Indentation in Source Code: A Randomized Control Trial on the Readability of Control Flows in Java Code with Large Effects

Johannes Morzeck[1], Stefan Hanenberg[2][a], Ole Werger[2][b] and Volker Gruhn[2][c]
[1]*Independent Researcher, Germany*
[2]*University of Duisburg–Essen, Essen, Germany*

Keywords: Programming, Indentation, Empirical Study, User Study.

Abstract: Indentation is a well-known principle for writing code. It is taught to progammers and applied in software projects. The typical argument for indentation is that it makes code more readable. However, taking a look into the literature reveals that the scientific foundatation for indentation is rather weak. The present work introduces a four factor experiment with focus on indentation in control flows. In the experiment, 20 participants (10 students and 10 professional developers) were asked to determine the results of given Java code consisting of if-statements and printouts. Measured was the time required to answer the question correctly. The experiment reveals that indentation has a strong ($p < .001$) and large ($\eta_p^2 = .832$) positive effect on the readability in terms of answering time. On average participants required 179% more time on non-indented code to answer the question (where the different treatment combinations varied on average between 142% and 269%). Additionally, participants were asked about their subjective impressions on the tasks using the standardized NASA TLX questionnaire (using the categories mental demand, performance, effort, and frustration). It turned out that participants subjectively perceived non–indented code with respect to all categories more negative ($p < .001$, $.4 < \eta_p^2 < .79$).

1 INTRODUCTION

Indentation is a known and often applied technique to format source code. Taking a look into tutorials for popular programming languages such as Java[1] or C++[2] shows that source code is mostly indented. And taking a look into modern IDEs such as IntelliJ, Visual Studio, etc. shows that indentation is massively considered there as well, either in terms of auto-formatting tools or simply in terms of user interfaces that indent code whenever it seems appropriate.

Considering this, one easily forgets that common indentation guidelines are not naturally given in languages. For example, common indentation guidelines for languages such as SQL are not as widely spread as for the previously mentioned programming languages: just taking a look into SQL tools such as PGAdmin[3] shows that no automatic support for indentation is given. The same statements hold for other languages such as regular expressions, etc. as well – languages that are today massively used (and often integrated in modern programming languages).

Given the fact that indentation is widely spread at least for popular programming languages, it is reasonable to ask whether there is a need for another empirical study. This question gets more obvious taking into account that indentation is quite an old technique – the first study by Weissman on that topic can be found in 1974 (Weissman, 1974). But it turns out that the scientific literature does not provide much evidence for positive indentation effects: the authors of the present paper are aware of only nine publications that explicitly focus on indentation. While for one publication it is unclear whether or not an effect was found, three publications found a positive effect of indentation – five did not find an effect. And one publication that did not find an effect is a replication of a study that found an effect.

[a] https://orcid.org/0000-0001-5936-2143
[b] https://orcid.org/0009-0007-3226-1271
[c] https://orcid.org/0000-0003-3841-2548
[1]https://docs.oracle.com/javase/tutorial/
[2]https://www.w3schools.com/CPP/default.asp
[3]https://www.pgadmin.org/

Morzeck, J., Hanenberg, S., Werger, O. and Gruhn, V.
Indentation in Source Code: A Randomized Control Trial on the Readability of Control Flows in Java Code with Large Effects.
DOI: 10.5220/0012087500003538
In *Proceedings of the 18th International Conference on Software Technologies (ICSOFT 2023)*, pages 117-128
ISBN: 978-989-758-665-1; ISSN: 2184-2833

Hence, one might say that so far only two publications showed a (positive) effect.

But for empirical studies it is not only a question of whether the effect exists. The question is also, how large it is. And it turns out that only one of the two publications report an effect size.[4] This remaining study by Albayrak and Davenport from 2010 (Albayrak and Davenport, 2010) measured the number of functional defects found by developers depending on whether or not indentation was present. While the difference in means was 25%, i.e., indentation increased the number of found defects by 25%, the effect was small (η^2=.047). However, the study does not give any details about the given code, the defects that were to be found, or the defects that were more often found in the presence of indentation. From that perspective, one can argue that despite its popularity, indentation has been hardly studied so far (9 publications known to the authors), or at least that there is not much evidence for a positive effect of indentation (three publications, for one of them a replication did not show comparable results), but even among those studies that found an effect, the reported effect is very small (just on single study with η^2=.047).

Programmers might argue that this does not matter: Indentation should be simply used because the costs of indentation seem negligible, no study has shown any negative effect of indentation, and tools for indentation are available. Still, we consider such argument as problematic. First, we think it is necessary in computer science education not only to convince students to use some technology because it is available, but to give students evidence that the application of a certain technology helps. Second, we need to take into account that for tool builders even the development of a rather simple tool such as a code formatter is an investment – and for all investments there should be the desire to determine upfront whether such an investment has a positive effect. Third, we also think that developers should not just rely on tools because they exist, but because they do help. I.e., the application of a technology should not be based upon a common belief in such technology, but should be based on documented evidence.

The present paper introduces a randomized control trial that studies indentation from the perspective of readability of control flows in programs. In the four-factor experiment (with the factors indentation, skipping, brackets, and background) Java code snippets were given participants to read. Each snippet consisted just of if- and print statements. The participants were asked what the output of the program was and the time required to give the correct answer was measured. The code snippets were chosen with respect to whether parts of the code could be skipped and whether or not brackets were used to mark the beginning and ending of the if-statements.

It turned out that indentation has a strong ($p < .001$) and large ($\eta_p^2 = .832$) positive effect on the readability in terms of answering time. On average participants required 179% more time on non-indented code to answer the questions in comparison to indented code (where the different treatment combinations varied on average between 142% and 269%). Additionally, participants were asked about their subjective impressions using the standardized NASA TLX questionnaire (Hart and Staveland, 1988). In the used categories mental demand, performance, effort, and frustration participants considered non-indented code more negative ($p < .001$, $.4 < \eta_p^2 < .79$). The participants' background (students or professional developers) had no effect on the dependent variable.

Altogether, we see the contribution of the present experiment in two ways. First, it answers a question about the effect of indentation (with the answer that the effect exists and is large in control flow with multiple conditionals). This is from our perspective urgently missing in the literature, taking the fundamental characteristics of indentation in programming into account. Second, the experiment shows that the effort for studing programming related technologies does neither necessarily require complex experimental designs nor huge effort. Running such studies (and getting answers) would help the field of software science to develop a stable background.

Finally, we think the study could encourage researchers to address even simple questions or simple technologies in empirical studies. Such studies should not be executed for the sake of running studies, but for the sake of getting documented evidence about effects of such technologies.

2 ON INDENTATION

The word indentation decribes an approach to organize source code in a way that some space in the beginning of a line of code is used in order to emphasize some code fragments (such as bodies of loops, bodies of conditions, etc.).

Figure 1 illustrates a simple Java program that

[4]Evidence standards such as CONSORT (Moher et al., 2010) consider the reporting of effect sizes as a minimum requirement for quantitative studies.

```java
// With indentation
class ArgumentPrinter {
  public static void main(String[] args) {
    if(argsl.length()==0) {
      print("No args");
    } else {
      for(int i=0; i<args.length(); i++) {
        print("Arg: " + args[i];
      }
    }
  }
}

// Without indentation
class ArgumentPrinter {
public static void main(String[] args) {
if(argsl.length()==0) {
print("No args");
} else {
for(int i=0; i<args.length(); i++) {
print("Arg: " + args[i];
}
}
}
}
```

Figure 1: Simple Java code example with and without indentation. Details such as `System.out.println` are abbreviated by `print`.

prints out the arguments passed to the program or `"No args"` in case no argument is passed. The indented version highlights that `main` is part of the class `ArgumentPrinter`: all lines that have two additional white spaces in the beginning represent a field or a method in the class. Furthermore, it emphasizes what code belongs to a method: all lines that have four or more white spaces in the beginning belong to a certain method. And within methods, additional space illustrates that some code belongs to another syntactical element. For example, the indentation of the line following the if-statement makes clear that the line is part of the body and the indentation of the line following the for-statement makes explicit that it belongs to the loop's body. It seems that non-indented code makes such relationships harder to detect.

Apart from the question how many whitespaces or tabs should be used for indentation, different programming styles differ with respect to what code should be indented. For example, in Java there are styles where opening brackets appear in a new line after an if-statement (without additional indentation). There are others, where the opening brackets appear at the end of a line. And finally there are programming languages where indentation is connected to the language's semantics. Examples for such languages are Python or Haskell (in contrast to, for example, Java or C++).

3 RELATED WORK

While there are countless resources on different code styles, empirical studies on them in general, respectively studies on indentation in particular are quite rare. Actually, identifying related work on indentation is not as trivial as it seems. A number of experiments were executed in the 70s and 80s and and it is not easy to get details about them. Additionally, there are indicators for experiments in the literature where it is unclear to the present authors whether or not they can be considered. For example, Vessey and Weber gave an overview of 12 experiments in programming in 1984 (Vessey and Weber, 1984) and mentioned two experiments by Sime et al. (Sime et al., 1977; Sime et al., 1973) that have according to Vessey and Weber indentation as a controlled variable. However, a closer look at these experiments reveals that it remains at least unclear to what extent indentation was indeed controlled.[5] Consequently, we do not consider these works in the following.

Table 1: Experiments on indentation (some publications contain more than one experiment). Column **Effect** describes whether or not an effect was detected, column **Size** describes whether traditional effect sizes or at least differences between groups were documented.

No	Pub	Experiment	Effect	Size
1	1	Weissman 1974 (1)	no	-
2	1	Weissman 1974 (2)	no	-
3	1	Weissman 1974 (3)	*unclear*	no
4	2	Shneiderman, McKay 1976	no	-
5	3	Love 1977	no	-
6	4	Norcio 1982 (1)	**yes**	**no**
7	4	Norcio 1982 (2)	**yes**	**no**
8	5	Norcio, Kerst	no	-
9	6	Miara et al. 1983 (replication by Bauer et al.)	***yes***	***unclear***
10	7	Kesler et al. 1984	no	-
11	8	Albayrak and Davenport 2010	**yes**	**yes**
12	9	Bauer et al. 2019 (replication of Kesler et al.)	no	-

Table 1 summarizes the experiments on indentation we are aware of.

In 1974, Weissmann (Weissman, 1974) reported different experiments while some of them also studied indentation. A first experiment using the programming language PL/I on the programs Quicksort and "Wirth's Eight Queens" was executed on 16 students. It is not 100% clear what the response variable in the experiment was, because

[5]Both experiments by Sime et. al considered if statements where the if-, then-, and else-clause was or was not separated by line breaks. While we see that this has definitively something to do with code formatting, we do not think that these experiments focus much on indentation but rather on line breaks.

the experiment protocol just mentions that the participants were *"asked for a self-evaluation"* (Weissman, 1974, p. 95). But whatever the dependent variable was, Weissman reported that the experiment did not reveal significant results. A second experiment studied different programs (Quicksort and Postfix) in the programming language Algol on 48 students. Again, Weissman reported that with respect to indentation no statistical differences were found (Weissman, 1974, p. 97). And finally, one experiment was performed again on PL/I on the programs Heapsort and "Shortest Paths in a Graph" using 24 participants: "*The subjects were given five minutes to read the program. They were next asked for a self-evaluation and given fifteen minutes for a detailed study, using any method they wished, and a quiz. They were then asked for a second self-evaluation, and given twenty minutes to make two specific modifications to the program. Finally, they were given ten minutes for a third self-evaluation and a second quiz.*" (Weissman, 1974, p. 131). Weissman reported that "*a paragraphed listing was significantly better than an unparagraphed listing on the first self-evaluation*", but the report did not mention any effect on the other self-evaluations nor on the program modifications which makes it unclear whether or not the experiment could be interpreted as a positive effect for indentation. Finally, one should take into account that even a possible positive effect is only a matter of the subjective self-evaluation of the participants.

In 1976, Shneiderman and McKay reported on an experiment where participants were given an indented and unindented PASCAL program and participants had to locate and repair bugs (Shneiderman and McKay, 1976). The dependent variable was credits for giving correct answers. The authors reported that none of the main effects were significant.

In 1977, a study by Love (Love, 1977) *"also found no reliable improvement in a reconstruction task for indented and unindented short Fortran programs"* (Sheil, 1981, p. 109).

In 1982, Norcio studied the effect of indentation in two trials executed on 70, respectively 60 participants (Norcio, 1982). In each, participants were given Fortan code containing a blank line that needed to be written. One of the variables in the experiment was indentation. The dependent variable was the number of correct statements. The result with regard to indentation was *"that groups with indented programs and interspersed documentation were able to supply significantly more correct statements"* (Norcio, 1982, p. 118). The paper only reports the p-value but does not describe the differences between both groups. Hence, one can only state that the evidence for the difference is strong (p<.05), but it is not possible to state how large the effect was. The second experiment was comparable to the first one. Again, indentation had a positive effect (p<.008), but the effect size of indentation was not documented.

In 1983, Norcio and Kesler gave 60 participants Fortran programs to read and memorize. Afterwards, the participants had to write the remembered code. A number of different treatments were considered (documentation, segmentation into logic boundaries) and one additional treatment was indentation. It turned out that indentation was no significant factor.

In 1983, Miara et al. (Miara et al., 1983) performed an experiment on 86 participants with seven different versions of the same Pascal program (with different kinds of indentation) taken from a textbook. The dependent variable was a comprehension quiz consisting of 13 questions. Additionally, the participants were asked to give a subjective rating. The results showed a strong effect of indentation (p<.05). Unfortunately, it cannot be derived from the paper what the effect is and taking a closer look into the graphical representation of the results (Miara et al., 1983, p. 866) gives the impression that an indentation level of 2 had probably a positive effect (in comparison to no indentation), while an indentation level of 6 rather had a negative effect.

In 1984, Kesler et al. found no significant effect of indentation in their study (Kesler et al., 1984). 72 students received 3 versions of Pascal programs (no indentation, excessive indentation, and moderate indentation). The participants were given a questionnaire consisting of 10 questions. The number of correctly answered questions was the dependent variable. The experiment did not show significant differences using a commonly accepted alpha-level of .05.

In 2010, Albayrak and Davenport divided 88 first year students into four groups with the focus on defects in indentation and naming on the detection of functional defects. Each group received a different version of a comparable piece of code consisting of "about 100 LOC of Java source code" (Albayrak and Davenport, 2010). Each version contained six functional defects, but the versions differed with respect to defects: one version did not contain any defect, one version contained ten naming defects, another version ten indentation defects and the final version contained four indentation defects and six naming defects. Each group's task was to detect functional defects. With respect to indentation defects it turned out that in the presence of indentation

```
1  class Example {
2
3  public static void main(String[] args) {
4  System.out.println("Hello World");
5  if(args.length!=0) {
6  do_something_else();
7  }
8  }
9
10 public static void do_something_else() {
11 System.out.println("Something else");
12 }
13
14 }
```

Figure 2: Unindented code where the effect of indentation is probably small.

defects 25% fewer functional defects were found (η_p^2 = .832). Unfortunately, no more details are given about the study, especially not the code nor the kind of functional defects to be found. Additionally, no information was given about the experiment protocol – but it seems plausible that the same amount of time was given to all groups.

In 2019, Bauer et al. replicated the experiment by Miara et al. on 39 participants where the original code base was migrated to the language Java and additionally an eye tracker was used. Altogether, the authors summarize that the "*results did not show any effect of indentation depth on program comprehension, perceived difficulty, or visual effort, indicating that indentation is indeed simply a matter of task and style, and do not provide support for program comprehension*" (Bauer et al., 2019, p. 163).

In summary, one can say that if we accept the work by Bauer et al. as a failed replication of the findings by Miara et al., we could just state that two publications reveal a positive effect of indentation: the work by Norcio and and the work by Albayrak and Davenport. However, only Albayrak and Davenport report how large the effect was, but the authors did not give any details about the studied code.

Hence, our conclusion is that not much evidence exists on the potential positive effect of indentation.

4 ON THE POSSIBLE EFFECTS OF INDENTATION

Before describing an experiment, it is necessary to understand the initial considerations that led to the experiment. Actually, we think that indentation has different effects for different tasks and for difference source codes.

Figure 2 illustrates a code example from which

```
class Example {                    | class Example {
                                   |
  ... main(...) {                  | ... main(...) {
    int i=...args[0];              | int i=...args[0];
    int j =...args[1];             | int j =...args[1];
    if (i!=j) {                    | if (i!=j) {
      if (j>10) {                  | if (j>10) {
        if (i<10) {                | if (i<10) {
          print(5);                | print(5);
        }else {                    | }else {
          print(10);               | print(10);
        }                          | }
      } else {                     | } else {
        print(12);                 | print(12);
      }                            | }
    } else {                       | } else {
      if(i<10) {                   | if(i<10) {
        print(23)                  | print(23)
      } else {                     | } else {
        print(15)                  | print(15)
      }                            | }
    }                              | }
  }                                | }
                                   |
}                                  | }
```

Figure 3: Indented and unindented Java code where the effect of indentation is probably large. Details are omitted (. . .) or abbreveated (`print`).

we do not believe that indentation has a major effect. If we ask developers about the number of methods, we do not think that the absence of indentation has a measurable effect because we think that the empty lines in line 2 and line 13 are good separators that help identify methods. Likewise, asking developers about the number of lines of code is probably also not influenced by indentation. Also, we think that asking developers how many lines of the code print on the console will not reveal any effects.

Actually, we do not even think that indentation contributes much to the understanding of `Example` because the code can be easily read from top to bottom. The only moment where we expect that indentation has a potential effect is the if-statement's body in line 6. There developers have to infer on their own whether or not the line belongs to the if-statement's body. But since no other lines follow within the same method body we do not think that developers would benefit much from indentation.

We believe the situation is different in Figure 3. When asking "*what is the result of the program if it is started with the parameters 5 and 5*?" indentation has probably a large effect. After someone becomes aware that both parameters are stored in local variables, one has to understand the functionality of the method which is determined by nested if-conditions. Reading such if-statements is from our perspective extraordinary hard without indentation, because even if someone understands that an if-condition does not hold, it becomes hard to determine the next line of code that needs to be read.

We think it makes sense to compare the code from Figure 2 and 3. Our argument is that indentation matters for if-statements, but we still argue that it rather does not count for the code in Figure 2. The reason is that the body of the if-statement in Figure 2 is rather trivial and it does not require major effort to identify the if-statement's body. From our perspective, unindented code becomes complex if larger then- or else-branches exist in the code (possibly even nested).

5 EXPERIMENT: INDENTATION, BRACKETS AND SKIPPING

Based on the previous discussion, we do not think that indentation has a measurable effect in general and we believe there is a need to distinguish between indentation that intends to highlight the control flow of programs (i.e. indentation in if-statements, loops, etc.) and indentation that intends to highlight structural elements (such as method bodies, etc). We believe it makes sense to study the different facets of indentation in separation and the goal of the present paper is to focus on control flows.

Algorithmic Complexity. When studying the control flow of programs, one has to keep in mind that the algorithmic complexity of programs plays a role, too, and we believe that a program's complexity has many facets. For example, we mentioned in Section 3 the study by Weissman where quite complex algorithms were given to participants. We believe understanding such complex algorithms takes time and this potentially hides the effect caused by indentation. Consequently, we see the need to keep the algorithmic part of programs as simple as possible. Keeping this in mind, we think it is reasonable to reduce the code to be studied only to if–statements, because it is plausible that reading and understanding e.g. loops adds an undesired complexity – a study by Ajami et al. gives evidence that loops are harder to understand than if-statements (Ajami et al., 2019). Finally, one needs to keep in mind that even the condition in an if–statement potentially becomes complex, because the Boolean expression might consist of multiple comparisons. Again, our argument is that we should keep this condition as simple as possible in order to avoid additional complexity caused by something that is not in the focus of the study. As a result, we use nested if-statements where the then- or else-branch hardly does anything except printing out something.

Tasks. We already discussed tasks that could be given to developers. However, while tasks such as asking for the lines of code or asking for the number of methods is a purely structural task, we think that – taking into account that we concentrate on the control flow of programs – asking for a program's result is reasonable. I.e., for programs such as the one given in Figure 3 we can simply ask what the output of such program is. Such tasks also have the benefit that articulating the output is quite trivial and does not require any complex explanations by participants.

In the experiment we only distinguish between indented and non-indented Java code. But we wanted to take into account that the identification of if–statement bodies matter instead of the pure lines of code. Our goal was to use the following approach:

- **Non-Skippable Code.** Every outer condition is true, so that all the inner statements have to be checked (which will then be false, except for the actual solution). Consequently, every statement has to be checked.
- **Skippable Code.** Every outer condition is false. That way inner statements can be skipped.
- **Mixed Code.** Mixes the previous ones. Some inner statements can be skipped, some have to be read.

Figure 4 shows an excerpt of the three kinds of skipping in indented Java code. However, we need to take into account that concentrating on skipping (again) leads to the question, whether or not brackets become a factor: it is quite plausible to assume that opening and closing brackets are appropriate separators to help people to identify the start or end of a code body. Consequently, we assume that this influences (potential) differences between indented and non-indented code.

Our goal was to additionally integrate subjective perceptions by participants as well. For that, we applied the NASA TLX (Hart and Staveland, 1988). The NASA TLX is a standard questionnaire to measure mental and physical demand, performance, effort, and frustration. This questionnaire is (although originally built for a different domain) often applied in software engineering (see for example (Fritz et al., 2014; Hollmann et al., 2017; Couceiro et al., 2019; Al Madi et al., 2022) among others). We applied this questionnaire without questions about physical demands (since we do not think that physical demand plays a larger role in our context).

```
int f=14;
int s=23;

if(f==7) {
  if (f==34) {
    print("Y");
  } else if (f==12) {
    print("T");
  }
} else if (s==10) {
  if (s==22) {
    print("A");
  } else if (s==45) {
    print("G");
  }
}
...
...
```

```
int f=15;
int s=26;

if(f==15) {
  if(f==13) {
    print("V");
  } else if (s==14) {
    print("Z");
  }
} else if (s==23) {
  ...
}
...
...
...
...
...
...
```

```
int f=8;
int s=32;

if(f==13) {
  ...
}
if(f==8) {
  if(s==11) {
    print("A");
  }
  if(s==30) {
    print("G")
  }
  ...
}
...
...
...
```

Figure 4: Code excerpts of skippable, non-skippable and mixed code. For illustration purposes, the code is slightly modified: the variable names in the experiment code is different (`firstInt` instead of `f`, `secondInt` instead of `s`), before and after the double equals sign white spaces occured, instead of `print` the experiment uses `System.out.println`.

5.1 Experiment Layout

The experiment was designed as a within-subject experiment where all subjects received 16 programming snippets. Four code snippets were used as warm–up tasks, 12 code snippets were used for the actual measurements.

- **Dependent Variables.**
  - **Time Until Correct Answer.** The time until the correct answer was given. The moment an answer was given, the time was stopped. If the answer was not correct, the participant was told so and and the measurement continued (right after the feedback was given to the participant).
  - **Subjective Ratings.** The NASA TLX with the the dimensions mental demand, performance, effort, and frustration (each measured on a scale from 1=low to 20=high). I.e., there are four different dependent variables, each representing a different dimension in the NASA TLX.
- **Independent Variables.**
  - **Indentation.** With the treatments indentation and non–indentation.
  - **Brackets.** With the treatments with and without brackets.
  - **Skipping Type.** With treatments non-skippable code, skippable code, mixed code.
- **Non-Controlled Variables.**
  - **Background (Between Subject).** Student or professional developer.
  - **Ordering.** The subjects received in the beginning the 4 warm-up tasks (in a random order), afterwards the 12 tasks in random order.[6]
- **Fixed Variable.**
  - **Lines of Code.** All code snippets had 48 lines of functional code (variable declarations, if-statements, print-statement without) including a method header. The code versions with brackets were longer due to the chosen code format (closing brackets in separate lines).
- **Task.** "*Say what the given program prints out*".

Giving each participant 12 tasks is the result of the independent variable: there are altogether 12 different treatment combinations (2 x indentation, 2 x bracket, 3 x skipping type, i.e. 2 x 2 x 3 = 12).

5.2 Execution and Results

The experiment was executed on 10 students and 10 professional developers chosen based on purposive sampling. The code snippets were shown to the participants on the screen. The time measurement was manually done. The experiment started with four warm-up tasks. The data was analyzed using a repeated measures ANOVA using SPSS v27. Table 2 shows the results of the statistical analysis.

If we reduce the analysis to the variable indentation, we receive the ratio of means

[6]We chose to assign the tasks randomly in order to counter–balance potential periodic effects (see for example (Madeyski and Kitchenham, 2018; Hanenberg and Mehlhorn, 2021)).

Table 2: Experiment Results – Confidence intervals (CI) and means (M) are given in rounded, full seconds. Non-significant interactions were omitted. N describes the number of datasets per treatment(-combination).

	df	F	p	η_p^2	**Treatment**	**N**	**CI$_{95\%}$**	**M**
Indentation (I)	1	89.372	<.001	.832	Indented	120	38; 58	48
					Non-Indented	120	115; 153	134
Brackets(B)	1	1.025	.325	.054	*omitted due to insignificant results*			
Skipping(S)	2	1.278	.291	.066	*omitted due to insignificant results*			
Background	1	.797	.384	.042	*omitted due to insignificant results*			
I * S	2	18.025	<.001	.500	Indented/Non-Skippable	40	49; 71	60
					Indented/Mixed	40	35; 52	43
					Indented/Skippable	40	24; 55	39
					Non-Indented/Non-Skippable	40	119; 171	145
					Non-Indented/Mixed	40	122; 170	146
					Non-Indented/Skippable	40	119; 171	144
I * S* B	2	3.820	.031	.175	*ommitted due to large number of combinations*			

$\frac{M_{Non-Indented}}{M_{Indented}} = \frac{134}{48} = 2.791$, i.e. on average non-indented code required 179% more time. However, it is necessary to take the interactions into account. Figure 5 illustrates the interaction Indentation*Skipping. It shows that the difference between indentation and non-indentation is relatively small in non-skippable code $\frac{M_{Non-Indented}}{M_{Indented}} = \frac{145}{60} = 2.417$, while the differences between mixed and skippable are larger ($\frac{146}{43} = 3.396$, respectively $\frac{144}{30} = 3.692$). However, the difference between mixed and skippable code was not significant.

The dataset also contained a significant interaction Indentation*Skipping*Brackets and the results were from our perspective counter-intuitive: while for non–skippable code the ratios between non-indented and indented code were almost constant (with brackets=$\frac{99}{54} = 1.833$, without bracket = $\frac{132}{66} = 2.00$), the introduction of brackets decreased the ratios in skippable code (with brackets = $\frac{158}{33} = 4.788$, without = $\frac{123}{45} = 2.73$).

Background was neither a significant variable, nor did it interact with any other variable.

5.3 Results of Subjective Ratings

We analyzed the responses of the NASA TLX in the same way as the previously measured times. Since four different responses were collected (mental demand, performance, effort, and frustration), four different repeated measures ANOVAs were executed.[7]

[7] We are aware that one could argue that the kind of response, i.e. the different categories, could also be considered as a separate variable in the analysis. However, we think that the main goal of the NASA TLX is to collect different kinds of responses of different categories independent of each other. Consequently, we analyze them independent of each other.

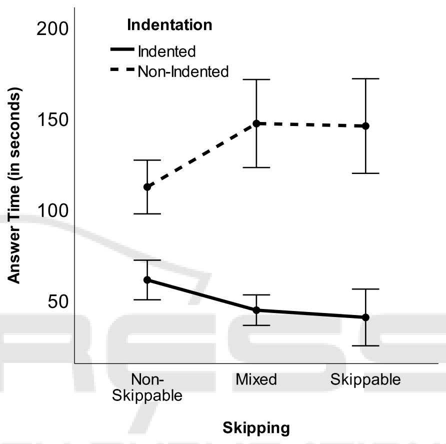

Figure 5: Interaction between indentation and skipping.

Taking into account that reporting each dependent variable in separation takes too much space, we reduce the reporting of the subjective ratings to a (from our perspective acceptable) minimum where we concentrate only on the variables indentation and bracket, and report for each dependent variable (mental demand, performance, effort, frustration) p-values, η_p^2 and means. We intentionally report the variables brackets and indentation, because indentation is the main goal of the current study, and brackets turned out to be a non-significant factor in the previous time measurements. Table 3 summarizes the results of the analyses.

With respect to indentation, the results of the NASA TLX are in line with the previous time measurements. The factor is significant in all dimensions of the NASA TLX: in the absence of indentation participants consider tasks as much more demanding, the effort as much higher, and participants feel less performant, and much more frustrated.

However, with regard to brackets, the results are not even closely in line with the time measurements.

Table 3: Pairwise Comparisons of the variables defined by the TLX

Category	Treatment	Means	p	η_p^2
Mental Demand	Indentation	$M_{Indented}$ = 5.33; $M_{Non-Indented}$ = 12.22	<.001	.789
	Brackets	$M_{Brackets}$ = 8.29 $M_{No-Brackets}$= 9.26	.002	.146
Perfor-mance	Indentation	$M_{Indented}$ = 12.79 $M_{Non-Indented}$ = 9.08	<.001	.396
	Brackets	$M_{Brackets}$ = 11.34 $M_{No-Brackets}$ = 10.53	.027	.081
Effort	Indentation	$M_{Indented}$ = 4.9 $M_{Non-Indented}$ = 11.52	<.001	.68
	Brackets	$M_{Brackets}$ = 7.79 $M_{No-Brackets}$ = 8.62	.005	.126
Frustra-tion	Indentation	$M_{Indented}$ = 3.14 $M_{Non-Indented}$ = 9.59	<.001	.626
	Brackets	$M_{Brackets}$ = 6.04 $M_{No-Brackets}$ = 6.69	.094	.047

The factor brackets is significant in all dimensions except frustation (where brackets approaches significance with p=.094): in the absence of brackets, participants consider the tasks slightly more demanding, the effort slightly higher, and the participants feel slightly more performant.

6 THREATS TO VALIDITY

The experiment does not rely on industrial code but uses artificial code: the conditions in the if-statements are trivial and so are the bodies of the then- or else-branches. And we need to point out that the present work only concentrates on control flows and we believe that the effects on other constructs will be different. While this is obviously an external threat to validity, we need to keep in mind that the goal of controlled trials is to keep as much as possible under control. I.e. using artificial code helps to focus on the variable being studied. But the implication is, that the effects of indentation that can be found in reality are probably much smaller than the ones reported in the present paper: due to other confounding factors, not due to a reduced effect of the main factor.

Another (external) threat is, that the present paper just measures answering times without giving participants the opportunity to interact with the code (via a debugger, etc.). While it is disputable to guess whether or not such situations should be studied at all, one needs to keep in mind that the pure reading of code probably still plays a role even in situations where development tools are available. Actually, so far we are not able to quantify how much reading time is actually required per day by developers, but at least there are studies available that give evidence that time spent in understanding code is probably larger than any other activity in development (see for example Murphy et al. (Murphy et al., 2006)).

We also need to keep in mind that the measurement technique was hand-measured time. We are aware that measuring time that way is quite imprecise and it probably influenced the results of the experiment as well. However, even if we assume that hand-measured time varies even by a few seconds, the measured differences are still large enough to compensate for this effect, i.e. we think it is not plausible that the results differ much if a different measurement technique would be used.

While we do not think it mattered in the experiment, the experiment design suffers from a serious internal threat: participants can compromise experiment results. This can be done simply by reading the code and saying the printed statements as quickly as possible. In principle, this could be solved by adding some algorithmic complexity to the code that does not permit to simply guess what the result of the code might be. However, we need to keep in mind that it was the experiment's intention to remove any kind of algorithmic complexity from the code. Another alternative could be to define an exclusion criterion such as a certain number of errors in the answers.

However, we need to keep in mind that another way to compromise the experiment is to sit down without doing anything. In such case, the answer time will be quite long and possible effects can still be hidden. I.e., we think that this kind of threat is inherent in experiments with time measurements based on participants's activities. So far, we are not aware of any solution to this problem.

7 SUMMARY AND DISCUSSION

This paper performed a randomized control trial on the readability of indented and non-indented code, because it turned out that evidence for an effect of indentation is missing in the literature – despite the often and common application of indentation in teaching and practice: Although the authors of the present paper are aware of 12 experiments where indentation was a controlled factor, only one experiment by Albayrak and Davenport (Albayrak and Davenport, 2010) was able to reveal an effect and report the effect size. However, the experiment did not report much on the code given to the participants nor does it say much about the defects that had to be found. Hence, we think it is quite hard to interpret the results and we see the need for more experiments on that topic.

The experiment presented in this paper (executed on 10 students and 10 professional developers) considered four factors: indentation, skipping, brackets and background. Reducing it only to the factor indentation, large effects could be found with on average 179% higher answer times for non–indented code up to 269% higher answer times in situations where parts of the code can be skipped. Moreover, the experiment showed that the difference in answer times between indented and non-indented code is influenced by the given code: not with respect to the code's length (which was kept constant in the experiment), but with respect to whether or not parts of the code can be skipped. However, it turned out that our notion of skippable and mixed did not reveal the expected results: the differences between indented and non–indented code were the same for both kinds of code. With respect to brackets, our original expectations were that brackets help in non–indented code where part of the code can be skipped. However, it turned out that the opposite was true.

Based on the NASA TLX, we were able to observe that the subjective ratings were in line with the time measurements. However, the subjective ratings also considered the presence of brackets as helpful (except with respect to TLX's dimension frustration where we did not measure a significant difference) which was not in line with the time measurements. Actually, the reason why we introduced brackets in the experiment was that we expected this would help participants especially in situations where indentation was not present. And while this assumption did not hold (insignificant variable brackets in time measurements), the subjective perception of participants matched our original assumption. We think that the latter statement rather expresses the strong belief among developers (and even our own belief) that brackets are helpful, although the non-subjective time measurements did not reveal such a phenomenon.

We should note that the experiment is just on control flows caused by if-statements. We think that such kind of code has one special characteristic: it can be read from top to bottom without the need to go back to other lines that were already read. We believe that other control-flow specific language constructs such as loops have other implications – we think that jumping back and forth in the code would reduce the effect of indentation and we think that other effects (such as the existence of beacons in the code – see for example (Crosby et al., 2002)) play a larger role. Actually, there is already evidence that if-statements are harder to read than loops (see (Ajami et al., 2019)). And we need to keep in mind that the effects reported here are caused by highly controlled source code and we need to keep in mind that source code one finds in reality contains many other factors that do not (or hardly) play a role in the presented experiment. One of these factors is the choice of identifiers: For example, the studies by Binkley et al. (Binkley et al., 2013) or by Hofmeister et al. (Hofmeister et al., 2019) showed that the style as well as the length of identifiers has a large effect on readabilty. Keeping in mind that different code in reality contains different identifiers implies that this is a potential source of deviation (i.e., a confounding factor). Consequently, we believe that it is quite likely that the effects of indentation one finds in industrial code is smaller than in the here reported experiment. While one might argue that this is rather an argument against the present paper (because its generalizability is quite limited and probably does not permit to make a statement about industrial code), we argue that one needs to keep in mind that the whole point in randomized control trials is to control variables. The goal is to identify the effect of the studied variable and to remove possible confounding factors. This is what the present paper did and this is probably the reason why the large effects could be found. It does not mean that we are not aware of multiple confounding factors that exist in industrial code.

Another facet of the study was that the background of the participants did not matter: no significant differences in answer times between students and professionals were found. From our perspective, this is caused by the quite simple control flows. Professional developers probably have more experience with APIs, tools, complex algorithms, or infrastructures, but we do not think that these skills help much when reading control flows.

We think that with respect to the main factor indentation the experimental results are quite clear: the effect of indentation is strong and large. This directly leads to the question how it comes that we hardly find experiments in the literature that revealed this effect.

We believe this is caused by two effects that often appear at the same time and that both possibly hide the effect of indentation. First, the code of experiments in the literature was relatively complex. We do believe that complex code increases the deviation in the dependent variable and that this possibly hides the effect of indentation (in case the effect of indentation is smaller than the deviation). An example for such an experiment was the one by Weissman who gave subjects source code with Quicksort and the Eight Queens Problem.

Another potential problem we find in the literature is the choice of dependent variables such as questionnaires consisting of multiple questions where correct answers give some credit. An example for such an experiment is the one by Shneiderman and McKay. The potential problem is, that some questions might or might not refer to the possible difference between indented or non–indented code but rather on some other facets of the code. For example, in Section 4 we discussed a possible question of how many lines of code some code snippet contains. If a questionnaire contains a number of questions that do not focus on the possible effect of indentation, we think that such questionnaires are rather inappropriate to be used in a study that focuses on indentation. I.e., the resulting studies that end up with null results do not indicate that indentation has no effect. Instead, it simply means that under the experimental conditions no effect could be measured (which is an important difference, because such null effects are possibly caused by inappropriate experimental designs).

Finally, we think that the present paper gives some more insights about a phenomenon that was studied by Johnson et al. (Johnson et al., 2019) as well as Ajami et al. (Ajami et al., 2019): both works identified that the deeper the nesting, the harder is it for developers to understand the code. Actually, the present paper says that nesting has another effect: the presence of nesting indicates code that can be skipped while reading (under some circumstances). I.e., the question is not only how deep a nesting is, but how much code is actually nested. We think that this issue deserves more studies in the future, because the present study was not able to reveal this effect: although the variable skipping was significant, the study did not find a difference between mixed and skippable. I.e., future work should address this in order to find an explanation for the readability differences between indented and non-indented code.

8 CONCLUSION

The main point of the present work is that indentation is an important issue in education and development – but there is so far not much evidence in the scientific literature that indentation actually matters.

The present work shows that indentation has – at least in the context of control flows – a large, positive effect in comparison to non-indented code. But it also shows something different: The effect of indentation is not a fixed factor. It depends on other factors as well and the present work identified "code skipping" as such an influencing factor with larger effects on the difference between indented and non-indented code. However, the experiment was not able to reveal a readability difference between skippable code, something that should be addressed in future work.

To conclude, we think that the present paper could be used to tell students or professional developers why indentation matters. The answer from the present experiment is: Because it saves a lot of time when reading code. We might not know how much time it saves for some given code basis (where multiple, additional factors influence the readability of the code). But in the reported experiment, non-indented code required between 142% and 269% more time to determine the output of the code.

Note

A replication package of the experiment is available via https://drive.google.com/drive/folders/1ZVDMfTBDIvpKF0uQtZemhmvH_LxoUODf. There, also the raw measurements of the experiment are available.

REFERENCES

Ajami, S., Woodbridge, Y., and Feitelson, D. G. (2019). Syntax, predicates, idioms — what really affects code complexity? *Empirical Software Engineering*, 24(1):287–328.

Al Madi, N., Peng, S., and Rogers, T. (2022). Assessing workload perception in introductory computer science projects using nasa-tlx.

Albayrak, O. and Davenport, D. (2010). Impact of maintainability defects on code inspections. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, New York, NY, USA. Association for Computing Machinery.

Bauer, J., Siegmund, J., Peitek, N., Hofmeister, J. C., and Apel, S. (2019). Indentation: Simply a matter of style or support for program comprehension? In *Proceedings of the 27th International Conference on Program Comprehension*, ICPC '19, pages 154–164. IEEE Press.

Binkley, D. W., Davis, M., Lawrie, D. J., Maletic, J. I., Morrell, C., and Sharif, B. (2013). The impact of identifier style on effort and comprehension. *Empir. Softw. Eng.*, 18(2):219–276.

Couceiro, R., Duarte, G., Durães, J. a., Castelhano, J. a., Duarte, C., Teixeira, C., Branco, M. C., de Carvalho, P., and Madeira, H. (2019). Biofeedback augmented software engineering: Monitoring of programmers' mental effort. In *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '19, pages 37–40. IEEE Press.

Crosby, M. E., Scholtz, J., and Wiedenbeck, S. (2002). The roles beacons play in comprehension for novice and expert programmers. In *Proceedings of the 14th Annual Workshop of the Psychology of Programming Interest Group, PPIG 2002, London, UK, June 18-21, 2002*, page 5. Psychology of Programming Interest Group.

Fritz, T., Begel, A., Müller, S. C., Yigit-Elliott, S., and Züger, M. (2014). Using psycho-physiological measures to assess task difficulty in software development. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 402–413, New York, NY, USA. Association for Computing Machinery.

Hanenberg, S. and Mehlhorn, N. (2021). Two n-of-1 self-trials on readability differences between anonymous inner classes (aics) and lambda expressions (les) on java code snippets. *Empirical Software Engineering*, 27(2):33.

Hart, S. G. and Staveland, L. E. (1988). Development of nasa-tlx (task load index): Results of empirical and theoretical research. In Hancock, P. A. and Meshkati, N., editors, *Human Mental Workload*, volume 52 of *Advances in Psychology*, pages 139–183. North-Holland.

Hofmeister, J. C., Siegmund, J., and Holt, D. V. (2019). Shorter identifier names take longer to comprehend. *Empir. Softw. Eng.*, 24(1):417–443.

Hollmann, N., Rossenbeck, T., Kunze, M., Tuerk, L., and Hanenberg, S. (2017). What's the effect of projectional editors for creating words for unknown languages? a controlled experiment. The 8th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU) at SPLASH 2017.

Johnson, J., Lubo, S., Yedla, N., Aponte, J., and Sharif, B. (2019). An empirical study assessing source code readability in comprehension. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 513–523.

Kesler, T. E., Uram, R. B., Magareh-Abed, F., Fritzsche, A., Amport, C., and Dunsmore, H. (1984). The effect of indentation on program comprehension. *International Journal of Man-Machine Studies*, 21(5):415–428.

Love, L. T. (1977). *Relating individual differences in computer programming performance to human information processing abilities.* PhD thesis.

Madeyski, L. and Kitchenham, B. A. (2018). Effect sizes and their variance for AB/BA crossover design studies. *Empir. Softw. Eng.*, 23(4):1982–2017.

Miara, R. J., Musselman, J. A., Navarro, J. A., and Shneiderman, B. (1983). Program indentation and comprehensibility. *Commun. ACM*, 26(11):861–867.

Moher, D., Hopewell, S., Schulz, K. F., Montori, V., Gøtzsche, P. C., Devereaux, P. J., Elbourne, D., Egger, M., and Altman, D. G. (2010). Consort 2010 explanation and elaboration: updated guidelines for reporting parallel group randomised trials. *BMJ*, 340.

Murphy, G. C., Kersten, M., and Findlater, L. (2006). How are java software developers using the eclipse ide? *IEEE Softw.*, 23(4):76–83.

Norcio, A. F. (1982). Indentation, documentation and programmer comprehension. In *Proceedings of the 1982 Conference on Human Factors in Computing Systems*, CHI '82, pages 118–120, New York, NY, USA. Association for Computing Machinery.

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

Shneiderman, B. and McKay, D. (1976). Experimental investigations of computer program debugging and modification. *Proceedings of the Human Factors Society Annual Meeting*, 20(24):557–563.

Sime, M., Green, T., and Guest, D. (1973). Psychological evaluation of two conditional constructions used in computer languages. *International Journal of Man-Machine Studies*, 5(1):105–113.

Sime, M., Green, T., and Guest, D. (1977). Scope marking in computer conditionals–a psychological evaluation. *International Journal of Man-Machine Studies*, 9(1):107..118.

Vessey, I. and Weber, R. (1984). Research on structured programming: An empiricist's evaluation. *IEEE Transactions on Software Engineering*, SE-10(4):397–407.

Weissman, L. M. (1974). *A Methodology for Studying the Psychological Complexity of Computer Programs.* PhD thesis. AAI0510378.