Software Quality Journal
https://doi.org/10.1007/s11219-023-09631-7

A replication study on the intuitiveness of programming language syntax

Vilma Lappi[1] · **Ville Tirronen**[1] · **Jonne Itkonen**[1]

Accepted: 17 April 2023

Abstract
In this article, we present a replication of an empirical experiment that evaluates intuitiveness and comprehensibility of keywords relating to different concepts in programming languages, originally conducted by Stefik and Gellenbeck. Novice programmers face many barriers when learning programming. One of these barriers is syntax, which for many languages is not designed based on empirical evidence. The purpose of the experiment was to provide more empirical evidence on the subject, to find out if the results of the original experiment can be replicated and if conducting the experiment in an environment where English is not the native language affects the results. The results of our experiment replicated most of the findings of the original study and provided further evidence that some syntactic choices in many popular programming languages are unintuitive for novice programmers. Our results suggest that the native language of participants who otherwise had good English skills had little effect when compared to the original study. These results may support programming language designers in making evidence-based design decisions and teachers of introductory programming courses in identifying some of the barriers novice programmers face.

Keywords Program comprehension · Syntax · Programming languages · Novice programmers · Native language in programming

1 Introduction

Developing software and educating new programmers is very expensive. In addition to educating software engineers at university level, considerable investments have been made to incorporate computer science into basic education in many countries (Department for Education (UK), 2013; Finnish National Agency for Education, 2014). Software engineers are also required to stay up to date on the most recent developments in numerous programming languages, libraries, and frameworks. Professionals who work with program code spend hours on reading it, e.g., debugging and reviewing purposes. Making programming

✉ Jonne Itkonen
jonne.itkonen@jyu.fi

[1] Faculty of Information Technology, University of Jyväskylä, Jyväskylä, Finland

Published online: 25 May 2023

languages easier to learn and comprehend would presumably spare large amounts of both time and money.

There is a profusion of programming languages used in both the software industry and the programming education. Despite the commonly stated adage "right tool for the job," there is little agreement on how the selection of the programming language will affect the outcomes in either of these fields. In regard to education, there is not enough systematic evidence to support any particular approach to how introductory programming courses should be taught or which language would be the ideal first programming language, as demonstrated by Pears et al. (2007) and Siegfried et al. (2008) among others.

The lack of evidence based practice in programming language design is explained to a certain degree by the dearth of empirical evidence in regard to the human aspects of programming languages, which has been documented multiple times, some examples being Stefik et al. (2014), Kaijanaho (2015), and Ko et al. (2015). Empirical evidence is essential in the scientific method and is held to high standards in other fields of research, such as medicine, psychology, chemistry, and physics. For programming language designers to be able to make informed decisions, it is important to gather more evidence that is reproducible and testable.

Many of the commonly used programming languages share a similar base (such as the tradition of C style syntax) but have different variations of syntax and semantics. Which of these variations are beneficial is up for debate, since many decisions in programming language design are based on opinions or personal experience of experts or committees instead of empirical evidence. While there is a case to be made for relying on the insight of experts, there is also evidence that their opinions might not line up with findings in empirical research. For example, Devanbu et al. (2016) found that while programmers have strong beliefs in certain claims about software engineering, these beliefs do not necessary line up with empirical evidence and are primarily based on personal experience. Brown and Altadmri (2017) researched whether expert educators understand the kind of problems beginner programmers face and found that the beliefs of the educators did not match the actual problems that were observed.

To understand and produce program code, every aspiring programmer must first understand the syntax of a programming language, that is, the combinations of characters that form the correct structure for program code. Denny et al. (2011) found that syntax errors were common with students of all levels of ability, even those that were performing better than average. Similarly, in a study by Bosse and Gerosa (2017), syntax errors were the problem most frequently reported by students regarding the difficulties of learning programming. Denny et al. continued their work in 2012 and found that a few types of syntax errors are not only considerably more common among students than others but also take the most time for the students to correct. As with the prevalence of syntax errors, the effect did not depend on the ability level of the students. This finding was supported by Tirronen et al. (2015) who found that in the context of a functional programming course most of student mistakes were reported by only three compiler error messages.

Compilers usually alert programmers about syntax errors with error messages that can lead to frustration if implemented poorly, as discussed by Marceau et al. (2011a, b). They found that syntax error messages are often misinterpreted by students. Some research has been conducted to determine whether improving these error messages affects the rate at which syntax errors are made. An article by Denny et al. (2014) suggests that enhancing syntax error messages does not significantly reduce the rate at which students encounter compilation errors. This finding was replicated by Pettit et al. (2017), though a study by Becker (2016) had the opposite results. Even though it is somewhat inconclusive whether

syntactic errors can be reduced with proper error messages, syntax remains a barrier to novices learning programming and more research points to the barrier remaining even with more informative error messages. Even if students start learning with a block-based programming language such as Scratch, where syntactic errors are avoided by allowing the programmer to combine only compatible blocks, sooner or later they will probably become acquainted with general purpose programming languages and syntax as well.

To identify some of the barriers students face, Stefik and Gellenbeck (2011) studied intuitiveness of specific word and symbol choices in programming languages. They asked both programmers and non-programmers to rate words and symbols by how intuitively they describe certain concepts in programming languages. They then compared the intuitiveness rates of the two groups to each other and created a ranking for the choices of each concept to find out which choices were more intuitive than others.

In this work, we replicate the study conducted by Stefik and Gellenbeck (2011) and investigate whether the results from a university in an English speaking country differ from those from a university in a non-English speaking country. We followed the design of the original study and compared our results to theirs to find out if the change of testing environment affected the perceived intuitiveness of the words and symbols. It should be noted that as with the original study, the goal of this experiment was not to find the single best word or symbol for certain concepts, but to get a more general understanding of how the words or symbols relate to the concepts they are supposed to represent.

Overall our results support the findings of Stefik and Gellenbeck (2011). The data suggests that indeed many popular syntactic choices are unintuitive for novice programmers. The native language of the participants did not seem to have much effect on the results, though the participants of our experiment were otherwise proficient in English.

The rest of this article is structured as follows. First, in Sect. 2, we will discuss related work and replication studies. Then, we will describe the design of this study in more detail in Sect. 3. In Sects. 4 and 5, the results of the study are presented and discussed. After that, in Sect. 6, we examine threats to the validity of the study and finally, in Sect. 7, we summarize our findings and conclude.

2 Background

In this section, we further discuss the relevant background for this study. It is divided into two parts. In Sect. 2.1, we present some of the earlier work that relates to our study. Then, in Sect. 2.2, we move to discuss replication and why it is important in science.

2.1 Related work

The article by Stefik and Gellenbeck (2011) consists of two empirical studies. The first study examined whether or not sound cues help programmers in debugging tasks. The results suggested that multimedia environments did not increase debugging performance at first, but with some practice they might be helpful. In the second study, which we replicated, it was found that novice programmers rated many of the commonly used words in programming languages, such as “for” and “while,” the least intuitive. Stefik and Gellenbeck’s results are further examined and compared to the results of this study in Sects. 4 and 5.

A few years later an article by Stefik and Siebert (2013) described four more empirical studies on syntax and novice programmers. The first two continued and expanded upon the research on intuitiveness of different constructs in programming languages. The first of the two partially replicated the study by Stefik and Gellenbeck (2011) but had a larger scope of concepts. The second study focused on the intuitiveness of larger constructs, such as a loop structure, instead of single words or symbols. The results of these studies support the findings of Stefik and Gellenbeck (2011) and reinforce the conclusions that there are differences in the perceived intuitiveness of different word choices and constructs in programming languages. Also, the perceived intuitiveness varies with the amount of programming experience and across different programming languages.

The two other studies reported in the article by Stefik and Siebert (2013) examined the accuracy at which novice programmers produced program code. The accuracy depended on the language they used and on syntax variations of programming language constructs, such as loops. In these studies, Stefik and Siebert found that both language and syntactic differences affect the accuracy rates. In fact, they found that novices who were using languages that adhere to C style syntax were not significantly more accurate than those using a language with randomized keywords. Their preliminary results also suggest that more intuitive word and symbol choices enhance the accuracy of novice programmers.

The effect of native language on programming has been researched increasingly in recent years. Work by Guo (2018) suggests that non-native English speakers face many obstacles in learning programming, when reading and writing code, relating concepts to their native language and finding suitable instructional materials. One of the barriers they identified was understanding abbreviated keywords or identifiers, which supports our hypothesis that native language affects the perceived intuitiveness of word choices in programming languages. Furthermore, when reading and discussing code with others, beginner programmers are forced to verbalize words in code. This can lead to multiple different verbalizations depending on if the student chooses their native language or English for pronunciation and word order (Hermans et al. 2018). This adds another layer of translation and potential burden. Despite these barriers, however, research by Reestman and Dorn (2019) indicates that native language does not have a significant effect on what compiler errors are the most frequent at least in Java.

Programming language localization is being researched as an option to make programming more accessible to non-native English speakers. Dasgupta and Hill (2017) found that novice programmers who used a version of Scratch environment and programming language keywords that were localized to their native language were able to produce more complex programs earlier than programmers from the same country that were using an English version of the environment and keywords. Similarly, a very recent study by Feijóo-Garcí et al. (2020) compared the performance of native and non-native English speakers in English Scratch environment. They found that while there was no significant difference in their performance in testing, non-native English speakers felt that that they would have performed better with a localized environment.

Though it seems that non-native English speakers might benefit from localized programming environments when the keywords are also translated, the participants in the study by Guo (2018) preferred English instructions instead of translated ones when the keywords of a programming language were in English. They felt that translated materials caused confusion and misinterpretation when combined with English keywords. Another very recent article by Piech and Abu-El-Haija (2020) analyzed the use of non-English languages on GitHub and introduced a tool called CodeInternational that could be used to translate comments and identifiers between human languages. While they maintain that

English is the primary language of programming, they propose that a tool like CodeInternational would benefit students in learning both English and programming.

As we can see, most of the studies outlined above point to similar conclusions but are not replications of each other, with the exception of Stefik and Siebert (2013) partially replicating the study by Stefik and Gellenbeck (2011). This brings us to the next section, in which we give a brief overview of the importance of replication in the empirical science, and discuss the use of replication in computer science and the goals we had in replicating the study by Stefik and Gellenbeck (2011).

2.2 Importance of replication

Replication is viewed as one of the cornerstones of any empirical science. It is defined as the action of repeating an experimental procedure with the intention of confirming or disconfirming its results (Schmidt, 2009). Despite being such a fundamental part of empirical science, in many fields of research, repetitions are rarely attempted (Begley & Ellis, 2012; Makel & Plucker, 2014; Stefik & Hanenberg, 2014; Aarts et al., 2015; Brown et al., 2018; Hao et al., 2018). Furthermore, a big portion of the replications are conducted by the same author as the original study (Makel & Plucker, 2014; Hao et al., 2018). Same-author replication introduces bias to the results more easily and it has been shown that the success rate of same-author replications is higher than replications that were not conducted by the same author (Makel & Plucker, 2014; Hao et al., 2018).

The field of computer science is no exception in regard to the low amount of replications. In 2014, Hornbæk et al. published a study on replication rate in human-computer interaction (HCI) research. The study showed that the HCI research field has a replication rate of 3%. Similarly, Hao et al. (2018) reported a replication rate of 2.38% in computer education research. Their research also confirms the existence of same-author bias in computer education research, with the success rate of replications rising from 58 to 72% when a replication study is conducted by the same author as the original study.

Over time there have been multiple different classification systems for replications, including Lykken (1968) and Schmidt (2009). The classification by Schmidt is more recent and widely used and will therefore be used in this discussion. According to Schmidt (2009), there are two kinds of replication studies. Direct replication means using the same methods and experimental design as the original study. Conceptual replication means testing the same hypothesis with different methods. Schmidt also discusses five functions for replication: controlling for sampling errors, controlling for internal validity, controlling for fraud, generalizing the results to a larger or different population, and verifying the hypothesis of an earlier study.

Since one goal of this study is to replicate the experimental procedure of Stefik and Gellenbeck (2011) as closely as possible while taking into consideration the different environment it is conducted in, the study should be considered a direct replication. Some changes to the original study design have been made with careful consideration in order to adapt the experiment for the different population it is aimed for. The changes are discussed further in Sect. 3.2.

This replication has two primary functions: verifying the results of Stefik and Gellenbeck (2011) and finding out whether or not the results of the original study can be generalized to a different population. The participants of the original study were students of Central Washington University. Although the authors do not report any information about the native language of the participants, it is expected that the participants either have English as their native language or are fluent enough to study in English. It is plausible that the perceived

intuitiveness of certain words might be dependent on an individual's ability to comprehend English. Also, since the participants in the original study were students of the same university, there is a possibility that their perceptions of the intuitiveness of certain words were affected by studying the same courses, and being taught by the same teachers among other factors that come with their shared background. Replicating the experiment in another environment also serves as a control for these effects. In this study, we want to examine whether the results of Stefik and Gellenbeck (2011) can be replicated both in general and in a setting where English is not the native language.

3 Study design

In this section, we describe the methodology and design of our study. We start with our research questions and assumptions, and discuss the tasks used in our experiment. Finally, we give an overview of the participants of the study.

3.1 Context

We are trying to answer the following research questions:

RQ1 What word and symbol choices for programming concepts are most intuitive to novice programmers?
RQ2 How does a novice programmer's native language affect the perceived intuitiveness of the word choices?
RQ3 Can the results of Stefik and Gellenbeck (2011) be replicated?

Since this study is a replication of the study by Stefik and Gellenbeck (2011), the first question is the same as theirs. We will look at RQ1 from two angles. First, whether the perceived intuitiveness for a particular word or symbol is different between programmers and non-programmers, and second, how the word and symbol choices compare to each other within a specific concept. The comparison between the two groups tests the assumption that programmers have learned to relate certain words and symbols to specific concepts in programming languages and may rate those more intuitive than non-programmers. The ranking of the different word and symbol choices within a concept gives us insight into which words might be more intuitive to novices than others to represent the concept.

The second research question is related to our hypothesis that the perceived intuitiveness of the word and symbol choices depends on the native language of the person interpreting them. Since the study by Stefik and Gellenbeck (2011) was conducted in the USA and ours in Finland, comparing the results of this study to theirs will give us some understanding of whether native English-speakers differ from non-native English speakers in this regard. Undoubtedly there are other factors than native language that might cause discrepancy between the results of the studies. Those factors are discussed in Sect. 6.

The background for the third research question was already discussed in detail in Sect. 2.2. As stated there, we want to confirm the validity of the experiment design and Stefik and Gellenbeck's (2011) results, and test whether the results of their experiment can be replicated in a different environment.

3.2 Materials and tasks

Following Stefik and Gellenbeck (2011), we presented the participants with 11 programming concepts and asked them to rate how intuitively given words or symbols described the concept on a scale from zero (no relation to the concept) to ten (perfect description of the concept). These ratings gave us numeric scores for each word and symbol that can be compared to each other and tested statistically.

Stefik and Gellenbeck (2011) conducted their study as a pen and paper survey and each question had the following structure: first, a short description of the programming concept and then a list of words or symbols that were each followed by a rating scale from zero to ten. We used the same structure for the questions, but instead of a pen and paper survey we decided to use an online survey built on the Webropol survey tool (Webropol, 2020). We chose the online survey to reach more participants with more variation in programming ability. Possible threats to validity that are introduced by this type of survey are discussed in Sect. 6.

We translated the questions of the original study from English to Finnish so that the participants' ability to comprehend English would only affect their understanding of the words they were rating instead of their understanding of the concept or question itself. The translations were balanced to the best of our ability to represent the original description by Stefik and Gellenbeck (2011) as closely as possible while making sure that they would be understood correctly by a Finnish reader and would create as little bias as possible towards any of the words or symbols. The translations pose some obvious threats to the validity of the study, which are again discussed in Sect. 6.

We tested the translations with a pilot study to make sure that they were correctly translated and described the same concepts as the original descriptions. In the pilot study, participants were given both the original description and the translated ones with the word list for each concept. They were asked to rate on the scale from one (extremely poorly) to five (extremely well), how well the translations corresponded to the original descriptions and how well the word lists fit the translated question (whether the translation described the right concept). The participants were asked to give reasoning for their ratings and feedback for each question. The pilot had four participants who had programming experience and could therefore evaluate how well the descriptions fit the concepts.

Based on the pilot study, the translations were adjusted. We had originally translated parts of the answer options in addition to the concept descriptions. For example, in the questions about comparison symbols, the structure for the answer options included two conditions ("It is raining" and "you have money"), which were originally translated into Finnish, and the symbol or word to be rated in between of the conditions. The participants of the pilot study found it confusing that the Finnish conditions had English keywords between them, and therefore in the final survey only the descriptions of the concepts were translated (Appendix).

3.3 Participants

The survey was sent to all students of the University of Jyväskylä and 158 participants decided to take part in the survey. The participants were divided into two groups, programmers and non-programmers, based on their programming experience. Stefik and Gellenbeck (2011) grouped their participants based on whether they attended a programming class or not. They reported the programming experience of the participants in self-reported years

of experience, but we decided to measure it also by self-evaluation because it seems to be a more reliable way to measure programming experience according to Siegmund et al. (2014). We did also measure it by self-reported years of programming experience, but that particular question seemed to cause some confusion in participants. For example, some participants answered with a specific year instead of a number of years. Also, when we compared self-evaluated programming skills with self-reported years of programming experience, some participants had evaluated themselves to be experienced programmers but still reported zero years of programming experience.

In the programming experience self-evaluation, participants rated themselves from 1 to 10, with 1 being very inexperienced and 10 being very experienced. Participants who rated their experience between 1 and 2 were counted as non-programmers and those who rated their experience between 3 and 10 were counted as programmers. The average rating was 1.30 for non-programmers and 5.38 for programmers. A few participants did not answer all necessary background questions and their answers were left out. All in all the study included 90 programmers and 63 non-programmers, with a total of 153 participants. Of the programmers, 64 were men, 24 were women, one reported their gender as "other," and one did not specify their their gender, while of the non-programmers 18 were men, 43 were women, one reported their gender as "other," and one did not specify their their gender.

All the participants were non-native English speakers and had Finnish as their native language. We received two answers from participants who reported their native languages as Russian and German, but their answers were left out since there were too few of them to compare to the native Finnish speakers. The participants rated their English reading comprehension ability using the Common European Framework of Reference for Languages (CEFR), which is an international guideline for describing foreign language ability (Council of Europe, 2020). The ability is rated on a 6-point scale, from A1 for basic users to C2 for proficient users. Both programmers and non-programmers rated themselves quite proficient on average. If the scale is converted to numbers from 1 to 6, with one being A1 and six being C2, the average score for programmers was 5.33 and the average score for non-programmers was 4.94. More detailed information about the participants is shown in Table 1.

4 Results

In this section, we will examine the results of our experiment. First, in Sect. 4.1, we go over the statistical methods we used to analyze and test our data. Then, in Sect. 4.2, we describe the results of the experiment and finally, in Sect. 4.3, we describe the differences between the results of our study and those of Stefik and Gellenbeck (2011).

4.1 Statistical methodology

We used the same statistical methods as Stefik and Gellenbeck (2011) to compare programmers and non-programmers. We used Student's *t*-tests to find out whether the ratings for each word or symbol choice were significantly different between programmers and non-programmers. To form rankings for the word or symbol choices for each question, we first ordered them by the mean of the rating they received. We then compared them pairwise with *t*-tests, starting from the highest rated word or symbol. If the *t*-test did not result in a significant difference between the words, they would receive the same rank and the next word would be compared to the highest rated word of the current rank. If the *t*-test

Table 1 An overview of the participants' gender, age, and self-reported English reading comprehension ability measured with the Common European Framework of Reference for Languages (CEFR)

Gender	**Programmers**	**Non-programmers**
	n (%)	*n* (%)
Male	64 (71.1%)	18 (28.6%)
Female	24 (26.7%)	43 (68.3%)
Other	1 (1.1%)	1 (1.6%)
Not specified	1 (1.1%)	1 (1.6%)
	90 (100%)	63 (100%)
Age	**Programmers**	**Non-programmers**
	n (%)	*n* (%)
Under 20	2 (2.2%)	1 (1.6%)
21 to 25	27 (30.0%)	38 (60.3%)
26 to 30	29 (32.2%)	13 (20.6%)
31 to 35	11 (12.2%)	2 (3.2%)
36 to 40	9 (10.0%)	4 (6.3%)
41 to 45	5 (5.6%)	3 (4.8%)
46 to 50	2 (2.2%)	0 (0%)
51 to 55	2 (2.2%)	2 (3.2%)
56 to 60	1 (1.1%)	0 (0%)
Over 60	2 (2.2%)	0 (0%)
Not specified	0 (0%)	0 (0%)
	90 (100%)	63 (100%)
CEFR	**Programmers**	**Non-programmers**
	n (%)	*n* (%)
A1	0 (0%)	1 (1.6%)
A2	0 (0%)	1 (1.6%)
B1	2 (2.2%)	2 (3.2%)
B2	13 (14.4%)	16 (25.4%)
C1	29 (32.2%)	20 (31.7%)
C2	46 (51.1%)	23 (36.5%)
	90 (100%)	63 (100%)
Programming experience	**Programmers**	**Non-programmers**
Self-reported	*n* (%)	*n* (%)
1 (very inexperienced)	0 (0%)	44 (69.8%)
2	0 (0%)	19 (30.2%)
3	30 (33.3%)	0 (0%)
4	8 (8.9%)	0 (0%)
5	11 (12.2%)	0 (0%)
6	8 (8.9%)	0 (0%)
7	13 (14.4%)	0 (0%)
8	14 (15.6%)	0 (0%)
9	4 (4.4%)	0 (0%)
10 (very experienced)	2 (2.2%)	0 (0%)

Table 1 (continued)

Programming experience	**Programmers**	**Non-programmers**
Self-reported	*n* (%)	*n* (%)
	90 (100%)	63 (100%)

resulted in a significant difference between the words, the latter word would become the first word in the next rank and the next word would be compared to it.

We compared the results of our experiment to those of Stefik and Gellenbeck (2011). First, we compared whether the ratings for individual words differed between the participants of our experiment and the participants of Stefik and Gellenbeck's experiment. These comparisons were made using 95% confidence intervals by observing which intervals did not overlap each other, therefore indicating a significant difference between the ratings. Then, we tested how similar the orders of the words and symbols in each questions were. To compare the orders of the ranked lists, we used Kendall's *W* and Kendall's tau-b. Kendall's *W* was used to compare the similarity of all four orders (programmers and non-programmers in our experiment and both groups in Stefik and Gellenbeck's experiments) for each question at the same time. Kendall's tau was used to compare two orders to each other (for example programmers in our experiment vs. programmers in Stefik and Gellenbeck's experiment). Appropriate formulas were used to accommodate for ties in the orders (Kendall, 1945).

4.2 The results of this study

In this section, we will present results that relate to **RQ1**. We will look at what word and symbol choices were rated highest and lowest by the programmers and non-programmers who participated in the experiment and compare the groups to each other. We identified three recurring patterns when comparing the ratings for each question and we will discuss the questions in the groups that show a similar pattern. There was one question that did not fit these patterns, which will be discussed on its own.

In four of the eleven questions, both programmers and non-programmers had clear most preferred options, giving a few words or symbols a significantly higher rating than the others. At some point, there was a clear gap between the ratings of subsequent ranks. Typically after the gap the rest of the options had a lot of ties between them in the rankings. A good example of this is **Question 7**, in which the participants were asked to rate how intuitively different symbols represent assigning a value for a variable. The results for this question can be seen in more detail in Fig. 7, which also shows the gap in the ratings that is typical of this pattern. The symbol = was rated the highest by both non-programmers (mean = 7.71, sd = 3.24) and programmers (mean = 8.97, sd = 2.15). The difference between the first and second highest rating was 3.55 for non-programmers and 3.45 for programmers. The programmers had a clear second ranked option as well, which was := (mean = 5.52, sd = 3.70), but non-programmers had seven options ranked for the second place.

This pattern can also be seen in **Question 11**, which was about how intuitive certain symbols were for concatenating text strings. The highest rating was given to +, again by both non-programmers (mean = 8.21, sd = 2.38) and programmers (mean = 8.51, sd = 2.13). Both groups had a clear second place too, &, with a difference of 2.53 (non-programmers) and 4.02 (programmers) to the highest rating. The rest of the options had even lower ratings with a lot of ties for the same ranks.

The three other questions with the same pattern were **Questions 4**, **5**, and **6**. The participants were asked to rate word and symbol choices for and-, or-, and xor-operators respectively. For and-operator, the highest rated option for non-programmers was `and` (mean = 9.30, sd = 1.70) and the second highest rated option was `&` (mean = 8.52, sd = 2.50). After that, there was a gap of 3.25 before the third highest rated option which was tied for the third rank with three others. The highest rated option for programmers was also `and` (mean = 9.27, sd = 1.62). Their second rank was a tie between `&&` (mean = 8.03, sd = 2–75) and `&` (mean = 7.79, sd = 2.86). They had two definitive gaps in their ratings with the rating of the third ranked option, ∧ (mean = 4.22, sd = 3.94), dropping from the second rank but also having a gap of 2.59 after it.

For or-operator, the highest rated option for both groups was the word `or` (non-programmers: mean = 8.59, sd = 2.73 and programmers: mean = 9.22, sd = 1.49). For non-programmers, the rest of the options were tied for either second or third rank, with the difference between the highest and second highest ranked options being 3.99. The programmers had the following commonly used symbols in ranks two through four: `||` (mean = 6.75, sd = 3.33), `|` (mean = 5.64, sd = 3.75) and ∨ (mean = 4.26, sd = 3.76). The rest of the options were tied for ranks five through seven. For xor-operator, both groups had again a clear highest rated word choice but this time they preferred different words. The non-programmers gave their best rating to `or` (mean = 7.57, sd = 3.41) with a difference of 3.17 to the second ranked option and programmers gave their best rating to `xor` (mean = 6.60, sd = 3.60) with a difference of 2.61 to the second ranked option. The rest of the words and symbols were tied for ranks 2 through four (non-programmers) or two through five (programmers).

The second recurring pattern, which we identified in three questions, was that neither group had a single highest rated option or the difference between the highest and second highest options was not as big as in the previous group. Also, the ratings were spread more linearly with high, medium-range, and low ratings. This linear descent of the ratings can be seen in Fig. 2, which shows the results of Question 2. In this group of questions, there were few significant differences between the ratings of programmers and non-programmers, with the highest rated options being mostly the same with some small variation in their order. An example of this pattern is **Question 2**, which was about rating which symbols were the most intuitive to surround the condition of a conditional statement. For non-programmers, the highest rated pair of symbols was parentheses (mean = 7.44, sd = 3.23), which was tied for first rank with square brackets, no symbols, and curly brackets. The highest rated option for programmers was also parentheses (mean = 8.45, sd = 2.44) and tied for the first rank was no symbols. The other symbols that were in first rank for non-programmers, square and curly brackets, were tied for second place.

In **Question 3**, the ratings were also more linear with no clear gap at any point between ranks. The participants were asked to rate potential word choices to represent a function or a method. Again, there was little disagreement between the ratings of the groups. The non-programmers had four words tied for first rank, the highest rated one being `function` (mean = 8.05, sd = 2.22) and the others `operation`, `action`, and `task`. The programmers had only one word in the first rank, `function` (mean = 8.49, sd = 1.74). Tied for second rank were `method`, `operation`, `task`, and `action`. The gap between the highest and second highest rated words was 1.15.

The third question in this group is **Question 8**, in which participants evaluated words and phrases to represent a conditional statement. The highest rated option for non-programmers was `only if` (mean = 8.48, sd = 2.41), which was tied for first rank with `if` and `with the condition that`. Similarly to Question 3, the programmers had one option in the first rank, which was `if` (mean = 9.28, sd = 1.63). The difference

between the second highest rated option, `only if`, and the highest rated one was 1.4. Also, tied for the second rank was `while`.

The third pattern we identified was a mix of the previous two. In this group, which includes two questions, the programmers had a clear top two in their ranking while the non-programmers were more undecided, with a lot of ties in their ranking. The first question in this group is **Question 9**, the results of which can be seen in more detail in Fig. 9. The figure shows how the ratings of non-programmers are close to each other while the programmers have a rating that jumps out of the line and is higher than the rest. In this question, the participants were asked to rate how intuitively different symbols represent calling a method of an object. For non-programmers, the highest rated symbol was `:` (mean = 6.11, sd = 3.23), which was tied for first place along with = and `->`. The rest of the options were tied in either second or third rank, except for `@#-` which was alone in rank four. The programmers, however, had `.` as their highest rated symbol (mean = 8.02, sd = 2.90) and `:` as their second highest rated symbol (mean = 6.28, sd = 2.90) in ranks one and two. The rest of the options were tied in ranks three to five, with `@#-` being alone in rank six, similarly to the non-programmers.

Question 10, in which participants rated which symbols were the most intuitive to surround a text string, followed the same pattern. Again, non-programmers had a lot of ties in their ranking, with 5 options tied for the first rank. The highest rated one was double quotes (mean = 7.25, sd = 3.18) and the others were parentheses, curly brackets, double slashes, and single quotes. The programmers had double quotes as the highest ranked option (mean = 8.42, sd = 2.68), single quotes as the second highest ranked option (mean = 7.52, sd = 3.00), and the rest of the options were close to each other with many ties (Figs. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, and 11).

There was one question, **Question 1**, which did not fit any of the patterns we identified with the other questions. In this question, the participants were asked to evaluate how intuitively different words described a loop. As shown in Fig. 1, the word `repeat` was at the top of the ratings of both groups. Otherwise, the words which received high ratings from one group seemed to receive low ratings from the other group and vice versa. Non-programmers had two words tied for the first rank: `repeat` (mean = 7.63, sd = 3.01) and `redo` (mean = 6.73, sd = 3.38), and three words tied for the second rank: `again` (mean = 6.46, sd = 3.03), `do`, and `rerun`. Programmers also had two words tied for the first rank and three for the second, but the words were mostly different. The words tied for the first rank were `loop` (mean = 7.90, sd = 2.56) and `repeat` (mean = 7.61, sd = 2.89) and for the second rank `while` (mean = 6.79, sd = 2.95), `foreach`, and `iterate`.

4.3 Comparison to the original study

In this section, we will describe the results that relate to **RQ2** and **RQ3**. We will present comparisons between our results and those of Stefik and Gellenbeck (2011). As explained in Sect. 4.1, we compared both the order in which the words and symbols were ranked by the participants, and the ratings that were given to individual words and symbols. When comparing the ordered rankings, we found a significant difference between the studies in only one question. When we compared the ratings of individual words, we found more differences. We grouped the questions into three categories based on between which groups of participants the individual differences were found. We will discuss each of these categories later in this section, but first we will look the difference in the order of the words and symbols.

The question in which the order of the words and symbols in our study was significantly different from the order in Stefik and Gellenbeck's study was **Question 5**. The difference in the

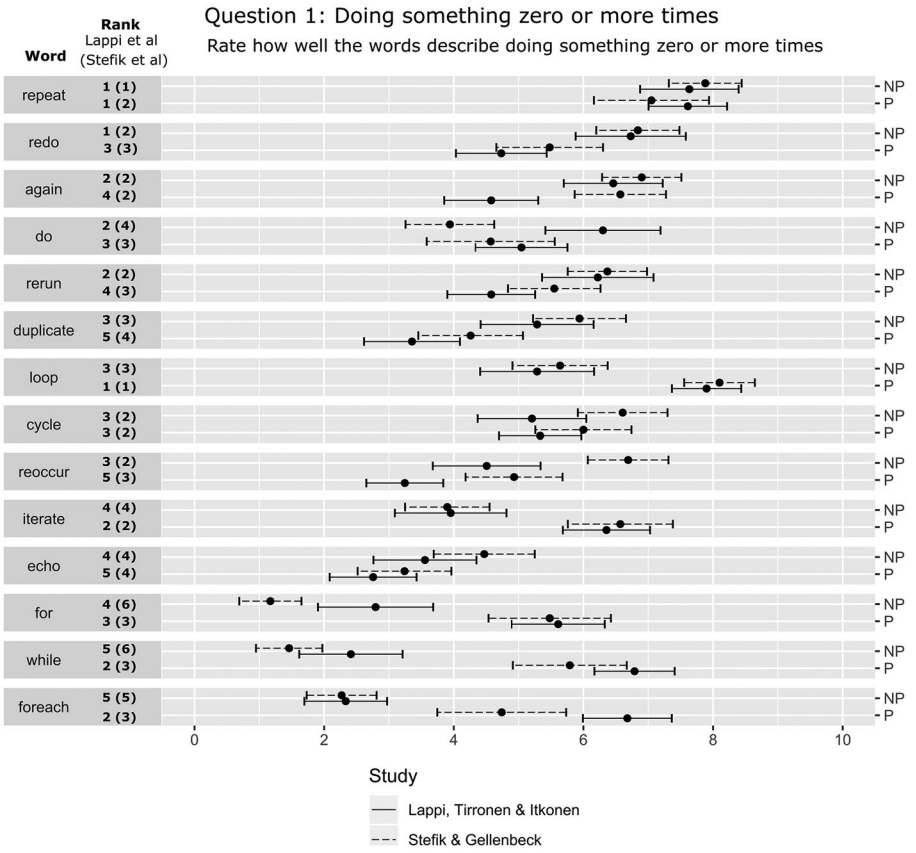

Fig. 1 A figure showing the rankings and ratings with 95% confidence intervals for Question 1 (NP, non-programmer; P, programmer)

order could only be found between the non-programmers of the different experiments (Kendall's tau-b = 0.24, $z = 0.96$, $p = 0.34$). Between the programmers in each study, there was no significant difference in word and symbol order. As can be seen in Fig. 5, the non-programmers in Stefik and Gellenbeck's experiment rated `and`, `&&`, and `&` significantly higher than the non-programmers in our experiment. Therefore, those options also placed higher in their ranking compared to ours, pushing some of the options that were ranked high in our ranking lower in theirs. This makes the orders of the words different enough to be significant.

Next, we will move on to describe the differences in the ratings of individual words and symbols. The first recurring pattern we found was in questions where the differences occurred only between the groups of programmers while non-programmers of both experiments gave similar ratings to all the words and symbols. Three questions fit this category: **Questions 6, 9**, and **10**. While the order in which the words and symbols were ranked in was similar between the groups, the programmers in Stefik and Gellenbeck's experiment rated many words or symbols significantly higher than the programmers in our experiment. In **Question 6**, the results of which can be seen in Fig. 6, the programmers in the original experiment gave a significantly higher rating to nine out of the thirteen words and symbols than the programmers in our experiment. In **Question 9**, both groups of programmers gave a

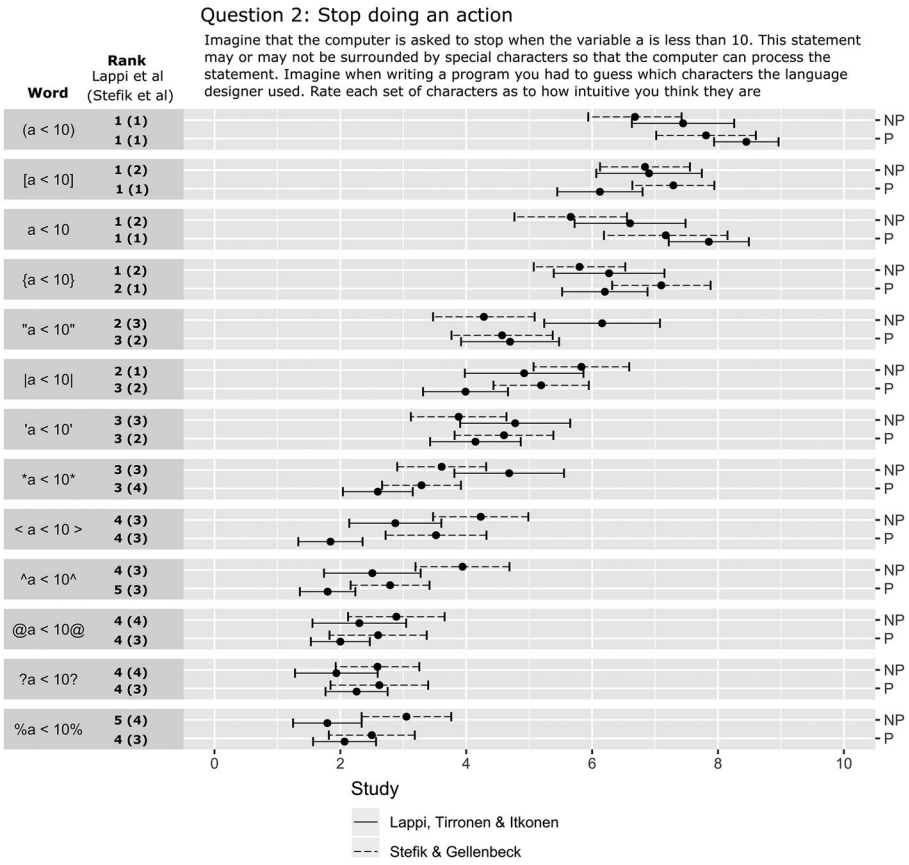

Fig. 2 A figure showing the rankings and ratings with 95% confidence intervals for Question 2 (NP, non-programmer; P, programmer)

similar rating to their highest rated symbol, `.`, but after that the ratings of our programmers drop more steeply. Similarly, in **Question 10**, the two highest rated pairs of symbols, `" "` and `' '`, received a similar rating but after that the ratings of our programmers declined faster. The results of Questions 9 and 10 can be seen in more detail in Figs. 9 and 10.

The second recurring pattern we found was questions where there were significant differences between both the programmers and the non-programmers of the studies. This pattern was identified in six questions: **Questions 1, 3, 5, 7, 8**, and **11**. We will point out only some of the individual differences, the rest can bee seen in the figures relating to each question. In **Question 1** (Fig. 1), the word `reoccur` was given a higher rating by both of Stefik and Gellenbeck's groups when compared to our groups. The same occurs in **Question 3** (Fig. 3) with the word `concern`, in **Question 7** (Fig. 7) with the symbol `=:`, and in **Question 8** (Fig. 8) with the words `for`, `loop`, and `redo`. In **Question 3**, the ratings of the word `function` were the other way around, with both groups in our experiment rating the word higher than the groups in the original experiment.

In **Questions 5** (Fig. 5) and **11** (Fig. 11), all the significant differences were such that the participants in Stefik and Gellenbeck's experiment rated the words or symbols higher than the participants in our study did. For **Question 5**, the biggest differences

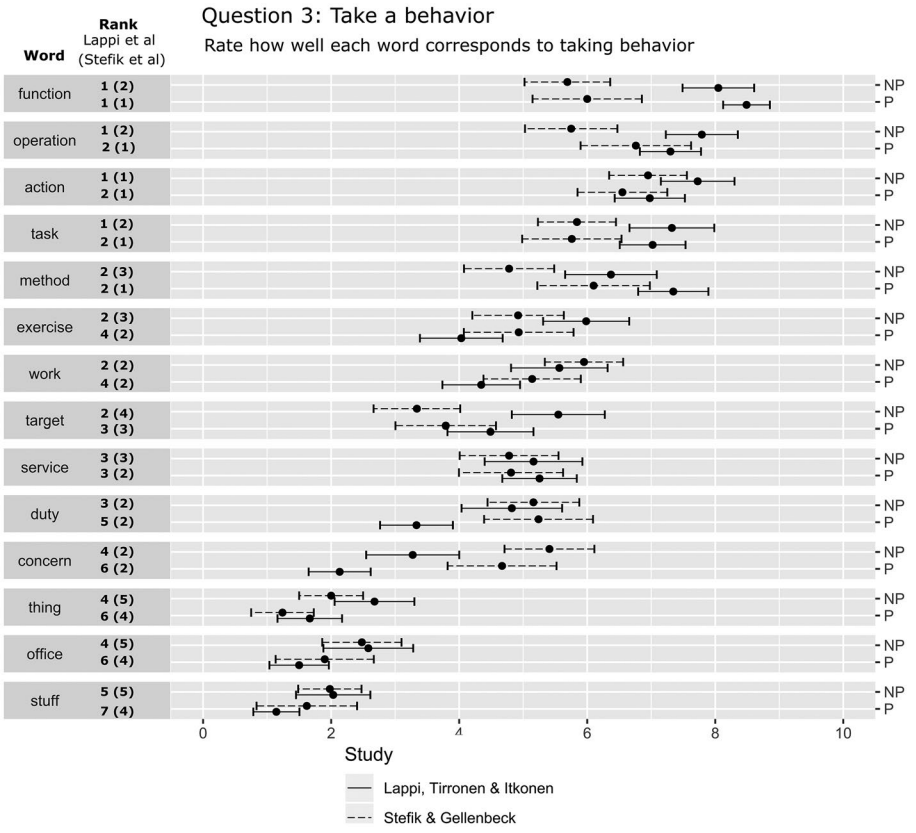

Fig. 3 A figure showing the rankings and ratings with 95% confidence intervals for Question 3 (NP, non-programmer; P, programmer)

were with the words or symbols `and` and `&` for which the difference was between non-programmers, and the word `xor` for which the difference was between programmers. For **Question 11**, the biggest differences were with the symbols `&` and `*` for which the difference was between the programmers of the studies.

The last recurring pattern we found was questions where there were little to no differences between the groups of the different experiments. In **Question 2**, there were little differences both within the studies and between them, as can be seen in Fig. 2. In **Question 4** (Fig. 4), there are more differences within the studies, but only one significant difference between them, with the programmers in Stefik and Gellenbeck's experiment giving a higher rating to the symbol `%%` than the programmers in our experiment.

5 Discussion

In this section, we will discuss the results presented in the previous section and how they relate to our research questions that were defined in Sect. 3.1. We will begin with **RQ1**. In Sect. 4.2, we divided the questions into three categories based on recurring patterns we identified.

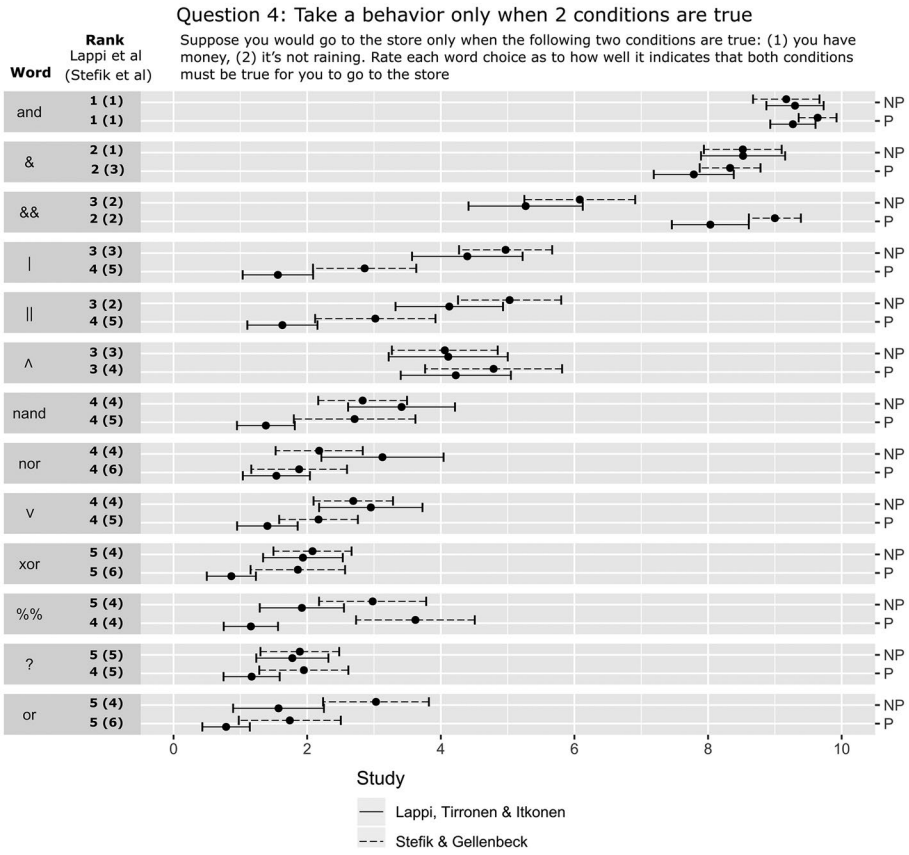

Fig. 4 A figure showing the rankings and ratings with 95% confidence intervals for Question 4 (NP, non-programmer; P, programmer)

In the first category, both programmers and non-programmers had clear top rated words or symbols. Both groups thought = was the most intuitive symbol for assignment and + for concatenating strings. The participants are probably familiar with these symbols from mathematics, where they have similar, though not exactly the same, functions. This might explain the intuitiveness of these symbols. The symbols are already commonly used in programming languages to represent these concepts and the results suggest that the symbols = and + are good choices regarding novice programmers.

The questions regarding logic operators also fell into this category. Both programmers and non-programmers seem to prefer words over symbols when it comes to these operators, since `and` and `or` were the top rated options for their corresponding operators. The groups disagreed on the most intuitive word for xor-operator, which will be discussed later in this section when comparing our results to those of Stefik and Gellenbeck (2011). Though logic operators are often represented by symbols, these results suggest that novice programmers might benefit from using words instead.

In the second category, the ratings of the words and symbols descended more linearly instead of having a clear gap at some point. Because of this, there seems to be a group of good options for these concepts instead of any clear top choices. For surrounding the condition

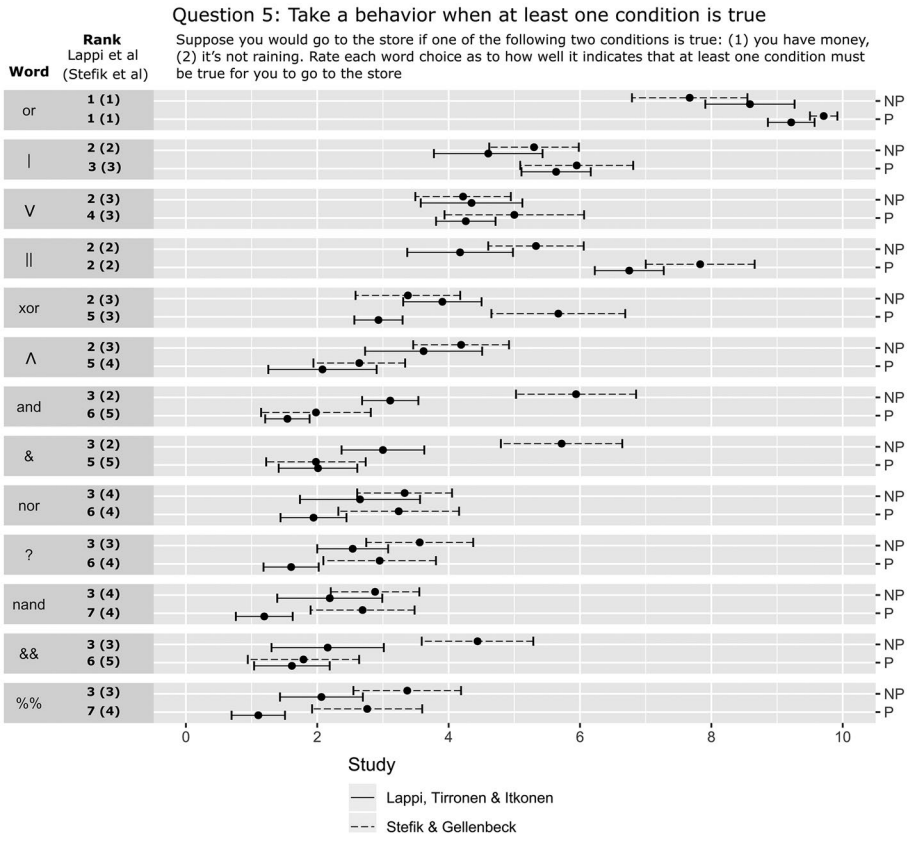

Fig. 5 A figure showing the rankings and ratings with 95% confidence intervals for Question 5 (NP, non-programmer; P, programmer)

in a conditional statement any kind of brackets or no symbols at all seem to work for non-programmers, though since parentheses are commonly used in programming languages, they are probably a good choice for familiarity. Whether the condition needs any symbols around it is something to consider. At least the participants in this experiment rated the option without any symbols highly, though the example they rated was quite simple. Drawing better conclusions regarding that would need more research with more complex examples.

As for conditional statements themselves, `if`, `only if`, and `with the condition that` were the highest rated options, but since `if` is commonly used already and the shortest of these options, it might continue to be the best choice. In this question, these highest rated options were quite similar to each other, which probably explains their similar ratings. There being few differences between the ratings of programmers and non-programmers in this category might indicate that these ratings are not affected by the programming experience of the participant.

In the third category, the programmers had clear top choices while the non-programmers had more even ratings. This might suggest that these questions were more affected by the programming experience of the participants. This might also be the result of the concepts being quite specific to programming and perhaps they were not as easy to understand for

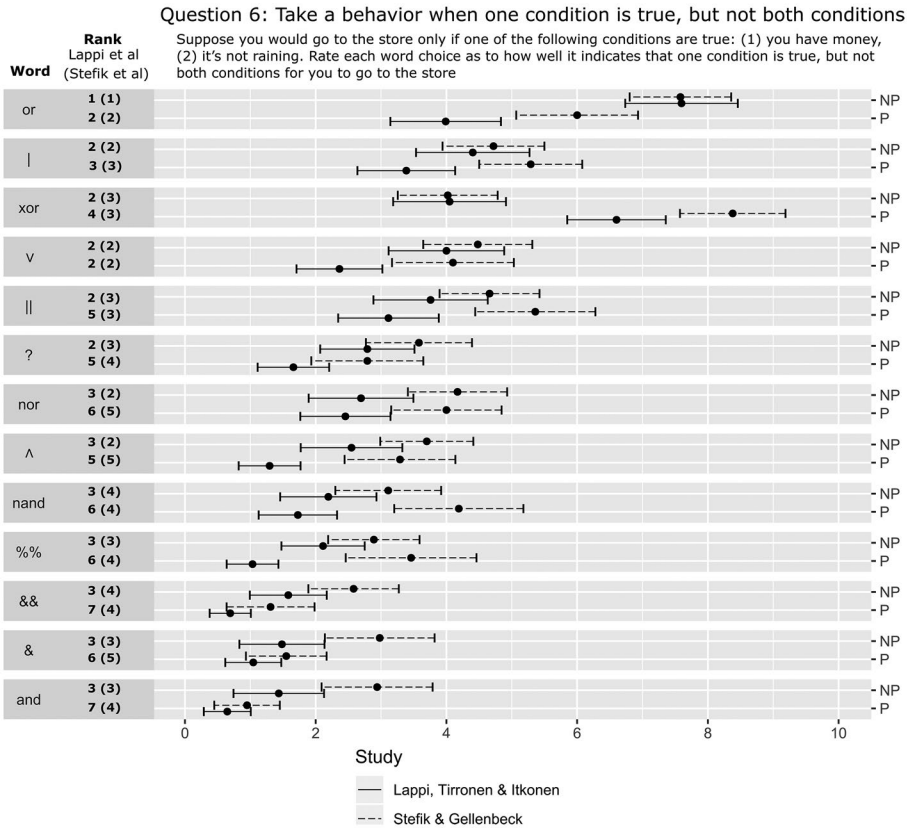

Fig. 6 A figure showing the rankings and ratings with 95% confidence intervals for Question 6 (NP, non-programmer; P, programmer)

novice programmers. The concept of calling a method of an object might have been especially obscure to the non-programmers since there is no clear real-life representation for it. There is also the possibility (as with any of these questions) that there are more intuitive options for non-programmers but they were missing from this study.

The concept that divided programmers and non-programmers most was loops. Though `repeat` seems to be the best choice according to both groups, otherwise, the ratings were quite different. The commonly used words `foreach`, `for`, and `while` were rated highly by the programmers but poorly by the non-programmers. This suggests that, though the common words for loops are very unintuitive for novice programmers, the experienced programmers have likely learned to associate them with the concept as they are so widely used in programming languages, and thus rated the common words so highly.

In **RQ2**, we were interested in whether the results of Stefik and Gellenbeck could be replicated in a setting where English is not the native language of the participants. The differences between the studies ended up being mostly in the ratings of individual words rather than in larger patterns. We expected the native language of the participants to have a bigger impact on the results, though it should be noted that, as shown in Sect. 3.3 and Table 1, the participants in our experiment understood English well based on their own

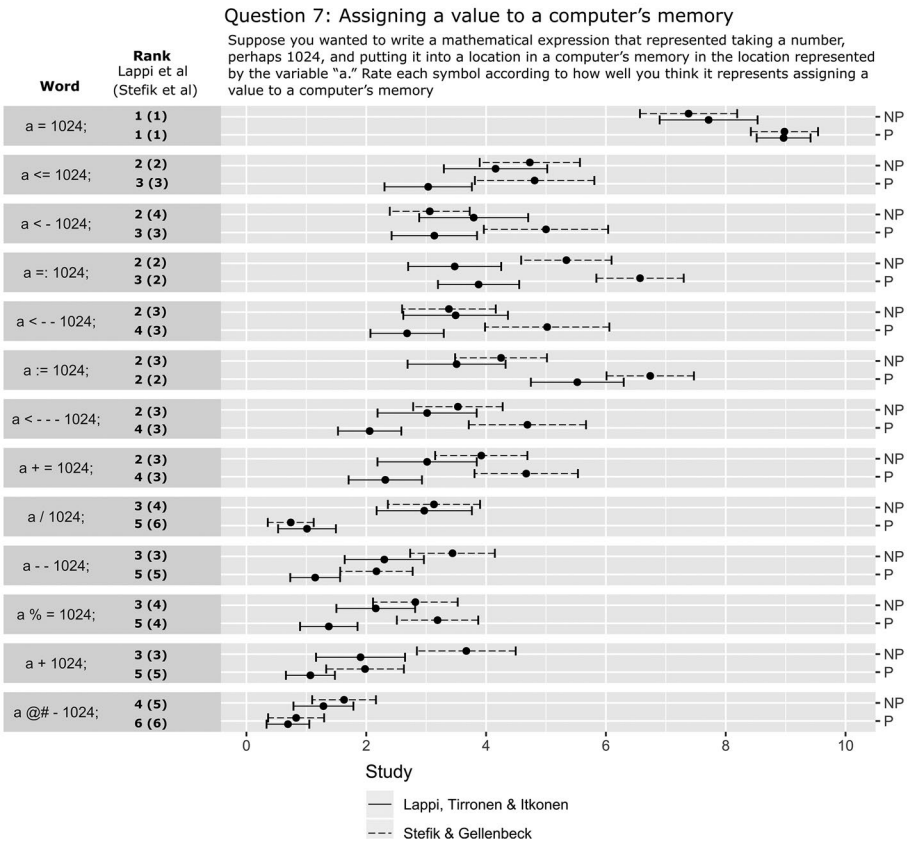

Fig. 7 A figure showing the rankings and ratings with 95% confidence intervals for Question 7 (NP, non-programmer; P, programmer)

evaluation. There might have been more differences if the participants were less proficient in English, but verifying this requires doing more experiments with suitable participants. We originally intended to compare the results within our experiment based on the English skills of the participants to get more insight into how their skills affected their ratings of different words and symbols. Unfortunately the distribution of out participants' English skills leaned too heavily toward the proficient side so that statistical testing was not possible. Therefore, we cannot answer **RQ2** other than that the native language of a programmer does not seem to have a significant effect on the perceived intuitiveness of these keywords as long as they are otherwise proficient in English.

Overall our results were quite similar to those of Stefik and Gellenbeck (2011) and most of the findings that they discussed in their article were supported by our experiment. This suggests that the answer to **RQ3** (Can the results of Stefik and Gellenbeck (2011) be replicated?) is yes, because it would seem that the experiment design gives consistent results and that the results can be replicated in a different setting.

One of Stefik and Gellenbeck's most notable findings was that the words that are often used to describe loops in programming languages are not very intuitive to novice programmers. Our results provide further evidence of this. In our study, the word `redo` was tied

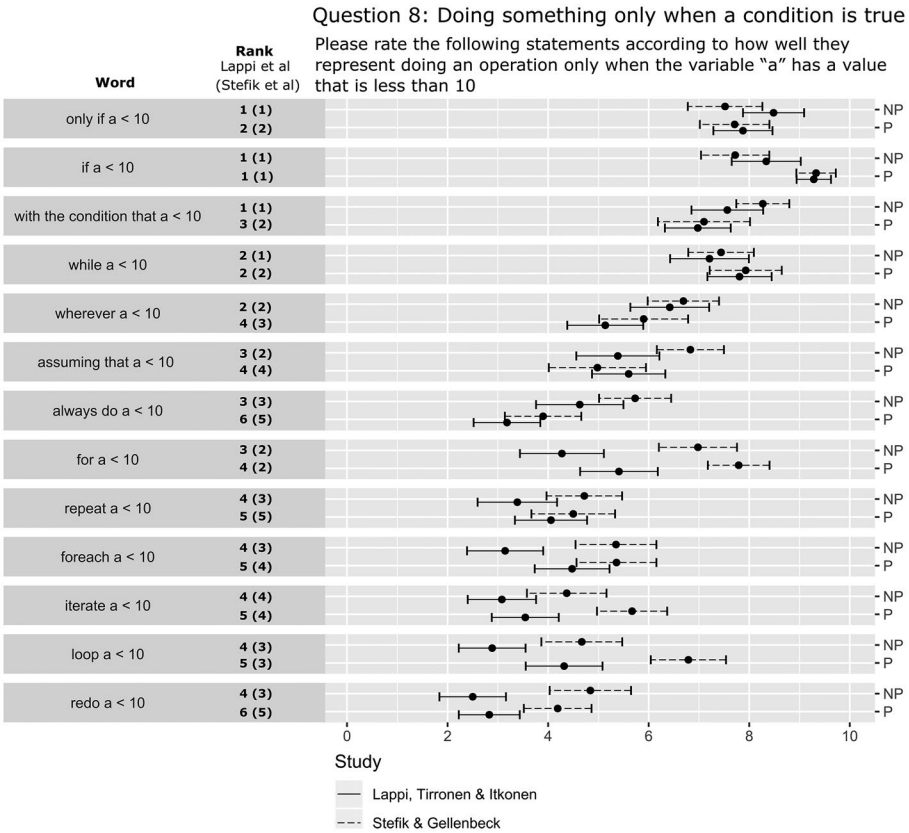

Fig. 8 A figure showing the rankings and ratings with 95% confidence intervals for Question 8 (NP, non-programmer; P, programmer)

with `repeat` and was high in the rankings of Stefik and Gellenbeck's as well, which suggests that it too might be a good alternative.

Another finding we replicated was that both programmers and non-programmers seem to prefer words instead of symbols when it comes to logic operators. Regarding the xor operator, our non-programmers also rated the word `or` highly, similarly to the non-programmers in Stefik and Gellenbeck's experiment. As they discussed, there might not be a word that well represents the concept of exclusive or operation, or at least it was not one of the options in these studies.

We also replicated the three findings which suggest that the symbols that are currently commonly used to represent a concept are intuitive to novice programmers. The symbol = was the clear first choice for assignment operator, the symbol + for concatenating text strings, and the symbols " " for distinguishing text strings.

Stefik and Gellenbeck were mixed on the results regarding the concepts of calling a method of an object and conditional statements. They stated that the options rated highly by non-programmers did not, in their opinion, clearly indicate the concepts they were supposed to represent, and that more research was needed to draw any conclusions. Our results replicated theirs so at least with these questions and options the non-programmers

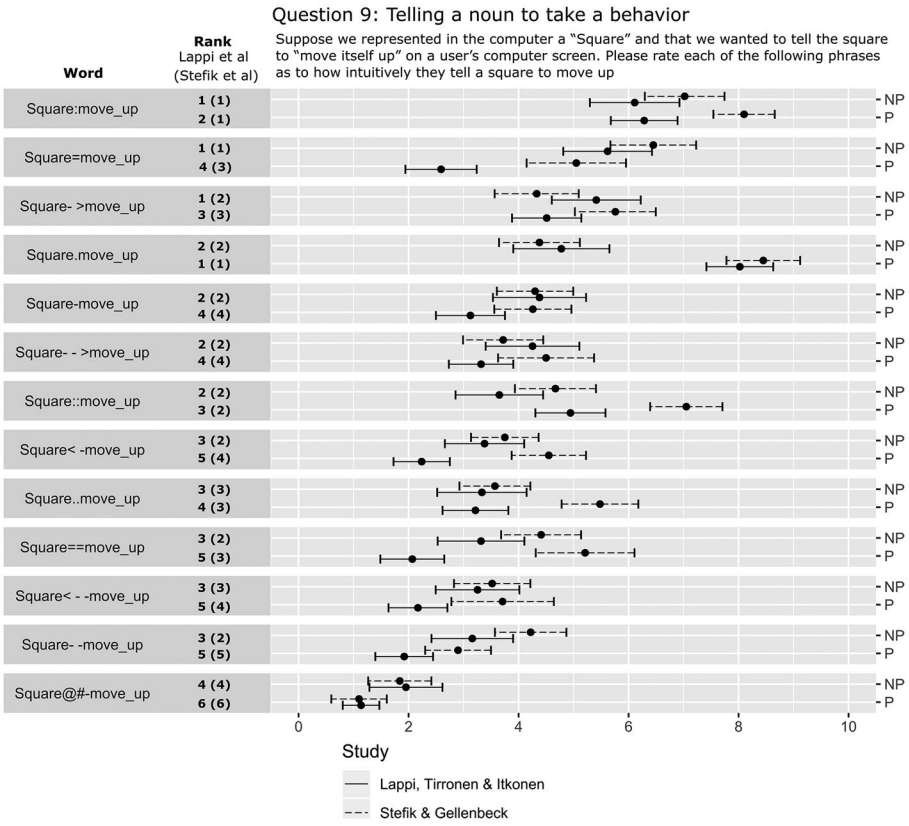

Fig. 9 A figure showing the rankings and ratings with 95% confidence intervals for Question 9 (NP, non-programmer; P, programmer)

prefer the same options. It is unclear if these options are intuitive to novice programmers or whether the concepts were not fully grasped by them. We too feel that more research is needed on these concepts.

Another question, the results of which Stefik and Gellenbeck were left mixed on, was **Question 3**, which was about functions. They were unsure whether their description of the concept ("Take a behavior") was clear and accurate enough to the participants, since the programmers in their study rated `operation` and `action` higher than the expected top answers `function` and `method`. We agree that the description is not necessarily clear enough. This question was the hardest one to translate into Finnish since there was not a way to translate the description as it was while still making sense to a Finnish speaker. After experimenting with a couple of alternatives and gathering feedback in the pilot study, we settled on a description that in our opinion described the concept better even though it was not a direct translation of the original question. Our description would translate roughly to "definition of action." Since the description was changed so much in the translation process, the results for this question cannot be directly compared.

Comparing the orders of words and symbols between the studies did not reveal any significant differences. This was probably affected by there being options in many of the questions that one would expect to get low ratings. An example of this is the option `@#-` in

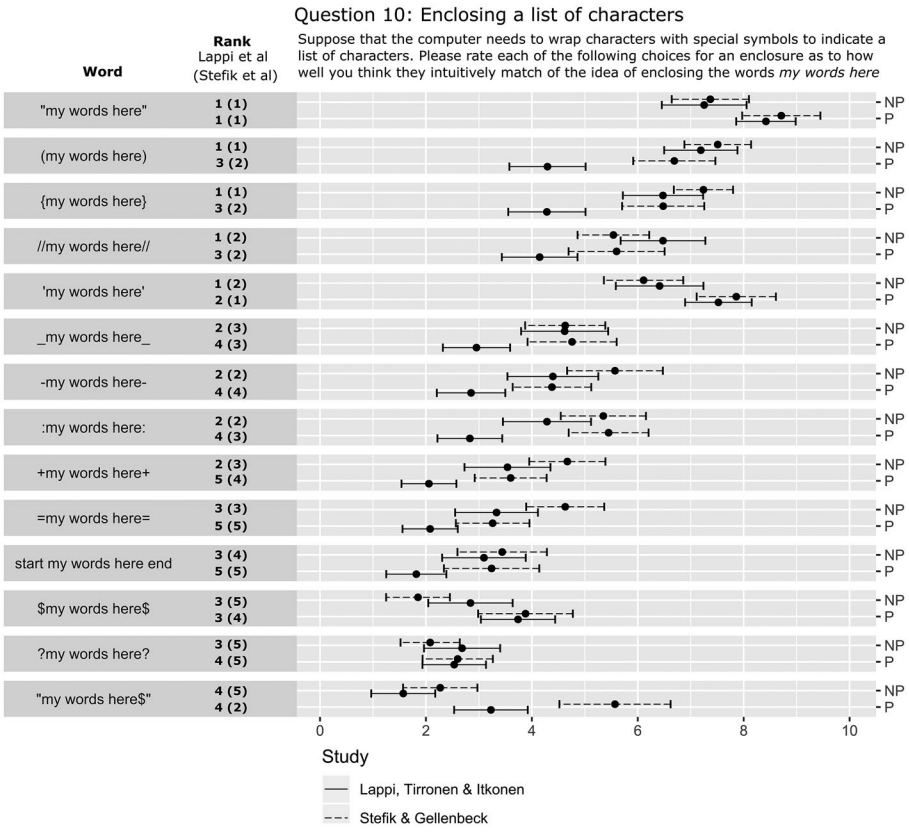

Fig. 10 A figure showing the rankings and ratings with 95% confidence intervals for Question 10 (NP, non-programmer; P, programmer)

Question 9. Since some options would probably end up at the bottom of the order every time, the orders are bound to be similar. It brings to question how were the word and symbol choices determined in the original study. Stefik and Gellenbeck mention that they conducted multiple pilots in advance to refine the testing process and probably the questions themselves, but do not make it clear how the options were chosen.

Another point we would like to make regarding the choice of words and concepts is that the scope of concepts which were chosen for the study was quite limited. Stefik and Gellenbeck touch on this in their text a little bit and mention that they wanted to focus on syntax and semantics which are common in most languages. It could be argued that the chosen concepts are common for many of the more object-oriented languages and especially those that are based on c when it comes to syntax, but not necessarily to other programming paradigms. With this said, it should be kept in mind that the original study was made in part to aid the authors in developing their own programming language, Quorum, which probably guided their choices for the concepts.

The last thing we would like to address is that the key words in a programming language are by far not the only thing that contributes to how easy it is to learn and comprehend. As Stefik and Gellenbeck also state, taking a programming language and simply exchanging

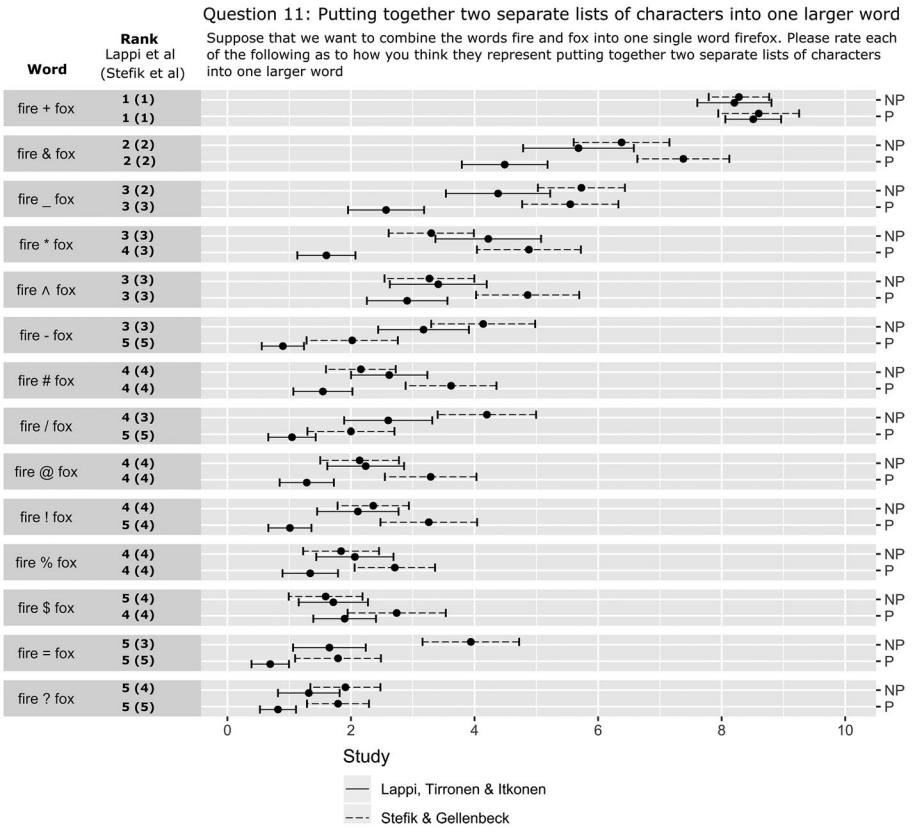

Fig. 11 A figure showing the rankings and ratings with 95% confidence intervals for Question 11 (NP, non-programmer; P, programmer)

its keywords to others is not likely to make much of a difference. As already discussed in Sect. 2.1, Stefik did go on to experiment with larger structures in the study by Stefik and Siebert (2013). There are also other things to consider in regard to key words such as how easy they are to write. Some amount of intuitiveness might be worth sacrificing for a shorter keyword, for example. We believe that intuitive keywords would support learnability and easier comprehension of programming languages but would not make a difference significant enough on their own.

6 Threats to validity

There are a number of threats to the validity of this study. With any survey, there are some threats that are always present, such as whether the wordings of the questions are biased towards some answers, and the limited possibility to conduct the survey with a random sample of the population (Wiersma, 2013). With this study, these threats also relate to the validity of this study as a replication because we translated the original questions to Finnish. While we believe that translation was the right choice in order to prevent the English skills of the participants from affecting their understanding of the question itself, it might have had

some effect on how the results of the studies can be compared to each other. There is a risk that the translation process might have slightly altered the way the questions are interpreted or benefited some answer options more than others. As discussed in Sect. 5, this was an issue with at least Question 3. This threat does not concern the results of our experiment, but whether the results can accurately be compared to those of Stefik and Gellenbeck (2011). The translations were evaluated in our pilot study and, other than Question 3, the translations are close enough to the original questions for comparisons to be made in our view.

Another change we made from the original study to our study design was that we conducted the survey online rather than on paper. Due to the anonymity of our survey, there is some uncertainty to the actual demographic of the participants in our study. Although we sent the link only to students of the University of Jyväskylä, we have no way of knowing whether it was forwarded to others. Had we done the survey on pen and paper, we would have had more control over this. Another possible threat that the online survey introduces is that it was harder for the participants to ask questions or get further clarification about the procedure, though we did provide an email address for them so they could contact us with any questions regarding the study. This threat adds some uncertainty to what kind of population the results could be generalized to, though the most relevant background information was asked in the survey. Whether the participants accurately reported their background information and answers to the research question is also a threat, but this kind of bias is a threat in any survey and not just those that are conducted online.

7 Summary and future work

In this article, we have presented an empirical study on the intuitiveness of certain word and symbol choices in programming languages. We replicated the findings of Stefik and Gellenbeck which show that while some of the words and symbols that are currently used in many programming languages are intuitive to novice programmers, others are not. We found that, at least with Finnish students that have good English skills, the participants not having English as their native language did not make many significant difference to the results.

For future work, this kind of research could be expanded almost infinitely with different concepts and words. It would, in our opinion, be beneficial to broaden the scope of the concepts examined in future studies and take into account more diverse programming paradigms. Furthermore, experimenting with larger structures in program code, as Stefik and Siebert did, might be more impactful than comparing single words or symbols. Gathering evidence from multiple angles would hopefully help future programming language designers make better informed design decisions and create more intuitive programming languages.

Regarding the effect the native language of a programmer has on their comprehension of and learning of programming languages, there is a need for more research. We did not get enough diversity in the English skills of our participants to be able to compare the effect it had on their ratings of different words though we originally planned to. Since English-based programming languages are used globally, getting more information on how they could be made more easily learnable and understood by those with only basic English skills would help, making them more accessible and might lessen the gatekeeping effect of having to learn English in order to learn programming.

Appendix. Survey questions and their translations

The questions from the original survey were translated into Finnish so that the English skills of the participants would not affect the results. The original questions and their Finnish translations are listed in Table 2.

Table 2 Finnish translations of the survey questions

Concept	Description
Doing something zero or more times	Rate how well the words describe doing something zero or more times
Jonkin asian tekeminen nolla kertaa tai useammin	Arvioi miten hyvin sanat kuvaavat jonkin asian tekemistä nolla kertaa tai useammin
Stop doing an action	Imagine that the computer is asked to stop when the variable a is less than 10. This statement may or may not be surrounded by special characters so that the computer can process the statement. Imagine when writing a program you had to guess which characters the language designer used. Rate each set of characters as to how intuitive you think they are
Toiminnan lopettaminen	Kuvittele, että tietokonetta pyydetään pysäyttämään sen hetkinen toiminta, kun muuttujan "a" arvo on pienempi kuin 10. Tämä lauseke saattaa olla ympäröitynä erikoismerkeistä koostuvalla merkkiparilla, jotta tietokone pystyy käsittelemään sen. Kuvittele, että kirjoittaessasi ohjelmaa sinun täytyy arvata mitä merkkejä ohjelmointikielen suunnittelija käytti. Arvioi miten intutiivisia merkkiparit ovat
Take a behavior	Rate how well each word corresponds to taking behavior
Toiminnan määrittely	Arvioi miten hyvin sanat kuvaavat jonkin tietynlaisen toiminnan määrittelyä
Take a behavior only when 2 conditions are true	Suppose you would go to the store only when the following two conditions are true: (1) you have money, (2) it's not raining. Rate each word choice as to how well it indicates that both conditions must be true for you to go to the store
Toiminnon suorittaminen, kun kaksi ehtoa täyttyy	Kuvittele, että menet ostoksille vain, kun seuraavat ehdot täyttyvät: (1) sinulla on rahaa (you have money), (2) ulkona ei sada (it's not raining). Arvioi miten hyvin seuraavat vaihtoehdot kuvaavat sitä, että molempien ehtojen tulee täyttyä, että menisit ostoksille
Take a behavior when at least one condition is true	Suppose you would go to the store if one of the following two conditions is true: (1) you have money, (2) it's not raining. Rate each word choice as to how well it indicates that at least one condition must be true for you to go to the store
Toiminnon suorittaminen, kun ainakin yksi ehto täyttyy	Kuvittele, että menet ostoksille vain, kun joku seuraavista ehdoista täyttyy: (1) sinulla on rahaa (you have money), (2) ulkona ei sada (it's not raining). Arvioi miten hyvin seuraavat vaihtoehdot kuvaavat sitä, että ainakin yhden ehdon tulee täyttyä, että menisit ostoksille

Table 2 (continued)

Concept	Description
Take a behavior when one condition is true, but not both conditions	Suppose you would go to the store only if one of the following conditions are true: (1) you have money, (2) it's not raining. Rate each word choice as to how well it indicates that one condition is true, but not both conditions for you to go to the store
Toiminnon suorittaminen, kun vain yksi ehto täyttyy	Kuvittele, että menet ostoksille, kun vain toinen seuraavista ehdoista täyttyy: (1) sinulla on rahaa (you have money), (2) ulkona ei sada (it's not raining). Arvioi miten hyvin seuraavat vaihtoehdot kuvaavat sitä, että yhden mutta ei molempien ehtojen tulee täyttyä, että menisit ostoksille
Assigning a value to a computer's memory	Suppose you wanted to write a mathematical expression that represented taking a number, perhaps 1024, and putting it into a location in a computer's memory in the location represented by the variable "a." Rate each symbol according to how well you think it represents assigning a value to a computer's memory
Arvon sijoittaminen tietokoneen muistiin	Kuvittele, että haluat kirjoittaa matemaattisen lausekkeen, joka esittää numeron (esimerkiksi 1024) sijoittamista tietokoneen muistiin paikkaan, jota edustaa muuttuja "a". Arvioi miten hyvin merkit kuvaavat arvon sijoittamista tietokoneen muistiin
Doing something only when a condition is true	Please rate the following statements according to how well they represent doing an operation only when the variable "a" has a value that is less than 10
Jonkin asian tekeminen vain ehdon täyttyessä	Arvioi miten hyvin seuraavat vaihtoehdot kuvaavat toiminnon suorittamista vain, kun muuttujan "a" arvo on pienempi kuin 10
Telling a noun to take a behavior	Suppose we represented in the computer a "Square" and that we wanted to tell the square to "move itself up" on a user's computer screen. Please rate each of the following phrases as to how intuitively they tell a square to move up
Kappaleen käskeminen suorittamaan jokin toiminto	Kuvittele, että esittäisimme tietokoneella neliön "Square" ja haluaisimme käskeä neliötä "liikuttamaan itseään ylöspäin" (move up) tietokoneen näytöllä. Arvioi miten intuitiivisesti seuraavat vaihtoehdot käskevät neliötä siirtymään ylöspäin
Enclosing a list of characters	Suppose that the computer needs to wrap characters with special symbols to indicate a list of characters. Please rate each of the following choices for an enclosure as to how well you think they intuitively match of the idea of enclosing the words *my words here*
Vapaan tekstin ympäröiminen	Kuvittele, että osoittaakseen joukon merkkejä olevan vapaata tekstiä, tietokoneen täytyy ympäröidä ne erikoismerkeillä. Arvioi miten intuitiivisesti seuraavat merkit sopivat ympäröimään sanoja *my words here*

Table 2 (continued)

Concept	Description
Putting together two separate lists of characters into one larger word	Suppose that we want to combine the words fire and fox into one single word firefox. Please rate each of the following as to how you think they represent putting together two separate lists of characters into one larger word
Kahden erillisen sanan yhdistäminen yhdeksi isommaksi sanaksi	Kuvittele, että halutaan yhdistää tietokoneen muistissa olevat sanat *fire* ja *fox* yhdeksi sanaksi *firefox*. Arvioi miten hyvin seuraavat vaihtoehdot kuvaavat kahden tietokoneen muistissa olevan sanan yhdistämistä yhdeksi isommaksi sanaksi

Funding Open Access funding provided by University of Jyväskylä (JYU).

Data availability The datasets generated during and/or analyzed during the current study are not publicly available due to data being possibly too identifying, but are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors declare no competing interests.

References

Aarts, A. A., Anderson, J. E., Anderson, C. J., et al. (2015). Estimating the reproducibility of psychological science. *Science*, *349*(6251). https://doi.org/10.1126/science.aac4716

Becker, B. A. (2016). An effective approach to enhancing compiler error messages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 126–131). Association for Computing Machinery, New York, NY, USA, SIGCSE '16. https://doi.org/10.1145/2839509.2844584

Begley, C. G., & Ellis, L. M. (2012). Raise standards for preclinical cancer research. *Nature, 483*(7391), 531–533. https://doi.org/10.1038/483531a

Bosse, Y., & Gerosa, M. A. (2017). Why is programming so difficult to learn? Patterns of difficulties related to programming learning mid-stage. *SIGSOFT Software Engineering Notes, 41*(6), 1–6. https://doi.org/10.1145/3011286.3011301

Brown, N. C. C., & Altadmri, A. (2017). Novice Java programming mistakes: Large-scale data vs. educator beliefs. *ACM Transactions on Computing Education*, *17*(2). https://doi.org/10.1145/2994154

Brown, N. C. C., Altadmri, A., Sentance, S., et al. (2018). Blackbox, five years on: An evaluation of a large-scale programming data collection project. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (pp. 196–204). Association for Computing Machinery, New York, NY, USA, ICER '18. https://doi.org/10.1145/3230977.3230991

Council of Europe. (2020). *Common European framework of reference for languages*. Retrieved June 11, 2020, from https://www.coe.int/en/web/common-european-framework-reference-languages/home

Dasgupta, S., & Hill, B. M. (2017). Learning to code in localized programming languages. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale* (pp. 33–39). Association for Computing Machinery, New York, NY, USA, L@S '1. https://doi.org/10.1145/3051457.3051464

Denny, P., Luxton-Reilly, A., & Carpenter, D. (2014). Enhancing syntax error messages appears ineffectual. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (pp. 273–278). Association for Computing Machinery, New York, NY, USA, ITiCSE '14. https://doi.org/10.1145/2591708.2591748

Denny, P., Luxton-Reilly, A., Tempero, E., et al. (2011). Understanding the syntax barrier for novices. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* (pp. 208–212). Association for Computing Machinery, New York, NY, USA, ITiCSE '11. https://doi.org/10.1145/1999747.1999807

Denny, P., Luxton-Reilly, A., & Tempero, E. (2012). All syntax errors are not equal. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (pp. 75–80). Association for Computing Machinery, New York, NY, USA, ITiCSE '12. https://doi.org/10.1145/2325296.2325318

Department for Education (UK). (2013). *National curriculum in England: Computing programmes of study*. Retrieved January 8, 2021, from https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study

Devanbu, P., Zimmermann, T., & Bird, C. (2016). Belief & evidence in empirical software engineering. In *Proceedings of the 38th International Conference on Software Engineering* (pp. 108–119). Association for Computing Machinery, New York, NY, USA, ICSE '16. https://doi.org/10.1145/2884781.2884812

Feijóo-García, P. G., McNamara, K., & Stuart, J. (2020). The effects of native language on block-based programming introduction: A work in progress with hispanic population. In *2020 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)* (pp 1–2). https://doi.org/10.1109/RESPECT49803.2020.9272513

Finnish National Agency for Education. (2014). *Perusopetuksen opetussuunnitelman perusteet*. https://eperusteet.opintopolku.fi/beta/#/fi/perusopetus/419550/tiedot

Guo, P. J. (2018). Non-native English speakers learning computer programming: Barriers, desires, and design opportunities. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (pp. 1–14). Association for Computing Machinery, New York, NY, USA, CHI '18. https://doi.org/10.1145/3173574.3173970

Hao, Q., Smith, I. V. D. H., Iriumi, N., et al. (2019). A systematic investigation of replications in computing education research. *ACM Transactions on Computing Education*, *19*(4). https://doi.org/10.1145/3345328

Hermans, F., Swidan, A., & Aivaloglou, E. (2018). Code phonology: An exploration into the vocalization of code. In *Proceedings of the 26th Conference on Program Comprehension* (pp. 308–311).

Hornbæk, K., Sander, S. S., Bargas-Avila, J. A., et al. (2014). Is once enough? On the extent and content of replications in human-computer interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 3523–3532). Association for Computing Machinery, New York, NY, USA, CHI '14. https://doi.org/10.1145/2556288.2557004

Kaijanaho, A. J. (2015). Evidence-based programming language design: A philosophical and methodological exploration. University of Jyväskylä, Jyväskylä. http://urn.fi/URN:ISBN:978-951-39-6388-0

Kendall, M. G. (1945). The treatment of ties in ranking problems. *Biometrika, 33*(3), 239–251. https://doi.org/10.2307/2332303

Ko, A., LaToza, T., & Burnett, M. (2015). A practical guide to controlled experiments of software engineering tools with human participants. *Empirical Software Engineering, 20*(1), 110–141. https://doi.org/10.1007/s10664-013-9279-3

Lykken, D. T. (1968). Statistical significance in psychological research. *Psychological Bulletin*, *70*(3, Pt.1), 151 – 159.

Makel, M. C., & Plucker, J. A. (2014). Facts are more important than novelty: Replication in the education sciences. *Educational Researcher, 43*(6), 304–316. https://doi.org/10.3102/0013189X14545513

Marceau, G., Fisler, K., & Krishnamurthi, S. (2011a). Measuring the effectiveness of error messages designed for novice programmers. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (pp. 499–504). Association for Computing Machinery, New York, NY, USA, SIGCSE '11. https://doi.org/10.1145/1953163.1953308

Marceau, G., Fisler, K., & Krishnamurthi, S. (2011b). Mind your language: On novices' interactions with error messages. In *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (pp. 3–18). Association for Computing Machinery, New York, NY, USA, Onward! 2011. https://doi.org/10.1145/2048237.2048241

Pears, A., Seidman, S., Malmi, L., et al. (2007). A survey of literature on the teaching of introductory programming. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer*

Science Education (pp. 204–223). Association for Computing Machinery, New York, NY, USA, ITiCSE-WGR '07. https://doi.org/10.1145/1345443.1345441

Pettit, R. S., Homer, J., & Gee, R. (2017). Do enhanced compiler error messages help students? Results inconclusive. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp 465–470). Association for Computing Machinery, New York, NY, USA, SIGCSE '17. https://doi.org/10.1145/3017680.3017768

Piech, C., & Abu-El-Haija, S. (2020). Human languages in source code: Auto-translation for localized instruction. In *Proceedings of the Seventh ACM Conference on Learning @ Scale* (pp. 167–174). Association for Computing Machinery, New York, NY, USA, L@S '20. https://doi.org/10.1145/3386527.3405916

Reestman, K., & Dorn, B. (2019). Native language's effect on Java compiler errors. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (pp. 249–257). Association for Computing Machinery, New York, NY, USA, ICER '19. https://doi.org/10.1145/3291279.3339423

Schmidt, S. (2009). Shall we really do it again? The powerful concept of replication is neglected in the social sciences. *Review of General Psychology, 13*(2), 90–100. https://doi.org/10.1037/a0015108

Siegfried, R. M., Chays, D., & Herbert, K. (2008). Will there ever be consensus on CS1? In *FECS* (pp. 18–23).

Siegmund, J., Kästner, C., Liebig, J., et al. (2014). Measuring and modeling programming experience. *Empirical Software Engineering, 19*(5), 1299–1334. https://doi.org/10.1007/s10664-013-9286-4

Stefik, A., & Gellenbeck, E. (2011). Empirical studies on programming language stimuli. *Software Quality Journal, 19*(1), 65–99. https://doi.org/10.1007/s11219-010-9106-7

Stefik, A., & Hanenberg, S. (2014). The programming language wars: Questions and responsibilities for the programming language community. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software* (pp. 283–299). Association for Computing Machinery, New York, NY, USA, Onward! 2014. https://doi.org/10.1145/2661136.2661156

Stefik, A., & Siebert, S. (2013). An empirical investigation into programming language syntax. *ACM Transactions on Computing Education (TOCE), 13*(4), 1–40. https://doi.org/10.1145/2534973

Stefik, A., Hanenberg, S., McKenney, M., et al. (2014). What is the foundation of evidence of human factors decisions in language design? An empirical study on programming language workshops. In *Proceedings of the 22nd International Conference on Program Comprehension* (pp. 223–231). Association for Computing Machinery, New York, NY, USA, ICPC 2014. https://doi.org/10.1145/2597008.2597154

Tirronen, V., Uusi-Mäkelä, S., & Isomöttönen, V. (2015). Understanding beginners' mistakes with Haskell. *Journal of Functional Programming, 25*, 30. https://doi.org/10.1017/S0956796815000179

Webropol. (2020). *Webropol survey tool website*. Retrieved June 11, 2020, from https://webropol.com/

Wiersma, W. (2013). The validity of surveys: Online and offline. *Oxford Internet Institute, 18*(3), 321–340.

Vilma Lappi was a M.Sc. student at the University of Jyväskylä, at the time of writing this article. The study described in this article was conducted as part of her master's thesis work. At present, she works at the Bank of Finland as a software developer. Her research interests include the design, usability, and accessibility of programming languages.

Ville Tirronen is a senior lecturer of Computer Science at University of Jyväskylä. His work focuses on teaching programming languages and researching related topics. He is presently studying real-world software construction as software engineer at Typeable.

Jonne Itkonen is a lecturer at the University of Jyväskylä, Finland.The focus of his teaching and research is in design, development, and tools of software.