Sympathy for the (Novice) Developer: Programming Activity When Compilation Mechanism Varies

Ioannis Karvelas
ioannis.karvelas@ucdconnect.ie
University College Dublin
Dublin, Ireland

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

ABSTRACT

In this work we investigate compilation behavior and error resolution time of thousands of novice programmers using two different versions of the BlueJ pedagogical Java programming environment. The two versions feature different compilation and error message presentation mechanisms. BlueJ 3 is a click-to-compile environment with enforced first error message presentation whereas BlueJ 4 features automatic compilation and on-demand error message presentation. We provide an overview of compilation activity based on novices' programming time spent in different compilation states and also present differences in the time it takes students to reach successful compilation states in each version. We find that compilation activity differs for each version and that students reach successful compilations faster in BlueJ 4. Based on our findings, we discuss the effectiveness of each version and argue against the constant red underlining of code – an error indicator mechanism present in BlueJ 4 – as this can lead to false positive errors that may mislead novices during their programming tasks.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; CS1.

KEYWORDS

Blackbox; BlueJ; compilation; compiler error messages; CS1; environments; novice programmers; programming; programming behavior; programming process data

ACM Reference Format:

Ioannis Karvelas and Brett A. Becker. 2022. Sympathy for the (Novice) Developer: Programming Activity When Compilation Mechanism Varies. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2022)*, March 3–5, 2022, Providence, RI, USA. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3478431.3499347>

1 INTRODUCTION

Whether or not programming is inherently ‘hard’ for novices is debated [5]. Regardless, learning to program can be complicated, involving a range of languages, tools, and approaches [28]. Novice programmers learn to code using different tools ranging from

industrial-strength IDEs to specially designed pedagogical environments to command-line interfaces. However, it is not known what constitutes an effective programming environment in terms of assisting students in avoiding and overcoming syntax errors, allowing them to focus on their actual programming tasks. Overall there is a lack of work on individual IDE features and how novices interact with them. For instance, are compilation and feedback mechanisms a matter of preference and is variety actually beneficial because different students have different needs? Such questions form the foundation of the work in this study and answering them requires multifaceted approaches – possibly driven from different perspectives. This research aims to provide evidence towards answering such questions from a data-driven perspective that explores novices' programming and interaction behavior when exposed to different compilation and error presentation mechanisms within the BlueJ [26] pedagogical programming environment.

1.1 BlueJ and Blackbox

BlueJ [26] is a popular introductory programming environment for Java and Stride [24] (a fusion of text-based and block-based programming called frame-based editing [4, 34]). BlueJ is used by thousands of CS1 students. Evolving from “Blue” [25] in the late 1990s, the environment has experienced changes and updates for over two decades as BlueJ. BlueJ 3 was the major version from 2010 to 2017. It features a standard click-to-compile mechanism (compilation can happen also in some other rare situations) and if any errors are present, only the first error is displayed at the bottom of the window along with red highlighting of the offending code line. BlueJ 4 was released in 2017 and followed a more dynamic approach to compilation. In addition to standard manual compilations, if users edit their code and change lines, an automatic compilation occurs, and if errors are present the corresponding code area(s) are underlined in red. However, no text-based error messages are presented by default. To see the error message text, users must hover the cursor over the offending code or click the compile button manually. In the presence of multiple error messages, clicking compile multiple times displays messages in round-robin fashion.

Blackbox [12] is BlueJ's programming process data repository, created and maintained by the BlueJ team. It collects anonymous data from users who agree to participate the first time they launch BlueJ. It contains millions of users' programming related information. All data used in this study were retrieved from Blackbox.

1.2 Interaction with a Programming Environment

In most programming environments, there are three core aspects of interaction between the system and the user: (1) writing code,

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGCSE 2022, March 3–5, 2022, Providence, RI, USA

© 2022 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-9070-5/22/03.

<https://doi.org/10.1145/3478431.3499347>

(2) evaluating code, and (3) feedback on the input code. Usually, (1) is achieved with an editor within which the user constructs their program. Syntactically, (2) is traditionally achieved through a programming language compiler that makes multiple passes on the given code and evaluates the syntax according to the specifications of the language. This evaluation can happen either in real time, as the user writes the code and the compiler continuously parses the modified code text, or it can be designed to only occur when the user explicitly requests it from the programming environment. (3) is primarily achieved by relaying the output of the compiler to the user. Most compilers generate a list of error messages if any syntactic rules of the programming language are violated. Much research on compiler error messages has been conducted in recent years as they are highly problematic, particularly in terms of their interpretation [6, 15, 16]. In some languages, compiler error messages can even be incorrect about the cause of an error, in terms of either content or location in the source code. In many cases, the first error message produced is the most important, as correcting the error it refers to often eliminates many (or all) subsequent error messages. In fact it is relatively common practice to instruct novices to ignore all errors other than the first one [9]. Accordingly, some environments filter the message list and show the first message only, even if multiple messages are generated by the compiler. Other systems present all error messages, or display messages on-demand, only when users request them.

1.3 Prior Work

A major part of compilation-related research aims to identify common errors and the time required to fix them or the evolution of error activity over time [1, 3, 10, 11, 22, 31], as well as common fixes that students use to correct their programs [14]. Other work aims to identify “at-risk” students by looking at their programming patterns [8, 13, 17], apply programming efficiency measures to investigate the behavior of users in different regions [20, 32], error message content and presentation [21, 23], provide categorization of error messages [29], improve them [7], automate the identification and correction of syntax errors [33], and developing standards for programming process data [30].

Apart from being a tool for correcting syntax errors, it has been suggested that compilers can also serve as psychological reassurance [21, 23], and also as a means to correct mental models of the problem domain and as a dynamic task-direction mechanism [27].

Jadud [18, 19] provided a behavioral overview of students using an early version of BlueJ (these studies were published in 2005/6). One of the aspects examined was the time between compilation events. Jadud made multiple observations regarding compilation behavior. It was found that students recompile their programs very quickly if their code contains an error compared to when their code is syntactically correct. Students also tended to write a lot of code without paying attention to errors, then went back to correct any errors, a strategy that seemed ineffective.

1.4 Research Questions

The present study expands Jadud’s [18, 19] approach to observing compilations and makes the following contributions: (1) A more

contemporary perspective of novice compilation behavior in a click-to-compile environment with thousands of users as a study cohort; (2) An investigation of compilation behavior of users exposed to two different BlueJ versions that feature different compilation and error message presentation mechanisms, but preserving most other environmental factors (see BlueJ 3 and BlueJ 4 in Section 1.1); and (3) An investigation of error resolution timings of students using both environments. We regard compilation functionality and error message presentation as very important aspects of a programming environment because they are fundamental to writing and interacting with code and they are common to almost all programming environments. We seek to answer the following research questions:

- RQ1:** How do novices spend their programming time with respect to the compilation state of their source code in BlueJ 3 and BlueJ 4?
- RQ2:** How long does it take novices to resolve syntactically incorrect source code in BlueJ 3 and BlueJ 4?

2 METHODOLOGY

2.1 Identifying Matched BlueJ User Cohorts

Researchers utilizing Blackbox data often follow a holistic approach of data retrieval, including data recorded in a particular academic period or all data present in the database [3, 20]. A general observation derived from our experience with Blackbox is that a very large number of users disappear from the database very quickly, either because they stop using BlueJ (which is the most probable scenario) or they opt-out from the data collection. However, users exposed to BlueJ so briefly fail to provide insight into the evolution of programming activity such as the effects of practice over time [22]. In the present work, we included users that have spent sustained lengths of time programming in BlueJ and made significant effort to create matched cohorts in terms of programming time for the two BlueJ versions under study. The procedure we followed for the data retrieval was as follows:

Step 1: We retrieved all users who used BlueJ 3 from September 2016 to December 2016 ($n = 3673$) and all users who used BlueJ 4 from September 2018 to December 2018 ($n = 3748$)¹.

Step 2: We filtered users to retain only those for whom the entirety of their programming activity was in Java 8² (BlueJ 3: $n = 3278$, BlueJ 4: $n = 3744$).

Step 3: In [2], twenty to thirty hours is reported as a realistic number of programming time over a semester for novices. We include users with more than ten and up to fifty hours of programming time, forming two cohorts of approximately the same size (BlueJ 3: $n = 1159$, BlueJ 4: $n = 1134$). This captures novices making genuine attempts at learning programming, at a reasonably similar level of activity.

We ensured that all remaining users had only been exposed to one BlueJ version. This is because transitioning between versions is known to have effects on compilation behavior [22]. Users were

¹The dates were used as proxies for an academic semester. We chose 2016 for BlueJ 3, as it was the last year before BlueJ 4’s release (September 2017), and 2018 for BlueJ 4 as students were more likely to be using it after a year had passed since its release.

²This was done in order to ensure consistency in compiler output, as different Java versions may have different error messages and may handle code differently [12].

similarly distributed between ten and fifty hours of programming time using BlueJ for both versions.

2.2 Constructing Compilation Sequences

After identifying suitable users we retrieved programming process data related to compilation, such as timings, success indicators and trigger descriptors that we used in post-processing.

BlueJ 3 supports only manual compilations. In this sense, compilation of each source code file can only have two states based on its success: Successful (S) or Failed (F). Therefore the possible state transitions of two consecutive compilations are: FF, FS, SF, SS where the letters describe the compilation success state in chronological order from left to right in each given sequence.

BlueJ 4 supports manual compilations that are explicitly requested by the user, and automatic compilations that are triggered by edits, loading source code, using the code pad and some other rare instances. Thus, we group compilations into two different categories based on the trigger state: Manual (M) and Automatic (A) – regardless of the exact reason for triggering the automatic compile. Thus, we can identify each sequence of consecutive compilations as AA, AM, MA, MM based on their trigger state. Depending on the success of the compilations we also have the four pairs of sequential states (FF, FS, SF, SS) as was the case in BlueJ 3, but augmented this time based on the trigger of the current and previous compilation. Thus in total, there are 16 different permutations for BlueJ 4 as shown in Table 1.

Table 1: All possible consecutive compilation sequences based on success and trigger states for BlueJ 4.

		Current Trigger State			
		Auto		Manual	
Previous Trigger State	Auto	SS	SF	SS	SF
		FS	FF	FS	FF
	Manual	SS	SF	SS	SF
		FS	FF	FS	FF

In SS sequences, users modify their already syntactically correct code successfully. In SF, they modify correct source code but end up having syntax errors. In FF, they attempt to fix a syntax error with no success. Finally, in FS, they succeed in correcting a syntax error. It is possible that not all users will actually be fixing an error (in the traditional sense) in this case. For instance, one could just delete a chunk of code causing the error resulting in a successful compilation. However, we believe this to be rare, and did not observe this in any source code we manually inspected.

We labeled consecutive compilation sequences in both BlueJ versions depending on previous and current success and trigger states as described above (a BlueJ 3 compilation trigger state can only be M). This allowed us to observe how often a given compilation sequence occurs, how much time it takes novices to complete them, and how these transitions feature in the programming process.

2.3 Error Resolution

Determining a precise and meaningful process to capture the resolution time of an error on large scale programming process data is a challenge. For instance, an error may be lingering on one part

Table 2: Proportion of total programming time occupied by each consecutive compilation sequence. Occurrence (Occur.) Proportion refers to the number of a sequence’s instance over the total number of sequences.

Sequence	Trigger	Time Proportion		Occur. Proportion	
		BlueJ 3	BlueJ 4	BlueJ 3	BlueJ 4
FF	AA		26%		
FF	AM		2%		
FF	MA		2%		
FF	MM	22%	1%		
Total FF		22%	31%	39%	50%
FS	AA		6%		
FS	AM		1%		
FS	MA		0%		
FS	MM	5%	0%		
Total FS		5%	7%	13%	7%
SF	AA		7%		
SF	AM		1%		
SF	MA		5%		
SF	MM	25%	1%		
Total SF		25%	14%	9%	7%
SS	AA		19%		
SS	AM		7%		
SS	MA		14%		
SS	MM	48%	8%		
Total SS		48%	48%	39%	36%

of the source code for a long time without users addressing it. Instead, a student could be working on other code areas for some time, and after a significant edit of the source, address the errors indicated in their last compile, as it has been observed happening before [18]. Difficulties such as this are amplified in our case as there are differences in the way compilations manifest themselves in each BlueJ version. We deemed suitable measures to observe in order to answer RQ2: (1), the time elapsed between two successful compilations (TSC) of the same source code file that are not necessarily consecutive and (2), the portion of programming activity that is spent on syntactically incorrect source code by aggregating the time spent on FF and FS compilation sequences.

3 RESULTS

In this section, we present the results for each research question in dedicated subsections. For a full reference of the results, the reader can consult Table 2 which shows the programming time occupied by different compilation states and also the occurrence proportion for each possible pair of consecutive compilations. A graphical representation of time allocation can be seen as a treemap in Figure 1. Additionally, Figure 2 displays the distribution of duration of all compilation sequences and Figure 3 shows a breakdown of the timings of consecutive compilations based both on success and trigger states for BlueJ 4.

Figure 1: Programming time allocation on consecutive compilation and trigger states (top: BlueJ 3, bottom: BlueJ 4). Key to compilation states and triggers: SF = Success→Failure, etc. MA = Manual→Automatic, etc.

Figure 2: Timing distributions of different compilation sequences in both BlueJ versions (green triangle: mean, horizontal line: median).

Figure 3: Timing distributions of BlueJ 4 consecutive compilation sequences separated by success and trigger indicators (green triangle: mean, horizontal line: median).

3.1 RQ1: Compilation Behavior

Our first research question was: *How do novices spend their programming time with respect to the compilation state of their source code in BlueJ 3 and BlueJ 4?*

In terms of occurrence proportion, time allocation, and time between compilations of the four different consecutive compilation sequences (FF, FS, SF, SS), we observe the following:

Failed → Failed (FF): States of consecutively failed compilations are more frequent in BlueJ 4 (50%) than in BlueJ 3 (39%) and users also spend more time between them (31% for BlueJ 4, 22% for

BlueJ 3). Consecutive automatic compilations dominate the picture in BlueJ 4, in which case users likely often experience errors that are false positives – for instance, pausing typing before a variable name is complete can result in an unknown variable error. Each compilation sequence lasts longer in BlueJ 3 ($\mu = 78s, SD = 705$) than in BlueJ 4 ($\mu = 25s, SD = 475$).

Failed → Successful (FS): Both versions show slightly different frequencies (7% for BlueJ 4 and 13% for BlueJ 3) but similar total time spent on correcting previously unsuccessful code until correcting and successfully compiling (7% in BlueJ 4 and 5% in BlueJ 3). The explanation behind the short total time spent between these compilations and the rarity of these sequences is that most of the errors or the biggest part of the faulty code have already been addressed during the FF states and that a small effort is required in the end to make the program compile with no issues. In BlueJ 3, the time between compilations in FS sequences is slightly longer ($\mu = 51s, SD = 400$) than in BlueJ 4 ($\mu = 41s, SD = 538$).

Successful → Failed (SF): Regarding erroneous compilation states that result from previously correct code, the two versions present similar frequencies (7% for BlueJ 4, 9% for BlueJ 3) but BlueJ 4 users spend less total time on those (13%, vs BlueJ 3 at 23%). This is due to the presence of automatic compilations when students start editing their code after a previous successful compile. In BlueJ 3, they have to write a chunk of code to progress, and then hit compile by themselves to check for errors, whereas BlueJ 4 does not require users to request a check by themselves. Time between compilation for this sequence is notably higher in BlueJ 3 ($\mu = 362s, SD = 1411$) than in BlueJ 4 ($\mu = 76s, SD = 850$).

Successful → Successful (SS): In this case, the events are similarly frequent for both versions in terms of occurrences (39% for BlueJ 3 and 36% for BlueJ 4). We can see that the percentage of total time spent between successful compilations is the same for BlueJ 3 and BlueJ 4 (48% for both). Regarding time between compilations, BlueJ 3 was again longer ($\mu = 168s, SD = 809$) compared to BlueJ 4 ($\mu = 53s, SD = 720$).

Given our observations, we conclude the following for BlueJ 3: users spend almost half of their programming time between successful compilations (48%), then spend half as long (25%) working on successful source code files until they introduce an error. They spend around the same amount of time (22%) trying to repeatedly make their code work, and after they have made changes that could possibly result in a successful state, they spend a small amount of time (5%) finishing up their code and make it syntactically correct.

Given our observations, we conclude the following for BlueJ 4: users spend almost half of their time between successful compilations (48%), then spend a notable amount of time (31%) trying to repeatedly make their code work with no success. They spend a smaller amount of time (14%) working on successful source code files until they introduce an error and an even smaller amount of time (7%) finishing up their code and make it syntactically correct. Consecutive automatic compilations account for 58% of users' total programming activity. 26% of the programming activity is spent between consecutive failed compilations and 19% is spent between consecutive successful compilations.

Our results for BlueJ 3 are in accord with Jadud's work on compilation behavior conducted in 2005 [18]. His analysis revealed occurrence proportions of similar magnitude (FF: 44%, FS: 16%, SF:

10%, SS: 30%). This is reasonable since BlueJ featured a click-to-compile mechanism similar to BlueJ 3 at that time. Additionally, Jadud noted that most users tend to compile in quick succession (51% of all compilation sequences last <30 seconds and 20% last >120 seconds). Similar results are present in our case (see time between compilations – TC – in Figure 2). Jadud also observed that FF and FS last very briefly while SS and SF last longer (more than 2 minutes 60% of the time in his work). Similar behavior can also be observed in our case by looking at Figure 2 while this phenomenon is even more visible in BlueJ 4 due to automatic compilations.

A particularly interesting observation is the fact that the duration of compile sequences with a previously successful manual compilation in BlueJ 4 (SS MA, SS MM, SF MA, SF MM) are notably higher than the rest of the sequences. This can be seen both in Figure 2 where SS and SF show higher durations in both versions, but it is even more visible in BlueJ 4 when the previous compilation is both manual and successful (see Figure 3). Jadud observed that in one-third of the SS and SF sequences there were significant edits in the source files, so these durations are justified. However, this cannot be the case for our BlueJ 4 users since there would be an automatic compilation if they were editing their program. In these long intervals, BlueJ 4 users could either be contemplating on their written code, planning their next steps or simply taking a break. We can be certain that they are not writing any code as an edit would trigger a compilation.

3.2 RQ2: Error Resolution Time

Our second research question was: *How fast do novices resolve syntactically incorrect source code in BlueJ 3 and BlueJ 4?*

The time between successful compilations (TSC) was a mean of 69s ($SD=796$) for BlueJ 4 and 227s ($SD=962$) for BlueJ 3 (see Figure 2). The time between successful manual compilations in BlueJ 4 is longer, at 217s ($SD=1225$), which is still slightly shorter than in BlueJ 3. This increase in time is due to the fact that in BlueJ 4, automatic compilations can occur before students manually compile and sometimes the syntax can intermediately be correct. Measuring TSC using all compilations captures these intermediately correct compilations that happen while users edit their code and decreases the timings. In BlueJ 3, students compile only when they feel they need to, and a reasonable assumption is that compilation happens on users' intrinsic milestones (when they think they've made significant changes, either semantic or syntax-related). In contrast, by observing the total time of students spent on syntactically incorrect code (see Table 2 and Figure 1) we see that novices spend 27% of their time (BlueJ 3) and 38% of their time (BlueJ 4) in FF and FS sequences. However, if we consider the significant presence of FF AA sequences in BlueJ 4 (26%) and assuming that a lot of the instances are false positives (partial code implementation that triggered a failed automatic compilation), we can infer that students recover from errors quicker in BlueJ 4 and spend more time working on syntactically correct code.

4 THREATS TO VALIDITY

Since the present analysis involves an evaluation of compilation related events of users' programming activity in unknown contexts (a problem of most Blackbox studies [12]), we cannot be certain of

the level of users' programming experience, their intention, and the setting of their coding time (lab session, practicing at home etc.). Additionally, the programming time of users is calculated as the time difference between a session start event (when users open BlueJ) and the last event recorded for that session. However, if there is a network disruption during the session, then Blackbox stops recording further events, leading eventually to some missing data. We do not regard this as a major limitation as the nature of our analysis is to calculate percentages of programming activity and time between consecutive compilations in the same session, and users are unlikely to significantly alter their programming behaviour after an unbeknownst network disruption.

5 DISCUSSION

In this paper we investigated the compilation behavior of thousands of novice programmers that have spent substantial time using one of two BlueJ versions – BlueJ 3 which features click-to-compile functionality and enforces that only the first error message is shown, and BlueJ 4 which features automatic compilation while retaining manual functionality and on-demand error message presentation, meaning users see error messages only when they want to. Although this study uses BlueJ as the experimental context, our results can be generalized outside of this context to a degree, since compilation and error message presentation are fundamental aspects of most programming environments.

Users in both BlueJ versions show a similar pattern regarding the programming time allocated in different consecutive compilation states: Around half of the time is spent between successful compilations. A very notable amount of time is spent between failed compilations and between a previous successful and a current failed compilation. At last, a small amount of time is spent between a previously failed compilation and a current successful one.

The major differences in users' compilation behavior between the two versions are: (1) users spend more total time between consecutive failed compilations in BlueJ 4, but the majority is due to automatic compilation capturing errors on-the-fly on often incomplete code, (2) users spend less total time working on syntactically correct code until they introduce an error for the first time in BlueJ 4, and (3) users spend slightly more total time on fixing an immediately previous syntactically incorrect code in BlueJ 4.

In general, users tend to compile again after a prolonged amount of time if their previous compile was successful when compared to instances where their previous compile resulted in errors, which is in accord with previous work [18]. A possible justification for this is that successful compiles serve as milestones for users, after which, they are contemplating on their written code, planning their next steps or simply taking a break. This is even more evident if we consider the fact that in BlueJ 4 users take notably more time in instances where their previous manual compile was successful than in the rest of the instances. Comparing the two versions in terms of different compilation sequences' duration revealed that in BlueJ 4, compilations occur much more often than in BlueJ 3, due to automatic compilations triggered by users editing their code. The order of different compilation sequences (FF, FS, SF, SS) based on their duration remains the same in both versions, which is puzzling

since automatic compilation offers a more fast-paced compilation mechanism than the standard click-to-compile.

Regarding error resolution, after observing the time between successful compilations and the time spent on syntactically incorrect code, we conclude that BlueJ 4 users overcome errors faster than BlueJ 3 users. Even after considering only manual compilations in BlueJ 4, error resolution timings were slightly shorter than in BlueJ 3. BlueJ 4 can provide early feedback (generated by the automatic compilations) that users may take into account and resolve trivial errors as they are editing instead of correcting them after a manual compile which is the only option in BlueJ 3.

Comparing the two versions in terms of compilation habits of users presents a significant complication: the fact that automatic compilation is present in BlueJ 4 inevitably leads to errors that users do not intend, or do not care about – a partially-completed variable name causing an error before typing is (correctly) finished is merely a distraction. Imagine the user writing code and when changing lines seeing the code underlined in red in several parts, due to a statement that has not been finished yet. The user can choose to ignore it, however that requires a higher level of understanding of compilation that novices are unlikely to possess. Alternatively, the user may address the offending code area indicated within the editor in which case, the compiler can either be right in identifying the cause of the error, but it can also be wrong and as a result the user will end up altering code that is not problematic. In terms of feedback, the environment is bound to perform unnecessary compilations leading to false positive errors that are meaningless, but users do experience them.

The aim of automatic compilation and on-demand feedback presentation in BlueJ 4 is to provide early access to error messages generated by the compiler in order to dynamically assist novices, instead of having them manually request feedback at the end of a significant edit. In this sense, our results suggest that automatic compilation succeeds in as much as we observe a shorter time between successful compilations in BlueJ 4 compared to BlueJ 3. The fact that users have access to all error messages generated instead of only the first one may also contribute to this. However, we want to note that a sizable portion of compilation sequences are triggered by failed automatic compiles that induce false positives that users may not care about – which is the good scenario – or may get confused about – a bad scenario. Some users, especially absolute beginners, may interpret these false positives as actual errors and may be negatively impacted psychologically by the constant red underlining of offending (in the compiler's opinion) code areas. A possible augmentation to BlueJ 4's dynamic approach to compilation and error message presentation could be a "smart" detection system that identifies suitable times for compilation, maximizing the impact of the environment's feedback while users edit their code, but without overwhelming them with red underlines every time the compiler detects syntax violations on partially implemented code.

In future work, we aim to perform additional analyses to reveal in which compilation sequences users actually see the errors generated in BlueJ 4, and investigate how users respond to the two BlueJ versions in terms of edits on their code after being exposed to feedback. Our aim is to determine exactly the effects that compilation functionality and error message presentation have on novice programmer behaviour, and which assist students the most.

ACKNOWLEDGMENTS

The work presented in this paper is funded by the Irish Research Council under grant agreement GOIPG/2020/1660.

REFERENCES

- [1] Alireza Ahadi, Raymond Lister, Shahil Lal, and Arto Hellas. 2018. Learning Programming, Syntax Errors and Institution-Specific Factors. In *Proceedings of the 20th Australasian Computing Education Conference* (Brisbane, Queensland, Australia) (ACE '18). ACM, 90–96. <https://doi.org/10.1145/3160489.3160490>
- [2] Joe Michael Allen, Frank Vahid, Alex Edgcomb, Kelly Downey, and Kris Miller. 2019. An Analysis of Using Many Small Programs in CS1. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) (SIGCSE '19). ACM, 585–591. <https://doi.org/10.1145/3287324.3287466>
- [3] Amjad Altadmri and Neil C. C. Brown. 2015. 37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Kansas City, Missouri, USA) (SIGCSE '15). ACM, 522–527. <https://doi.org/10.1145/2676723.2677258>
- [4] Amjad Altadmri, Michael Kölling, and Neil C. C. Brown. 2016. The Cost of Syntax and How to Avoid It: Text versus Frame-Based Editing. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. 748–753. <https://doi.org/10.1109/COMPSAC.2016.204>
- [5] Brett A. Becker. 2021. What Does Saying That 'Programming Is Hard' Really Say, and About Whom? *Commun. ACM* 64, 8 (jul 2021), 27–29. <https://doi.org/10.1145/3469115>
- [6] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L. Pearce, and James Prather. 2019. Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland UK) (ITICSE-WGR '19). ACM, 177–210. <https://doi.org/10.1145/3344429.3372508>
- [7] Brett A. Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin, and Catherine Mooney. 2016. Effective Compiler Error Message Enhancement for Novice Programming Students. *Computer Science Education* 26, 2-3 (2016), 148–175. <https://doi.org/10.1080/08993408.2016.1225464>
- [8] Brett A. Becker and Catherine Mooney. 2016. Categorizing Compiler Error Messages with Principal Component Analysis. In *12th China-Europe International Symposium on Software Engineering Education (CESEE 2016)*, Shenyang, China, 28-29 May 2016.
- [9] Brett A. Becker, Cormac Murray, Tianyi Tao, Changheng Song, Robert McCartney, and Kate Sanders. 2018. Fix the First, Ignore the Rest: Dealing with Multiple Compiler Error Messages. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) (SIGCSE '18). ACM, 634–639. <https://doi.org/10.1145/3159450.3159453>
- [10] Neil C. C. Brown and Amjad Altadmri. 2014. Investigating Novice Programming Mistakes: Educator Beliefs vs. Student Data. In *Proceedings of the 10th Annual Conference on International Computing Education Research* (Glasgow, Scotland, United Kingdom) (ICER '14). ACM, 43–50. <https://doi.org/10.1145/2632320.2632343>
- [11] Neil C. C. Brown and Amjad Altadmri. 2017. Novice Java Programming Mistakes: Large-Scale Data vs. Educator Beliefs. *ACM Trans. Comput. Educ.* 17, 2, Article 7 (may 2017), 21 pages. <https://doi.org/10.1145/2994154>
- [12] 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). ACM, 223–228. <https://doi.org/10.1145/2538862.2538924>
- [13] Karo Castro-Wunsch, Alireza Ahadi, and Andrew Petersen. 2017. Evaluating Neural Networks as a Method for Identifying Students in Need of Assistance. In *Proceedings of the 48th ACM Technical Symposium on Computer Science Education* (Seattle, Washington, USA) (SIGCSE '17). ACM, 111–116. <https://doi.org/10.1145/3017680.3017792>
- [14] Draylson Micael de Souza, Michael Kölling, and Ellen Francine Barbosa. 2017. Most Common Fixes Students Use to Improve the Correctness of Their Programs. In *2017 IEEE Frontiers in Education Conference (FIE)*. 1–9. <https://doi.org/10.1109/FIE.2017.8190524>
- [15] Paul Denny, James Prather, and Brett A. Becker. 2020. Error Message Readability and Novice Debugging Performance. In *Proceedings of the 25th ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) (ITICSE '20). ACM, 480–486. <https://doi.org/10.1145/3341525.3387384>
- [16] Paul Denny, James Prather, Brett A. Becker, Catherine Mooney, John Homer, Zachary C. Albrecht, and Garrett B. Powell. 2021. On Designing Programming Error Messages for Novices: Readability and Its Constituent Factors. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, Article 55, 15 pages. <https://doi.org/10.1145/3411764.3445696>
- [17] Gregory Dyke. 2011. Which Aspects of Novice Programmers' Usage of an IDE Predict Learning Outcomes. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) (SIGCSE '11). ACM, 505–510. <https://doi.org/10.1145/1953163.1953309>
- [18] Matthew C. Judud. 2005. A First Look at Novice Compilation Behaviour Using BlueJ. *Computer Science Education* 15, 1 (2005), 25–40. <https://doi.org/10.1080/08993400500056530>
- [19] Matthew C. Judud. 2006. Methods and Tools for Exploring Novice Compilation Behaviour. In *Proceedings of the 2nd International Workshop on Computing Education Research* (Canterbury, United Kingdom) (ICER '06). ACM, 73–84. <https://doi.org/10.1145/1151588.1151600>
- [20] Matthew C. Judud and Brian Dorn. 2015. Aggregate Compilation Behavior: Findings and Implications from 27,698 Users. In *Proceedings of the 11th Annual International Conference on International Computing Education Research* (Omaha, Nebraska, USA) (ICER '15). ACM, 131–139. <https://doi.org/10.1145/2787622.2787718>
- [21] Ioannis Karvelas, Joe Dillane, and Brett A. Becker. 2020. Compiler Error Messages: Their Content and Accessibility in Novice Programming Environments. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) (SIGCSE '20). ACM, 1310. <https://doi.org/10.1145/3328778.3372617>
- [22] Ioannis Karvelas, Joe Dillane, and Brett A. Becker. 2021. Exploring Novice Programming Behavior over Time. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 2* (Virtual Event, Germany) (ITICSE '21). ACM, 651. <https://doi.org/10.1145/3456565.3460063>
- [23] Ioannis Karvelas, Annie Li, and Brett A. Becker. 2020. The Effects of Compilation Mechanisms and Error Message Presentation on Novice Programmer Behavior. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) (SIGCSE '20). ACM, 759–765. <https://doi.org/10.1145/3328778.3366882>
- [24] 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). ACM, 63–69. <https://doi.org/10.1145/3287324.3287462>
- [25] Michael Kölling and John Rosenberg. 1996. Blue—a Language for Teaching Object-Oriented Programming. In *Proceedings of the 27th ACM Technical Symposium on Computer Science Education* (Philadelphia, Pennsylvania, USA) (SIGCSE '96). ACM, 190–194. <https://doi.org/10.1145/236452.236537>
- [26] Michael Kölling, Bruce Quig, Andrew Patterson, and John Rosenberg. 2003. The BlueJ System and its Pedagogy. *Computer Science Education* 13, 4 (2003), 249–268. <https://doi.org/10.1076/cs.ed.13.4.249.17496>
- [27] Justin Lubin and Sarah E. Chasins. 2021. How Statically-Typed Functional Programmers Write Code. *Proc. ACM Program. Lang.* 5, OOPSLA, Article 155 (oct 2021), 30 pages. <https://doi.org/10.1145/3485532>
- [28] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory Programming: A Systematic Literature Review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) (ITICSE 2018 Companion). ACM, 55–106. <https://doi.org/10.1145/3293881.3295779>
- [29] Davin McCall and Michael Kölling. 2014. Meaningful Categorisation of Novice Programmer Errors. In *2014 IEEE Frontiers in Education Conference (FIE)*. 1–8. <https://doi.org/10.1109/FIE.2014.7044420>
- [30] Thomas W. Price, David Hovemeyer, Kelly Rivers, Ge Gao, Austin Cory Bart, Ayaan M. Kazerouni, Brett A. Becker, Andrew Petersen, Luke Gusukuma, Stephen H. Edwards, and David Babcock. 2020. ProgSnap2: A Flexible Format for Programming Process Data. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) (ITICSE '20). ACM, 356–362. <https://doi.org/10.1145/3341525.3387373>
- [31] David Pritchard. 2015. Frequency Distribution of Error Messages. In *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools* (Pittsburgh, PA, USA) (PLATEAU 2015). ACM, 1–8. <https://doi.org/10.1145/2846680.2846681>
- [32] Brad Richards and Ayse Hunt. 2018. Investigating the Applicability of the Normalized Programming State Model to BlueJ Programmers. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research* (Koli, Finland) (Koli Calling '18). ACM, Article 10, 10 pages. <https://doi.org/10.1145/3279720.3279731>
- [33] Eddie Antonio Santos, Joshua Charles Campbell, Dhvani Patel, Abram Hindle, and José Nelson Amaral. 2018. Syntax and Sensibility: Using Language Models to Detect and Correct Syntax Errors. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 311–322. <https://doi.org/10.1109/SANER.2018.8330219>
- [34] David Weintrop and Nathan Holbert. 2017. From Blocks to Text and Back: Programming Patterns in a Dual-Modality Environment. In *Proceedings of the 48th ACM Technical Symposium on Computer Science Education* (Seattle, Washington, USA) (SIGCSE '17). ACM, 633–638. <https://doi.org/10.1145/3017680.3017707>