Compile Much? A Closer Look at the Programming Behavior of Novices in Different Compilation and Error Message Presentation Contexts

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

Joe Dillane
joe.dillane@itcarlow.ie
University College Dublin
Dublin, Ireland

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

ABSTRACT

Learning to program is a process that relies on learning theoretical fundamentals as well as practice, and almost always involves some type of programming environment. In order to build effective environments that support good learning for novices, it is important to explore the interaction between novices and these environments. A variety of feedback mechanisms are employed by various environments in use in classrooms today. Some, such as text-based error messages are common to almost all. Other interaction mechanisms, such as invoking the compiler, can vary rather drastically.

In this study we investigate the difference between BlueJ 3 and BlueJ 4, two versions of a pedagogical programming environment that offer different mechanisms for compilation and error message presentation. We find evidence that these differences provide users with fundamentally different programming experiences. Specifically, we find that programming process data produced by BlueJ 3 users follow a very deterministic distribution compared to BlueJ 4. Based on this, we present a formula that describes the behaviour of BlueJ 3 users in terms of compilation and error metrics. Conversely, we demonstrate that BlueJ 4 allows users to interact more freely in terms of compilation mechanism as well as how they receive error messages, and their quantity. Which is more beneficial to novices however, is an open question.

CCS CONCEPTS

• **Social and professional topics → Computing education**; *K-12 education*; **CS1**.

KEYWORDS

Blackbox; BlueJ; compiler error messages; CS1; editors; educational data mining; IDE; novice programmers; programming; programming environments; programming process data; tools

ACM Reference Format:
Ioannis Karvelas, Joe Dillane, and Brett A. Becker. 2020. Compile Much? A Closer Look at the Programming Behavior of Novices in Different Compilation and Error Message Presentation Contexts. In *United Kingdom & Ireland Computing Education Research conference. (UKICER '20), September 3–4, 2020, Glasgow, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3416465.3416471

1 INTRODUCTION

A central focus of computing education research is the teaching and learning of a first programming language [3]. It is also common for educators to claim that "programming is hard" [26, 31]. However, this has been recently called into question [30]. Regardless, parts of this claim could be attributed to the tools and techniques used for teaching programming, rather than the content itself. It was stated in 1977 that computer programming could be made easier for novices [29]. We argue that this is still true today, and novices need to be facilitated with realistic expectations and suitable environments [25]. Novices in CS1 contexts are usually enrolled in introductory programming courses that involve teaching fundamental concepts and practicing through developing solutions for exercises using a programming environment.

Some educators encourage students to use a specific programming environment – often (but not always) a pedagogical one. These are often devoid of distracting and complex features meant for professionals, should theoretically improve the learning experience. In cases where students are free to choose the environment themselves, often industry-grade Integrated Programming Environments (IDEs) such as Eclipse, NetBeans and IntelliJ are selected, as students believe that these will better prepare them for their careers in industry. Although we agree that familiarisation with these environments can be beneficial, it is not clear if working with them at early stages is optimal. The main focus of most introductory programming courses does not require industry-grade IDEs. Spending time on learning the intricacies of such IDEs takes time that could be better utilized focusing on the actual programming tasks. In addition, the presence of multiple tools and features in the environment can potentially overload cognitive channels that could be utilized more constructively [15].

In order to build effective and evidence-based programming tools, we must first explore the difference in the interaction between novices and the feedback mechanisms that programming environments offer from the programmer's point of view [32, 33]. Although research in the field of Human-Computer Interaction has established principles for designing user interfaces based on findings on human cognition, many of these principles originate from experience and sometimes advice from experts and practitioners in the subject domain [16]. This makes it all the more important to examine feedback mechanisms individually and thoroughly by studying their effect on novice programmers.

This study builds on prior work [19–21] that investigates the effect of exposure to different compilation mechanisms as well as different error message presentation in the BlueJ [24] pedagogical programming environment. The current study further explores the effects of the fundamentally different feedback mechanisms that two BlueJ versions (version 3 and version 4) offer to novices.

BlueJ 3 features a standard "click-to-compile" mechanism that only presents the first error message to users. Users can only evaluate their source code if they specifically request it from the environment by clicking the compile button. BlueJ 4, in addition to the option of manual compilation, features automatic background compilation when users perform certain actions, such as changing to a different line in the source, loading the source code in the environment, etc. However, only error indicators are displayed in the corresponding source code areas by default; users have to hover over the indicators or click the compile button in order to see the error message(s) which pop up as text boxes next to the offending code. Pressing the compile button multiple times sequentially cycles over all the errors present in the source code (round-robin).

We select BlueJ 3 and BlueJ 4 for two primary reasons. First, amongst the largest changes between the two versions are the compilation and error message presentation mechanisms. Much of the other features remain the same. This allows us to reasonably conclude that behavioral differences that arise are due to these factors. Second, programming process data from both versions are automatically captured by the Blackbox data collection project [5].

1.1 Research Questions

In this study, a more detailed look at the distribution of users' activity is presented, further supporting previous findings, and provides insights about the way users interact with each BlueJ version. Furthermore, we present a model for describing the distribution of BlueJ 3 interaction regarding Displayed Compiler Error Messages per Hour (DCEMpH), Compilations per Hour (CpH), and Percentage of Successful Compiles (PSC), defining a relationship between them. Following this, possible implications of using each version are discussed and further investigation is proposed. This work seeks to answer the following research questions:

RQ1: How do different programming environments affect user behaviour in terms of interaction with error messages and compilation mechanisms?

RQ2: Can user behaviour profiles be modelled mathematically (or otherwise) for different programming environments?

2 RELATED WORK

Research on tools aimed at assisting novices typically involve the development of full or prototype environments [11], intelligent tutoring systems [7–9], new languages [12–14, 23], and combinations of the above. Sometimes evaluation of these tools is based on empirical evidence, but not always. Recently, it was stated that evaluation of programming tools should involve a wide breadth of approaches [10], urging for even more focus on exploring the effects that different mechanisms have on novices.

2.1 Compilation Mechanisms

Little work has been performed on the effects of different compilation mechanisms on novices. This is surprising given the wide array of mechanisms employed by modern environments. Snap! [17], Scratch [28] and other block-based environments rarely have a dedicated compilation mechanism, but have "run", "go" or similar buttons which in effect compiles then executes code. Environments for text-based languages range from similar compile-and-execute buttons as in Visual Studio[1] to compile-on-save features as in Netbeans[2]. Other environments have separate compile and execute buttons. Several IDEs, such as Eclipse, incrementally compile as code is edited[3]. BlueJ 3 was a manual click-to-compile editor, but BlueJ 4 introduced incremental compilation (automatic error checking[4]), while retaining the manual compile button allowing both automatic and manual compilation mechanisms.

2.2 Error Messages

Work on the effects of error message content and presentation is more extensive than that for compilation mechanisms. Becker et al. published an ITiCSE working group report detailing the history of compiler error message work as well as a set of literature- and evidence-backed guidelines for error message design [2]. Specific to the present study is the presentation of Java error messages in BlueJ. Java messages are notoriously difficult to understand/interpret as noted by McCall & Kölling as: 1) different logical errors frequently result in the same diagnostic message, and 2) the same logical error can – depending on context – produce different messages [27]. It is important to note that although programming process data such as errors have long been used to predict performance (for example the error quotient [18] and repeated error density [1]), such efforts are affected by context and replication studies show varying results [35].

As stated in Section 1, BlueJ 3 presents only one error message at a time [34]. This was a design decision that was carried over from Blue, described in detail in [22]. This has two immediate benefits: simplicity, and avoiding multiple error messages and issues they bring (such as cascading and spurious messages) [4]. BlueJ 4 on the other hand, can present multiple error messages, but not all at once (thus, still avoiding issues such as cascading and likely most spurious messages). In BlueJ 4, only error indicators (squiggly red underlines) are displayed by default in each compilation, and clicking the compile button (without altering the source) causes the next message to be displayed, and so on. If the final message is being displayed and the compile button is clicked again, the first message is displayed (again), and so on. In addition, hovering over an error indicator in the source will present a pop-up with an error message (which is occasionally truncated).

2.3 Comparing BlueJ Versions

In [21], programming activity of users using BlueJ 3 and Blue 4 was analyzed to identify potential differences in the frequency of manual compilations, error messages as well as percentage of success of manual compilations. To explore these, three metrics were created and calculated for each user: Displayed Compiler Error Messages per Hour (DCEMpH), manual Compilations per Hour (CpH), as well as Percentage of Successful manual Compilations (PSC). Statistical tests showed significant differences between the two BlueJ versions.

[1]docs.microsoft.com/en-us/visualstudio/windows/?view=vs-2017
[2]netbeans.org/kb/docs/java/quickstart.html#run
[3]www.codejava.net/ides/eclipse/why-does-eclipse-use-its-own-java-compiler
[4] bluej.org/versions.html

In particular, in BlueJ 4 users seem to be exposed to more compiler error messages, and compile manually less frequently – but when they do, they seem to have higher rates of compilation success.

3 DATA

The data that were used in this study were retrieved from the Blackbox dataset [6]. Blackbox records programming process data from BlueJ sessions if users agree to contribute the first time they launch BlueJ. All programming activity is recorded in the form of programming events with different labels depending on the event type. The following sections outline the process of retrieving and processing the data, step by step.

3.1 User Cohorts

The initial step was to identify all users who used BlueJ 3.x or BlueJ 4.x exclusively. In other words, all users were selected except those who used both versions. From all the users retrieved, 3500 were picked randomly from each of the two sets (BlueJ 3 and BlueJ 4).

3.2 Compiler Version

As one of the primary aspects of analysis in this work is compiler error messages, we restricted the data to events associated with Java version 8. This is because different Java versions are known to produce variations in numbers and content of messages [6]. This is in accord with the prior work [21], ensuring consistency in methodology.

4 METHODOLOGY

4.1 Data Filtering

For consistency with the methodology discussed in [21], the events were also reduced in post-processing to only include activity between the 14th of January 2016 and the 24th of May 2019.

4.2 Programming Time

The first information constructed from the programming activity was how much time (in hours) users spend on BlueJ. This is essential as programming time (H) is used as a component for the metrics that are examined in this study. However, Blackbox is known to have missing events for a large number of programming sessions. This can be caused by disruptions of the Internet connection or by system interruptions from the user's end. Thus, to calculate the programming time of users, the last event recorded in every session is assumed to be the terminating event for that session regardless of it being the "true" final event (e.g. a Blackbox end session event). The difference in each session's first and final events were calculated for each user. This is a known Blackbox issue, and is noted as a potential threat to validity in Section 6.

4.3 Metrics

For every user, the programming time spent on BlueJ was used to construct the Displayed Compiler Error Messages per Hour (DCEMpH) and manual Compilations per Hour (CpH) metrics. Percentage of Successful manual Compilations (PSC) was also constructed (all metrics were calculated based on the methodology presented in [21]). Briefly, the construction of the metrics involved counting the numbers of displayed compiler error messages, manual compilations and successful manual compilations and dividing them appropriately.

4.4 Removing Outliers

As BlueJ is a free and widely used tool, it is expected that substantial noise will be present in the data. Thus, a significant portion of the analysis is spent on identifying and removing activity that seems irregular. First, BlueJ 3 users that exceeded in programming time the maximum programming time of users in BlueJ 4 were removed from the data in order to eliminate a handful of users who had unrealistic amounts of time spent on BlueJ 3 (i.e. tens of thousands of hours). Additionally, users whose programming time and DCEMpH exceeded a threshold three standard deviations above their means were also removed. The thresholds for DCEMpH and H were distinctly calculated.

5 RESULTS

The final groups after data pre-processing included 727 users in BlueJ 3 and 536 users in BlueJ 4. We discuss the results presented here in Section 7.

5.1 RQ1: Distributions

Research question 1 was: *How do different programming environments affect user behaviour in terms of interaction with error messages and compilation mechanisms?*

Displayed Compiler Error Messages per Hour (DCEMpH) are higher for BlueJ 4. Manual Compilations per Hour (CpH) are higher in BlueJ 3. The Percentage of Successful manual Compilations (PSC) is higher in BlueJ 4. Table 1 shows a detailed description of the distributions including exact values for the above. While there are notable variations in the data, the differences at most quartiles of the distributions are consistent with the differences in the means.

Figures 1-3 show the relationship for all three possible pairs of these metrics.

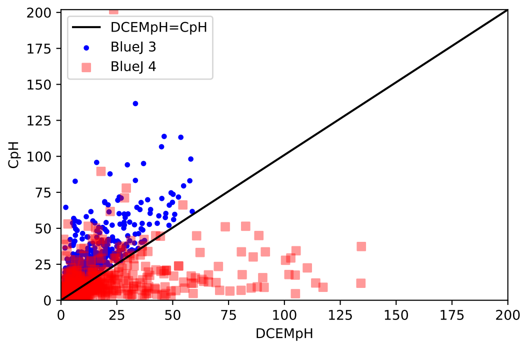


Figure 1: Displayed Compiler Error Messages per Hour (DCEMpH) and manual Compilations per Hour (CpH) for BlueJ 3 and 4. Each data point represents a user.

In Figure 1, the Cartesian space is bisected by the $y = x$ line which represents the lower bound of the relationship between DCEMpH

Table 1: Descriptive statistics for programming time in Hours (H), Displayed Compiler Error Messages per Hour (DCEMpH), manual Compilations per Hour (CpH) and Percentage of Successful manual Compilations (PSC) for BlueJ 3 and BlueJ 4.

	H		DCEMpH		CpH		PSC	
	BlueJ 3	BlueJ 4	BlueJ 3	BlueJ 4	BlueJ 3	BlueJ 4	BlueJ 3	BlueJ 4
M	18.604	30.485	10.657	17.102	21.873	11.625	0.515	0.761
SD	53.952	69.172	10.805	23.109	19.204	14.993	0.207	0.205
min	0.033	0.049	0.015	0.003	0.024	0.005	0.027	0.091
25%	0.670	0.908	3.253	2.451	8.367	2.910	0.357	0.626
50%	1.556	3.833	7.056	8.278	17.050	7.165	0.500	0.800
75%	8.008	19.058	14.402	21.601	29.336	15.173	0.668	0.930
max	437.732	488.914	58.752	134.44	136.667	201.948	1.000	1.000

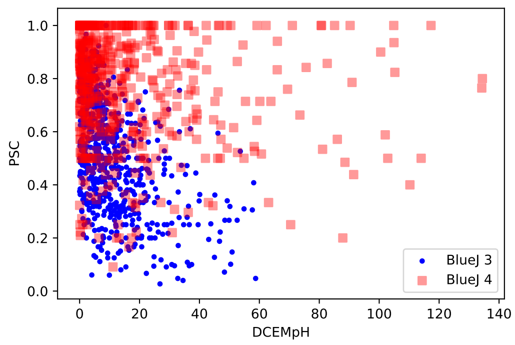


Figure 2: Displayed Compiler Error Messages per Hour (DCEMpH) and Percentage of Successful manual Compilations (PSC) for BlueJ 3 and 4. Each data point represents a user.

and CpH for BlueJ 3. This is due to the relationship between the number of compilations and the number of error messages displayed - for instance, a BlueJ 3 user can't see more than one error message per compilation. There is overlap between BlueJ 3 and BlueJ 4 for small values of CpH and DCEMpH. However, as values increase, it can be seen that the two distributions diverge, with BlueJ 4 moving along the DCEMpH axis, and BlueJ 3 along the CpH axis. The divergence of the distributions can also be seen in Figure 4, which we discuss later.

Figure 2 shows that BlueJ 4 users see more compiler error messages per hour than BlueJ 3. Additionally, a large number of BlueJ 4 users have high percentages of successful manual compilations (PSC), often 100%. Interesting, as manual compilations are not required in BlueJ 4. It is possible that students only manually compile as reassurance that there are no errors, when that appears to be the case (from a lack of other error indicators).

Figure 3 shows that BlueJ 3 users have more manual compilations per hour (as this is their only option), and again that the percentage of successful compilations is lower as in Figure 2.

5.2 RQ2: Modelling

Research question 2 was: *Can user behaviour profiles be modelled mathematically (or otherwise) for different programming environments?*

In Figures 4a and 4b it is shown that the BlueJ 3 distribution is localised to a nearly planar surface. We decided to model this distribution using the bivariate quadratic polynomial (see Eq. 1) to represent the relationship between the three metrics. Figure 5 shows a visualisation of the polynomial of Eq. 1 (the nearly planar surface) where $x = PSC$, $y = CpH$, $z = DCEMpH$, and coefficients a-f are described in Table 2.

$$z = f(x, y) = ax^2 + by^2 + cxy + dx + ey + f \qquad (1)$$

The surface produced by the polynomial (see Figure 5) provides a robust fit for the data and the equation can be used to describe the relationship between the three metrics when users program using BlueJ 3. This relationship seems to be restricted due to conditions imposed by BlueJ 3, such as the fact that users can not see more compiler error messages than they have compiles. BlueJ 4 interaction seems to be more complex.

6 THREATS TO VALIDITY

Blackbox usage data are anonymous, and without manual inspection of the source code, it is impossible to know either the nature of the tasks that users are programming or the level of mastery required to tackle them. Thus, these data are devoid of contextual information outside of environmental context. However, since this study focuses on an abstract level of interaction involving a large

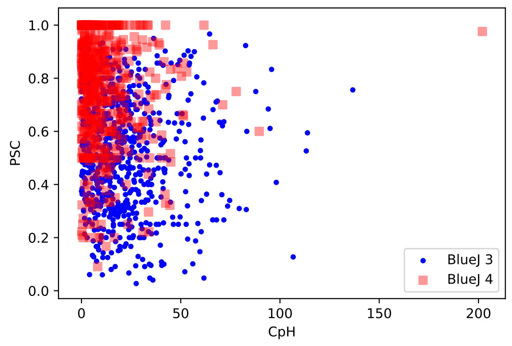


Figure 3: Manual Compilations per Hour (CpH) and Percentage of Successful manual Compilations (PSC) for BlueJ 3 and 4. Each data point represents a user.

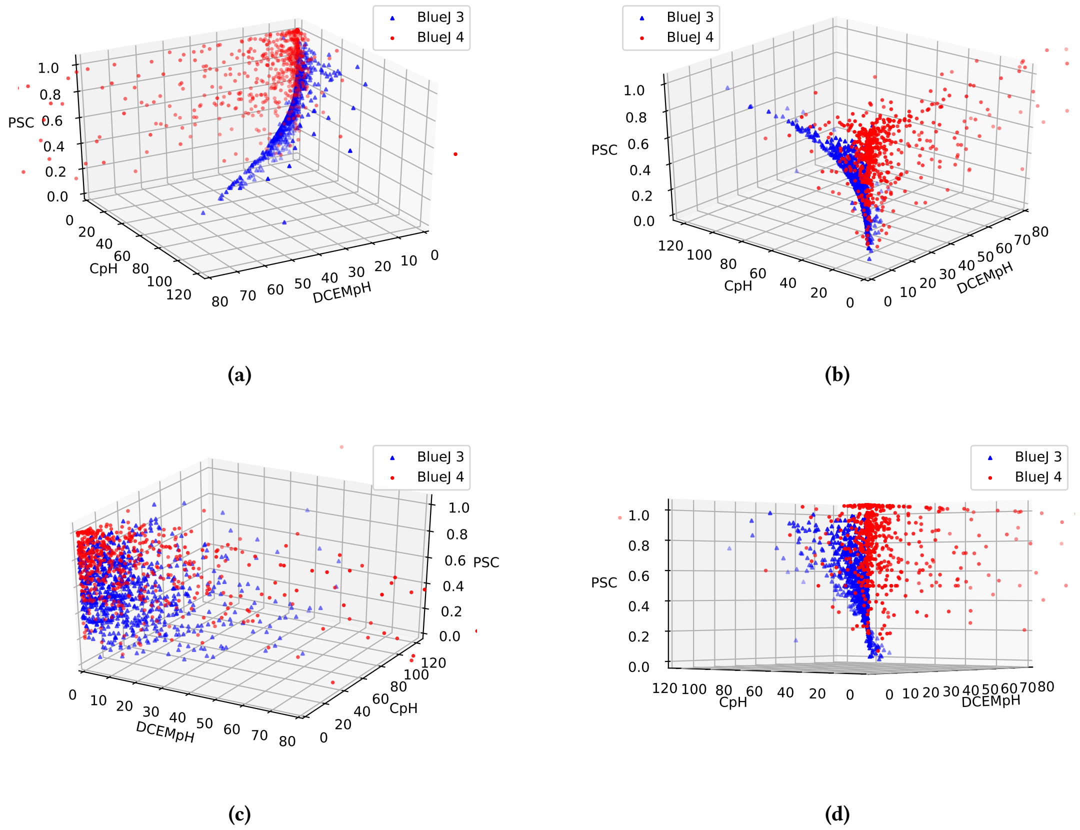


Figure 4: Three-dimensional scatter-plots describing Displayed Compiler Error Messages per Hour (DCEMpH), Compilations per Hour (CpH), and Percentage of Successful manual Compilations (PSC) from different angles. Each data point represents a user. The nearly planar form of the BlueJ 3 distribution is visible in (a) and (b). This is shown in more detail in Figure 5.

Table 2: Coefficients of Eq. 1.

Coefficient	Value	95% confidence interval
a	-1.106	(-3.63, 1.418)
b	-0.0006824	(-0.0008652, -0.0004997)
c	-0.8647	(-0.8944, -0.8349)
d	-1.236	(-4.016, 1.545)
e	0.9655	(0.9444, 0.9867)
f	0.7388	(-0.065, 1.543)

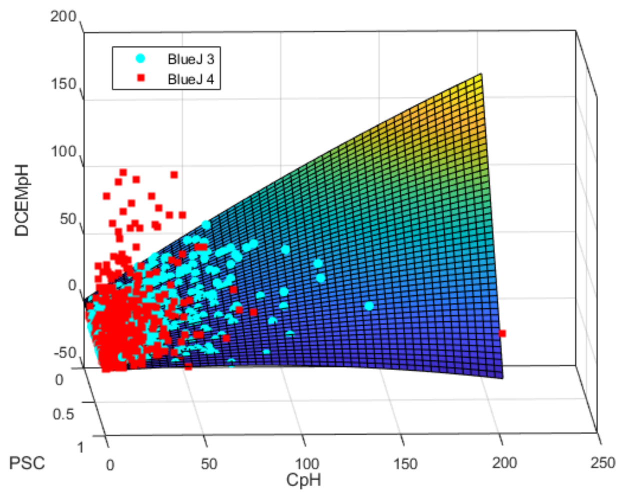
number of users, we believe that this is not a significant threat. We do plan on investigating the role that tasks play in the future.

Additionally, as mentioned in Section 4.2, there is an assumption regarding the actual time that users spend programming in BlueJ. Disruptions on the client side can cause Blackbox to stop recording events for a session. Since the present study regards the last recorded event as a final event of user activity for a session, it is possible that the retrieved data are incomplete. However, since we examine both BlueJ versions and the main point of the current study is a comparison between the two, we can hypothesize that this issue is factored out as the probability of having an incomplete session is the same in both versions. Regardless, this is an issue inherent in Blackbox which affects all studies using these data.

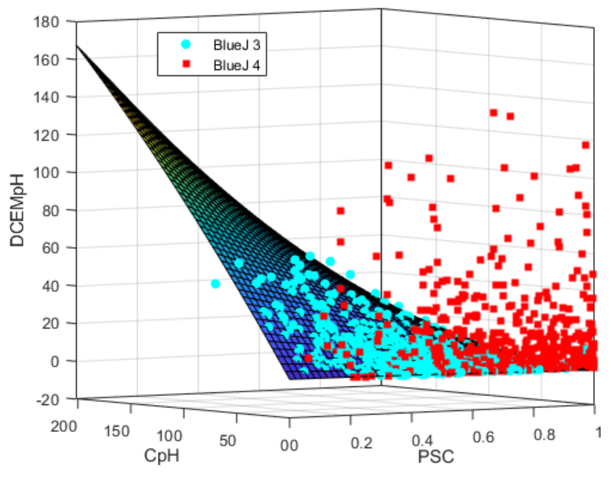
The error message presentation mechanism of BlueJ 4 is not limited to a maximum of one error message per compilation like BlueJ 3. However, users have to perform certain actions in order to see the error messages: either by clicking the compile button (possibly multiple times) or by hovering over the area of the error indicator(s). There is also a possibility that users trigger a shown error message if they click on the offending code to fix it, even if they didn't want to, or need to, see the message. In fact, we don't know that such messages are *read* even when they are displayed. Future work will focus on isolating these instances from the shown error messages that were triggered on purpose.

7 DISCUSSION

Each of the metrics used in the current study describes an aspect of the programming process when using an environment (which are not necessarily limited to, or unique to, BlueJ). Programmers provide the system with code, the system processes this code, and produces output in the form of error messages in the case of errors.



(a)



(b)

Figure 5: Displayed Compiler Error Messages per Hour (DCEMpH), manual Compilations per Hour (CpH), and Percentage of Successful manual Compilations (PSC) from a 3D perspective. The surface represents the fitting of BlueJ 3 using Eq. 1.

Thus, CpH represents the user request for evaluation, DCEMpH the output of the system, and PSC represents a form of evaluation of this interaction. Note that this evaluation is not necessarily representative of the user's effectiveness, but an evaluation of the interaction between the system and the user. Although, ultimately the aim is to define a system which maximises the effectiveness of the user in the smallest amount of time possible (while quality of learning is the same or better). This study explores users' actions, how they are constrained by the system, and aims to make some hypotheses about this interaction.

We can choose to view the three-dimensional space created by the three metrics as an enclosure of all the possible combinations of the metrics that in the end define a reasonable portion of the spectrum of possible interactions between user and the programming environment. The distributions of the two BlueJ versions are substantially different based on our findings in Section 5. In BlueJ 4, users see more error messages (either with intent or inadvertently as outlined in Section 6), and they do so by compiling manually less frequently. Additionally, their manual compilations are more often successful when compared to BlueJ 3 users.

BlueJ 3 interactions regarding manual compilations, their success rate, and the frequency of displayed error messages can easily be modelled as a nearly planar surface, while BlueJ 4 interactions are more complex and not easily represented by a single polynomial formula. This is mainly due to the fact that BlueJ 3 inherently restricts the behavior of the user by providing a maximum of one error message per compilation. This creates a bound for the amount of information sent by the system to the user. In BlueJ 4 this restriction does not occur, because the system allows users to have more displayed error messages than compilations.

We do observe many BlueJ 4 users with high numbers of displayed error messages, yet some of these also have high percentages of successful manual compilations (that producde no error messages). It is worth noting again that BlueJ 4 users need not manually compile. The reasons behind this are currently unknown. As mentioned in [21], manual compilations in BlueJ 4 could serve as psychological reassurance. If users don't see any compiler error messages, they may want to make sure that their code is syntactically correct.

The results of this study bring about a general observation and a corresponding question which we are working towards answering. The observation is: Given the freedom to explore an interaction space as they wish, users seem to choose to act in more complex ways. The question is: Which is more effective, a system that is more restrictive in terms of possible user actions, or one that allows more complex interactions?

While these findings provide useful insights for studying novice programming behaviour, more work is needed to infer any superiority of using one version over the other. Future work will focus on types of error messages that students are exposed to in each BlueJ version, as well as exploring the interaction before and after users manually compile. We will also investigate the resolution time of errors in each version.

REFERENCES

[1] Brett A. Becker. 2016. A New Metric to Quantify Repeated Compiler Errors for Novice Programmers. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (Arequipa, Peru) *(ITiCSE '16)*. Association for Computing Machinery, New York, NY, USA, 296–301. https://doi.org/10.1145/2899415.2899463

[2] 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, New York, NY, USA, 177–210. https://doi.org/10.1145/3344429.3372508

[3] Brett A. Becker and Thomas Fitzpatrick. 2019. What Do CS1 Syllabi Reveal About Our Expectations of Introductory Programming Students?. 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, 1011–1017. https://doi.org/10.1145/3287324.3287485

[4] 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, New York, NY, USA, 634–639. https://doi.org/10.1145/3159450.3159453

[5] Neil C. C. Brown, Amjad Altadmri, Sue Sentance, and Michael Kölling. 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* (Espoo, Finland) *(ICER '18)*. Association for Computing Machinery, New York, NY, USA, 196–204. https://doi.org/10.1145/3230977.3230991

[6] 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

[7] Elizabeth Carter. 2015. Its Debug: Practical Results. *J. Comput. Sci. Coll.* 30, 3 (Jan. 2015), 9–15.

[8] Elizabeth Carter and Glenn D Blank. 2013. An Intelligent Tutoring System to Teach Debugging. In *International Conference on Artificial Intelligence in Education*. Springer, Springer, Berlin, Heidelberg, 872–875.

[9] Elizabeth Carter and Glenn D. Blank. 2014. Debugging Tutor: Preliminary Evaluation. *J. Comput. Sci. Coll.* 29, 3 (Jan. 2014), 58–64.

[10] Michael Coblenz, Jonathan Aldrich, Brad A. Myers, and Joshua Sunshine. 2018. Interdisciplinary Programming Language Design. In *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Boston, MA, USA) *(Onward! 2018)*. Association for Computing Machinery, New York, NY, USA, 133–146. https://doi.org/10.1145/3276954.3276965

[11] Natalie J Coull. 2008. *SNOOPIE: Development of a Learning Support Tool for Novice Programmers Within a Conceptual Framework*. Ph.D. Dissertation. University of St Andrews.

[12] Evan Czaplicki. 2012. Elm: Concurrent FRP for Functional GUIs. *Senior thesis, Harvard University* 30 (2012).

[13] Joe Dillane. 2020. Frame-Based Novice Programming. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 583–584. https://doi.org/10.1145/3341525.3394007

[14] Linda Farragher and Simon Dobson. 2000. *Java Decaffeinated: Experiences Building a Programming Language From Components*. Technical Report. Trinity College Dublin, Department of Computer Science.

[15] Christoph Hannebauer, Marc Hesenius, and Volker Gruhn. 2018. Does Syntax Highlighting Gelp Programming Novices? *Empirical Software Engineering* 23, 5 (01 Oct 2018), 2795–2828. https://doi.org/10.1007/s10664-017-9579-0

[16] H. Rex Hartson. 1998. Human–computer Interaction: Interdisciplinary Roots and Trends. *Journal of Systems and Software* 43, 2 (1998), 103 – 118. https://doi.org/10.1016/S0164-1212(98)10026-2

[17] Brian Harvey, Daniel D. Garcia, Tiffany Barnes, Nathaniel Titterton, Daniel Armendariz, Luke Segars, Eugene Lemon, Sean Morris, and Josh Paley. 2013. SNAP! (Build Your Own Blocks) (Abstract Only). In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 759. https://doi.org/10.1145/2445196.2445507

[18] Matthew C. Jadud. 2006. Methods and Tools for Exploring Novice Compilation Behaviour. In *Proceedings of the Second International Workshop on Computing Education Research* (Canterbury, United Kingdom) *(ICER '06)*. Association for Computing Machinery, New York, NY, USA, 73–84. https://doi.org/10.1145/1151588.1151600

[19] Ioannis Karvelas. 2019. Investigating Novice Programmers' Interaction with Programming Environments. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE '19)*. Association for Computing Machinery, New York, NY, USA, 336–337. https://doi.org/10.1145/3304221.3325596

[20] 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)*. Association for Computing Machinery, New York, NY, USA, 1310. https://doi.org/10.1145/3328778.3372617

[21] 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)*. Association for Computing Machinery, New York, NY, USA, 759–765. https://doi.org/10.1145/3328778.3366882

[22] Michael Kölling. 1999. *The Design of an Object-oriented Environment and Language for Teaching*. Ph.D. Dissertation. Basser Department of Computer Science, University of Sydney.

[23] 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

[24] 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/csed.13.4.249.17496 arXiv:https://doi.org/10.1076/csed.13.4.249.17496

[25] Andrew Luxton-Reilly. 2016. Learning to Program is Easy. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (Arequipa, Peru) *(ITiCSE '16)*. Association for Computing Machinery, New York, NY, USA, 284–289. https://doi.org/10.1145/2899415.2899432

[26] 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 '18)*. ACM, New York, NY, USA, 55–106. https://doi.org/10.1145/3293881.3295779

[27] Davin McCall and Michael Kölling. 2019. A New Look at Novice Programmer Errors. *ACM Trans. Comput. Educ.* 19, 4, Article 38 (July 2019), 30 pages. https://doi.org/10.1145/3335814

[28] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11 (2009), 60–67.

[29] ME Sime, AT Arblaster, and TRG Green. 1977. Structuring the Programmer's Task. *Journal of Occupational Psychology* 50, 3 (1977), 205–216.

[30] Simon, Andrew Luxton-Reilly, Vangel V. Ajanovski, Eric Fouh, Christabel Gonsalvez, Juho Leinonen, Jack Parkinson, Matthew Poole, and Neena Thota. 2019. Pass Rates in Introductory Programming and in Other STEM Disciplines. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE-WGR '19)*. Association for Computing Machinery, New York, NY, USA, 53–71. https://doi.org/10.1145/3344429.3372502

[31] Juha Sorva. 2019. Splashing the Surface of Research: A Study of Koli Abstracts. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '19)*. Association for Computing Machinery, New York, NY, USA, Article 26, 2 pages. https://doi.org/10.1145/3364510.3366148

[32] Andreas Stefik, Bonita Sharif, Brad. A. Myers, and Stefan Hanenberg. 2018. Evidence About Programmers for Programming Language Design (Dagstuhl Seminar 18061). *Dagstuhl Reports* 8, 2 (2018), 1–25. https://doi.org/10.4230/DagRep.8.2.1

[33] Andreas Stefik and Susanna Siebert. 2013. An Empirical Investigation into Programming Language Syntax. *ACM Trans. Comput. Educ.* 13, 4, Article 19 (Nov. 2013), 40 pages. https://doi.org/10.1145/2534973

[34] Kelsey Van Haaster and Dianne Hagan. 2004. Teaching and Learning with BlueJ: an Evaluation of a Pedagogical Tool. *Issues in Informing Science & Information Technology* 1 (2004).

[35] Daniel Zingaro, Michelle Craig, Leo Porter, Brett A. Becker, Yingjun Cao, Phill Conrad, Diana Cukierman, Arto Hellas, Dastyni Loksa, and Neena Thota. 2018. Achievement Goals in CS1: Replication and Extension. 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, 687–692. https://doi.org/10.1145/3159450.3159452