A Frame of Mind: Frame-based vs. Text-based Editing

Neil C. C. Brown
neil.c.c.brown@kcl.ac.uk
King's College London
London, UK

Charalampos Kyfonidis
c.kyfonidis@kcl.ac.uk
King's College London
London, UK

Pierre Weill-Tessier
pierre.weill-tessier@kcl.ac.uk
King's College London
London, UK

Brett Becker
brett.becker@ucd.ie
University College Dublin
Dublin, Ireland

Joe Dillane
Joe.DILLANE@itcarlow.ie
University College Dublin
Dublin, Ireland

Michael Kölling
michael.kolling@kcl.ac.uk
King's College London
London, UK

ABSTRACT

Block-based programming has become popular for children and young school students, but at university level almost all programming is still text-based. A third intermediate option is the use of frame-based editors that combine elements of both block- and text-based systems. However, there have been few evaluations of the efficacy of frame-based editing, so its suitability for school use is uncertain. This paper describes an experiment comparing the use of frame-based and text-based editing in a UK school setting. A total of 85 teenage students from five different schools each completed three sessions of object-oriented programming tasks and a programming quiz, with each school assigned to use either a text-based editor or frame-based editor. We found no difference in understanding of object-oriented concepts between the two editors, and no difference in task completion times. This provides some evidence to suggest that frame-based editing is a viable option for use in a school setting, in place of text-based editing.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; • **General and reference** → *Evaluation.*

KEYWORDS

BlueJ, frame-based editing, Java, K-12, object orientation, Stride

ACM Reference Format:
Neil C. C. Brown, Charalampos Kyfonidis, Pierre Weill-Tessier, Brett Becker, Joe Dillane, and Michael Kölling. 2021. A Frame of Mind: Frame-based vs. Text-based Editing. In *United Kingdom and Ireland Computing Education Research conference. (UKICER '21), September 2–3, 2021, Glasgow, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3481282.3481286

UKICER '21, September 2–3, 2021, Glasgow, United Kingdom

ACM ISBN 978-1-4503-8568-8/21/09…$15.00
https://doi.org/10.1145/3481282.3481286

1 INTRODUCTION

School teachers are interested in effective methods and tools to teach programming to school students [4]. Programming environments for young novices commonly use block-based editors such as Scratch [11]. Programming environments for older learners (e.g. university students) and professionals are most commonly text-based [15, 16]. A third intermediate option is the use of frame-based program editors [8], which are designed to combine the best of both approaches and thus be suitable for high-school learners.

Frame-based program editing treats each programming construct (e.g. 'if' statements) as a single coherent entity, much like block-based program editors do. The key differences between block-based editing and frame-based editing are that:

- Block-based editors use draggable blocks at all levels of the abstract syntax tree, whereas frame-based editors only use such draggable syntax units down to the level of statements (so: functions, control flow, assignment, etc) but use text entry for entering and editing expressions;
- Frame-based editors feature a "frame cursor" that allows full keyboard navigation and manipulation of programs in addition to the mouse/touch manipulation of most block-based editors, with the aim of improving productivity and accessibility [9, 12];
- Frame-based editors lay out the code in a single linear representation similar to text-based programs, rather than the unstructured canvas of block-based editors, where independent small scripts can be placed arbitrarily in two dimensions.

The main frame-based editor available is named Stride[1], that uses a programming language semantically equivalent to Java. Stride is available in the BlueJ integrated development environment [9], along with a text-based editor for Java. This allows study of the differences between the two editors, with the language semantics and the wider development environment (BlueJ), otherwise unchanged.

To date there has only been one study [14] on the effectiveness of frame-based program editing with novices (and one with non-majors who have done some programming [8]). This means that educators and other editor designers do not have a strong body of evidence to determine how frame-based editing compares to other styles of program editing.

[1]We note that for text-based programming, a programming language (Java) can be clearly separated from its editor (IntelliJ, Notepad, etc). For frame-based and block-based programming there is generally no such distinction: like Scratch, Stride is both the name of the programming language but also the name of the editor.

In this paper we present the results of a study that compares frame-based program editing to text-based program editing. A total of 85 students aged 14-17 from five different UK schools performed the same tasks in BlueJ, using either the text-based Java programming language or the frame-based Stride programming language.

1.1 Research Questions

Our research questions (RQs) for comparing Java and Stride are:

- **RQ1 Speed**: Is there a difference in time taken to complete tasks when using Java vs Stride?
- **RQ2 Understanding**: How does the use of Java vs Stride impact understanding of object-oriented programming concepts?
- **RQ3 Ease**: What are the differences in subjective ease of use in Java vs Stride?

For all three RQs we are also interested in how these effects are moderated by student age; we believe that in a relative comparison Stride may be preferable at younger ages and Java at older ages. Franklin et al. [5] found no performance difference in programming learners between ages when faced with simple concepts, but for advanced concepts, age was found to be a factor, so we believe it could be a factor in this study of object-oriented programming.

RQ1 (speed) is relevant to the usability of the editor. If an editor can allow students to finish a task faster then it may allow a teacher to spend more time teaching concepts. But speed cannot be considered in isolation: an editor that generates the solution with a click of one button is fast, but it does nothing to educate the learner.

The goal of using any programming editor in a classroom is to facilitate the learning of programming concepts, so it is important whether students' understanding of these is (positively or negatively) impacted. The result of RQ2 (understanding) is thus very important for educators. Some programming concepts have no discernible difference between Java and Stride – for example the concept of integers or strings. We focused instead on some key differences[2] between Java and Stride that may affect understanding:

- the use of pre-provided slots to provide guidance and rigid structure for syntactic constructs (e.g. method declarations);
- depiction of general syntax (e.g. the assignment operator, which is <= in Stride); and
- display of object-oriented terminology in the editor (e.g. the labeled fields, methods sections).

See Figure 1 for a Stride screenshot showing some of these features.

The items that we felt would be impacted by these differences that we chose to measure were:

- understanding of syntax, e.g. in Java and Stride's variable declaration "vehicle car" would they be able to distinguish which is the type and which is the variable;
- assignment control flow;
- inheritance; and
- object-oriented terminology.

Another aspect of programming tools that can affect adoption success is how easy students find it to use the tools. We collected some subjective perceptions of ease for RQ3, to capture some of their experience of using tools.

[2]A more complete description of Stride can be found in a paper by Kölling et al. [8].

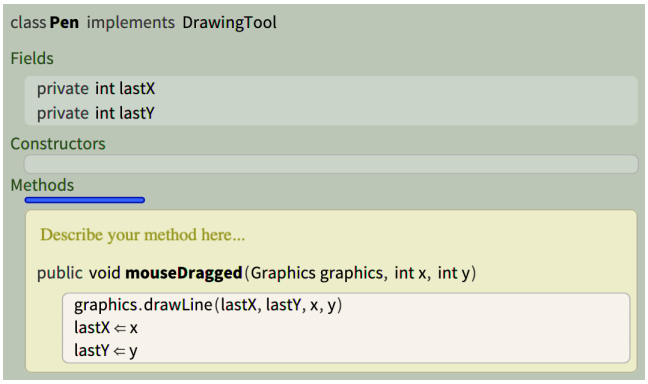

Figure 1: Screenshot of the Stride editor. The editor has specific fixed sections for fields, constructors and methods. The horizontal blue bar is the frame cursor. Although they do not have an explicitly drawn border, each field declaration and statement – like the yellow method – is itself a frame that can be dragged around or deleted as a whole item.

2 RELATED WORK

There have been many previous studies comparing block-based editing with text-based editing. Grover and Basu [6] suggested that the choice of program editor may help with understanding syntax, but may not be as effective at understanding semantics, at least without specific supporting pedagogy. Weintrop and co-authors have published several studies comparing aspects of block-based and text-based editing [17–19] and Price and Barnes [13] also published a study comparing blocks to text. This set of studies have generally found modest positive support for using block-based editors in teaching, thus implying that choice of program editor can make a difference in programming education. We have followed a similar study design to many of these studies but we are testing a different style of editor: a frame-based editor.

Alrubaye et al. [1] presented and evaluated a hybrid block-text tool that allows using a block palette to drag new code into a text program. They found advantages of this hybrid approach compared to both blocks and text, suggesting that it may be possible to provide better designs than blocks or text. Kölling et al. [7] offered a classification of the relative advantages of blocks and text, and some reasons why frame-based editing may confer the same benefits.

There have been two previous evaluations of the frame-based Stride editor. Price et al. [14] compared a Java group and Stride group from three classes in a single school as part of an outreach program. All data was gathered in a single session. Price et al. found small benefits of Stride but hypothesised that a single session was not yet enough to overcome the learning effort required to master the novel Stride interface. The present study therefore has three separate sessions and also includes more participants, from multiple schools. The other evaluation of Stride also featured a single session, examining only Stride, among university students with some programming experience [8]. This study only collected subjective rating data and comments, without any tests of understanding.

3 STUDY DESIGN

We designed a multi-session, multi-institution study to compare Java and Stride. Questionnaires were administered to students before and after three in-class sessions using either Java or Stride. To limit the workload required of the teacher, and to mirror realistic conditions, whole classes were assigned either to Java or to Stride; a *between* factor design where no participant saw both editors.

Classes were run in physical classrooms (not virtually), but due to the constraints of the COVID-19 pandemic, it was deemed infeasible to attempt in-person observations of the classes during the experiment. Data was collected remotely, via online questionnaires and passive observation facilitated by BlueJ's 'Blackbox' data collection mechanism [2]. This had the benefit of ensuring a realistic classroom setting, untainted by the presence of an observer.

3.1 Timeline

Each classroom followed this timeline:

- Session 1: Students fill in pre-questionnaire and quiz, taking 10-15 minutes. Then they do the first programming session tasks.
- Session 2: Second programming session is carried out.
- Session 3: Third programming session is carried out, then students fill in the post-questionnaire featuring the same quiz.

The three sessions were carried out in computing lessons in the standard school timetable. Due to the differing timetables for computing lessons across all the schools involved, it was infeasible to guarantee identical session lengths and identical spacing between the sessions across all classes. For example, some schools only had computing lessons once a week, and some had multiple lessons each week. We advised that the ideal interval was one week between lessons, but understood that teachers could not guarantee this. We included this spacing as a factor in the analysis.

3.2 Materials

To measure the tools' impact on understanding of programming, we needed a test that would not take too much time away from the programming sessions, and that would test object-oriented concepts and be translatable to Java and Stride. We did not find any existing tests in a catalogue in the literature [3], so we constructed our own. We used ten multiple choice questions, designed to assess understanding of object-orientation, especially in Java's semantic model. The order of the multiple choice responses was randomised within each question by the questionnaire software to avoid any positioning bias, but the questions were always presented in the same order. Students assigned to the Java condition saw the answers as Java code, while students assigned to the Stride condition saw the answers formatted as Stride code. The exact same quiz was presented to the students both in the pre- and the post-questionnaire.

We designed three sessions of programming tasks that could be used in Java or Stride within the BlueJ programming system (which we used because of its support for the Blackbox data collection project [2]). Sessions were split into several smaller tasks that feature code writing, editing, and comprehension. Both the programming quiz and the set of tasks are available for inspection in an Open Science Foundation (OSF) repository[3].

[3]Link to OSF repository: https://osf.io/t2kzg/

3.3 Ethics & Anonymisation

Ethical approval was granted by King's College London (ref: LRS-20/21-20902). Teachers were required to get permission from their school management before participating. Teachers signed an informed consent form, as did the students, and the students' parents if the student was under 16. Students were anonymous, and were given arbitrary identifiers to be able to track them during the study.

3.4 Educational Context

All of the schools who took part in were in the UK. The government guidance differs between the UK's constituent countries, but broadly schools should offer computer science to all ages, and students are usually allowed to choose which non-core subjects to take from around the age of 14 onwards. The schools were all non-selective, and were not fee-paying. Two of the five schools who completed the study were all-male schools, which is noticeably above the national average (only around 5% of non-selective non fee-paying schools in the UK are single-sex).

3.5 Recruitment & Attrition

To aid recruitment during the COVID-19 pandemic, we gave an incentive of 500 GBP for spending on the teacher's classroom (e.g. on hardware). Recruitment for the study began in autumn 2020. At this time, thirteen teachers/classes across twelve different schools were recruited to the study, and assigned a condition: six Java and seven Stride. Due to pandemic-related lockdowns, ultimately, only five classes from five different schools completed the study: three using Java and two using Stride.

4 DATA PROCESSING

4.1 Data Cleaning

The five classes were examined to determine the timings of the sessions. Activity outside class times (found in one of the five classes) was removed from the analysis. Unfortunately, some of the students' data was missing or curtailed, because their connection to the Blackbox server had not been stable, especially in one of the Stride classes. Our task time analyses were designed to be tolerant to this lack of data, and it did not affect the analyses based on questionnaires. In total, usable task data was collected from 65 students out of the 85 who completed the questionnaires.

4.2 Task Data Extraction

The source code written by the students and their interaction activity was uploaded in real-time to the Blackbox data repository. This enabled us to afterwards examine activity traces and identify when a task was completed. We chose to use a simple classification: a task could either be *incomplete* (including the case where it was not attempted), *partially correct* (e.g. they wrote the right method call for drawing an object, but used the wrong numbers as coordinates), or *correct*. If a task was correct, the first point in time where the correct solution was present in the code was marked, even if the user later refactored it or re-introduced errors. If a task had an incorrect solution (e.g. the wrong name for a method) but a future task was correct, accounting for the earlier mistake, we did not penalise the later task and marked it correct.

All participants' traces were first marked independently by two markers each (picked randomly per-participant from five markers overall). Then all markers met again to clarify rules that had repeatedly caused confusion. Markers each reviewed their own judgments and individually adjusted marks based on the new rules. The remaining disagreements were resolved by all five markers in a joint meeting. Tasks which were not completed or were completed out of order were excluded (25 out of 393 completed tasks).

It had been our hope to create tasks of roughly equal sizes. However, a Kruskal-Wallis test on task time vs task showed a significant effect of task at the $\alpha = 0.05$ level ($p < 0.001$). Therefore, we chose not to analyse task times in a single large analysis. Task times were – as expected – highly variable across students with a long tail. Informally, our markers report that some students sometimes take much longer than others on a task, because they did not understand a particular compiler error, misunderstood the instructions, or lacked the knowledge required for the question. Using task time as a linear measure would unduly penalise such issues, so we instead converted task times to per-task ranks for analysis.

5 ANALYSIS DESIGN

All tests performed in our analysis use frequentist methods with $\alpha = 0.05$ to check for statistical significance.

5.1 Preregistration & Open Science

The statistical analysis was voluntarily preregistered (albeit after data collection – but before statistical analysis was carried out). This can be found in an OSF repository[4].

The code used to perform the analysis was written in R and will be made available in the same OSF repository prior to publication. The data files (survey responses and task marking) are available on request from the lead author, as permission was not sought to make the data public.

5.2 Statistical Modelling

The data set had one chosen independent variable (CIV): Java or Stride. However it also had several extraneous independent variables (EIVs), both at individual student and classroom level (age, experience, effect of teacher), which could influence the results. This meant that a simple analysis comparing all Stride students to all Java students could be invisibly confounded by these extra factors. Thus we chose to use Hierarchical Linear Models (HLMs, a type of mixed model) for our core analyses that could take into account these extra factors.

An ideal model would feature levels for school, teacher, class, and student (if not more). In the current study we are restricted in how much of this we can model due to the available data. All of our classes come from different schools; each teacher only appears with one class. With such constraints it is impossible to separate the effects of teacher from those of class or school. Thus for practical simplification of the model, we have two levels: teacher/class/school (hereafter referred to as "classroom") and student. Given that our primary interest is in student performance, we believe it is a reasonable simplification to combine the higher-level sources of variance into one level. The available variables for model construction are as follows:

- **Age**: student age in years (per-student).
- **Language**: Java or Stride (per-classroom).
- **Spacing**: spacing between sessions (per-classroom).
- **Classroom**: classroom label (per-classroom).

5.2.1 Analyses. Our analysis involves building a model for each of the dependent variables (DVs), which are listed in subsection 5.3. The models are constructed using backward stepwise model construction [10], which starts with a full model (containing all the specified CIVs and EIVs) and then removes variables which do not make a useful difference to the model fit. The result is a model containing only statistically relevant factors, and from that we can examine the factor weights to determine the level of the effect. The full model for the analysis, in R syntax is:

$$\text{DV} \sim \textbf{Age}*\textbf{Language} + \textbf{Spacing} + (1 \mid \textbf{Classroom})$$

This means that the level of DV (a placeholder for each dependent variable) is modelled as the sum of the interaction between student age and programming language (Java or Stride) and the spacing between sessions, with a grouping factor of classroom. The former two are known as fixed effects, and classroom is a random effect. Even though we are using backward stepwise model construction, we start with a very small model to help ensure good model fit.

5.2.2 Task time encoding. As described in subsection 4.2, tasks were marked as correct or partially correct at specific points in time, which was used to calculate a time taken to complete a task.

There are two issues with task times. One is that the task times are skewed, which causes mean values to be less informative – the mean times are increased by a small number of very long task times. Thus, we converted the task times into ranks, compared across all students, with rank 1 given to the students who finished the task fastest. Partially complete tasks were ranked below completed tasks (but compared within each other for speed), and incomplete tasks were always assigned the (possibly joint) bottom rank.

The second issue with task times is comparison between different classrooms. Not every classroom spent the exact same amount of time programming in each session, and neither did all students within that classroom (some arrived late, and some showed possible data loss at the end of their sessions). Therefore it seems inappropriate to compare their task performance across all tasks without somehow accounting for total time spent.

To mitigate against this, we adopted the following algorithm. First, all tasks where less than 10% of students completed them were discounted entirely. Then, for each task that a student completed, the rank of that task was included into an average for that student. The task after the last one they completed, where it existed, was also included in the average – thus dragging down the average for a single unfinished task but no more than that. So if there were five tasks in a session where at least 10% of students completed them, a student who completed three tasks would have their automatic last place on the fourth task factored into their overall rank average, but not their automatic last place in the fifth task.

[4]Link to OSF repository: https://osf.io/t2kzg/

5.3 Hypothesis Testing

Our specific hypotheses (mapped from the respective research questions in subsection 1.1) and their statistical tests are as follows:

- Hypothesis 1: Students will differ in speed of completing the session tasks between the Stride and Java editors. This will be assessed by looking for statistical significance in the **Language** factor in the models for the DV **AvgTaskRank**, the average rank as described in the previous section.
- Hypothesis 2: Students' performance improvement on the quiz will differ between the Stride and Java editors. This will be assessed by looking for statistical significance in the **Language** factor in the model for the DV **Quiz_delta**, which is the difference in quiz score over the experiment (post minus pre).
- Hypothesis 3.1/3.2/3.3/3.4: Students' ratings will differ on the four DVs **LStudentLearn**, **LStudentWrite**, **LStudentEdit**, and **LStudentRead** – four 7-point Likert scales asking students to rate the editor on ease of learning, writing, editing and reading respectively – in the Stride editor compared to the Java editor. This will be assessed by looking for statistical significance in the **Language** factor in the model for each of these dependent variables.

We have a cross-cutting additional hypothesis: for each measure the effect of Stride may differ with age. This will be assessed by looking for statistical significance in the **Age*Language** interaction in each model.

6 RESULTS

Results are presented with coefficient estimates to 2 significant figures and p-values to 3 decimal places. Where applicable, 95% confidence intervals (CIs) are provided.

6.1 Data Overview

There were 85 students, aged 14–17 inclusive (mean 15.3) with 76 identifying as male, 6 as female and 3 selecting an "other" option. In terms of experience, 54 listed experience with a block language, 53 listed experience outside lessons (30 indicated both, 8 had neither). There were a further 67 pre-questionnaires that did not have a matching post-questionnaire and were discarded: 6 from classes that completed the experiment, and 61 from classes that did not complete the experiment. All schools that completed the study were the same type: non-selective non-fee publicly-operated.

Performance on the quiz could range from 0 to 10 (the number of correct answers to 10 multiple choice questions). The results are shown in Figure 2, with pre-results plotted against post-results. A number of possible uncertainties relating to quiz performance were:

- Existence of a ceiling or floor effect. Only one participant scored 0 in pre-test, and only one participant scored 0 in post-test. Similarly, only two participants scored 10 (both in post). The median score was 5 in pre- and 6 in post-test, suggesting that the quiz provided a good range of scores between its 0 and 10 limits.
- Significant improvement effect from sitting the same quiz in pre- and post-test. It is possible students could have remembered the questions and searched online for answers afterwards. However, with only a small difference in medians between pre- and post-questionnaire, we did not observe evidence for this. Additionally, students were not told what the correct answer was during or after the pre-questionnaire, nor were they told that they would be given the same questions again in future.
- The small difference raises another question: Is the low level of performance improvement suggestive of a lack of effort on the students' part? Chance performance on the task[5] was 2.75. A one-sample Wilcoxon signed rank test confirmed that both the pre- and post-quizzes were above chance performance ($p < 0.001$ in both cases).

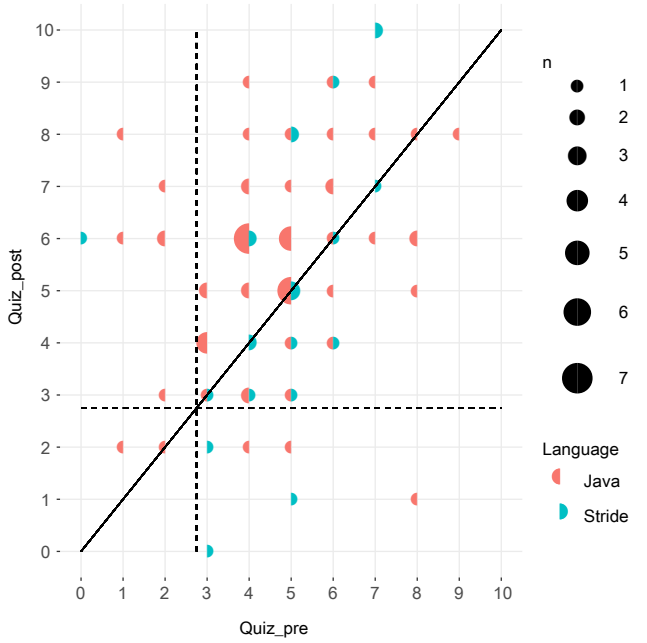

Figure 2: Scores on the pre Quiz (x-axis) and post Quiz (y-axis). Dot size is proportional to the number of students present at a given score combination, separately for Java students (left semicircles) and Stride students (right semicircles). The diagonal solid line indicates matching pre and post performance; students who performed better on post than pre are above/left of the line. The dashed vertical/horizontal lines indicate chance performance; scores to the right/above these dashed lines respectively indicate better than chance performance.

6.2 Task Completion Times

We tested hypothesis 1 by modelling the **AvgTaskRank** variable, which is calculated by comparing average ranks of the students across multiple tasks.

6.2.1 Analysis. The analysis for tasks showed only an effect of classroom, and not any other factors. There was thus no difference between Java and Stride, and no effect of **Age**.

[5] 7 questions had 4 options, 3 questions had 3 options, so chance is $7 \times \frac{1}{4} + 3 \times \frac{1}{3} = 2.75$.

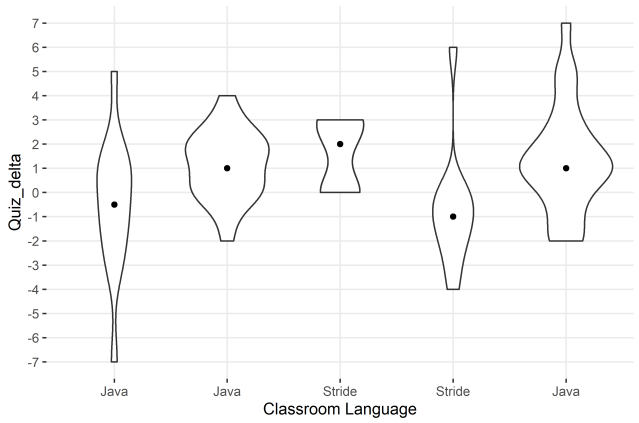

Figure 3: Violin plot of the Quiz delta (changes in performance from pre to post, positive means improved, only integer values are possible). There is one "violin" shape for each classroom, labelled by their Language condition on the x-axis. The width of each shape shows the density of delta scores (y-axis) at that point. The plotted point is the median delta for that classroom.

6.3 Quiz performance difference

We tested hypothesis 2 by modelling the **Quiz_delta** variable, which is the change in performance between the pre- and post-quiz. We used the modelling approach described in subsection 5.2.

A Kruskal-Wallis test revealed that pre-quiz performance varied by classroom ($p = 0.020$). However, this does not invalidate our analyses: the **Quiz_delta** variable is a difference in performance between pre and post that already accounts for these baseline differences (thus is akin to a "value added" score) – and classroom is included as a factor in all our models, which will help take into account any classroom-based differences in prior ability.

6.3.1 Analysis. After stepwise reduction of the core analysis model for **Quiz_delta**, the only factor that remained was classroom. This can be seen in Figure 3 which shows the five classrooms. There was a difference in performance change, but informally: since the highest and lowest classroom medians were the two Stride classrooms, there was no effect of language.

6.4 Opinions on Ease of Use

Students were asked for their opinions on whether the language they used was easy to: read, edit, write, learn. The four responses were each on a 1–7 Likert scale with higher being more positive (easier to read, etc).

6.4.1 Analysis. The models for the learn, read, and edit Likert items each showed an effect of classroom, but not of any other factor. The model for write showed an effect of language (but not classroom), with Stride scoring 0.98 points (95% CI: -1.65 to -0.31, $p = 0.005$) lower on the Likert scale.

7 LIMITATIONS

Due to the effects of COVID-19, our sample size was not as large as intended, although it still involved multiple classrooms across multiple schools.

Although our study was run over multiple sessions, there is still a confound with Stride that the students had to learn a new interface and a new language, whereas the Java students were only learning a new language.

The gender balance of our sample was 7% female. This was lower than the national average for computing in UK schools (21% female at age 16, 15% at age 18). This low figure is because two of our five classes were from all-male schools (the other schools in the original sign-ups were mixed or all-female), which skewed our gender balance. This could make our findings less representative.

8 CONCLUSIONS

We conducted a multi-institution study to compare two pedagogical programming editors: a text-based editor (Java) and a frame-based editor for a semantically identical programming language (Stride). All other factors were held constant: the development environment (BlueJ), programming language semantics, programming tasks, and outcome measures. Our study was impacted by COVID-19 but five schools completed the study: three using Java and two using Stride, with 85 students in total.

The overall finding was that there was no difference between Java and Stride across almost all our measures. Specifically, the outcomes of our research questions were:

- RQ1 (Speed): Task completion times did not differ between the two conditions.
- RQ2 (Understanding): No difference was found in understanding of concepts between the two conditions.
- RQ3 (Ease): Students rated Stride as being harder to write code in, but otherwise did not rate them differently on ease of use, reading and editing.

Almost none of the students had prior experience in Java, so they were all learning a new language. In the case of the Stride group, participants were additionally learning a new way to edit source code. Despite this additional challenge, we did not see a meaningful difference between the editors.

We suggest that this study provides evidence that both Java and Stride are equally suited for use in secondary education, especially taken in combination with the previous study [14] that showed a similar result. We believe that educators can have confidence that Stride is a viable option for high-school education, and that designers should continue to explore the rich space of interaction design that lies between block-based and text-based editing, in the knowledge that hybrid approaches can be viable for school education.

ACKNOWLEDGMENTS

We are grateful to Jane Waite for her help with recruiting participants, and we are grateful to our participants for taking part during such a tumultuous time.

REFERENCES

[1] Hussein Alrubaye, Stephanie Ludi, and Mohamed Wiem Mkaouer. 2019. Comparison of Block-Based and Hybrid-Based Environments in Transferring Programming Skills to Text-Based Environments. In *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering* (Toronto, Ontario, Canada) *(CASCON '19)*. IBM Corp., USA, 100–109.

[2] Neil C. C. Brown, Michael Kölling, Davin McCall, and Ian Utting. 2014. Blackbox: A Large Scale Repository of Novice Programmers' Activity. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA) *(SIGCSE '14)*. Association for Computing Machinery, New York, NY, USA, 223–228. https://doi.org/10.1145/2538862.2538924

[3] Adrienne Decker and Monica M. McGill. 2019. A Topical Review of Evaluation Instruments for Computing Education. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 558–564. https://doi.org/10.1145/3287324.3287393

[4] Paul Denny, Brett A. Becker, Michelle Craig, Greg Wilson, and Piotr Banaszkiewicz. 2019. Research This! Questions That Computing Educators Most Want Computing Education Researchers to Answer. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. ACM, New York, NY, USA, 259–267. https://doi.org/10.1145/3291279.3339402

[5] Diana Franklin, Gabriela Skifstad, Reiny Rolock, Isha Mehrotra, Valerie Ding, Alexandria Hansen, David Weintrop, and Danielle Harlow. 2017. Using Upper-Elementary Student Performance to Understand Conceptual Sequencing in a Blocks-Based Curriculum. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 231–236. https://doi.org/10.1145/3017680.3017760

[6] Shuchi Grover and Satabdi Basu. 2017. Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 267–272. https://doi.org/10.1145/3017680.3017723

[7] Michael Kölling, Neil C. C. Brown, and Amjad Altadmri. 2015. Frame-Based Editing: Easing the Transition from Blocks to Text-Based Programming. In *Proceedings of the Workshop in Primary and Secondary Computing Education* (London, United Kingdom) *(WiPSCE '15)*. Association for Computing Machinery, New York, NY, USA, 29–38. https://doi.org/10.1145/2818314.2818331

[8] Michael Kölling, Neil C. C. Brown, and Amjad Altadmri. 2017. Frame-Based Editing. *Visual Languages and Sentient Systems* 3 (7 2017), 40–67.

[9] Michael Kölling, Neil C. C. Brown, Hamza Hamza, and Davin McCall. 2019. Stride in BlueJ – Computing for All in an Educational IDE. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 63–69. https://doi.org/10.1145/3287324.3287462

[10] Alexandra Kuznetsova, Per B. Brockhoff, and Rune H. B. Christensen. 2017. lmerTest Package: Tests in Linear Mixed Effects Models. *Journal of Statistical Software, Articles* 82, 13 (2017), 1–26. https://doi.org/10.18637/jss.v082.i13

[11] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. *ACM Trans. Comput. Educ.* 10, 4, Article 16 (Nov. 2010), 15 pages. https://doi.org/10.1145/1868358.1868363

[12] Lauren R. Milne and Richard E. Ladner. 2018. Blocks4All: Overcoming Accessibility Barriers to Blocks Programming for Children with Visual Impairments. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3173574.3173643

[13] Thomas W. Price and Tiffany Barnes. 2015. Comparing Textual and Block Interfaces in a Novice Programming Environment. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (Omaha, Nebraska, USA) *(ICER '15)*. Association for Computing Machinery, New York, NY, USA, 91–99. https://doi.org/10.1145/2787622.2787712

[14] Thomas W. Price, Neil C. C. Brown, Dragan Lipovac, Tiffany Barnes, and Michael Kölling. 2016. Evaluation of a Frame-based Programming Editor. In *Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER '16)*. ACM, 33–42. https://doi.org/10.1145/2960310.2960319

[15] Latifa Ben Arfa Rabai, Barry Cohen, and Ali Mili. 2015. Programming Language Use in US Academia and Industry. *Informatics in Education* 14, 2 (2015), 143–160.

[16] Simon, Raina Mason, Tom Crick, James H. Davenport, and Ellen Murphy. 2018. Language Choice in Introductory Programming Courses at Australasian and UK Universities. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 852–857. https://doi.org/10.1145/3159450.3159547

[17] David Weintrop and Nathan Holbert. 2017. From Blocks to Text and Back: Programming Patterns in a Dual-Modality Environment. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 633–638. https://doi.org/10.1145/3017680.3017707

[18] David Weintrop, Heather Killen, Talal Munzar, and Baker Franke. 2019. Block-Based Comprehension: Exploring and Explaining Student Outcomes from a Read-Only Block-Based Exam. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 1218–1224. https://doi.org/10.1145/3287324.3287348

[19] David Weintrop and Uri Wilensky. 2017. Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Trans. Comput. Educ.* 18, 1, Article 3 (Oct. 2017), 25 pages. https://doi.org/10.1145/3089799