscore style than subjects without training'. Additionally, Binkley found that longer identifier names took longer to read than shorter names. Another finding in the study was that a longer reading time decreased the correctness.

**Buse and Weimer (2010):** In a larger study performed on 120 participants, Buse and Weimar extracted 100 code snippets from five open-source projects [2] and gave these snippets the participants who rated them with respect to readability (scoring 1-5). Then, Buse and Weimar extracted from the snippets 19 different factors (such number of identifiers, line length, etc.) and generated a model from these factors. Then, they compared the resulting model with the participants' responses by running a regression analysis, showing that there is a large correlation between the model and the responses. Finally, they compared the model with other readability metrics and showed that there is a high correlation between such metrics and the proposed model.

### 2.2 Programming Language Syntax

In addition to the previously mentioned studies, there are studies that refer to programming language syntax. Stefik and Siebert performed three studies, reported in one paper [22].

The first study was a survey about preferred words that appear in a programming language syntax. The participants were grouped according to their expertise into the groups 'programmers' and 'non-programmers' and both groups were asked to rate a number of words that are often used in common programming languages. An interesting observation was, that there seems to be some large disagreement between non-programmers and programmers.

In the second study, different syntactical versions of programming language constructs in nine different languages (C++, Java, Smalltalk, PHP, Perl, Ruby, Go, Python, and Quorum) where shown to the subjects who rated for each construct how intuitive it is from their perspective. The different constructs were grouped into different categories such as loops, conditions, functions, etc. It turned out that developers considered even language constructs from major languages (such as C++, Go, or Perl) as problematic.

In a final study, subjects were asked to write some small piece of code based on a given sheet that introduced the programming language via an example. Six different programming languages were tested (Quorum, Perl, Randomo, Java, Ruby, Python). Then, the authors measured the probability that single tokens were used in the right way. The experiment revealed that only the programming languages Quorum, Python and Ruby were significantly different from the randomly created language Randomo – for Java and Perl this statement did not hold.

### 2.3 Readability and Eye-Tracking

Finally, there are other studies that refer to the readability of code - especially those ones that make use of eye-tracking systems.

The first study to be mentioned is the one by Crosby and Stelovsky [4]. They gave subjects different kinds of texts (natural texts as well as algorithms) and analyzed the different reading strategies. The algorithms provided to the subjects consisted of loops, conditions and side-effects. The analysis was performed based on data received from an eye-tracking system. The study showed that there are differences in reading algorithms and reading natural texts in almost all cases. However, with respect to the code, large variations were found among the subjects. Participants in the low experience group spent more attention to comments than participants from the high-experience group, but in both groups people were found that did or did not spend most of the attention on comments.

The study by Busjahn et al. [3] measured in more detail the effect of different groups of people on code reading. In the experiment, novices were asked to read three programs (from a few lines of code up to to an entire screen). Two of them were in pseudo-code, one a complete Java class. Experienced developers were given six programs: two were the same as given to the novices. The participants were asked different kinds of questions from giving a summary of the code up to answering some multiple choice questions. The main result of the study is that both – novices and programmers – showed a different reading strategy when reading text. While novices showed a high linear eye movements, this was not the case for experienced developers: it turned out that experienced developers adapt their reading strategy when switching from a natural text to source code.

## 3. Experiment Description

### 3.1 Research Questions , Initial Considerations

The experiment addresses two research questions:

- RQ1: Does the number of object identifiers in a piece of code influence the readability of the code?
- RQ2: Does the similarity of object identifiers in a piece of code influence the readability of the code?

As described above our goal was to run this study not on source code but on natural texts. In order to do so, we needed to translate object names found in code to something we find in natural texts. One criterion that we already decided in the beginning was that the texts we gave the subjects should be

in their mother tongue, in our case this means that we wanted to give subjects German texts.

**Object identifiers:** In our understanding, *nouns* that describe actual objects in natural texts are representatives for object names in source code. Next, we needed to define what the texts should be about – and what kind of questions should be asked about the text in order to find indicators for the readability of the text.

**Equality of text:** First, we decided to give subjects texts with equal length. Although the phrase "equal length" is easy to articulate, we found it rather hard to say what exactly this should mean. Possible candidates could be equal with respect to the number of characters in the text, equal with respect to the number of words or equal with respect to the number of sentences. While we thought that each of them are possible criteria, we found it extraordinary hard to write a natural text that is actually equal with respect to all of these criteria, i.e. texts that still do represent valid sentences and that are able to express "something meaningful". We have chosen here a much more relaxed criteria. Instead of measuring equality with respect to number of characters, etc. we just defined *number of lines* as a criteria: each text given to the participants should be equal with respect to its length measured in lines. For measuring the lines, we used a google docs document and used in that way google's underlying algorithm for setting words into lines as a line measurement.

**Number of lines:** Next, we needed to decide how many lines we wanted to give to the subjects. It became relatively clear to us (by running very small initial studies) that the less lines we give, the harder it is for us to measure anything at all, because the effort for searching and finding fragments in the text would be too low. After some trials, we decided to give text with five lines to the participants (plus/minus half a line).

**Criteria for object identifiers:** Next we needed to define some criteria for the object identifiers we used in the text. First, we defined that each noun should have between 8-12 characters. Additionally, participants should be able to articulate the nouns, i.e. technical terms that might be complicated to read should not be part of the text. Note that we do not restrict here ourselves to non-artificial nouns. We just say that a noun should be a word that should be easy to read and to pronounce.

**Similarity of object identifiers:** Since similarity is a factor to be studied in the experiment, we needed to agree on some similarity measurement. We decided to classify nouns in the following way. First, we considered nouns to be very similar if they differ in less than three letters, partially different if they differ in half of their letters, and different if they have at most three letters in common.

**Number of object identifiers:** Since we are interested in studying the effect of number of object identifiers, we had to decide on the different numbers that we would like to test. We decided to design texts with three object identifiers and texts with five object identifiers. Taking into account that the text length was fixed, it turned out to be hard to write meaningful texts with more identifiers within five lines.

**Style of the text:** Additionally, we needed to agree on how the style of the text should look like. We decided to use texts where a number of nouns are mentioned, each noun representing some object, person or location, and where some verbs and adjectives were associated to each noun. And finally, the text should express something meaningful, i.e. the mentioned nouns should be somehow connected in the text in a meaningful way.

**Appearances of object identifiers:** Next, we decided that each object identifier should appear exactly two times in the text, i.e. no object identifier should play a larger role in the text than another one.

> *FirstObject* wants to repair with *secondObject* the beautiful mansion *thirdObject*, owned by the gentle and small man *fourthObject in the* village *fifthObject*. The old *secondObject* lives as well in the small *fifthObject* and *firstObject* is a neighbor of *fourthObject* and they like each other a lot. The old mansion *thirdObject is* landmarked.

**Figure 3.** Example text (translated for illustration purposes into English, concrete object names were removed)

Figure 3 illustrates one text we used in the experiment: for illustration purposes we translated the text into English (original in German) and we removed the object identifiers from the text, because in the given example the identifiers were similar, but we did not find appropriate translations that reflect on this similarity[2]. Another difference between the German text and the translation here is, that it is possible in German to compose nouns to larger words (for example, an English phrase such as "*word length*" is in the German language just one single word). As a consequence, building larger and similar word are no major problem in German. It should be made explicit that each word chosen in the text (including of course the object identifiers) are valid German words.[3]

**Questions to the subjects:** We were interested in the readability and the comprehension of the text. Hence, we needed to design corresponding questions to be answered by the participants. After some trials, we decided to ask for each text the participants two questions (one after the other). The first question asked for an object identifier that fulfills some criteria mentioned in the text. The second question included an object identifier and asked for some criteria associated to it. For example, the first question for the text illustrated in

[2] Additionally, due to the translation and the replacement of the object identifiers, the illustrated text is slightly longer than the original one and some word appear at different places (due to differences in the grammar of the German and English language).

[3] Another difference between English and German that is worth mentioning here is, that nouns in the German language always start with a capital letter. I.e. it might be the case that it is easier to identify nouns in the German language because of the different style in comparison to other words.

Figure 3 asked for the village name (with the correct answer *fifthObject*), the second question asked how *fourthObject* is described (with the correct answer "gentle and small").

### 3.2 Experimental Design and Procedure

We decided to measure time (which corresponds to the measurement as taken in programming experiments such as [6, 8]) until we received the right answer from the participants. The measurement started when we showed a question to the participant and stopped when we received the right answer. In case we got a wrong answer, we told the participants so. The measurement was done by a timer that was started and stopped by the persons who collected the data by hand.

During prototyping the experiment, we became aware that different subjects differ widely with respect to reading time and response time. As a consequence, we expected that in order to show any effect in the study we either need to drastically increase the number of subjects or to perform within-subject measurements. We decided to perform within-subject measurements. But since there are potentially learning effects or novelty effects, we decided to do some counter-balancing by organizing the texts given to the subjects as a repeated Latin square design: one group of subjects received the questions in order ABCD, the next group received the questions in order BCDA, etc. For the ordering of the tasks, we decided to switch between similarities, i.e. one group received first a text with similar identifiers, the next group received a text with different identifiers, etc. However, taking into account that three different kinds of similarities of texts were chosen (similar, different, partially different), six different groups would be required. Since we expected 30-35 subjects in the experiment, six groups were from our perspective too many: this would imply five participants per group – again, taking into account the large deviation between participants, we expected that we would naturally measure differences between such groups just because of the low number of participants per group. As a compromise we organized the similar and different questions in the Latin square and left the partially different texts at the end. Our goal was to use mainly the similar and different identifier texts for the analysis and using the partially different identifiers only when needed.

I.e. we have in principle a 3-factor (within-subject) experiment: the first factor is the similarity of object identifiers (with the treatments similar, different, and partially different), the second factor is the number of object identifiers (with the treatments three and five) and the third factor is the question (with the treatments question 1 and question 2). Due to the ordering of tasks into different groups we have an additional (between-subject) factor group that permits to determine whether our measurements depend on the ordering of the tasks.

The experiment procedure was as follow:

- We gave participants some warm-up task in order to get used to the experimental procedure (example text followed by a first question, followed by a second question).
- For each text, we gave participants as much time they required to read the text (we measured the corresponding time).
- After a participant told us that he finished reading, we gave him the first question to read. This was the starting point of the measurement. For answering the question, the participant had access to the original text. When the participant told us the correct answer we stopped the time. When the participant gave us an incorrect answer, we told him, that the answer was incorrect.
- We repeated the previous procedure with the second question. After question two was finished, we moved to the next text.

| Group | Ordering of texts |
|---|---|
| 1 | $S_3$, $D_5$, $S_5$, $D_3$, $PD_3$, $PD_5$ |
| 2 | $D_5$, $S_5$, $D_3$, $S_3$, $PD_3$, $PD_5$ |
| 3 | $S_5$, $D_3$, $S_3$, $D_5$, $PD_3$, $PD_5$ |
| 4 | $D_3$, $S_3$, $D_5$, $S_5$, $PD_3$, $PD_5$ |

**Figure 4.** Ordering of tasks: S=similar, D=different, PD=partially different, the index describes the number of object identifiers in the text

The texts and the questions were available as printouts (on DIN A4 pages). No additional material was available.

## 4. Results and Analysis

We have asked arbitrary subjects – people working at different companies and students from our institute – to participate in the study. Finally, 32 subjects participated in the experiment who were randomly assigned to the four groups. The age of the people varied between 18 and 58.

Table 1 shows the raw measurements in the experiment. *Group* shows the different groups (according to the Latin square arrangement), *R* is the reading time in seconds for each text, *Q1* is the response time in seconds for question 1, and *Q2* is the response time in seconds for question 2. We analyzed the data using repeated measures ANOVAs using the statistics package SPSS (version 22).

### 4.1 Results on Similar/Different Identifier Texts

First, we checked whether the reading time for the different tasks were different. We did that by performing the repeated measures ANOVA on the within-subject variable *numberOfIdentifiers* (with the two treatments *three identifiers* and *five identifiers*) and the within-subject variable *similarity* (with the two treatments similar and non-similar). Neither identifiers nor similarity were significant ($p$ = .27, $\eta_p^2$ = .038, respectively $p$ = .5, $\eta_p^2$ = .015). However, there was a significant disordinal interaction between both ($p$ = .033, see

| | | Similar 3 | | | Similar 5 | | | Different 3 | | | Different 5 | | | Indifferent 3 | | | Indifferent 5 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Subject** | **Group** | R | Q1 | Q2 | R | Q1 | Q2 | R | Q1 | Q2 | R | Q1 | Q2 | R | Q1 | Q2 | R | Q1 | Q2 |
| 1 | 1 | 32,8 | 4,2 | 6,6 | 40,3 | 6,3 | 8,8 | 50,1 | 7,6 | 7,4 | 35,0 | 6,4 | 7,2 | 39,6 | 7,4 | 6,6 | 46,1 | 8,6 | 11,2 |
| 2 | 1 | 15,3 | 10,2 | 8,6 | 13,7 | 12,3 | 14,9 | 18,2 | 11,4 | 10,7 | 16,2 | 10,8 | 11,0 | 17,5 | 8,3 | 8,9 | 15,4 | 11,1 | 14,2 |
| 3 | 1 | 35,6 | 3,0 | 4,6 | 38,4 | 4,5 | 8,0 | 40,3 | 6,3 | 5,4 | 39,8 | 5,8 | 5,7 | 44,3 | 5,5 | 7,6 | 41,0 | 11,3 | 8,3 |
| 4 | 1 | 27,0 | 4,5 | 12,2 | 24,3 | 5,7 | 11,0 | 38,2 | 8,6 | 3,6 | 28,4 | 7,4 | 5,6 | 53,0 | 5,8 | 7,8 | 58,9 | 14,0 | 8,6 |
| 5 | 1 | 28,7 | 5,1 | 4,7 | 36,8 | 11,0 | 8,6 | 40,5 | 8,1 | 7,9 | 35,1 | 7,4 | 8,4 | 36,4 | 6,9 | 4,9 | 45,1 | 8,8 | 14,0 |
| 6 | 1 | 65,1 | 3,8 | 4,8 | 59,7 | 5,2 | 4,7 | 66,6 | 5,3 | 6,7 | 55,6 | 7,0 | 8,1 | 70,2 | 3,5 | 3,5 | 56,4 | 7,0 | 8,0 |
| 7 | 1 | 21,3 | 8,7 | 7,6 | 27,8 | 8,8 | 9,4 | 31,4 | 11,5 | 9,4 | 25,8 | 6,4 | 8,6 | 24,2 | 4,4 | 6,5 | 29,0 | 7,9 | 6,5 |
| 8 | 1 | 45,3 | 10,4 | 9,8 | 50,2 | 8,2 | 8,7 | 49,1 | 11,0 | 11,1 | 43,2 | 9,1 | 15,7 | 56,6 | 3,7 | 8,7 | 51,4 | 9,4 | 7,7 |
| 9 | 2 | 16,8 | 6,9 | 7,1 | 17,4 | 12,7 | 16,7 | 15,8 | 10,7 | 9,7 | 14,3 | 7,5 | 8,2 | 16,3 | 6,7 | 7,5 | 18,0 | 8,4 | 9,9 |
| 10 | 2 | 40,4 | 8,5 | 8,6 | 45,5 | 9,7 | 13,7 | 49,7 | 10,4 | 9,1 | 36,7 | 7,8 | 11,4 | 35,5 | 17,4 | 10,1 | 39,7 | 7,7 | 9,8 |
| 11 | 2 | 35,3 | 5,6 | 6,6 | 33,1 | 4,1 | 4,1 | 37,0 | 4,9 | 6,2 | 27,4 | 4,7 | 6,3 | 19,3 | 10,2 | 4,4 | 29,2 | 6,7 | 7,0 |
| 12 | 2 | 14,7 | 3,5 | 9,5 | 17,8 | 4,7 | 4,8 | 10,6 | 3,7 | 8,9 | 21,8 | 10,9 | 16,4 | 11,3 | 3,2 | 3,5 | 13,8 | 10,7 | 4,3 |
| 13 | 2 | 22,1 | 11,0 | 9,8 | 30,3 | 6,4 | 6,6 | 22,6 | 4,9 | 15,5 | 23,3 | 7,7 | 14,1 | 26,6 | 8,5 | 5,0 | 26,5 | 10,0 | 6,8 |
| 14 | 2 | 16,5 | 2,8 | 3,0 | 18,0 | 6,0 | 8,7 | 16,0 | 1,4 | 16,8 | 17,9 | 2,2 | 18,6 | 17,7 | 6,2 | 2,0 | 23,2 | 3,9 | 9,2 |
| 15 | 2 | 19,2 | 8,5 | 8,3 | 20,2 | 2,4 | 3,8 | 16,9 | 7,6 | 14,6 | 21,1 | 4,1 | 11,3 | 19,3 | 2,9 | 7,0 | 18,3 | 8,6 | 9,5 |
| 16 | 2 | 12,7 | 5,8 | 3,8 | 15,8 | 2,6 | 6,1 | 19,8 | 4,1 | 16,0 | 23,0 | 7,2 | 11,6 | 22,2 | 5,3 | 8,8 | 24,8 | 5,7 | 4,3 |
| 17 | 3 | 20,3 | 4,1 | 3,9 | 20,9 | 4,9 | 4,0 | 18,9 | 6,0 | 17,6 | 23,1 | 5,4 | 27,6 | 18,7 | 6,6 | 5,6 | 20,6 | 11,8 | 8,6 |
| 18 | 3 | 24,4 | 2,9 | 7,2 | 21,4 | 2,5 | 5,9 | 31,5 | 4,3 | 17,6 | 33,3 | 3,1 | 4,0 | 18,9 | 2,2 | 1,6 | 23,0 | 3,4 | 7,1 |
| 19 | 3 | 12,7 | 4,1 | 11,3 | 15,4 | 3,0 | 15,7 | 15,5 | 1,8 | 16,6 | 17,5 | 11,6 | 32,8 | 16,7 | 3,0 | 9,0 | 15,6 | 2,8 | 5,7 |
| 20 | 3 | 18,8 | 2,7 | 3,3 | 25,7 | 2,2 | 3,0 | 23,6 | 2,2 | 14,5 | 33,3 | 1,8 | 14,4 | 23,2 | 2,2 | 2,6 | 29,9 | 3,9 | 10,1 |
| 21 | 3 | 20,2 | 5,6 | 8,0 | 22,8 | 6,6 | 7,6 | 19,6 | 7,6 | 60,0 | 27,0 | 4,3 | 10,7 | 21,2 | 5,0 | 4,7 | 21,3 | 8,1 | 4,5 |
| 22 | 3 | 22,8 | 4,1 | 4,9 | 33,6 | 3,0 | 11,0 | 21,9 | 4,2 | 47,2 | 48,1 | 5,1 | 11,7 | 26,5 | 2,8 | 10,2 | 27,4 | 10,7 | 23,8 |
| 23 | 3 | 32,7 | 4,1 | 13,9 | 27,0 | 4,6 | 3,9 | 30,6 | 11,3 | 19,5 | 24,8 | 8,0 | 3,7 | 22,1 | 5,0 | 3,0 | 30,7 | 3,0 | 4,4 |
| 24 | 3 | 24,0 | 3,1 | 44,0 | 19,0 | 4,3 | 8,5 | 24,3 | 2,3 | 49,8 | 23,4 | 3,7 | 6,5 | 24,3 | 2,3 | 3,1 | 19,3 | 9,7 | 3,5 |
| 25 | 4 | 23,6 | 7,4 | 4,5 | 25,6 | 11,3 | 10,8 | 19,0 | 7,4 | 10,0 | 20,0 | 6,8 | 20,6 | 22,9 | 5,1 | 7,2 | 23,2 | 3,4 | 7,4 |
| 26 | 4 | 16,5 | 7,0 | 9,2 | 21,6 | 12,2 | 9,6 | 17,6 | 9,6 | 20,9 | 19,4 | 6,7 | 10,1 | 17,7 | 5,7 | 5,6 | 19,4 | 6,6 | 6,6 |
| 27 | 4 | 22,1 | 14,4 | 9,0 | 17,8 | 8,3 | 6,2 | 16,3 | 5,5 | 13,9 | 21,7 | 3,5 | 4,3 | 34,8 | 3,0 | 4,2 | 30,3 | 19,7 | 4,5 |
| 28 | 4 | 37,4 | 11,2 | 7,6 | 30,2 | 7,1 | 4,0 | 36,5 | 9,5 | 15,5 | 27,2 | 3,4 | 22,8 | 28,7 | 5,8 | 7,2 | 29,2 | 11,7 | 11,8 |
| 29 | 4 | 27,9 | 6,8 | 4,3 | 28,4 | 5,4 | 9,3 | 26,4 | 7,0 | 30,5 | 26,8 | 4,5 | 10,6 | 21,9 | 3,6 | 4,0 | 27,1 | 4,0 | 4,5 |
| 30 | 4 | 23,2 | 7,9 | 5,5 | 35,8 | 13,1 | 10,2 | 31,7 | 7,1 | 26,0 | 32,3 | 4,4 | 6,5 | 32,9 | 3,4 | 4,3 | 29,1 | 10,3 | 16,7 |
| 31 | 4 | 30,4 | 5,9 | 10,4 | 21,1 | 10,0 | 3,8 | 21,8 | 2,6 | 35,8 | 31,7 | 8,9 | 9,7 | 29,6 | 7,3 | 5,3 | 32,0 | 17,6 | 6,2 |
| 32 | 4 | 19,5 | 3,8 | 6,3 | 17,8 | 5,5 | 7,5 | 20,4 | 6,4 | 10,7 | 17,2 | 4,8 | 4,4 | 18,4 | 2,6 | 4,7 | 20,3 | 6,7 | 19,6 |

**Table 1.** Raw Measurements of all subjects (all times in seconds): R = reading time, Q1 = time for question 1, Q2 = time for question 2

Figure 5), where the reading time for three different identifiers was higher than the reading time for the three similar identifiers (and even higher than for five different identifiers) – we will speak about this interaction later.

Next, we ran a repeated measures ANOVA with the between-subject variable *group* (with 4 treatments), the within-subject variable *question* (with the treatments *question one* and *question two*), the within-subject variable *similarity* (with the treatments *similar* and *different*) and the within-subject variable *numberOfIdentifiers* (with the two treatments *three identifiers* and *five identifiers*).

First, the factor group was non-significant ($p > .17$, $\eta_p^2 = .16$) which indicated that there were no differences between the response times for the different questions and similarities between the groups. In case this factor would have been significant it would indicate that the ordering of the tasks influenced the resulting measurements which means that a common analysis of all groups would be problematic.

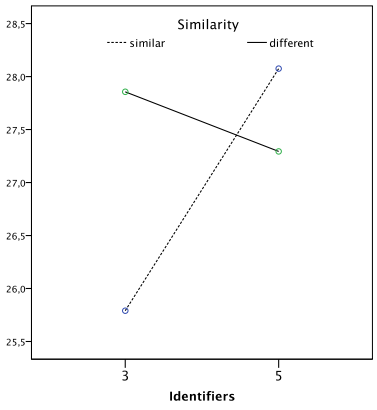


**Figure 5.** (Surprising) disordinal interaction of identifiers and similarity on reading time

Second, the factor *question* was significant ($p < .001$, $\eta_p^2 = .643$) and the subjects required less time to answer question one (M=6.4) compared to question two (M=11.4).

Third, the factor *numberOfIdentifiers* was significant ($p < .02$, $\eta_p^2 = .18$) where responses to texts with three identifiers required less time (M=8.1) than responses to texts with five identifiers (M=9.8).

Fourth, the factor *similarity* was only approaching significance (p = .072, $\eta_p^2 = .11$) with the tendency that different identifiers (M=8.1) required less time than similar identifiers (M=9.71).

Taking into account that *question* was a significant factor and taking into account that the two questions were intentionally designed in a different way, it is reasonable to analyze both questions, i.e. question one and question two, in separation. Hence, we performed a repeated measures ANOVA on the within-subject factors *similarity* and *numberOfIdentifiers.*

For question one, none of the identifiers is significant: neither *numberOfIdentifiers* ($p > .24$, $\eta_p^2 = .044$) nor *similarity* ($p > .9$, $\eta_p^2 < .001$).

For question two, the situation was fundamentally different. *NumberOfIdentifiers* was significant ($p = .023$, $\eta_p^2 = .156$) and the three identifier texts (M = 9.961) revealed a smaller response time than the texts with five identifiers (M = 12.886). *Similarity* was again approaching significance (p = .075, $\eta_p^2 = .099$). Again, we received an interaction effect between *numberOfIdentifiers* and *similarity* with a large effect size (p < .001, $\eta_p^2 = .419$): a disordinal interaction between both variables, comparable to the one described in Figure 5.

Again, the interaction effect was not plausible to us – but we received from the subjects one comment over and over again: the subjects considered the text with three different identifiers extraordinary hard to read. Not because of the identifiers or their number, but because of the written text itself, which was not plausible to them. This information is consistent with the disordinal interaction we detected in the beginning.

Because of this comment, and because the previous result that *group* was not a significant factor, we decided to repeat the analysis but this time without the texts $similar_3$ and $similar_5$, but with the texts partially $different_3$ and partially $different_5$. We are aware that the last two texts were not given to the subjects in different orders and that a possible ordering effect could influence the results. But since we were not able to detect such an effect for the first four texts, we assumed that this was not the case as well for the texts partially $different_3$ and partially $different_5$.

### 4.2 Results on Similar/Non-Similar Identifier Texts

Instead of repeating the whole previous analysis, we just perform the two separate tests for the two different questions.

For question one, *numberOfIdentifiers* was significant (p < .001, $\eta_p^2 = .393$) where three identifiers (M = 5.8) required less time than five identifiers (M = 7,6). Similarity was not significant (p > .28, $\eta_p^2 = .036$). However, we receive again a significant hybrid interaction effect between both variables (p < .026, $\eta_p^2 = .151$): with an increase of similarity the five identifiers reduced the response time, while the three identifiers increased the response time (see Figure 6). While this is not intuitive from first glance, there seems to be an easy interpretation for this phenomenon: the first question asks for an identifier to whom a certain adjective is associated, i.e. a given adjective has to be found in the text. The more identifiers are similar in the text, the easier it is to find this adjective (because it is easy to skip text while reading similar identifiers, because it is clear that none of them is the adjective to be found). Such an effect must be stronger than in situations where the number of identifiers is rather low.

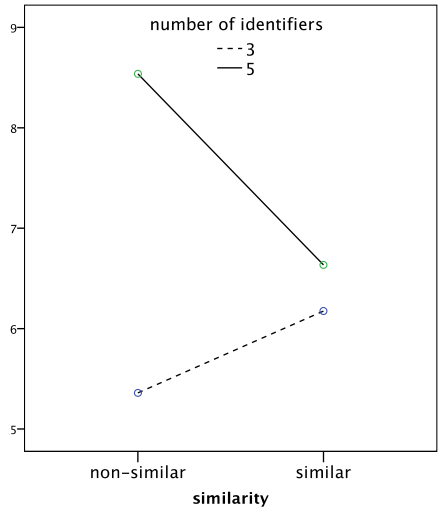


**Figure 6.** Interaction of number of identifiers and similarity for question 1

For question two, the results are different. *NumberOfIdentifiers* was significant (p < .001, $\eta_p^2 = .443$) and three identifiers (M = 7.1) required less time than five identifiers (M=13.3). Similarity was significant (p < .002, $\eta_p^2 = .263$) and the more similar those identifiers were, the more time required the responses (M=7.3 vs M=13.03). There is a significant, ordinal interaction between both (p = .008, $\eta_p^2 = .204$), i.e. an increase of identifiers and similarity increases much more the difference in response time (see Figure 7).

## 5. Summary and Interpretation

This work was motivated by the works by Lawrie et al. [17], Hofmeister [14], and Binkley et al. [1] who showed that the kind of identifiers (full name identifiers vs. abbreviations vs. single letter identifiers) as well as the style of identifiers (camel case vs. underscore) have an effect: while the first authors studies the effect on source code, Binkley studied the effect in a game-like environment (where subjects had to

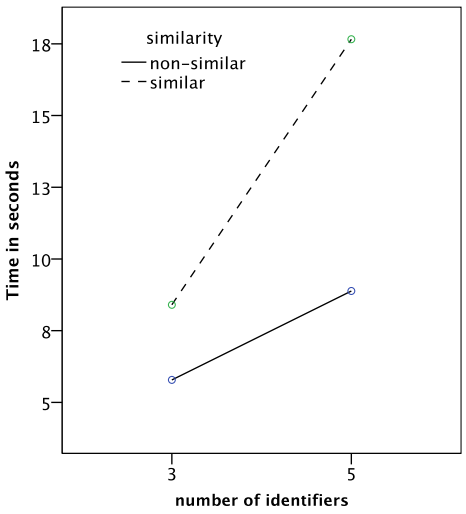


**Figure 7.** Interaction of number of identifiers and similarity for question 2

chose between different options).

One of our goals was to identify more factors that influence the readability of source code. Our original idea was to check, whether the number of passed parameters to a method influence the understandability of such a method. Additionally, we wanted to check, whether the similarity of such parameter names do influence the understandability of such a method.

However, we followed another goal in this experiment. A secondary goal was to build up an experiment that is closely related to a programming experiment - without actually doing anything directly related to programming. Here, we were heavily influenced by the Binkley experiment [1] where the participants were not working on real source code but in a game-like environment.

By combining both goals we defined tasks on natural texts, i.e. participants did not read source code but a natural text that described situations where adjectives were assigned to object identifiers. Altogether, we designed six of such texts. Three or them contained three nouns, the other three contained five nouns. Each noun appeared two times in the text where corresponding adjectives were assigned to them. Then, we measured the participants' response times.

Finally, we asked ourselves whether the experiment's results depend on the kind of questions given to the subjects. For each text, we asked two questions. The first question asked to what object identifier a criteria matches, the second question asked for the criteria for a given object identifier. We expected first that the kind of question has a influence on the responses and that the number of object identifiers and their similarity has a different effect for the two different questions.

The results of the experiment were as follows:

1. When we asked for the criteria by mentioning an object identifier (question 2), both, the number of identifiers as well as their similarity effect the response time: an increase of number of object identifiers and their similarity increased the response times. We think that this result is quite plausible, because the more similar object identifiers are, the harder it is for readers to detect differences between a word the reader currently reads and the object identifier we asked for. Additionally, if there are more identifiers in the text, the reader has to ask himself more often whether the word he currently reads corresponds to the one he has been asked for.
2. When we asked for an identifier by mentioning objects' criteria (question 1), only the number of identifiers was a significant factor, but not their similarity. Since the number of criteria are larger the more object identifiers are in the text, is is plausible that the number of identifiers (indirectly) increases the number of comparisons the reader has to do in mind. We think that the interaction effect can be explained in a similar way: the similarity does not (directly) play a role, because readers need to identify criteria in the text. However, if a larger number of object identifiers is similar, it is easier to detect other words such as the criteria the reader has to search for. If the number of object identifiers is small, this effect is small, too (and maybe hidden by other factors such as text complexity).

However, we should also keep in mind that the experiment was not as straight forward as we expected: one text that was originally intended to be used for the comparison seemed to be faulty, at least we got feedback from the participants that indicated this and which was consistent with a rather peculiar interaction measurement.

## 6. Interpretation from the Source Code Perspective

The goal of the experiment was not only to reduce the effort of designing and running an experiment by using natural texts, the goal was to learn something about the readability of source code. From the (object-oriented) source code perspective, the object identifiers in the text represent named objects (which might be local variables or parameters) while the other criteria represent fields or assigned values. The whole text does represent a code snippet that does not depend on any other variable (i.e. it does not refer to anything that is not defined within the text). The used text is just a very simplified source code: just object identifiers and their associations with other criteria are being used. From our perspective, this is comparable to source code that just consists of assignments. Other features that can be found in source code such as method calls, conditions, loops, etc. were not part of the natural text.

What we varied is the number of object identifiers and their similarity. By varying the number of nouns we tried to mimic the idea of methods with different numbers of param-

eters (or local variables). Next, we varied the similarity of nouns. By making the nouns more similar we tried to mimic situations that can be found in source code where different parameters have similar names.

Finally, we varied the kind of information that had to be extracted from the text. Instead of asking questions such as "what does the code do" or "what is the output of the code", we asked for associations between object identifiers and other described criteria. Applied to actual source code this means that we asked, to what object a certain field is assigned (question 1), respectively what fields or assignment belong to a certain object (question 2). By showing differences between both questions with respect to the measurement, we have shown that there are different effects depending on what exactly has to be known from the code.

## 7. Threats to Validity

The experiment suffers from a number of threats to validity.

**Generalizability to source code:** The most obvious one is that we used natural texts instead of source code. It is unclear whether the results can be generalized to source code – a similar situation holds for example for the Binkley study [1]. But even in case the texts should be comparable to source code, it is still a very restricted kind of code where only object identifiers and assignments appear.

**Imprecise measurement:** Next, the measurements are quite imprecise (without knowing exactly how imprecise they are): starting and stopping the timer was done by hand. Although we are aware of this general threat, we think that this approach makes it easy to run the experiment because no special equipment is required to collect the data.

**Text styles:** In order to make the texts comparable, we made explicit the kind of text style we used. We are not aware to what extent there are other influencing factors such as the ordering of the object identifiers in the text, the way how criteria of such identifiers appear in the text, etc. The same objection holds for the text length: we are not aware to what extent the chosen length of the text introduced additional confounding factors.

**Text similarity:** We built different texts for the different number of identifiers and different similarities. This also has to do with the decision to perform within-subject measurements: too similar texts would probably increase the carry-over effects when switching from one task to the other. However, it is possible that the texts differed with respect to some other confounding factors. In fact, we have found such indicators by the users' feedback and the measured interaction that made us change the analysis.

**Differences in reading strategies:** The eye tracking studies by Crosby and Stelovsky [4] and by Busjahn et al. [3] indicated different reading strategies by different groups of participants (at least for source code). Due to our restricted measurement (time measurement) we are not able to detect to what extent the results were driven by different reading strategies. However, we actually achieved differences caused by the different treatments. I.e. although we are not able to find better explanations for why there were such differences, the measurements say that there are such differences. From the perspective of source code readability this means that the experiment was able to detect influencing factors but provided no data that permit to reason about why these differences appeared.

**No additional help for reading:** Finally, we should mention that the participants did not get any tools that are intended to help them reading the text. In regard to source code we know that modern IDEs provide abilities such as highlighting opening and closing brackets or highlighting the appearances of marked identifiers in the code. We think that such tools help people reading and understanding code, but we are not able to say how large this effect actually is. Hence, it should be up to future studies to check the possible effect of such tools on the readability of source code. At least, we think that such tools could have helped our participants in the experiment: since words from the text were mentioned in the questions, we assume that even simple find-functions known from arbitrary modern text editors would have had a measurable effect.

## 8. Lightweight Experimentation: An Alternative to Pilot Studies?

While the previous sections described and discussed the experiment from the perspective of the original research question (readability of source code in general, and possible effect of the number of identifiers and their similarity), it is still necessary to discuss the very initial question in more detail: Can the here proposed approach be used as a template for other programming experiments?

From the perspective of effort for designing and running the experiment, the experiment can be considered as a success: it was relatively easy to design the experiment and to collect data from a number of subjects. For each single subject it required in the average less than 10 minutes to participate in the experiment. This means, that collecting data from 32 subjects was done in less than six hours – even if the subjects were sequentially tested. If we compare this to programming experiments such as the ones reported in [5, 6, 8, 10, 11, 13, 19], this is an extraordinary small number: in programming experiments, a single subject typically requires 2-3 hours, others even 4 hours and more (see [23]). I.e. in a sequential execution of the experiment, the data collection for the lightweight experiment required between 4% and 8% of the time required for the data collection in comparable full-blown programming experiments.[4] The design of the experiment also required relatively low effort: The construction of the whole experiment required less than one

[4] Assumed 2 hours respectively 4 hours per subject for full-blown experiment and 10 minutes for lightweight experiment.

week. This means that in about six working days the experiment was constructed and the data collected.

However, we must ask ourselves, whether the collected data is valid from the perspective that the collected data gives insight into the original domain. Of course, it can be doubted whether the achieved results have anything to do with reading and understanding source code in general. However, what we can do is to compare the results of the study with the study by Buse and Weimer. They concluded, based on an experiment with 120 programmers that *"the number of identifiers and characters per line has a strong influence on our readability metric [...]. It would appear that just as long sentences are more difficult to understand, so are long lines of code*" [2]. This means that the results of Buse and Weimar with respect to the number of identifiers is quite comparable to the here produced results – but the here produced results are retrieved with low costs. Hence, at least for the given research question there is an indicator that not arbitrary numbers have been generated, but numbers that do actually give insights into the original research question.

Of course, the executed experiment does something that is not directly comparable to typical programming experiments: the participants just had to read something and to respond to some questions. I.e., the participants were not required to construct something in the experiment (such as the construction of source code that solves a certain problem). We absolutely think that this is a potential restriction so far and we do not have a direct answer to this problem: we think that in order to build an experiment that requires people to construct a solution to a problem would require to design a completely different experiment. We are not aware so far whether it is possible at all to translate such constructive programming tasks to a different domain. However, we think that it is worth to think about possible ways. For example, the mentioned programming experiments [5, 6, 8, 10, 11, 13, 19] did not require participants to develop code that includes conditionals or loops. It might be the case that such experiments can be translated to natural texts as well, such as that references to text elements should be written by participants (the mimic method calls).

However, when designing the experiment, we also became aware that we needed to address problems that we probably do not have (at least not to the same extent) in programming. While writing the texts, there was the need to discuss different styles of texts, different words, etc. Here, we had the feeling that the new domain brought in additional problems as well – and there might be situations where such problems are larger than the ones that need to be solved by the experiment.

Finally, we should be even more critical with the whole approach. The typical argument by people running controlled experiments is often, that they doubt that the discipline of software science is based on any solid empirical foundation. If we now propose to build such a foundation by translating the original research question to a different domain, there could be a very strong argument against such an approach:

**This is pseudo-science, because it does not study the original research question, but studies something that can be considered as (the trial of) an analogy that might or might not give insights to the original question.**

In fact, this is a really serious objection and we must not be too eager to ignore it. Yes, there is the great danger that by translating the research question to another domain, we lose any relationship to the original research question. Hence, we should be more than skeptical with results gathered **purely** from experiments that are translated to a different domain. Hence, we think that such lightweight experiments should only be used as small steps in the research process: situations where it is desirable to run a (small) experiment and where the results of such an experiment give some guidance about how the next (small) step in the research process could look like. However, in the very end of the research process, there should be (still) a controlled trial that actually checks whether a given hypothesis in the domain of software science holds – and this trial should be executed in the domain of software science. I.e., we share the doubt that only the execution of lightweight experiments will (probably) not be able to give enough insight in software science – but they could play a part in the process.

More precisely, our interpretation of the present experiment is that we gathered evidence that the number of identifiers and (with weaker evidence) their similarity play a role in readability. Based on that, we should now start designing a larger experiment. The pilot for such an experiment would no longer focus on the research question in order to check whether the underlying hypothesis holds – instead, the pilot would concentrate more on whether or not the experimental procedure works out.

## 9. Conclusion

We think that the here presented experiment could be used as a starting point for lightweight experiments in software science that can be used to gather insights in our field without investing too much in the process of experimentation. More precisely, we think that the experiment can be used especially as a starting point for experiments about the readability and understandability of source code where the effort for running such experiments is much lower than in traditional programming experiments.

However, we should not forget that the way how measurements are performed in this experiment are much more restrictive than for example eye-tracking measurements as used by Busjahn [3] that permit not only to identify an effect, but to reveal possible explanation models for such an effect. However, just identifying factors that do influence the readability and understandability of code would be already a

huge help for the programming community or the software engineering community in general.

Additionally, we think that the proposed approach of performing lightweight experiments also contains a number of risks – namely that the results gathered from such experiments cannot be translated back to the original research question. Hence, while we would like to encourage other researchers to think about such lightweight experiments, we also would like to warn that it is not the intention of this paper to propose such experiments as an alternative to traditional programming experiments. They should only be used as "additional studies" (used in early stages in experimentation in order to detect possible factors) – the overall evaluation of hypotheses should still be done within the studied domain. We think that such larger experiments can be executed after a longer chain of rather cheap experiments. We think that in that way it is possible to gain more benefit out of experimentation in comparison to for example the experiment series such as [5, 6, 8, 10, 11, 13, 19] where we think that the effort for running such experiments is rather high.

We think that especially in the area of program syntax or programming language syntax, there is a larger opportunity to run such lightweight experiments and that in the area of syntax there is a large gap that still needs to be filled. While there are studies available about things such as identifier style (full names, abbreviations, camel case, underscore style, etc.) there is still a huge gap in knowledge with respect to what exactly identifiers should express (what is a good name for an identifier?) or identifier styles for different elements (variables vs. parameters vs. method names), etc. Here, the execution of lightweight experiments such as the one described in this paper could help – because it seems plausible that such questions are closely related to the question how people read text (and not so much, how people read source code).

In general, this paper's contribution is to give a different perspective on experimentation than what's currently found in the literature. We hope that in that way we show researchers opportunities to run experiments, who are currently rather reluctant to run any controlled trial.

## References


[1] David Binkley, Marcia Davis, Dawn Lawrie, and Christopher Morrell. To camelcase or under_score. In *The 17th IEEE International Conference on Program Comprehension, ICPC 2009, Vancouver, British Columbia, Canada, May 17-19, 2009*, pages 158–167, 2009.

[2] Raymond P. L. Buse and Westley Weimer. Learning a metric for code readability. *IEEE Trans. Software Eng.*, 36(4):546–558, 2010.

[3] Teresa Busjahn, Roman Bednarik, Andrew Begel, Martha E. Crosby, James H. Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. Eye movements in code reading: relaxing the linear order. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, ICPC 2015, Florence/Firenze, Italy, May 16-24, 2015*, pages 255–265, 2015.

[4] Martha E. Crosby and Jan Stelovsky. How do we read algorithms? a case study. *Computer*, 23(1):24–35, January 1990.

[5] Stefan Endrikat and Stefan Hanenberg. Is aspect-oriented programming a rewarding investment into future code changes? A socio-technical study on development and maintenance time. In *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, ICPC '11, pages 51–60, Kingston, CA, 2011. IEEE Computer Society.

[6] Stefan Endrikat, Stefan Hanenberg, Romain Robbes, and Andreas Stefik. How do api documentation and static typing affect api usability? In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 632–642, New York, NY, USA, 2014. ACM.

[7] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., Boston, MA, USA, 1998.

[8] Lars Fischer and Stefan Hanenberg. An empirical investigation of the effects of type systems and code completion on api usability using typescript and javascript in ms visual studio. In *Proceedings of the 11th Symposium on Dynamic Languages*, DLS 2015, pages 154–167, New York, NY, USA, 2015. ACM.

[9] M. Fowler and K. Beck. *Refactoring: Improving the Design of Existing Code*. Object Technology Series. Addison-Wesley, 1999.

[10] Stefan Hanenberg. Doubts about the positive impact of static type systems on programming tasks in single developer projects - an empirical study. In *ECOOP 2010 - Object-Oriented Programming, 24th European Conference, Maribor, Slovenia, June 21-25, 2010. Proceedings*, LNCS 6183, pages 300–303. Springer, 2010.

[11] Stefan Hanenberg. An experiment about static and dynamic type systems: doubts about the positive impact of static type systems on development time. In *Proceedings of the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2010, October 17-21, 2010, Reno/Tahoe, Nevada, USA*, pages 22–35, 2010.

[12] Stefan Hanenberg. Faith, hope, and love: An essay on software science's neglect of human factors. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '10, pages 933–946, New York, NY, USA, 2010. ACM.

[13] Stefan Hanenberg, Sebastian Kleinschmager, and Manuel Josupeit-Walter. Does aspect-oriented programming increase the development speed for crosscutting code? an empirical study. In *Proceedings of Empirical Software Engineering and Measurement (ESEM) 2009*, pages 156–167, 2009.

[14] Johannes C. N. Hofmeister. *Influence of identifier length and semantics on the comprehensibility of source code*. Department of Psychology, University of Heidelberg, Germany, 2015.

[15] N. Juristo and A.M. Moreno. *Basics of Software Engineering Experimentation*. Springer, 2001.

[16] Antti-Juhani Kaijanaho. *Evidence-based programming language design : a philosophical and methodological exploration*. University of Jyväskylä, Finnland, 2015.

[17] Dawn Lawrie, Christopher Morrell, Henry Feild, and David Binkley. What's in a name? a study of identifiers. In *Proceedings of the 14th IEEE International Conference on Program Comprehension*, ICPC '06, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.

[18] Gary T. Leavens, Yoonsik Cheon, Curtis Clifton, Clyde Ruby, and David R. Cok. How the design of JML accommodates both runtime assertion checking and formal verification. *Sci. Comput. Program.*, 55(1-3):185–208, 2005.

[19] Pujan Petersen, Stefan Hanenberg, and Romain Robbes. An empirical comparison of static and dynamic type systems on api usage in the presence of an ide: Java vs. groovy with eclipse. In *Proceedings of the 22Nd International Conference on Program Comprehension*, ICPC 2014, pages 212–222, New York, NY, USA, 2014. ACM.

[20] B. A. Sheil. The psychological study of programming. *ACM Comput. Surv.*, 13(1):101–120, 1981.

[21] Ben Shneiderman. *Software Psychology: Human Factors in Computer and Information Systems*. Winthrop Publishers, August 1980.

[22] Andreas Stefik and Susanna Siebert. An empirical investigation into programming language syntax. *Trans. Comput. Educ.*, 13(4):19:1–19:40, November 2013.

[23] Andreas Stuchlik and Stefan Hanenberg. Static vs. dynamic type systems: An empirical study about the relationship between type casts and development time. In *Proceedings of the 7th symposium on Dynamic languages*, DLS '11, pages 97–106, Portland, Oregon, USA, 2011. ACM.

[24] Walter F. Tichy. Should computer scientists experiment more? *IEEE Computer*, 31:32–40, 1998.

[25] Walter F. Tichy, Paul Lukowicz, Lutz Prechelt, and Ernst A. Heinz. Experimental evaluation in computer science: A quantitative study. *Journal of Systems and Software*, 28(1):9–18, 1995.

[26] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, and Björn Regnell. *Experimentation in Software Engineering*. Springer, 2012.