<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>The Role of Compilation Mechanisms in Novice Programming Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Authors(s)</strong></td>
<td>Karvelas, Ioannis</td>
</tr>
<tr>
<td><strong>Publication date</strong></td>
<td>2022</td>
</tr>
<tr>
<td><strong>Publisher</strong></td>
<td>University College Dublin. School of Computer Science</td>
</tr>
<tr>
<td><strong>Item record/more Information</strong></td>
<td><a href="http://hdl.handle.net/10197/13282">http://hdl.handle.net/10197/13282</a></td>
</tr>
</tbody>
</table>

The UCD community has made this article openly available. Please share how this access benefits you. Your story matters! (@ucd_oa)

© Some rights reserved. For more information
The Role of Compilation Mechanisms in Novice Programming Behaviour

By

IOANNIS KARVELAS
STUDENT NUMBER: 17200836

The thesis is submitted to University College Dublin in fulfilment of the requirements for the degree of Doctor of Philosophy

School of Computer Science

Supervisor: Dr. Brett A. Becker

Head of School: Assoc. Prof. Neil Hurley

Doctoral Studies Panel Members

Assoc. Prof. Fintan Costello  Assoc. Prof. Catherine Mooney  Assoc. Prof. Mark Scanlon

NOVEMBER 2022
To Anna, the most improved versions of ourselves.
Ioannis Karvelas was born and raised in the suburbs of Athens in Greece. He obtained a Bachelor of Science in Computer Science from Athens University of Economics and Business in 2016. During his studies, he was fascinated by Algorithms, Complexity Theory, and Game Theory.

Ioannis’s curiosity about the human mind led him to Ireland to pursue a Master of Science in Cognitive Science in University College Dublin (UCD), which he obtained in 2018. Drawn by computational approaches to exploring cognition, he focused on Decision Making, Cognitive Modelling, Artificial Intelligence, and User Experience.

Ioannis proceeded as a PhD student in Computer Science in UCD in 2018, with a focus on Computing Education Research and Human-Computer Interaction. During his PhD, Ioannis developed an interest in teaching Computer Science, and specifically programming and robotics to younger students.

Ioannis’s research interests revolve around the intersection of Computer Programming Education, Human-Computer Interaction, and Microergonomics. His further interests include everything that is guitar-related, music (particularly when $bpm > 180$ and $d > 3/4$, where $d := distortion$), miniature painting, and motorcycles.
Without any doubt, this work could not have been completed without my mentor, Dr. Brett Becker. Brett has been a remarkable companion through this Odyssey and I would like to sincerely thank him for his support, patience, and for providing me the opportunity to grow as a person beside him. His professionalism, extensive knowledge of the field, pragmatic approach to overcoming obstacles, as well as his wits are things that I appreciated the most during my PhD.

I want to thank all the members of my Research Studies Panel, Assoc. Professors Fintan Costello, Catherine Mooney, and Mark Scanlon, for all the insightful conversations we had about my research. Furthermore, I would like to thank Prof. Eleni Mangina for providing me with so many opportunities to enhance my teaching skills while studying for my PhD and for all the wonderful chats and experiences we shared through the years. At the same time, I want to thank Assoc. Prof. Fred Cummins, the most intriguing lecturer I had the luck of meeting. His persistence on making his students wear their critical hats all the time made all the difference, I believe. Many thanks to all my colleagues, Joe, Joyce, and Eddie for being a fantastic company which I truly wish would have appeared much sooner and to Dr. Lan Wei and Yuhan for all the laughs and cries we shared, and for the fantastic food that they made me eat during my time in Ireland.

A big thank you to Prof. Michael Kölling and Dr. Neil Brown, for all their shared wisdom and technical assistance on BlueJ and Blackbox all these years.

I want to thank the Irish Research Council (IRC) for funding my PhD studies (under grant agreement GOIPG/2020/1660) and the UCD School of Computer Science for their funding, excellent faculty and staff, and friendly environment. The School felt like an extended family since day 1.

I would also like to thank and partially dedicate this thesis to the memory of Prof. Ioannis Milis. He was the catalyst that made me embark on this journey many years ago.

Further, I want to thank my parents Mary and Christos, my brother Giorgos, and our family’s dire wolf Hionoula for their unconditional love and support. Thank
you for listening to me ranting about the weather in Ireland all the time and for
enduring all the troubles I caused all these years. I love you all very much.

I feel that I could not be more lucky to have met Olga and Fiach, the coolest,
loveliest, most Greek non-Greek persons anyone could ever wish to spend so many
long nights learning about history, 90’s hip-hop, and heavy metal cover bands
with. Huge thanks and love to my friend Apollo, for all the adventures we had
all these years, for being there for me always and for (still) teaching me what to
truly appreciate in life. I also thank Giota, the most brilliant and talented school
teacher/papyrologist/dancer I have ever met. Moreover, my idyllic summers in
Greece would not have been possible without Fanis (aka, Armodios), Socrates, and
Stefanis - looking forward to many more summers with all of you!

I feel obliged to mention that this thesis would not have been possible without
the aid of my favourite band The Haunted, whose music kept me sound along the
way.

I sense that I have been overly extensive with the acknowledgements, but I
have to include a final addition. I want to dedicate this thesis and express my
deepest gratitude to Dr. Anna Markella Antoniadi. There are so many words one
could say in my stead, but I don’t feel like doing so right now. I dedicate the poem
“T Sit Beside the Fire and Think” written by J. R. R. Tolkien to her instead.
Abstract

Compilation mechanisms, like many features in programming environments, are understudied in terms of their effects on users, especially in the case of programming students. In this thesis we explore how these mechanisms affect the programming behaviour of novices towards better understanding their roles in the programming process. We conduct our study through different compilation mechanism setups that are present in two major version releases of the BlueJ introductory pedagogical environment for the Java programming language. BlueJ 3 features “manual compilation with enforced first error message presentation”, while BlueJ 4 supports “automatic and manual compilation with on-demand any/all error message presentation”. We examine the programming behaviour of thousands of novices by utilizing programming process data mined from Blackbox, a large repository that stores programming-related information organically generated by BlueJ users.

We begin our investigation by exploring how the differentiation in compilation mechanisms affects the interaction between the user and the programming environment in terms of quantity of displayed compiler error messages, manual compilations, and rates of manual compilation success. We found that novices tend to see more error messages, compile manually less frequently, while their manual compilations have higher success rates in BlueJ 4 over BlueJ 3. We also observed that novices’ quantified interaction with BlueJ 3 follows a more deterministic distribution compared to the interaction of users with BlueJ 4, due to the environment restrictions. In addition, we found more intensified differences in the interaction with the two BlueJ versions when users are exposed to a single compilation mechanism than when exposed to multiple mechanisms, with users in the latter group showing a moderated interaction, possibly due to adapting their habits accordingly.

Further, we explored the compilation behaviour of novices in terms of the syntax state of their source code and the trigger state of their compilations, the error resolution time, as well as their interaction with compiler error messages during the programming process. We identified multiple similarities and differences in compilation behaviour when using the different compilation mechanisms, and also observed lower error resolution times in BlueJ 4. As the interaction of users with error messages is inherently more complicated
in BlueJ 4 than in BlueJ 3, we provide details on the manner that displayed error messages emerge while programming.

As part of our investigation, we also conducted surveys with programmers with different experience levels that targeted the perceived usefulness of compilation mechanisms, and asked them to provide suggestions for improvement. We found that programmers perceive the error message presentation present in BlueJ 4 as more useful and their overall suggestions on improvement aligned more towards the functionality of the mechanisms present in BlueJ 4 than those in BlueJ 3. Finally, we conducted an observational study with postgraduate students in order to draw contextualized inferences and evaluate prior assumptions that derived from non-contextual data. Substantial differences in the manner that users interact with compilation mechanisms and approach the resolution of errors were observed, depending on how these mechanisms are facilitated within the environment.

Based on our findings, we discuss the implications of the studied compilation mechanisms in the programming process, since substantial changes emerged in the behaviour of novices, and provide a set of design guidelines for programming environments. This thesis aims to inform the research community with an inquiry on mechanisms that influence the programming process of students, and provide information to pedagogical tool designers about the suitability of their design decisions for the first steps of learning how to program.
Conference Papers


Conference Posters


Conference Abstracts

I hereby certify that the submitted work is my own work, was completed while registered as a candidate for the degree stated on the Title Page, and I have not obtained a degree elsewhere on the basis of the research presented in this submitted work.

SIGNED: ............................................ DATE: ..................................

3 November 2022
COLLABORATORS

• **Joe Dillane**, *School of Computer Science, University College Dublin, Dublin 4, Ireland*
  Assisted in the mathematical modelling of the interaction metrics in Chapters 4 and 5. Contributed to sections in Chapters 2 and 7.

• **Annie Li**, *University of Michigan, Ann Arbor, Michigan, USA*
  Implemented procedures for testing the interaction metrics with different time heuristics to account for different inactive time intervals of the programming activity in Chapter 3.

• **Eddie Antonio Santos**, *School of Computer Science, University College Dublin, Dublin 4, Ireland*
  Assisted in the investigation of the volatility of displayed compiler error messages and contributed to the associated section in Chapter 7.
# Table of Contents

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
</tr>
<tr>
<td>List of Figures</td>
</tr>
<tr>
<td>I Introduction and Background</td>
</tr>
<tr>
<td>1 Introduction</td>
</tr>
<tr>
<td>1.1 Motivation</td>
</tr>
<tr>
<td>1.2 Objectives</td>
</tr>
<tr>
<td>1.3 Research Questions</td>
</tr>
<tr>
<td>1.4 Contributions</td>
</tr>
<tr>
<td>1.5 Outline</td>
</tr>
<tr>
<td>2 Background and Related Work</td>
</tr>
<tr>
<td>2.1 Introductory Programming</td>
</tr>
<tr>
<td>2.2 Programming Environments</td>
</tr>
<tr>
<td>2.2.1 Programming Environments in the Classroom</td>
</tr>
<tr>
<td>2.2.2 What is a Programming Environment?</td>
</tr>
<tr>
<td>2.2.3 Compilation and Error Message Presentation</td>
</tr>
<tr>
<td>2.2.4 Programming Environments as Complex Tools</td>
</tr>
<tr>
<td>2.2.5 Lack of Empirical Evaluation in Programming Environments</td>
</tr>
<tr>
<td>2.3 Related Work</td>
</tr>
<tr>
<td>2.3.1 Development of Prototype Tools</td>
</tr>
<tr>
<td>2.3.2 Impact of Programming Tools</td>
</tr>
<tr>
<td>Section</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>2.3.3</td>
</tr>
<tr>
<td>2.3.4</td>
</tr>
<tr>
<td>2.3.5</td>
</tr>
<tr>
<td>2.4</td>
</tr>
<tr>
<td>2.5</td>
</tr>
<tr>
<td>2.5.1</td>
</tr>
<tr>
<td>2.5.2</td>
</tr>
<tr>
<td>2.5.3</td>
</tr>
<tr>
<td>2.6</td>
</tr>
</tbody>
</table>

II The Triptych of the Interaction Between Novices and Programming Environments

3 Interaction Between Programming Environments and Novices Exposed to Multiple Compilation Mechanisms

3.1 Introduction                                            29
3.2 Data                                                   30
  3.2.1 Data Filtering                                      30
  3.2.2 Calculating Programming Time                       31
3.3 Methodology                                            32
  3.3.1 Per-User Calculation                               32
  3.3.2 Removing Outliers                                  33
  3.3.3 Global Calculation                                 34
  3.3.4 Trimming Inactive Intervals                        34
3.4 Results and Discussion                                 35
  3.4.1 RQ3.1: Displayed Compiler Error Messages            36
  3.4.2 RQ3.2: Manual Compilations                          37
  3.4.3 RQ3.3: Success of Manual Compilations               38
  3.4.4 RQ3.4: Session Time Heuristics                      39
3.5 Threats to Validity                                    40
3.6 Conclusions and Future Direction                       42
# TABLE OF CONTENTS

4 Interaction Between Programming Environments and Novices Exposed to a Single Compilation Mechanism  
4.1 Introduction ........................................... 44 
   4.1.1 Research Questions ................................. 44 
4.2 Methodology ........................................... 45 
4.3 Results ................................................. 45 
   4.3.1 RQ4.1: Distributions ............................... 45 
   4.3.2 RQ4.2: Modelling .................................. 47 
4.4 Threats to Validity ................................... 48 
4.5 Discussion ............................................. 50 

5 Compilation Mechanisms: Exclusive Versus Transitioning Use  
5.1 Introduction ........................................... 54 
   5.1.1 Research Questions ................................. 54 
5.2 Methodology ........................................... 55 
   5.2.1 User Cohorts ...................................... 55 
   5.2.2 Compilation and Error Message Presentation Metrics ... 55 
   5.2.3 Removing Outliers .................................. 56 
   5.2.4 Categorizing Transition Users ..................... 56 
   5.2.5 Metric Restriction in BlueJ 3 ...................... 57 
   5.2.6 Similarity Calculation .............................. 57 
5.3 Results ................................................. 58 
   5.3.1 RQ5.1: Exclusive vs Transition Use ............... 59 
   5.3.2 RQ5.2: Order of Transition ......................... 62 
5.4 Threats to Validity ................................... 63 
5.5 Discussion ............................................. 64 

III Compilation Behaviour and Interaction with Compiler Error Messages  

6 Compilation Behaviour and Error Resolution Time of Novices Exposed to Different Compilation Mechanisms  
6.1 Introduction ........................................... 70
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1.1</td>
<td>Research Questions</td>
<td>70</td>
</tr>
<tr>
<td>6.2</td>
<td>Methodology</td>
<td>70</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Identifying Matched BlueJ User Cohorts</td>
<td>70</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Constructing Compilation Sequences</td>
<td>71</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Error Resolution</td>
<td>73</td>
</tr>
<tr>
<td>6.3</td>
<td>Results</td>
<td>73</td>
</tr>
<tr>
<td>6.3.1</td>
<td>RQ6.1: Compilation Behaviour</td>
<td>73</td>
</tr>
<tr>
<td>6.3.2</td>
<td>RQ6.2: Error Resolution Time</td>
<td>78</td>
</tr>
<tr>
<td>6.4</td>
<td>Threats to Validity</td>
<td>79</td>
</tr>
<tr>
<td>6.5</td>
<td>Discussion</td>
<td>79</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Interaction Between Novices and Compiler Error Messages</td>
<td>83</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>84</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Objectives and Research Question</td>
<td>84</td>
</tr>
<tr>
<td>7.2</td>
<td>Methodology</td>
<td>84</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Blackbox Cohort</td>
<td>84</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Data Pre-processing</td>
<td>85</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Most Common Error Messages</td>
<td>86</td>
</tr>
<tr>
<td>7.2.4</td>
<td>Identifying Specific Compilations that Result in Displayed Error Messages</td>
<td>86</td>
</tr>
<tr>
<td>7.2.5</td>
<td>Viewing Order of Error Messages</td>
<td>86</td>
</tr>
<tr>
<td>7.3</td>
<td>Results</td>
<td>87</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Most Common Displayed Error Messages</td>
<td>88</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Proportion of Compilations that Result in Displayed Error Messages</td>
<td>90</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Specific Compilations that Result in Displayed Error Messages</td>
<td>91</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Distributions of Displayed Error Messages per Failed Compilation</td>
<td>93</td>
</tr>
<tr>
<td>7.3.5</td>
<td>Order in Which Error Messages Are Displayed</td>
<td>95</td>
</tr>
<tr>
<td>7.4</td>
<td>Threats to Validity</td>
<td>95</td>
</tr>
<tr>
<td>7.5</td>
<td>Discussion</td>
<td>97</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

## IV Usability Testing

8 Evaluation of Compilation Mechanisms: An Online Survey of Programmers 100

8.1 Introduction 102

8.1.1 Objectives and Research Question 102

8.2 Methodology 103

8.2.1 Participant Recruitment 103

8.2.2 Participant Screening 103

8.2.3 Survey Materials 104

8.3 Results 105

8.3.1 Demographics 105

8.3.2 Compilation 105

8.3.3 Error Indication 106

8.3.4 Error Message Presentation 108

8.4 Threats to Validity 110

8.5 Discussion 111

9 Observations on the Programming Behaviour of Students Using Different Compilation Mechanisms 113

9.1 Introduction 114

9.1.1 Aims and Objectives 114

9.2 Methodology 115

9.2.1 Participant Recruitment 115

9.2.2 Interview Materials 116

9.2.3 Quantifying the Interaction Between Participants and BlueJ 117

9.3 Results 118

9.3.1 Demographics 118

9.3.2 Interaction with BlueJ 118

9.3.3 Survey Questions 124

9.4 Threats to Validity 125

9.5 Discussion 127
## V Discussion

10 Discussion, Implications, and Future Work 130

10.1 Programming Environments: What Do We Want From Them? .... 130

10.2 Programming Process Data Repositories: What Do We Want From Them? ................................................................. 132

10.2.1 Blackbox Implications .............................................. 132

10.3 Thesis Overview .......................................................... 133

10.4 RQ1. Quantified Interaction and Transitioning Between Mechanisms 134

10.5 RQ2. Compilation Behaviour and Interaction with Compiler Error Messages ........................................................................ 137

10.6 RQ3. Programmers’ Perception of Compilation Mechanisms .... 140

10.7 RQ4. Changes in the Programming Process ........................ 140

10.8 Suggestions on Designing Compilation Mechanisms ............. 141

10.9 Suggestions to the Research Community ............................ 143

10.10 The Verdict ........................................................................ 143

## VI Appendices

A The Beast in Black: From Blackbox to Data 146

A.1 Author’s Note ................................................................. 147

A.2 Challenges of Working with Blackbox ................................. 147

A.3 Blackbox Data in the Present Thesis ................................ 148

B User Studies Materials 153

B.1 Participant Recruitment Advertisement ............................... 153

B.2 Consent Form .................................................................. 155

B.3 Programming Task .......................................................... 159

B.4 Online Survey and Interview Questionnaires .................... 161

B.4.1 BlueJ 3 Survey .............................................................. 161

B.4.2 BlueJ 4 Survey .............................................................. 171

Bibliography 184
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Programming time (H), displayed compiler error messages per hour (DCEMpH), manual compilations per hour (CpH), and percentage of successful manual compilations (PSC) in both BlueJ versions per user (*the first two rows are global calculations and values for H in this case are discussed in Section 3.3.3). We report p values that are &lt; .001 as such, per APA guidelines [6] which are recommended by [137]. We used an alpha level of .05 for all statistical tests.</td>
</tr>
<tr>
<td>4.1</td>
<td>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.</td>
</tr>
<tr>
<td>4.2</td>
<td>Coefficients of Equation 4.1.</td>
</tr>
<tr>
<td>5.1</td>
<td>Complete set of statistical results for all metrics examined.</td>
</tr>
</tbody>
</table>
5.2 Mean values of 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 of all user cohorts. Effect sizes (ES) using Cohen’s $d$ are displayed alongside each pair of metrics. The direction of effect is omitted in ES since it is consistent in all metrics across all cohorts. ES Sum refers to the cumulative effect size derived by summing all ES values in that row. The last three columns display the values of distance between BlueJ 3 and BlueJ 4 using the methods described in Section 5.2.6. In the first row, $n_3$ and $n_4$ refer to the number of users in cohorts X3 and X4 respectively.

6.1 All possible consecutive compilation sequences based on success and trigger states for BlueJ 4.

6.2 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.

7.1 Frequencies of the most common displayed compiler error messages in the two BlueJ versions. The negative sign in “Diff.” (Difference) indicates if there was a decrease in frequency from BlueJ 3 to BlueJ 4. The messages are sorted based on the absolute difference in frequencies between BlueJ 3 and 4 in descending order.

7.2 Distributions of the number of compiler error messages per failed compilation in BlueJ 4.

7.3 Distributions of displayed error message quantity in mapping order in the source code (the higher and more left an error message is in the source – the lower its ranking number) and viewing order in terms of time (max rank=85).

8.1 Percentages of participants’ preferences regarding the proper location of error messages within a programming environments, grouped by cohort and survey version.
9.1 Distributions of the interaction metrics for the participants performing
the programming task. The DCEMpH values under 4* represent the
deliberate appearance of compiler error messages. .......................... 119
9.2 Distribution of the session times for participants and the time elapsed
until they manually compiled for the first time. The time is expressed
in minutes. ................................................................. 119
9.3 Breakdown of mean rating values provided by participants regard-
ing compilation and error message presentation-related mechanisms
present in the two BlueJ versions. Numbers inside the parentheses
represent the measured standard deviations. ................................. 125
9.4 Participant responses when asked about their strategy of addressing
errors. ............................................................................ 125
9.5 Participant preferences on the desired presentation of error messages. 125
9.6 Participant preferences on the desired location display of error messages. 126
9.7 Participant preferences on the desired manner of display of error mes-
sages. ............................................................................ 126

A.1 Blackbox database fields that were used through the work described in
the present thesis. The labels in the horizontal parallel lines signify the
database table that the subsequent fields are located in Blackbox. ..... 149
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The outline of the programming process of a novice programmer.</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>An example of inaccurate and confusing error messages.</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>BlueJ 3 editor window.</td>
<td>24</td>
</tr>
<tr>
<td>2.4</td>
<td>BlueJ version 4 editor window.</td>
<td>25</td>
</tr>
<tr>
<td>3.1</td>
<td>Schematic representation of how programming time is calculated from Blackbox data for a single user. The first term in the subscripts refers to the number of each session in chronological order ranging from 1 to “m”, while the second to the number of each event in that session in chronological order ranging from 1 to “l”. “l” always represents the last recorded event of a session.</td>
<td>31</td>
</tr>
<tr>
<td>3.2</td>
<td>Programming time in BlueJ 3 &amp; 4 for each user. All but the lower left quartile were considered outliers.</td>
<td>33</td>
</tr>
<tr>
<td>3.3</td>
<td>Displayed compiler error messages per hour (DCEMpH) in BlueJ 3 &amp; 4 per user. All but the lower left quartile were considered outliers.</td>
<td>34</td>
</tr>
<tr>
<td>3.4</td>
<td>Displayed compiler error messages per hour (DCEMpH) for BlueJ 3 &amp; 4. The orange horizontal line in the box conveys the median and the green arrow the mean.</td>
<td>37</td>
</tr>
<tr>
<td>3.5</td>
<td>Manual compilations per hour (CpH) for BlueJ 3 &amp; 4. The orange horizontal line in the box represents the median and the green arrow the mean.</td>
<td>38</td>
</tr>
<tr>
<td>3.6</td>
<td>Percentage of successful manual compilations (PSC) for BlueJ 3 &amp; 4. The orange horizontal line in the box represents the median and the green arrow the mean.</td>
<td>39</td>
</tr>
</tbody>
</table>
3.7 Global displayed compiler error messages per hour (DCEMpH) in BlueJ 3 & 4, treating time intervals between events as inactive according to different thresholds. ................................................................. 40

4.1 Displayed Compiler Error Messages per Hour (DCEMpH) and manual Compilations per Hour (CpH) for BlueJ 3 and 4. Each data point represents a user. ................................................................. 46

4.2 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. ................................................................. 47

4.3 Manual Compilations per Hour (CpH) and Percentage of Successful manual Compilations (PSC) for BlueJ 3 and 4. Each data point represents a user. ................................................................. 48

4.4 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 4.5. ................................................................. 50

4.5 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 Equation 4.1. ................................................................. 51

5.1 Surface representing Equation 5.1 and each BlueJ 3 user mapped by their metric coordinates (blue triangles). ................................................................. 58

5.2 Three-dimensional scatter-plots describing Displayed Compiler Error Messages per Hour (DCEMpH), Compilations per Hour (CpH), and Percentage of Successful manual Compilations (PSC) for all user cohorts. Each point represents a user. ................................................................. 62

6.1 Programming time allocation on consecutive compilation and trigger states (top: BlueJ 3, bottom: BlueJ 4). Key to compilation states and triggers: \textbf{SF} = Success$\rightarrow$Failure, etc. \textbf{MA} = Manual$\rightarrow$Automatic, etc. 75
LIST OF FIGURES

6.2 Timing distributions of different compilation sequences in both BlueJ versions (green triangle: mean, horizontal line: median). TC is the time between compilations, while TSC is the time between successful compilations. ................................................. 76

6.3 Timing distributions of BlueJ 4 consecutive compilation sequences separated by success and trigger indicators (green triangle: mean, horizontal line: median). A stands for an automatic compilation, M for manual. 77

7.1 Rank variations of the most common shown compiler error messages based on frequency in the two BlueJ versions. The figure was created with the aid of LineUp [59]. ................................................. 92

7.2 Proportion of displayed compiler error messages corresponding to different consecutive compilation sequences based on syntax states for BlueJ 3 & 4. “-t” denotes a terminating compilation for a given session. 93

7.3 Proportion of displayed compiler error messages corresponding to different consecutive compilation sequences based on trigger states for BlueJ 4. “-t” denotes a terminating compilation for a given session. The “AA: 78%” sequence was omitted in favor of a clear representation of the rest of the sequences. ............................................................ 94

8.1 Participant ratings for the compilation mechanism featured in each BlueJ version (rectangle: mean, horizontal line: median). ................. 106

8.2 Participant ratings for the error indication featured in each BlueJ version (rectangle: mean, horizontal line: median). ......................... 107

8.3 Participant ratings for the error message presentation featured in each BlueJ version (rectangle: mean, horizontal line: median). ................ 108

9.1 Displayed Compiler Error Messages per Hour (DCEMpH) boxplots of participants during the programming session. “BlueJ 4 Deliberate” reflects the error messages that were triggered consciously by participants during the session (rectangle: mean, horizontal line: median). ........ 120

9.2 Manual Compilations per Hour (CpH) boxplots of participants during the programming session (rectangle: mean, horizontal line: median). 121
9.3 Boxplots describing the distribution of the time elapsed since the start of the programming session until the first time a manual compilation occurred divided by the total time of the programming session (rectangle: mean, horizontal line: median). ................................................................. 122

9.4 Percentage of Successful Manual Compilations (PSC) boxplots of participants during the programming session (rectangle: mean, horizontal line: median). ................................................................. 123
Part I

Introduction and Background
Resnicow and Vaughan published a paper in 2006 [132], in which they stated:

“One additional concept from Chaos Theory, fractal patterns, may also be relevant to understanding human behavior. Fractals, which have been identified in natural science in the mapping of the microvascular system and snowflake geometry, are recurring patterns within larger systems that are self-similar, that is, a shape appears similar at all scales of magnification. In terms of human behavior, there may be common patterns of behavior change within and across individuals that follow certain complex, non-linear patterns. Thus, although behavior change may unfold in an almost infinite combination of knowledge, attitude, efficacy, and intention, there may be recurrent patterns of change that may be used to identify audience segments which could be targeted by common interventions.”

Human behavior is studied across countless disciplines from different prisms and with different aims and objectives each time. Understanding how we - as humans - act, react, behave, and interact with our surroundings is not only beneficial for understanding human nature, but also aids in optimizing our tasks. In the present work, one such interaction is studied and presented; the interaction between
humans and tools. The epistemological approach of the present thesis includes a series of empirical studies that serve as a medium through which we attempt to explore and evaluate this interaction, with the ultimate goal of understanding the human, the tool, and improving the interaction between them.

1.1 Motivation

Programming environments are the main tools that code developers use in order to implement, debug, and test their code before deployment of the final product. One of the main mechanisms present in the software development process is compilation; the translation of source code written in a programming language to a lower-level set of command statements. Compilers are usually large and complicated programs that perform series of analyses, checking the input code against the specifications of the programming language that the source code is written. If no violations of those specifications exist, then the developer can proceed to execution of their program. However, if any inconsistencies are detected by the compiler during the compilation process, then it usually generates a list of error messages, and the programming process is halted. The developer is then required to identify and correct the mistakes that are present in the source code, in order for the programming process to resume.

Compiler error messages (or programming error messages) have a long history of being notoriously difficult to interpret, as they are often poorly worded, and most of the times not aimed for inexperienced programmers. Further, they can be inaccurate at times. Finally, since it is difficult to establish a consistent mapping between specific errors in the source code and compiler error messages, they are a source of confusion for developers, especially for beginners.

Novice programmers learn the fundamentals of programming, while often being enrolled as students in an introductory programming course. Novices typically solve programming exercises assigned by their educators while using a programming environment. Sometimes students are advised to use a specific programming environment (that could be designed for educational purposes) or a commercial environment that the educator uses. In other instances, students are allowed to pick an environment of their own preference. As a result, a lot of students turn to
big industrial-grade Integrated Developments Environments (IDE), tools that offer a multitude of assisting integrated mechanisms and customization option, mostly aimed to assist experienced programmers. However, it is uncertain which of these mechanisms are suitable for novice programmers in their first steps of learning how to program.

Considering all the above, combined with the fact that introductory programming courses suffer from high dropout rates from students [158], it seems all the more important for institutions, educators, researchers and tool designers to work towards facilitating an improved experience for students undertaking computer programming. This task is far from trivial, as it requires multi-disciplinary approaches and constant revision due to computing being a phenomenally fast progressing field. In the present work, we examine the relationship between novice programmers’ behaviour and the compilation mechanisms present in an educational programming environment. Through this work, future researchers and tool designers can be better informed about the effects that their products have on novices and assist them while learning how to program.

1.2 Objectives

There are many different tools and mechanisms that have been developed to assist programmers during their programming task, ranging from carefully designed user interfaces, to debuggers, embedded unit testers, programming languages, compilers, etc. It is not only difficult nowadays to track down every tool that is available for use, it is often hard to categorize them based on their functionality. However, one of the core mechanisms that are present in almost every single programming environment, whether that is a notepad editor that the programmer utilizes to write the source code and a command line terminal to invoke the compiler or a complicated Integrated Development Environments (IDE) such as IntelliJ or Eclipse, is compilation. The compilation mechanism allows users to check their source code for compile-time errors, and provides feedback in the form of compiler error messages. Although the compiler is the main agent in this process, the programming environment is responsible for when the compilation will occur, and which error messages the environment will show. One design decision is to
present the compiler output without any interference. Another is to process the list of messages and present a reworded alternative. All these are mechanisms that the user (in our case the novice programmer) experiences, and in extent altering them also alters this experience.

In older environments, it was common for the compilation to be invoked manually.\(^1\) This was achieved by including a dedicated button or binding a combination of buttons from the keyboard that would trigger a compilation of the source code to occur. More modern systems follow a more dynamic approach to compilation, which is triggered automatically while users edit their code. The rationale behind this implementation is early feedback. Usually errors present in the source code are marked on the editor while the user is typing new code, with the designers’ intent being that the programmer will identify and fix these errors early, saving them time that would otherwise be devoted in the end of the programming task. However, the effect design decision on both novices and experienced developers has not been explored. Yet, it is hard to find IDEs with the option for compilation not set to dynamic by default.

One approach from an educational point of view regarding compiler error messages is to limit the level of novices’ exposure to multiple, possibly cascading and inter-related, error messages when they want to fix compile-time errors. Furthermore, some educators instruct their students to pay attention to the first error message that appears on the list of messages as an output from the compiler, as fixing the error associated with the first message often eliminates some (or all) subsequent errors (and in extent, error messages).

In this thesis, we explore how these design decisions regarding the compilation mechanism affect the programming behaviour of novice programmers, while at the same time providing evidence that these play an important role in the programming process, especially for that of inexperienced programmers, since they not only learn how to program, but they also learn how to use and interact with a programming environment. We focus on two versions of the BlueJ introductory programming environment (described in Chapter 2), that feature changes in how

\(^1\)Unless the programmer was using a text editor to write code and invoking the compiler from the command-line as an alternative practice. However, it could be argued that this is also a variation of compiling manually.
the compilation mechanisms are facilitated within the editor, while keeping the rest of the environment features mostly unchanged. We specifically focus our study on the two following compilation mechanism set-ups:

1. “Manual” compilation with “enforced first error” message presentation
2. “Automatic and manual” compilation with “on-demand any/all” error message presentation

Through this study, we aim to: (1) provide evidence on the importance of design decisions of compilation mechanisms in the novice programming process, and (2) discover how programmers interact with these mechanisms.

1.3 Research Questions

Through the work described in this thesis, we are guided by the following research questions:²

**RQ1:** How is the quantified interaction between novice programmers and programming environments affected by changes in the way compilation mechanisms operate within a programming environment and how does transitioning between mechanisms affect this interaction? (Part II, based on [84, 80, 83])

**RQ2:** How is the compilation behaviour and interaction with compiler error messages altered by changes in the way compilation mechanisms operate within a programming environment? (Part III, based on [79])

**RQ3:** How do programmers from different experience levels perceive these mechanisms in terms of their usefulness and what are their suggestions on improving them? (Part IV, Chapter 8)

**RQ4:** How do the interaction and behavioural findings of this research emerge while observing programmers using the compilation mechanisms during the programming task? (Part IV, Chapter 9)

²The research questions demonstrated in this section represent the inquiry frames overarching multiple studies. While presenting the individual work that was conducted, we will present “lower-level” research questions, which we will abbreviate according to their respective chapter. For example, research question y in chapter x will be denoted as RQx.y.
1.4 Contributions

This work has:

1. Provided empirical evidence on the programming behaviour of novices and their interaction with different core compilation mechanisms including their error resolution times through multiple studies that use a large sample (that amounts to several thousands in total) of novice programmers, allowing for reasonable generalization to the general population. These findings can also be generalized beyond the BlueJ programming environment to an extent, since the environment is very simplistic in its design, allowing for an investigation of an almost direct interaction with the user without distracting features.

2. Explored how using a single compilation mechanism affects the quantifiable interaction with the user, as opposed to when the user transitions through multiple mechanisms.

3. Provided evidence on how programmers from different experience levels perceive these compilation mechanisms in terms of usefulness at an individual basis and a number of suggestions on improving them based on those programmers’ experiences.

4. Provided an outline on how users resolve their errors when compilation mechanisms vary by conducting an observational study with students as participants, while at the same time providing a high-level vignette of the observed differences in the programming process.

5. Evaluated the findings from the studies that relied on non-contextual data by performing interviews with students performing a programming task using the compilation mechanisms under investigation.

6. Provided a series of suggestions for designing compilation mechanisms and suggested avenues for future work to the Computing Education Research community.
1.5 Outline

In Part I of this thesis, we present a preamble to the covered topic and provide a more comprehensive background of the work, including the motivations, aims, contributions, and outline (Chapter 1). We proceed to providing a more detailed view of the background and the related work, describing the investigated topic, and providing information about the BlueJ programming environment and the Blackbox database (Chapter 2).

In Part II, the quantitative aspect of the interaction between a programming environment and the novice is explored. This is investigated through three successive studies. In the first study, we explore the interaction of users who used both variations of the examined compilation mechanisms (Chapter 3). In the second study, the experiments are replicated using a cohort of novices who were exposed only to a single compilation mechanism. The interaction with one variation of compilation mechanisms is mathematically modelled (Chapter 4). In the third study, we focus on comparing the interaction of users who used both mechanisms to those who used one. We also explore if and how the interaction habits of users are affected by the order of transitioning between compilation mechanisms (Chapter 5).

In Part III, we investigate how the compilation behaviour, error resolution time (Chapter 6), and the interaction with error messages (Chapter 7) are affected by different compilation mechanisms.

In Part IV, we conduct an online survey that aims to provide insights into how the different compilation mechanisms are perceived in terms of usefulness by programmers from different experience levels (Chapter 8). Furthermore, we present an observational study (Chapter 9) through which (1) we evaluate previous findings regarding the quantified interaction of users with compilation mechanisms (Part I), (2) we evaluate our findings from the online survey, by having participants use the compilation mechanisms, (3) we observe how our findings and insights from non-contextual data emerge during the programming process, (4) we describe how the programming process is affected on a more abstract level.

In Part V (Chapter 10), we present an overarching discussion based on the findings of our studies along with their implications, and propose design guidelines for compilation mechanisms and future work on this domain.
2.1 Introductory Programming

Introductory programming at university/college level (commonly referred to as CS1) [67, 19] receives a substantial amount of focus in computing education research. It is common for educators to claim that “programming is hard” [146, 104]. However, this has been recently called into question based on the pass rates of introductory programming courses not being substantially lower than those of introductory courses in several other disciplines [145], and whether or not programming is inherently ‘hard’ for novices is debated [12]. Parts of this claim could be attributed to the tools and techniques used for teaching programming, rather than the content itself. In 1977, Sime et al. stated that “computer programming could be made easier” for novices [144, p205]. Arguably, this is still true today and novices need to be facilitated with realistic expectations and suitable environments [103]. Regardless, learning to program can be complicated, involving a range of languages, tools, and approaches [104].

A 2018 review cited over 700 references in CS1 research from 2003-2017 [104] (starting with > 5000) and another focused just on the Association for Computing Machinery’s (ACM) Special Interest Group on Computer Science Education
(SIGCSE) Technical Symposium, processed hundreds of papers [14]. The effects of many variables in the CS1 experience have been explored, including various teaching approaches and languages [93], as well as features in programming environments such as level of Integrated Development Environment (IDE) assistance [46], simplicity [134], graphic output [88], and support for collaboration [74].

2.2 Programming Environments

2.2.1 Programming Environments in the Classroom

Most typical university-level CS1 courses involve theory and practice. It is common for students to put programming knowledge and theory into practice using a first programming language [135, 14] while writing their code in a programming environment [104], editor, or Integrated Development Environment. As a result, students end up facing multiple challenges as they learn the theoretical aspects of programming and familiarize themselves with the software development process [104] (see Figure 2.1), which aim not only to enable students to put theory into practice, but also to expose them to the often strenuous task of debugging and testing their code. Debugging has been mentioned as a core programming concept and novices need to have the skill as part of the “whole set” of programming skills [101]. Tools that facilitate the effective learning of debugging could help novice acquire this skill more effectively. Ideally, these tools should be educational and functional at the same time.

2.2.2 What is a Programming Environment?

Programming environments are software applications that programmers use to implement, debug, and test their programs. In most programming environments, there are three core aspects of interaction between the system and the user: (1) the user is writing code, (2) the system is evaluating the code, and (3) provides 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 processes the given code (often through
Figure 2.1: The outline of the programming process of a novice programmer.

multiple passes, depending on the compiler) 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.

Students usually engage with programming while following small cycles of editing, compiling, and executing code [72] using a programming environment. Since environments act as a medium through which users create and interact with programs, it is important that this interaction is appropriate for student learning and efficient in assisting them in improving skills such as syntax mastery in order to overcome issues that may disrupt the core learning experience. Most instructors would agree that learning to program should not be complicated with learning the intricacies of an elaborate environment, or tools that otherwise hinder the learning of programming concepts. Although some educators may encourage their students
to use a pedagogical environment, this is not always the case and students may end up using IDEs designed for experienced programmers. At a minimum, more advanced environments would likely impart a higher extraneous cognitive load on the student.\footnote{Although, the same argument can be made for poorly-designed educational environments.} For novices, there is also a particularly important feature that all environments should provide: constructive and informative (ideally formative [76]) feedback on the code written by the student. This puts error messages, their mechanisms, and their presentation in the spotlight.

The fundamental mechanisms that are common to almost all programming editors are (1) compilation mechanism(s), which allow the source code to be passed through a compiler and checked against the specification of the programming language, and (2) compiler (or programming) error messages that serve as feedback regarding the state of the source code when compiled without success [13]. There are multiple ways that these mechanisms are facilitated by modern programming environments, however there is little research on their impact on novices, and ultimately what constitutes their effectiveness in the programming process, a point that has been under discussion since at least 1987 [118].

### 2.2.3 Compilation and Error Message Presentation

Compilation (or interpretation) mechanisms and feedback via diagnostic messages are core aspects of programming and at their most fundamental level these features are common to all environments. Compilation serves two purposes: first, to check that the source code is well-formed according to the specification of the programming language [13]; and second, to translate the source code into an executable representation. If violations of the language’s specification exist, feedback is presented to the user as programming error messages (often called “compiler error messages”) [13]. When this occurs the compilation is halted and no executable is able to be produced until the errors are rectified [152]. When errors are present, it is up to the programmer to interpret the error message(s) and fix the errors in their code [11, 13]. There is a long history of research in the computing education community that investigates compiler error messages and their effects on novices, since they are considered problematic [127, 125, 9, 18, 40] (see an example in
Figure 2.2). \(^2\) Strong cases have been made recently that more in-depth work is needed [13].

Programming editors vary in terms of scope and levels of assistance [46], resulting in a plethora of different designs and supported tools, usually packaged within large Integrated Development Environments [57, 114] that feature customization options that may aid the programmer during the programming process [130]. These environments also have various characteristics that are inherent to the environment, such as specific programming language support, user interfaces, feedback mechanisms, and core facilities, such as build and compile procedures [109]. Many programming environments used in educational contexts are in fact industry-grade complex tools supporting a wide variety of features aimed to assist professional developers. Given the vast array of tools available and the range of features that each employ, it is reasonable to hypothesize that programmer behaviour will be affected by the programming environment used, particularly during learning. Although some of these systems are designed to assist experienced programmers,

\(^2\)Error messages can sometimes have serious repercussions [97].
they are often plagued by complicated mechanisms that can be overwhelming for novices [130]. Many educators are aware that learning programming extends far beyond syntax; although this remains a barrier [149]. However, it is often overlooked that students also need to learn how to work with the programming environment. Exposing novices to an IDE with a plethora of tools and features can potentially impose extra cognitive load or otherwise waste cognitive channels that could be utilized more constructively in the learning process [62]. Further, there is evidence supporting a relationship between the environment and student metacognition [126, 68] which demands further exploration. However, this does not necessarily mean that environments should be as simplistic as possible. Although such approaches surely have their merits, they are often anecdotally supported [46].

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, that 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. Learning the intricacies of such IDEs takes time that could be better utilized focusing on the actual programming tasks. Research on the exact effects that different supporting features embedded within these systems induce when it comes to programming behaviour – especially for novices – is scarce.

2.2.5 Lack of Empirical Evaluation in Programming Environments

Designing or even selecting an environment for use ideally takes into account evidence of the impact that individual features have on novice programming behaviour. In order for developers to include effective functionalities and feedback
mechanisms when designing environments (pedagogical or professional), they should ideally be basing decisions on empirical evidence. A better understanding of how the interaction between novices and environments manifests itself can allow for advancements in the field of human cognition during programming that could inform Human Computer Interaction researchers and User Experience designers on how to improve existing educational environments and develop new ones by incorporating grounded principles that are more optimal for learning into the design process. This however, is a complex undertaking as there is a multitude of context-dependent variables starting with the students themselves and the language being used. To achieve this, more studies should focus on establishing evidence, frameworks and guidelines for designing these tools. This would ideally result in environments that benefit users in terms of programming patterns, compilation habits and the usefulness of information that users receive from the environment.

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 [148, 149]. 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 [65]. This makes it all the more important to examine feedback mechanisms individually and thoroughly by studying their effect on users.

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 thesis and answering them requires multifaceted approaches – possibly driven from different perspectives.
2.3 Related Work

2.3.1 Development of Prototype Tools

Improving the programming experience is an active field in software engineering and computing education research. Research on tools aimed at assisting novices typically involve the development of full or prototype environments [37, 36, 74], intelligent tutoring systems [30, 31, 29], plugins for already existing environments [130, 138], new languages [51, 38, 91, 44], and combinations of the above. For example, Whyline, an interactive debugging tool for Java that provides information on a program’s execution was found to be effective in terms of users’ debugging speed [88]. JavaWIDE is a browser-based IDE that was designed to assist programming collaboration and some of its features were widely accepted by students who used it [74]. Reis and Cartwright [130] built a plug-in for Eclipse that supported DrJava’s [2] programming interface. Santolucito et al. [138] extended an IDE in order to incorporate automatic compilation as they deemed it a novel approach for programming. Decaf is a Java editor that ‘enhances’ standard Java compiler error messages [15]. Sometimes evaluations of these tools are performed, but evidence-based approaches [147] supporting their use, explaining their behavioural effect on novices or highlighting their strengths and weaknesses are rare. Evaluation of programming tools that involves a wide breadth of approaches [34] urges for even more focus on exploring the effects that different mechanisms have on novices and instead of relying on anecdotal evidence from students, educators or advice from experts and practitioners.

2.3.2 Impact of Programming Tools

Research on the impact of programming environment features on student behaviour is relatively scarce. Often such studies focus on introducing prototype tools without presenting rigorous empirical findings on how these tools affect programmer behaviour and they are assessed based on “opportunistic” criteria [60]. Some studies have stressed the need and importance of investigating the effects of environments on novice behaviour, but few have taken place. Burnett et al. [28] investigated differences in environment feature preferences across genders, indicat-
ing that a more thorough and multi-layered investigation of their suitability needs to be conducted. Kelleher and Pausch [85] provided a taxonomy of programming environments and argued that it is important for beginners to avoid becoming entangled in the mechanics of the programming process and focus on the actual task. McGill and Decker [112] proposed a taxonomy to classify tools, languages and environments based on categories that emerged through a systematic literature review. Xu et al. [157] provided a meta-analysis that examined block and text-based programming environments, and argued that there exists great heterogeneity between different studies, suggesting the need for a more intensive and consistent effort from the research community. In [46], three different IDEs providing differing levels of assistance were assessed with students. They found that novices use some moderately assistive IDEs more effectively, compared to those with lower levels of assistance. However, syntax highlighting (as used in many IDEs), has recently been suggested to not have a substantial effect on the program comprehension of hundreds of novices programming in Java [62], although the authors acknowledged considerable limitations in their study. A slightly older study (albeit with only 10 participants) found that syntax highlighting did reduce task completion time and context switches, noting that no effect on gaze fixations (number or duration) was found [140]. Regarding transitioning between environments, Weintrop and Holbert [154] investigated users switching between settings in a dual-modality environment of block-based and text-based programming.

2.3.3 Compiler Error Messages

Most compilers generate a list of error messages if rules of the programming language are violated. One research area that has seen increased attention in recent years is on Compiler Error Messages (CEMs) [11, 41]. Research on CEMs has been conducted as they are highly problematic, particularly in terms of their interpretation [13, 43, 42]. In some languages, CEMs 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 [18]. 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.

Substantial research has been conducted regarding CEMs [13, 152], including how developers utilize them [9], their impact on novices [119, 125, 150], and investigations on their effective enhancement [127, 32], theory-driven design [8], error message content and presentation [81, 78], categorization [111], and improvements [15]. Another strain of research is focused on identifying common errors that students make during programming so that research on the improvement of messages is directed towards the most common ones [4].

It has been found that students do read CEMs [7], and that they may be able to provide a mechanism to identify at-risk students [17]. Also, the effects of their wording, presentation, and enhancement/improvement have been explored [127, 16, 40]. A 2019 study found that changing the presentation of CEMs had a positive effect on program comprehension rates, resolution rates and user preference [48] – specifically, the spacing and color of text, and employing progressive disclosure. However the sample size was small – 52 (13 in each group) and statistical significance and effect size were not determined. Additionally, the participants were developers (not students) programming in the Dart language on the Flutter platform. Regardless, there is not yet a consensus as to whether altering the presentation or content of CEMs has an effect on learning and more work is needed [16].

Work on the effects of error message content and presentation is more extensive than that for compilation mechanisms. Becker et al. published an Innovation and Technology in Computer Science Education (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 [13]. Specific to the research described in this thesis is the presentation of Java error messages in BlueJ [94]. 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 [110].
2.3.4 Novice Programming Behaviour

A major part of compilation-related and programming behaviour research aims to identify common errors and the time required to fix them or the evolution of error activity over time [82, 1, 4, 26, 22, 129], as well as common fixes that students use to correct their programs [39], and error identification strategies [136, 155]. Other work aims to identify “at-risk” students by looking at their programming patterns [33, 49, 17], apply programming efficiency measures to investigate the behaviour of users in different regions [73, 133], automate the identification and correction of syntax errors [139], and developing standards for programming process data [128]. 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 [72] and repeated error density [10]), such efforts are affected by context (e.g. institution, students’ motivation complex) based on replication studies [159].

2.3.5 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! [66], Scratch [131] 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. Mechanisms for text-based languages range from similar compile-and-execute buttons as in Visual Studio\(^3\) to compile-on-save optional features in several IDEs. Other environments have separate compile and execute buttons. Several IDEs, such as Eclipse, incrementally compile as code is edited.\(^4\) BlueJ 3 was a manual click-to-compile editor, but BlueJ 4 introduced incremental compilation (automatic error checking).\(^5\) while retaining the manual compile button allowing both automatic and manual compilation mechanisms. In order to execute the code, users have to navigate to the main window of BlueJ (which is separate from the editor) and interact with an automatically generated UML diagram of their program using their mouse.

---

\(^3\)docs.microsoft.com/en-us/visualstudio/windows/?view=vs-2017

\(^4\)www.codejava.net/ides/eclipse/why-does-eclipse-use-its-own-java-compiler

\(^5\)bluej.org/versions.html
Lubin and Chasins [102] identified that compilers serve as means to correct mental models of the problem domain and as a dynamic task-direction mechanism. They also found that often functional programmers invoke the compiler even when they are certain that their code will not successfully compile. They found that reasons of using the compiler go beyond functional use, that often programmers want intermediate feedback before their code is compilable, and that some error message may be caused by programmers explicitly compiling because they want compiler feedback.

Jadud [71, 72] provided a behavioural overview of students using an early version of BlueJ in 2005/6. One of the aspects examined was the time between compilation events. Jadud made multiple observations regarding compilation behaviour while at the same time developing the Error Quotient [72, 124] measure for predicting programming performance. One of the findings was that students recompile their programs very quickly if their code contains an error compared to when their code is correct. Students also tend to write a lot of code without paying attention to errors, then went back to correct any errors, a strategy that seemed ineffective.

### 2.4 Problem Statement

The effects of compilation functionality in programming environments on programming behaviour and how it affects the users is under-studied, which is puzzling, considering the fact that it is one of the most critical aspects of interacting with code. For example, Visual Studio features compilation and execution triggered by the same user action; Eclipse on the other hand compiles while the code is modified. Often these settings are configurable to provide flexibility and be in accord with the user’s preference. Regarding pedagogical environments [131, 66], it is a common practice to merge compilation and execution into one during the design and not distinguish between two separate actions, particularly for more simple environments. However, why a given environment offers a given choice is most often not clearly based on evidence.

Research on delivering effective feedback through the environment is mostly accomplished using compiler error messages [152, 13, 121]. Questions include if
users read them, how much time they spend on reading them, how they can be effectively worded [9, 7] and what are the most common mistakes that students make during programming [26]. However, there is not enough focus on the manner in which messages are delivered to the user and how users’ reactions are affected by different presentation methods. Some environments convey the compiler output untouched. Others employ multiple combinations of color, highlighting, red squiggles, lightbulbs, popups, tooltips and more. An alternative approach is that showing less information provides more simplicity, and deters users from getting lost among multiple cascading errors and other distractions. Repeatedly addressing the first error message has been positively correlated with assignment grades [115] and is advocated by many [18]. Again however, empirical evidence on these choices is scarce. Both the process by which the user compiles code and the manner in which error messages are reported to the user differ depending on the environment used. Further, these can evolve with time. Often, this evolution is an almost mystical “wind of change” that large-scale IDE designers and developers implement in new versions. Once a feature is introduced in one IDE, it soon spreads to others. However, it is unclear if the decisions behind releasing new programming environment mechanisms and features (or modifying existing ones) are based on any empirical research, although, these decisions may stem from unpublished research. In addition, although nowadays IDEs feature a vast number of different intricacies and plugins, the core of these environments is subject to a very slow change. This has come to be expected, as these environments evolve in tandem with the evolution of the programming process and the evolution of programming languages which are influenced by market pressures [25]. However, from a user and human computer interaction perspective, it is often unclear if changes in programming environments are tested in terms of the effects on programmer behaviour before release, and there is a lack of research on such after release. In short, many changes in programming environments come and go over time, but why, and the effects they have on programmer behaviour — particularly students — are not questioned or investigated enough. This is not a contemporary phenomenon; in 1996, Pane and Myers [122] provided a collection of design principles for developing assisting programming environments for novices, raising concerns about programming environments not implementing design paradigms resulting from research. Instead,
environments are built and improved around technical goals, such as speed, and size.

From a user experience perspective, compilation can be engineered to work in two broad ways. First, it can be invoked by the user taking some action outside of the act of writing code, such as clicking a dedicated button or executing a keyboard shortcut. In this sense, compilation is strictly manual and only occurs when the programmer decides to stop writing code and chooses to evaluate the soundness of their source code's syntax in its present state. Second, many contemporary programming environments feature “automatic background compilation”. This is often triggered by actions involved in writing code itself such as keystrokes, line changes, etc. There is very little strong evidence supporting advantages of one over the other, but no shortage of conjecture that automatic compilation provides intermediate feedback [87], which may be more beneficial to the programmer than writing large chunks of code and performing a compilation after many changes. In addition, it has been suggested that dynamic code execution mechanisms facilitate the early correction of users’ mental models and assist in avoiding misconceptions instead of burdening the user until explicitly requesting feedback from the programming environment [100].

In this work, we present a thorough exploration of how novices engage with two different compilation and error message presentation functionalities (we will be referring to these using the broader term “compilation mechanisms”). We will investigate this subject through interactions of novices with the BlueJ programming environment, and more specifically with two versions BlueJ 3 and BlueJ 4 (explained later). Our aim is to establish a better understanding of programming behaviour and to what degree changing the compilation mechanisms alters the interaction between the novice programmer and the system. We select BlueJ 3 and BlueJ 4 for three primary reasons. First, amongst the largest changes between the two versions are the compilation functionality and error message presentation. Much of the other features remain the same. This allows us to reasonably conclude that behavioural differences that arise are due to these factors. Second, programming process data from both versions are automatically captured by the Blackbox data collection project [23], allowing for substantially sized samples of novices to be included in our studies. Finally, the simplistic design of BlueJ offers an almost...
direct exposure of novices programming in Java to the compilation mechanisms present in each BlueJ version, without any further assisting features. This allows us to reasonably generalize our findings beyond the BlueJ environment, specifically to those that feature compilation mechanisms closely similar to BlueJ.

2.5 BlueJ

BlueJ [94] is an introductory pedagogical programming environment for text-based programming in Java, and Java is a popular CS1 language [14]. BlueJ is utilized in multiple regions and across many introductory programming courses. It is used by thousands of CS1 students at an annual rate. Evolving from “Blue” [92] in the late 1990s, the environment has experienced changes and updates through different versions for over two decades as BlueJ by a research team currently based in King’s College London.

2.5.1 BlueJ version 3

BlueJ 3 (2010-2017) is a standard “click-to-compile” environment that allows users to check their code for compiler errors by clicking a dedicated button at the top left of the BlueJ window or by using a specific keyboard shortcut. When the code is compiled, if there are errors present, then only the code area that the compiler detects as the first error’s location is highlighted in red. The error message corresponding to the first error is displayed at the bottom of the editor in a dedicated pane, allowing one error message at a time [153]. This was a design decision that was carried over from Blue, described in detail in [89]. Even if there are multiple errors present in the source code, the environment only allows the display of the first. This has two immediate benefits: simplicity, and avoiding multiple error messages and issues they bring (such as cascading and spurious messages) [18]. Figure 2.3 shows a typical BlueJ 3 editor window.
2.5.2 BlueJ version 4

In 2017, BlueJ 4 was introduced, featuring changes in the compilation mechanism and the presentation of errors and error messages to users. BlueJ 4 features "automatic background checking" which is triggered whenever users change lines in the source code by moving the cursor. BlueJ 4 also retains the manual compilation functionality present in BlueJ 3 by including a "Compile" button at the top left of the editor window. Whenever errors are detected in the source code by the compiler, all the code areas that are recognized as containing an error are underlined in red error squiggles. In addition, red indication bars appear on the left of the respective source code lines. By default, BlueJ 4 does not present any error messages. In order to see a message, users have to navigate with their mouse or keyboard to an underlined area that was identified as an error location. This triggers a small black box that contains the corresponding error message in white font right next to the error location on the code. In contrast to BlueJ 3 which only presents the error message corresponding to the first error, users in BlueJ 4 are able to see error messages for all errors present in their source code. Additionally, clicking the compile button multiple times (in the presence of multiple errors) causes error messages to be presented from first to last, in round-robin fashion. BlueJ 4 also supports Stride [91, 45] (a fusion of text-based and block-based programming called
frame-based editing [5, 154, 90]). Figure 2.4 shows a typical BlueJ 4 editor window. Apart from the aforementioned differences from BlueJ 3, BlueJ 4 is mostly similar in terms of the features and general appearance of the environment. More details on the differences across BlueJ versions can be found at the BlueJ website.\footnote{https://bluej.org/versions.html}

Figure 2.4: BlueJ version 4 editor window.

### 2.5.3 BlueJ version 5

In January 2021, BlueJ 5 became the major BlueJ version. As with every new version of BlueJ, new features and improvements were added to the environment. However, the compilation mechanisms in BlueJ 5 have not been altered and remain identical with those present in BlueJ 4 (automatic and manual compilation and on-demand any/all error messages presentation). The rest of the environment features, as well as the general look of the user interface are mostly unchanged. Thus, the findings regarding the interaction between novices and BlueJ 4 that are presented through the studies described in this thesis can also be applied to the newest version of BlueJ.
2.6 Blackbox

The team that develops and maintains BlueJ also developed Blackbox [24, 23], a data collection project that records Programming Process Data (PPD) [70] generated from BlueJ users who agree to participate in the data collection. Blackbox contains a formidable amount of programming information from million of users, such as compilation syntax states, error messages, source code snapshots and other interaction events, each accompanied by a timestamp, a user ID, etc. Blackbox records the programming activity as events that are labeled automatically based on their type. All analyses of programming process data in the present work (Part II and III) have been conducted using data mined exclusively from the Blackbox database. The experiments described in this work feature users of the two versions of BlueJ implementing source code in the Java programming language.

---

7BlueJ users get an option for participating in the Blackbox data collection the first time they launch BlueJ.
Part II

The Triptych of the Interaction Between Novices and Programming Environments
CHAPTER 3

INTERACTION BETWEEN PROGRAMMING ENVIRONMENTS AND NOVICES EXPOSED TO MULTIPLE COMPILATION MECHANISMS

Abstract

It is generally accepted that learning to program could be easier for many students. One of the most important components of this experience is the programming environment. Novices learn in a variety of environments, from basic command-line interfaces to industry-strength IDEs. These environments can differ substantially in compilation behaviour and error message presentation – arguably two of the most important mechanisms through which users interact with the programming language. In this study, we utilize Blackbox data to compare the programming behaviour of thousands of users programming in Java, who all used BlueJ versions 3 and 4. We find that the programming experience and behaviour of these users can be substantially affected by changes in these mechanisms, causing numbers of manual compilations, successful compilations, and error messages presented in each version to differ, in cases, markedly. Our results provide evidence on how changes in programming environment affect user behaviour in conditions that reasonably control for variables other than the programming environment.
3.1 Introduction

In this study, we explore the role of the programming environment – specifically that of compilation mechanisms – on novice programmer behaviour. It has been shown that the presentation of word problems can affect student behaviour [21] and here we extend this idea to the presentation of error messages. Additionally, it has been shown that the number of error messages displayed to students has an affect on the frequency of error messages [18]. We utilize the BlueJ editor and the Blackbox database, providing conditions that reasonably control for variables other than the programming environment. Through the Blackbox programming process data, we explore (1) the number of displayed compiler error messages over time, (2) the number of manual compilations over time, and (3) the percentage of successful manual compilations that occur when novice programmers are exposed to both BlueJ 3 and BlueJ 4. The two BlueJ versions feature fundamental differences in the manner that compilation mechanisms are facilitated within the editor. These changes present an excellent opportunity for research on the impact of the effect of these features on novices' programming behaviour in the relatively controlled ecosystem of BlueJ/Blackbox. The present study aims to answer the following research questions:

RQ3.1: How did changes in BlueJ affect the frequency of error messages presented to users?

RQ3.2: How did changes in BlueJ affect the frequency of manual compilations? (In BlueJ 3 users have to manually click to compile, but in BlueJ 4 this is possible but not required.)

RQ3.3: How did changes in BlueJ affect the percentage of successful manual compilations?

RQ3.4: How are these results affected by different choices of heuristics for calculating programming session time?

The contribution of this work is in providing reliable information on the effects of changes in a programming environment with conditions that reasonably control for variables other than those changes.
CHAPTER 3. INTERACTION BETWEEN PROGRAMMING ENVIRONMENTS AND NOVICES EXPOSED TO MULTIPLE COMPILATION MECHANISMS

3.2 Data

In order to find a study group, the Blackbox database was queried for users who within a specified time-frame had BlueJ sessions in both versions. Eventually, users with activity in both versions between October 2017 and January 2018 were selected. The choice of these months was not arbitrary. As BlueJ version 4.1.1 (the first BlueJ 4.x version that was “Blackbox-enabled”) came out in late September 2017, it was assumed that many users would switch from version 3 to 4 in the following months. Our query resulted in 3176 BlueJ users. However, it was discovered that many users had atypical behaviour – writing code but never compiling, etc.¹ These users were discarded, leaving 2062 users. The number of users in the analysis that is described in the following sections fluctuates as outliers were removed as discussed later in Section 3.3.2, so the number of users who were present in each case will be reported when required. For the selected users, we gathered the following: (a) meta-information such as BlueJ and Java versions, (b) events such as compilations, (c) reason of compilation (for BlueJ 4 only - manual/automatic) and state of success, (d) error messages generated by the compiler and if they ended up being presented to the users, and (e) timing information for sessions and events.

3.2.1 Data Filtering

It is well known that Java Compiler Error Messages (CEM) may not be stable across Java versions [24]. To achieve consistency in terms of CEMs, the current study focuses on events that were linked to a single Java version. Since BlueJ 4 supports Java 8 and above, only events that were linked to Java 8 (the most commonly used in both BlueJ versions) were included. The results on CEMs refer only to actual errors in code that failed to compile (not counting warnings, etc.).

We reduced the data in post-processing so that the day the first user used BlueJ 4 (September 18, 2017, Blackbox time) was equidistant to the start and end dates of the data period. This resulted in a data period ranging from January 14, 2016 to May 24, 2019. We also removed outliers as discussed in Section 3.3.2.

¹This could be the result of network interruptions.
3.2.2 Calculating Programming Time

Every time a user action occurs in BlueJ, an event accompanied by a timestamp is recorded in Blackbox. When users open BlueJ, an event that marks the start of a session is recorded in the database. Another event that marks the end of the session is recorded when users terminate BlueJ. Thus, the duration of the session can be calculated by subtracting the timestamp of its ‘begin session’ event from the timestamp of the ‘end session’ event. However, early in the analysis it was discovered that the number of sessions that had an ‘end session’ event recorded in Blackbox did not match the total number of sessions started. This could be due to many factors, such as users experiencing a drop in their internet connection or BlueJ terminating unexpectedly. Additionally, when an event fails to be sent from BlueJ to Blackbox, the rest of the events in that session are not recorded. For that reason, we used the last recorded event (even if not an ‘end session’ event) from each session as the session-ending event (see Figure 3.1). Thus, a session was considered complete even if it didn’t have an actual ‘end session’ event. The number of sessions with no ‘end session’ event was too high to justify not including these sessions in our analysis. We note this as a potential threat to validity in Section 3.5. The total programming time for \( n \) users with \( m \) recorded sessions in Blackbox can be expressed as Equation 3.1.

\[
\text{Total programming time} = \sum_{i=1}^{n} \left( \sum_{j=1}^{l} t_{ij} \right)
\]

Figure 3.1: Schematic representation of how programming time is calculated from Blackbox data for a single user. The first term in the subscripts refers to the number of each session in chronological order ranging from 1 to “\( m \)”, while the second to the number of each event in that session in chronological order ranging from 1 to “\( l \)”. “\( l \)” always represents the last recorded event of a session.
CHAPTER 3. INTERACTION BETWEEN PROGRAMMING ENVIRONMENTS 
AND NOVICES EXPOSED TO MULTIPLE COMPILATION MECHANISMS

(3.1) \[ T = \sum_{i=1}^{n} t_i = \sum_{i=1}^{n} \sum_{j=1}^{m} (t_{j\text{last}} - t_{j\text{first}}) \]

Where \( t_i \) is the total programming time for user \( i \), and \( t_{j\text{first}}, t_{j\text{last}} \) are the first and last recorded events for session \( j \) respectively.

3.3 Methodology

3.3.1 Per-User Calculation

The first step of our analysis was to profile each user, gathering information related to answering the research questions. For each user’s activity, the following information was collected and/or calculated for both versions of BlueJ independently:

- \( H \): Programming time spent on BlueJ (\( t_i \) from Equation 3.1)
- \( C \): Number of compilations (manual for BlueJ 4)
- \( SC \): Number of successful compilations (manual for BlueJ 4)
- \( DCEM \): Number of displayed compiler error messages

Based on this information, the following metrics were calculated:

- Compilations per hour (manual for BlueJ 4):
  \[ CpH = \frac{C}{H} \]

- Percentage of successful compilations (manual for BlueJ 4):
  \[ PSC = \frac{SC}{C} \]

- Displayed compiler error messages per hour:
  \[ DCEM_{pH} = \frac{DCEM}{H} \]
3.3.2 Removing Outliers

Due to the lack of contextual information on the programming activity of BlueJ users, it is difficult to establish if users are actually programming for all of their session time as recorded by Blackbox. It is highly possible that many users leave BlueJ open for long periods of time, while they engage in other activities. There were some users with extreme values of H – for instance, one user had several thousand hours recorded for BlueJ 3 – which is extremely unlikely to represent genuine programming time. In some cases, less extreme yet prolonged activity might have originated from an institutional machine, where many physical users have access. Moreover, there were users with high DCEMpH (e.g. >200).

We removed outliers whose programming time (measured in hours) was greater than three standard deviations above the mean (BlueJ 3 $M=144.8$, $SD=301$; BlueJ 4 $M=58.7$, $SD=163.1$). This was applied separately for both BlueJ versions, leaving 1705 users – see Figure 3.2. The same rule was applied on DCEMpH (BlueJ 3 $M=7.3$, $SD=8.7$; BlueJ 4 $M=11.1$, $SD=15.5$) leaving 1661 users – see Figure 3.3.

![Figure 3.2: Programming time in BlueJ 3 & 4 for each user. All but the lower left quartile were considered outliers.](image-url)
3.3.3 Global Calculation

A second approach for calculating \( Cph \), \( Psc \), and \( DCEMpH \) was also undertaken. Instead of profiling individual users, we aggregated each metrics’ values from all users, after removing outliers as described in Section 3.3.2. This simple strategy served as a ‘bird’s eye view’ of the data from a BlueJ 3 vs BlueJ 4 perspective, compared to looking at individual users. It was clear that there was a difference between the two methods for each metric, however, the values observed were small in terms of \( DCEMpH \), due to large variations in users’ programming time. We discuss this in Section 3.4. Other than \( DCEMpH \), changes in the global metrics roughly correlated with those when calculated on a per-user basis.

3.3.4 Trimming Inactive Intervals

As reported in Section 3.3.3, \( DCEMpH \) values from the global calculation were small (e.g. 2.6 for BlueJ 3, to 4.0 for BlueJ 4) because of variations in \( H \) across users. In Section 3.3.2, it was stated that many users might leave the software running while engaging in other activities. Based on that, it seemed appropriate to explore how \( DCEMpH \) would be affected by removing long time intervals between
CHAPTER 3. INTERACTION BETWEEN PROGRAMMING ENVIRONMENTS AND NOVICES EXPOSED TO MULTIPLE COMPILATION MECHANISMS

Table 3.1: Programming time (H), displayed compiler error messages per hour (DCEMpH), manual compilations per hour (CpH), and percentage of successful manual compilations (PSC) in both BlueJ versions per user (* the first two rows are global calculations and values for H in this case are discussed in Section 3.3.3). We report p values that are < .001 as such, per APA guidelines [6] which are recommended by [137]. We used an alpha level of .05 for all statistical tests.

<table>
<thead>
<tr>
<th></th>
<th>H</th>
<th>DCEMpH</th>
<th>CpH</th>
<th>PSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlueJ 3* μ</td>
<td>171202.8</td>
<td>2.6</td>
<td>6.1</td>
<td>0.6</td>
</tr>
<tr>
<td>BlueJ 4* μ</td>
<td>62846.2</td>
<td>4.0</td>
<td>4.1</td>
<td>0.8</td>
</tr>
<tr>
<td>BlueJ 3 - μ (SD)</td>
<td>103.0 (180.8)</td>
<td>6.6 (6.1)</td>
<td>14.6 (12.7)</td>
<td>0.5 (0.2)</td>
</tr>
<tr>
<td>BlueJ 4 - μ (SD)</td>
<td>37.8 (73.9)</td>
<td>9.874 (9.9)</td>
<td>10.9 (13.9)</td>
<td>0.7 (0.2)</td>
</tr>
<tr>
<td>Mann-Whitney U (p &lt; .001)</td>
<td>-</td>
<td>1147333.5</td>
<td>1078703</td>
<td>576532</td>
</tr>
<tr>
<td>A_w</td>
<td>-</td>
<td>0.42</td>
<td>0.61</td>
<td>0.21</td>
</tr>
<tr>
<td>Robust d</td>
<td>-</td>
<td>-0.30</td>
<td>0.39</td>
<td>-1.14</td>
</tr>
</tbody>
</table>

events. A filtering algorithm that iteratively calculated the timestamp difference between users’ consecutive events, determining if the difference was greater than a predefined threshold was employed. In these cases, the time interval between the two events was subtracted from the total H that was calculated by the method described in Section 3.3.3. We ran this algorithm seven times for thresholds set at 1, 5, 10, 15, 30, 60, and 90 minutes, calculating DCEMpH each time. This method serves as reinforcing evidence for increased DCEMpH in BlueJ 4 as discussed in Section 3.4 below. As CpH is dependent on H in the same way as DCEMpH, and the difference in DCEMpH was quite consistent for different trimming intervals, we did not re-calculate CpH for different intervals.

3.4 Results and Discussion

We ran experiments in two pipelines: (1) raw data, and (2) with outliers removed. Both the per-user and global data discussed in Sections 3.3.1 and 3.3.3 were run through each pipeline. The results from both pipelines were very similar in terms of differences between the two BlueJ versions for all the metrics that are examined throughout this study. For that reason, only the results from pipeline 2 are presented and discussed in this section.
A Shapiro-Wilk test [143] revealed that the distributions of our three metrics were not normal. We therefore used a Mann-Whitney $U$ test (two-tailed) [107] for statistical significance. As Cohen’s $d$ technically may not be robust for some non-parametric distributions and heterogeneity of variances between samples [99], the probability of superiority effect size measure ($A_w$) was calculated and converted to a more robust ‘Robust $d$’ metric [99]. $A_w$ indicates the probability of a random element in a sample being larger than a random element in another sample. A 2016 study of six effect size measures for non-normal, non-homogeneous, two independent samples cases (including Cohen’s $d$, Robust $d$, and $A_w$) found that $A_w$ and the Robust $d$ were generally robust to five possible violations including distribution normality [99]. Our results are in Table 3.1.

It is often possible to use parametric tests with non-parametric data. For instance, many studies dating back to the 1930s consistently show that parametric statistics are robust with respect to violations of assumptions of normality and generally, parametric methods can be utilized on non-parametric distributions without concern [120]. We therefore also performed $t$-tests (two-tailed) and calculated Cohen’s $d$ [35] to measure effect size. The $t$-tests showed very similar results to the Mann Whitney $U$ tests and our values for Cohen’s $d$ showed similar results to the Robust $d$. Specifically our values for Cohen’s $d$ were: DCEMpH (-0.39 – between small and medium effect size); CpH (0.28 – same); and PSC (-1.10 – large effect size). Our Cohen’s $d$ interpretations are from [77]. The largest difference between respective Cohen’s $d$ and Robust $d$ values was 0.11. This demonstrates that we would draw very similar conclusions from the data regardless of the violation of normality assumptions.

### 3.4.1 RQ3.1: Displayed Compiler Error Messages

Our first research question was: How did changes in BlueJ affect the frequency of error messages presented to users?

Table 3.1 shows a statistically significant increase in DCEMpH from BlueJ 3 ($M = 6.6$) to BlueJ 4 ($M = 9.7$). $A_w$ indicates that a random BlueJ 4 value has a $1 - 0.42 \approx 68\%$ chance of being larger than a random BlueJ 3 value, and the Robust $d$ indicates an effect size of -0.30. See Figure 3.4 for a comparison of the
distributions. The global calculation method shows a very similar increase (about 1.5×), from 2.6 in BlueJ 3 to 4.0 in BlueJ 4.

It is appropriate to point out that users choose to see these messages in BlueJ 4 as they are not presented by default – the user needs to hover over the error to see the message. In addition, as Blackbox records only the first time each error message is displayed for a given compilation, the values presented in this study account for the minimum number of times that users actually see these error messages. These results may support two hypotheses: novices take initiative to explore the feedback that BlueJ 4 provides, and users feel they need (or want) more compiler error messages than what was provided in BlueJ 3. Further (most likely qualitative) study would be needed to confirm these hypotheses.

3.4.2 RQ3.2: Manual Compilations

Research question 2 was: How did changes in BlueJ affect the frequency of manual compilations?

Table 3.1 shows a statistically significant decrease in Cph from BlueJ 3 ($M = 14.6$) to BlueJ 4 ($M = 10.9$). $A_w$ indicates that a random BlueJ 3 value has a 61% chance of being higher than a random BlueJ 4 value and the Robust $d$ indicates an
effect size of 0.39. See Figure 3.5 for a comparison of the distributions. Again the global calculation shows a decrease similar in magnitude, from 6.1 in BlueJ 3 to 4.1 in BlueJ 4.

![Figure 3.5: Manual compilations per hour (CpH) for BlueJ 3 & 4. The orange horizontal line in the box represents the median and the green arrow the mean.](image)

Interestingly, these results show that users still exercise the manual compilation mechanism despite not having to (unlike in BlueJ 3 where manual compilation is the only mechanism). The exploration of this, despite the automatic error checking constantly informing students about the state of their syntax, may be interesting. A possible explanation may be that users who programmed in BlueJ 3 before moving to BlueJ 4 grew accustomed to interacting with in a “manual-compilation-only” manner.

### 3.4.3 RQ3.3: Success of Manual Compilations

Research question 3 was: *How did changes in BlueJ affect the percentage of successful manual compilations?*

Table 3.1 shows a statistically significant increase in percentage of successful compilations (PSC) from BlueJ 3 \( M = 0.5 \) to BlueJ 4 \( M = 0.7 \). \( A_w \) indicates that there is a \( 1 - 0.21 = 79\% \) chance that a random BlueJ 4 value is larger than a
random BlueJ 3 value and the Robust $d$ metric shows a considerable effect size of -1.14. See Figure 3.6 for a comparison of the distributions. The global calculation shows quite similar results, with BlueJ 3 at 0.6 and BlueJ 4 at 0.8.

This considerable increase in manual compilation success could be due to users manually compiling as an extra verification mechanism – if no error is being highlighted in the BlueJ 4 window, users may want explicit reassurance that their code is indeed correct (meaning it compiles). We intend to explore this in future work.

![Figure 3.6: Percentage of successful manual compilations (PSC) for BlueJ 3 & 4. The orange horizontal line in the box represents the median and the green arrow the mean.](image)

### 3.4.4 RQ3.4: Session Time Heuristics

Research question 4 was: *How are these results affected by different choices of heuristics for calculating programming session time?*

The DCEMpH values obtained by treating different time intervals between successive events as inactive (described in Section 3.3.4) are shown in Figure 3.7. DCEMpH was always higher in BlueJ 4, regardless of the threshold by which time intervals between programming events were treated as inactive. Additionally, global calculations vs per-user calculations showed similar trends in the results,
with the former producing small numbers for DCEMpH compared to the latter. Subsequently, treating different time intervals between events as inactive showed variation in the DCEMpH numbers (as seen in Figure 3.7), but the difference between the two versions is similar for each interval. We conclude that the results from the global calculation and trimming intervals methods align with the rest of the study and support our results.

![Figure 3.7: Global displayed compiler error messages per hour (DCEMpH) in BlueJ 3 & 4, treating time intervals between events as inactive according to different thresholds.](image)

### 3.5 Threats to Validity

As Blackbox data are anonymous, there is missing contextual information. As a result, all users are considered novice programmers and equal in terms of experience (and other possible factors of programming proficiency). However, as BlueJ was designed for novices, and there is no evidence of it being used non-educationally, it seems highly unlikely that a significant proportion of users in Blackbox will be advanced programmers. That being said, some users might be educators, experimenting with the software, or preparing and testing assignments. Such threats are largely unavoidable and are shared by all studies utilizing Blackbox data. Brown
provides a review of all Blackbox experiments through the summer of 2018 [23]. Many of these studies acknowledge similar threats.

Next, Blackbox might capture a displayed error message event in BlueJ 4, if the user moved the cursor on the error to fix it or if triggered by inadvertently hovering over the error (in the source) for a certain amount of time. However, given the fact that a shown error message is recorded in Blackbox only the first time it is displayed in a compilation, users might actually see the error message more than once (despite it only being recorded once), potentially countering this to some extent. This is something inherent to BlueJ and Blackbox that all studies using this data need to take into account.

Additionally, we selected users with activity in both BlueJ versions. It is likely that their BlueJ 4 behaviour was influenced by their experience with BlueJ 3 (or technically vice versa if a user went from BlueJ 4 to BlueJ 3 which is unlikely). We plan to repeat this analysis on BlueJ 4 users who did not use BlueJ 3 in future work.

Finally, our methodology of calculating programming time in Blackbox includes subtracting the first event’s timestamp in each session from the last. We noted in Section 3.2.2 that the number of sessions with ‘end session’ events was very small compared to the total number of ‘start session’ events. This might occur due to lost connectivity or by closing BlueJ prematurely/improperly. For that reason, the event that signals the end of a session in our analysis is the last one that is present in the database. This assumption might influence the programming time that was considered in calculating the metrics described in 3.3.1. It also indicates that there is most likely some missing data, however this is again shared by all Blackbox studies. In addition, we make the assumption that the calculated programming time spent on BlueJ is a realistic representation of the actual time-on-task. A recent study found that novices engage with a programming task for a median of about 8 minutes, and disengage for about 1 to 4 minutes [50]. We made efforts to account for different times of inactivity within sessions (RQ3.4) and found the differences between BlueJ 3 and BlueJ 4 to be consistent for all time intervals considered.
3.6 Conclusions and Future Direction

In this study, we compared the programming behaviour of users programming with BlueJ 3 and BlueJ 4. These versions differ drastically in their compilation mechanisms as well as their error message presentation. Our results support hypotheses that (1) users see more compiler error messages in BlueJ 4, likely because they choose to – a choice afforded by a more flexible presentation mechanism. They also show that (2) BlueJ 4 users manually compile less frequently, but (3) have more successful manual compilations, possibly indicating that these are done as reassurance that their code is correct when there is no error highlighting present. In addition, (4) the heuristics that were used for session time calculation showed consistency across results, and thus support these results further.

Future work includes exploring the psychological factors behind users’ initiative to compile manually in BlueJ 4, as part of a larger study examining users who transition from BlueJ 3 to BlueJ 4 with users who start programming for the first time in BlueJ 4. This will provide insights about the role that prior exposure to a given programming environment affects experience in another. We will also investigate the high percentage of success for manual compilations in BlueJ 4. Again this may be due to the users’ BlueJ 3 experience. It will also reveal its effect on programming behaviour, when users are exposed to a different mechanism. Exploring the effect of transitioning from BlueJ 3 to BlueJ 4 and vice versa would be a strong indicator of whether prior exposure to a specific mechanism affects how novices interact with another.

Our results support hypotheses that user behaviour is substantially affected by the programming environment, specifically by the compilation mechanism and error message presentation. Further exploration of user interaction with programming environments may yield important insights that could help pave the way toward developing tools that are evidence-based in their effectiveness.

The next Chapter includes a replication of the experiments conducted in the present study on BlueJ users who used only one BlueJ version exclusively. It also provides a more in-depth view of the metric distributions that were presented in this Chapter, as well as an exploration of their mathematical modelling.
INTERACTION BETWEEN PROGRAMMING ENVIRONMENTS AND NOVICES EXPOSED TO A SINGLE COMPILATION MECHANISM

Abstract

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. In the study described in the previous Chapter, we utilized BlueJ users who had been exposed to both BlueJ version 3 and 4. We replicate our previous experiments by utilizing a new sample of users; novice programmers who have been exposed to a single version, either 3 or 4. We find evidence that the differences in compilation mechanisms 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.
CHAPTER 4. INTERACTION BETWEEN PROGRAMMING ENVIRONMENTS AND NOVICES EXPOSED TO A SINGLE COMPILATION MECHANISM

4.1 Introduction

This study builds on the work described in the previous Chapter that investigates the effect of exposure to different compilation mechanisms as well as different error message presentation in the BlueJ 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, by including a new sample of users who have been using only a single BlueJ version. In Chapter 3, 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. 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.

4.1.1 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:

RQ4.1: How do different compilation mechanisms affect user behaviour in terms of compilation habits and interaction with error messages when users have been exposed to exclusively a single compilation mechanism?
**RQ4.2:** Can user behaviour profiles be modelled mathematically for different compilation mechanisms?

### 4.2 Methodology

The data that were used in this study were retrieved from Blackbox [24]. The initial step was to identify all users in Blackbox 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). We followed the same methodology for data pre-processing as the one described in Chapter 3 from Section 3.2.1 to Section 3.3.2.

### 4.3 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 4.5.

#### 4.3.1 RQ4.1: Distributions

Research question 1 was: *How do different compilation mechanisms affect user behaviour in terms of compilation habits and interaction with error messages when users have been exposed to exclusively a single compilation mechanism?*

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 4.1 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 4.1 to 4.3 show the relationship for all three possible pairs of these metrics.

In Figure 4.1, the Cartesian space is bisected by the $y = x$ line which represents the lower bound of the relationship between DCEMpH and CpH for BlueJ 3. This is due to the relationship between the number of compilations and the number of
Table 4.1: 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.

<table>
<thead>
<tr>
<th></th>
<th>H</th>
<th></th>
<th>DCEMpH</th>
<th></th>
<th>CpH</th>
<th></th>
<th>PSC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td></td>
<td>3</td>
<td>4</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>μ</td>
<td>18.604</td>
<td>30.485</td>
<td>10.657</td>
<td>17.102</td>
<td>21.873</td>
<td>11.625</td>
<td>0.515</td>
</tr>
<tr>
<td>SD</td>
<td>53.952</td>
<td>69.172</td>
<td>10.805</td>
<td>23.109</td>
<td>19.204</td>
<td>14.993</td>
<td>0.207</td>
</tr>
<tr>
<td>min</td>
<td>0.033</td>
<td>0.049</td>
<td>0.015</td>
<td>0.003</td>
<td>0.024</td>
<td>0.005</td>
<td>0.027</td>
</tr>
<tr>
<td>25%</td>
<td>0.670</td>
<td>0.908</td>
<td>3.253</td>
<td>2.451</td>
<td>8.367</td>
<td>2.910</td>
<td>0.357</td>
</tr>
<tr>
<td>median</td>
<td>1.556</td>
<td>3.833</td>
<td>7.056</td>
<td>8.278</td>
<td>17.050</td>
<td>7.165</td>
<td>0.500</td>
</tr>
<tr>
<td>75%</td>
<td>8.008</td>
<td>19.058</td>
<td>14.402</td>
<td>21.601</td>
<td>29.336</td>
<td>15.173</td>
<td>0.668</td>
</tr>
<tr>
<td>max</td>
<td>437.732</td>
<td>488.914</td>
<td>58.752</td>
<td>134.44</td>
<td>136.667</td>
<td>201.948</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Figure 4.1: Displayed Compiler Error Messages per Hour (DCEMpH) and manual Compilations per Hour (CpH) for BlueJ 3 and 4. Each data point represents a user.

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 4.4, which we discuss later.

Figure 4.2 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 4.2: 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.](image)

Figure 4.2 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 4.2.

### 4.3.2 RQ4.2: Modelling

Research question 2 was: *Can user behaviour profiles be modelled mathematically for different compilation mechanisms?*

In Figures 4.4(a) and 4.4(b) 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 Equation 4.1) to represent the relationship between the three metrics. Figure 4.5 shows a visualisation of the polynomial of...
CHAPTER 4. INTERACTION BETWEEN PROGRAMMING ENVIRONMENTS AND NOVICES EXPOSED TO A SINGLE COMPILATION MECHANISM

Equation 4.1 (the nearly planar surface) where \( x = PSC \), \( y = CpH \), \( z = DCEM pH \), and coefficients \( a-f \) are described in Table 4.2.

\[
(4.1) \quad z = f(x, y) = ax^2 + by^2 + cxy + dx + ey + f
\]

The surface produced by the polynomial (see Figure 4.5) 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 cannot see more compiler error messages than they have compiles. BlueJ 4 interaction seems to be more complex.

4.4 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 number of users, we believe that this is not a significant threat.

Additionally, 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.
Figure 4.4: 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 4.5.

4.5 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. 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 the user and the compilation mechanisms. The distributions of the two BlueJ versions are substantially different based on our findings in Section 4.3. In BlueJ 4, users see more error messages (either with intent or inadvertently as outlined in Section 4.4), 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 behaviour 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. The parabolic surface structured by the BlueJ 3 distribution in
the graphs is a result of the relationship between successful manual compilations and displayed error messages per compilation. As the percentage of success in compilation increases, compilations happen more frequently than error messages are displayed. 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 produce 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 Chapter 3, 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? In addition, the quantified aspect of the interaction between novices exposed to a single BlueJ version and the compilation mechanisms present shows a similar pattern with the work conducted in Chapter 3, in which the sample of users that was included in the analysis had been exposed to both BlueJ versions.

In the next Chapter, we provide an in-depth comparison of the interaction of users exposed to a single compilation mechanism and those who have been exposed to both. We additionally investigate the role that the order of transitioning across mechanisms plays in this interaction.
Abstract

In this study, we utilize Java programming process data to investigate the interaction between novices and two different versions of the BlueJ pedagogical IDE, which differ substantially in terms of compilation mechanisms. Specifically, we compare novices that used both BlueJ 3 and BlueJ 4 with those who exclusively used either and the effects of the order in which they transition between BlueJ versions. We find substantial differences between different cohorts in terms of error messages and compilation which provides evidence that programming environments play an important part in influencing the programming practices of novices. This work supports the hypothesis that the choice of programming environment significantly affects user behaviour with respect to specific programming interactions and therefore it is reasonable to expect a difference in how these environments affect learning.
CHAPTER 5. COMPILATION MECHANISMS: EXCLUSIVE VERSUS TRANSITIONING USE

5.1 Introduction

Previous findings from the studies presented in Chapter 3 and Chapter 4 showed that novice programming interaction with BlueJ shows substantial differences between BlueJ 3 and BlueJ 4. In BlueJ 4, users get exposed to more compiler error messages in the same amount of time, they compile manually less frequently and their manual compilations are more often successful. In Chapter 3, we focused on users who had used both BlueJ versions. In Chapter 4, we replicated our experiments on users who had used only one of the two, and investigated their quantified interaction with the compilation mechanisms in more detail. We found a similar pattern in terms of the interaction with compilation mechanisms across the two samples of users: in BlueJ 4, users see more compiler error messages over time, compile manually less frequently, and their manual compilation are more often successful than in BlueJ 3.

5.1.1 Research Questions

In this work, we investigate the differences in the interaction between novices and the compilation mechanisms present in the two BlueJ versions, and the effect of transitioning between mechanisms by focusing on distinct user cohorts: users that used BlueJ 3 exclusively, users that used BlueJ 4 exclusively, and those that transitioned between BlueJ versions. For the latter, we study various subcohorts further, depending on the transition order (3 → 4), (3 ← 4) and (3 ↔ 4). Our research questions are:

RQ5.1: How does transitioning between compilation mechanisms affect novice interaction regarding compilation and error messages as opposed to being exposed to a single mechanism?

RQ5.2: How does the order of transition between compilation mechanisms affect this interaction? Possibilities include: (BlueJ 3 → BlueJ 4), (BlueJ 3 ← BlueJ 4) and (BlueJ 3 ↔ BlueJ 4).
5.2 Methodology

5.2.1 User Cohorts

We initially selected two cohorts from the Blackbox database:

1. Transition Users (TR): Users who switched between BlueJ 3 and BlueJ 4 versions between October 2017 and February 2018. We chose these dates as this coincides with the introduction of BlueJ 4. This cohort includes users regardless of their transition status (e.g. a user in this cohort could be switching from BlueJ 3 to BlueJ 4 or vice versa). All users in this cohort had programming activity in BlueJ 3 and BlueJ 4. We break these users down further later.

2. Exclusive Users (X): These users were selected randomly from users who had only a single BlueJ version (either BlueJ 3 or BlueJ 4) installed on their machine during the period their data were logged by Blackbox. All users in this cohort had programming activity only in one of these two versions. We study these separately later (we refer to users who only used BlueJ 3 as X3 and BlueJ 4 as X4).

Only programming events that were associated with Java version 8 were retrieved (which is also the most common version in Blackbox for the dates studied).\(^1\) In addition, the programming activity of both cohorts was expanded to the range of the 14th of January 2016 and the 24th of May 2019.\(^2\)

5.2.2 Compilation and Error Message Presentation Metrics

After retrieving the programming events of the users as described in Section 5.2.1, the programming time (H) in hours that every user spent programming in BlueJ was calculated. This was done by summing all time differences between the first and the last programming events of every session\(^3\) for each user. This methodology presents a complication: sometimes, connection interruptions cause Blackbox to stop logging events requiring a manual means of calculating session duration. We discuss this further in Section 5.4.

\(^1\)This was done as compiler error messages are known to differ across Java versions [24].
\(^2\)The range limits are equidistant from the first day that transition to BlueJ 4 was observed.
\(^3\)A session is bounded by two distinct events sent from the user to the Blackbox database, indicating the launch and termination of BlueJ.
We used the following metrics for describing the interaction regarding compilation and error message presentation for every user: (1) Displayed Compiler Error Messages per Hour (DCEMpH), (2) Manual Compilations per Hour (CpH), and (3) Percentage of Success of manual Compilations (PSC).

5.2.3 Removing Outliers

When dealing with large repositories of programming process data like that in Blackbox, it is expected that there will be many cases of irregular activity. In our case, there were users with unrealistically high programming time (for instance, tens of thousands of hours) or displayed compiler error messages over time (for instance, several hundred per hour). Extremely high programming times can be a result of idle activity in BlueJ, whereas many compiler error messages could be triggered by stuck keyboard keys or similar hardware failures or even a book falling on a keyboard – with hundreds of thousands of users total and millions of events per day, strange things happen. Although these users were few in number, such extreme values could distort results. In order to mitigate against this, we excluded users in all TR and X cohorts independently, based on the following procedure:

1. Removal of users whose programming time in BlueJ 3 was greater than the maximum programming time in BlueJ 4. This was done to eliminate few cases where programming time was exceptionally high in BlueJ 3, something not observed for BlueJ 4.

2. Recalculation of the means and standard deviations after Step 1 and removal of users whose programming time (H) was greater than the mean increased by three standard deviations.

3. Removal of users whose DCEMpH was greater than the mean increased by three standard deviations.

5.2.4 Categorizing Transition Users

As mentioned in Section 5.2.1, the nature of the transition between BlueJ 3 and BlueJ 4 was not initially known when the TR cohort was retrieved from Blackbox.
In this stage of analysis, we classified transition users based on all three possible transition possibilities: transitions from BlueJ 3 to BlueJ 4 (we will use the acronym 3t4 later for these), transitions from BlueJ 4 to BlueJ 3 (4t3), and transitioning repeatedly between the two versions (Overlap).

### 5.2.5 Metric Restriction in BlueJ 3

In BlueJ 3, each compilation causes at most one error message to be displayed. In other words, users see a maximum number of error messages equal to the number of compilations they invoke (if all compilations involve an error). Based on this, we can define a relationship between the three metrics that are examined in this work (DCEMpH, CpH, PSC) using the formula described in Equation 5.1. We will refer to this equation in later sections as the *BlueJ 3 equation*.

\[
DCEMpH = (1 - PSC) \cdot CpH
\]

### 5.2.6 Similarity Calculation

To gain a high-level view of differences between the BlueJ 3 and BlueJ 4 distributions, we quantified the differences between versions regarding our metrics for each cohort using three approaches:

**Method 1 (M1) – Average minimum distance from BlueJ 3:** Figure 5.1 represents the surface defined by Equation 5.1. This method involves the calculation of the shortest Euclidean distance between the BlueJ 4 interaction of a user and this surface. Specifically, we defined a new function describing the distance between user interaction in BlueJ 4 and BlueJ 3. For every user, we used the Nelder-Mead downhill simplex algorithm [117] to obtain the minimized function value. This can be viewed as a process of answering the question: "What is the closest possible BlueJ 3 behaviour to this particular BlueJ 4 user’s behaviour?".

**Method 2 (M2) – Distance between BlueJ 3 and BlueJ 4 mean coordinate**

---

*Equation 5.1 can be used to describe every user's interaction with BlueJ 3 (or any environment that uses a similar compilation mechanism). This equation is in fact the inherent relationship that we attempted to approximate in Chapter 4 (see Equation 4.1). Values that do not satisfy the equation are a result of missing data in Blackbox.*

57
values: In this method, we created a hypothetical “average user” using the mean values of DCEMpH, CpH, and PSC of all users for each BlueJ version, and calculated the Euclidean distance between them.

**Method 3 (M3) – Minimum distance between BlueJ 4 mean coordinate values and BlueJ 3:** In this method, we used the mean coordinate values of BlueJ 4 (in the respective user cohort) to come up with one “average” BlueJ 4 user profile, and calculated its minimum Euclidean distance from the surface represented by the BlueJ 3 equation.

![Figure 5.1](image)

*Figure 5.1: Surface representing Equation 5.1 and each BlueJ 3 user mapped by their metric coordinates (blue triangles).*

### 5.3 Results

Table 5.2 summarizes results discussed in this section. Figure 5.2 displays the metric distributions for each of the user cohorts. Statistical tests were performed for each individual cohort to reveal the statistical significance in the difference between the metrics in BlueJ 3 and BlueJ 4. We carried out a Shapiro-Wilk test for normality [143] and after the null hypothesis for normality was rejected for all distributions, a Mann-Whitney U test for statistical significance [107] was performed along with a calculation of Cohen’s $d$ as a measure of effect size (ES) [35]. Cohen’s $d$ was calculated using BlueJ 4 as the experimental group and BlueJ 3 as the control group in all cases. Since the effect direction was consistent for every
metric in all user cohorts (increase in DCEMpH, decrease in CpH, increase in PSC), only absolute values are displayed in Table 5.2. All tests revealed statistical significance with $p < 0.05$ and the effect sizes support our results in Table 5.2 and Sections 5.3.1 and 5.3.2. As there are two results (Mann-Whitney and Cohen’s $d$), three metrics (DCEMpH, CpH, PSC) and four data cohorts (3t4, 4t3, Overlap and X), there are 24 independent results. We present the complete set of the statistical results in Table 5.1. As is evident from the plots in Figure 5.2, these distributions are obviously distinct. Further, BlueJ 3 users are observed to closely follow the BlueJ 3 Equation (5.1), displaying a much more confined behaviour while BlueJ 4 users are much more dispersed representing a wider variation in behaviour.

Table 5.1: Complete set of statistical results for all metrics examined.

<table>
<thead>
<tr>
<th></th>
<th>Mann Whitney U</th>
<th>Cohen’s d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DCEMpH</td>
<td></td>
</tr>
<tr>
<td>3t4</td>
<td>statistic=442503.5, $p&lt;.001$</td>
<td>-0.31771759</td>
</tr>
<tr>
<td>4t3</td>
<td>statistic=12929.0, $p&lt;.001$</td>
<td>-0.548625824</td>
</tr>
<tr>
<td>Overlap</td>
<td>statistic=85313.0, $p&lt;.001$</td>
<td>-0.484115739</td>
</tr>
<tr>
<td>Exclusive</td>
<td>statistic=183571.0, $p=.039$</td>
<td>-0.357252925</td>
</tr>
<tr>
<td></td>
<td>CpH</td>
<td></td>
</tr>
<tr>
<td>3t4</td>
<td>statistic=372857.5, $p&lt;.001$</td>
<td>0.338510481</td>
</tr>
<tr>
<td>4t3</td>
<td>statistic=15866.0, $p=.020$</td>
<td>0.149006546</td>
</tr>
<tr>
<td>Overlap</td>
<td>statistic=90244.0, $p&lt;.001$</td>
<td>0.225779148</td>
</tr>
<tr>
<td>Exclusive</td>
<td>statistic=116725.5, $p&lt;.001$</td>
<td>0.594828088</td>
</tr>
<tr>
<td></td>
<td>PSC</td>
<td></td>
</tr>
<tr>
<td>3t4</td>
<td>statistic=193780.0, $p&lt;.001$</td>
<td>-1.199747215</td>
</tr>
<tr>
<td>4t3</td>
<td>statistic=10479.0, $p&lt;.001$</td>
<td>-0.742902374</td>
</tr>
<tr>
<td>Overlap</td>
<td>statistic=45798.0, $p&lt;.001$</td>
<td>-1.062483818</td>
</tr>
<tr>
<td>Exclusive</td>
<td>statistic=77153.5, $p&lt;.001$</td>
<td>-1.19623504</td>
</tr>
</tbody>
</table>

### 5.3.1 RQ5.1: Exclusive vs Transition Use

Our first research question was: *How does transitioning between compilation mechanisms affect novice interaction regarding compilation and error messages as opposed to being exposed to a single mechanism?*
Table 5.2: Mean values of 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 of all user cohorts. Effect sizes (ES) using Cohen’s d are displayed alongside each pair of metrics. The direction of effect is omitted in ES since it is consistent in all metrics across all cohorts. ES Sum refers to the cumulative effect size derived by summing all ES values in that row. The last three columns display the values of distance between BlueJ 3 and BlueJ 4 using the methods described in Section 5.2.6. In the first row, \( n_3 \) and \( n_4 \) refer to the number of users in cohorts X3 and X4 respectively.

<table>
<thead>
<tr>
<th>User Cohort (n)</th>
<th>H V3</th>
<th>H V4</th>
<th>DCEMpH V3</th>
<th>DCEMpH V4</th>
<th>CpH V3</th>
<th>CpH V4</th>
<th>PSC V3</th>
<th>PSC V4</th>
<th>ES Sum</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>X (( n_3 = 727, n_4 = 536 ))</td>
<td>19</td>
<td>30</td>
<td>11</td>
<td>17</td>
<td>.36</td>
<td>.59</td>
<td>.52</td>
<td>.76</td>
<td>1.2</td>
<td>2.15</td>
<td>1.39</td>
<td>1.22</td>
</tr>
<tr>
<td>3t4 (( n = 1008 ))</td>
<td>101</td>
<td>41</td>
<td>7</td>
<td>9</td>
<td>.32</td>
<td>.34</td>
<td>.53</td>
<td>.74</td>
<td>1.2</td>
<td>1.86</td>
<td>.8</td>
<td>.66</td>
</tr>
<tr>
<td>4t3 (( n = 190 ))</td>
<td>62</td>
<td>9</td>
<td>7</td>
<td>12</td>
<td>.55</td>
<td>.15</td>
<td>.58</td>
<td>.73</td>
<td>.74</td>
<td>1.44</td>
<td>.78</td>
<td>.54</td>
</tr>
<tr>
<td>Overlap (( n = 463 ))</td>
<td>125</td>
<td>44</td>
<td>6</td>
<td>10</td>
<td>.48</td>
<td>.23</td>
<td>.54</td>
<td>.73</td>
<td>1.06</td>
<td>1.77</td>
<td>.76</td>
<td>.64</td>
</tr>
</tbody>
</table>
Displayed Compiler Error Messages per Hour (DCEMpH) are greater in BlueJ 4 than in BlueJ 3 for all cohorts. The mean values in BlueJ 3 and BlueJ 4 for each cohort are: 11 and 17 for cohort X, 7 and 9 for cohort 3t4, 7 and 12 for cohort 4t3, and 6 and 10 for cohort Overlap. Manual Compilations per Hour (CpH) are lower in BlueJ 4 than in BlueJ 3 in all cohorts. The mean values in BlueJ 3 and BlueJ 4 for each cohort are: 22 and 12 for cohort X, 15 and 10 for cohort 3t4, 17 and 15 for cohort 4t3, and 14 and 11 for cohort Overlap. Regarding Percentage of Success of manual Compilations (PSC), the differences between BlueJ 3 and BlueJ 4 are very similar for all user cohorts, with more successful manual compilations in BlueJ 4. Manual compilations in BlueJ 4 are 73-76% successful compared to BlueJ 3, in which they are 52-58% successful.

All three methods discussed in Section 5.2.6 show a larger difference between X4 and X3 interactions for users in cohort X than for the rest of the cohorts. Specifically: (1) the average minimum distance between cohorts and the BlueJ 3 equation is 1.39 for X4 and around 0.76-0.8 for the transition cohorts, (2) the distance between the means of the metrics for users in X3 and X4 is 12.11, while for the transition cohorts, the distances are between 4.92 and 5.68, and (3) the minimum distance between X4 mean coordinate values and the BlueJ 3 equation is 1.22, whereas for the rest of the cohorts it is between 0.54 and 0.66.

The results of these three methods align with the cumulative ES reported in Table 5.2. This is expected since all discussed methods and the sum of the absolute values of the ES incorporate all three metrics in their calculation.

Based on these findings we conclude that the difference in programming behaviour between using BlueJ 3 and using BlueJ 4 is substantially greater for users that used one of the two versions exclusively than for those who program using both versions. This is primarily the result of changes in numbers of displayed compiler error messages and manual compilations, as the difference in successful manual compilations between BlueJ versions remains relatively stable across different cohorts of users.
5.3.2 RQ5.2: Order of Transition

Our second research question was: How does the order of transition between compilation mechanisms affect this interaction?

Regarding DCEMpH, the difference between the means in BlueJ 3 and BlueJ 4 for each of the transition cohorts from highest to lowest are: (1) 71% for cohort 4t3, (2) 67% for cohort Overlap, (3) 29% for cohort 3t4. Regarding CpH, the differences between the means in BlueJ 3 and BlueJ 4 for each of transition cohorts from highest to lowest are: (1) 33% for cohort 3t4, (2) 21% for cohort Overlap, (3) 12% for cohort 4t3. Regarding PSC, the differences between the means in BlueJ 3 and BlueJ 4 are very similar for all user cohorts.

Methods 1-3 (discussed in Section 5.2.6) present similar numbers for the interaction difference between BlueJ 3 and BlueJ 4 for the users in cohorts 3t4, 4t3,
CHAPTER 5. COMPILATION MECHANISMS: EXCLUSIVE VERSUS TRANSITIONING USE

and Overlap. Although there are minor differences, they are not on the same order of magnitude as those for the X cohort (which is around twice as high for all three methods). Cumulative effect sizes using Cohen’s $d$ also align with this as they range between 1.44 and 1.86 for all transition cohorts.

Based on these findings, we conclude that the order in which users transition from one BlueJ version to the other does not play a substantial role in the programming behaviour change between BlueJ 3 and BlueJ 4 and the interaction regarding DCEMpH, CpH and PSC is not significantly altered.

5.4 Threats to Validity

Blackbox data are anonymous, and do not contain any information about the programming level of BlueJ users. Although BlueJ is not practical for experienced programmers, some users could be educators trying out the environment or making sure that the environment can execute certain exercises without issues. Additionally, one Blackbox user could in fact be several users working on the same machine, which is very common in institutional labs. This limitation is inherent to all studies that use these data [23].

The metrics involved in the current work incorporate the time spent on BlueJ. However, network interruptions can cause Blackbox to stop logging events for a session. In our analysis we treat the last event logged in the session as the true final event. This approach inevitably results in some missing data – if a user’s connection is disrupted, the last logged activity for that session may not be complete. We regard this as a minor threat, since we are comparing two different BlueJ versions and the probability of an incomplete session should be the same for both versions, potentially mitigating this otherwise unavoidable issue to some degree.

Finally, one of the metrics we used (Displayed Compiler Error Messages per Hour) involves counting the numbers of shown compiler error messages that are logged in Blackbox. These logged events can sometimes be triggered by users inadvertently in BlueJ 4. Since these events are triggered by moving the cursor to the area of the code responsible for the error, if users accidentally move their cursor or if they click the offending code area to fix the error, this counts as a shown error.
message in the current analysis. We will work towards isolating these instances in the future.

5.5 Discussion

In this study, we explored the programming interaction between novices and two versions of the BlueJ pedagogical environment that differ fundamentally in their incorporated compilation mechanisms. BlueJ 3 features a click-to-compile mechanism and enforced first error message presentation. BlueJ 4 features automatic error checking and on-demand error message presentation. The aims of our research were: (1) to investigate how exposure to a single compilation mechanism affects the interaction between novices and the environment regarding compilation and error messages compared to being exposed to multiple compilation mechanisms, and (2) to what extent the order in which this exposure takes place affects this interaction.

The analysis was conducted using programming process data from four distinct user cohorts using two different BlueJ versions. The cohorts included users who exclusively used only one of the two BlueJ versions, users who transitioned from BlueJ 3 to BlueJ 4, users who transitioned from BlueJ 4 to BlueJ 3, and users who switched multiple times between the two. In order to answer our research questions, we utilized three metrics that describe user interaction with the environment regarding compilation and error message presentation: displayed compiler error messages per hour, manual compilations per hour, and percentage of successful manual compilations. Including these metrics in our study could allow researchers to generalize our findings outside of the BlueJ context, as the interaction they measure is common to programming environments that feature similar compilation mechanisms.

Regardless of whether users were exposed to a single BlueJ version or multiple versions, we made a common observation: in BlueJ 4, users see significantly more error messages, compile manually less frequently and have higher manual compilation success. This is in accord with previous findings from Chapters 3 and 4 that followed a more holistic approach to analyzing programming activity in BlueJ.
3 and BlueJ 4, without comparing transitioning users with those who programmed only in one of the two.

In terms of differences in displayed compiler error messages over time, the cohort of users who moved from BlueJ 4 to BlueJ 3 exhibited the largest variation, followed by those who kept switching repeatedly between versions, those who used exclusively one BlueJ version, and those who moved from BlueJ 3 to BlueJ 4. All cohorts showed changes situated between the small to large spectrum of effect size interpretations [141].

In terms of differences in manual compilations over time, the cohort of users who exclusively used only one BlueJ version exhibited the largest variation between versions, followed sequentially by those who moved from BlueJ 3 to BlueJ 4, those who kept switching between versions and those who moved from BlueJ 4 to BlueJ 3. All cohorts showed changes situated between the very small to large spectrum of effect size interpretations.

In terms of successful manual compilations, differences between versions are very high in all cohorts of users, and the magnitude of the differences is relatively stable. All cohorts showed changes situated between the medium (users who moved from BlueJ 4 to BlueJ 3) to very large spectrum (users who exclusively used one BlueJ version and users who moved from BlueJ 3 BlueJ 4) of effect size interpretations.

By quantifying the cumulative interaction comprising the three metrics, we conclude that programming interaction difference between programming in BlueJ 3 and BlueJ 4 is higher for users who only used one of the two versions exclusively than for users who transitioned between versions, primarily due to the numbers of displayed compiler error messages and manual compilations. The order of transitioning between versions does not seem to play a role of similar magnitude in this difference however. It is reasonable for novices who learn programming while using a single environment to adapt their interaction habits according to how the environment operates while displaying substantially diverging behaviour from those using another environment. In contrast, novices who transition between pro-

\[\text{Users who started programming in BlueJ 3 show a moderating effect regarding displayed error messages taking place in BlueJ 4, which is expected, given the deterministic nature of BlueJ 3 which only displays a maximum of one error message per compilation.}\]
gramming environments (regardless of the reason behind such transitions) could be influenced by mechanisms present in both versions and display a moderated behaviour.

A further observation is that users exposed only to a single environment generated more error messages and compiled manually more frequently in both BlueJ versions than users exposed to both. This may be due to users who are comfortable using an environment following a more fast-paced programming behaviour than those who switch between environments, attempting to find an environment more suitable to their needs. An exception to this are novices that moved from BlueJ 4 to BlueJ 3, who seem to compile more frequently than users who programmed only in BlueJ 4. This requires further investigation, possibly by including more users.

Due to the nature of Blackbox data, we have no access to the reasons why users transition between environments. It could be that their educator instructed them to switch BlueJ versions or that they discovered that one of the two versions aligns with their personal preferences better, although this does not explain the reason for repeatedly switching between versions. It is possible that some of these users have multiple BlueJ installations on their machines, and they alternate between them by mistake. Another explanation could be that these users are in fact multiple different students using an institutional machine. A survey targeting the reasons behind this could reveal important insights into the motivations for such behaviour. We can speculate that some novices who used only one environment felt comfortable while programming in it, therefore not attempting to switch to another version. Others may be constrained by institutional or other factors. Users who moved from one version to another may have had trouble with the manner in which the environment operated. For example, those who moved from version 3 to 4 could feel restricted by the need to compile manually or by only having access to the first error message. On the other hand, users who moved from 4 to 3 could feel overwhelmed by the constant red underlining of errors triggered by automatic compilations and desired a simpler approach. Again, institutional and other factors beyond student control could also be at play.

Our research indicates that programming behaviour is largely determined by the mechanisms of the programming environment, and that the transfer of behaviour from one environment to another, if it occurs, is affected substantially by
the restrictions of the environment. This is evidence that the choices of environment
designers heavily affect novice programmer behaviour and possibly their learning
opportunities.

The present study marks the end of Part II of the present thesis. In Part III, we
delve deeper into the behavioural differences of novices using the compilation
mechanisms present in BlueJ as well as their interaction with error messages. In
particular, in Chapter 6, we investigate the role that these mechanisms play in the
syntax state of the source code, the relationship between the syntax state and the
trigger state (i.e., the reasons) of compilation, and the error resolution of novices in
each BlueJ version.
Part III

Compilation Behaviour and Interaction with Compiler Error Messages
Abstract

In this work we investigate compilation behaviour 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.
CHAPTER 6. COMPILATION BEHAVIOUR AND ERROR RESOLUTION TIME OF NOVICES EXPOSED TO DIFFERENT COMPILATION MECHANISMS

6.1 Introduction

The present study expands Jadud’s [71, 72] approach to observing compilations and makes the following contributions:

1. A more contemporary perspective of novice compilation behaviour in a click-to-compile environment with thousands of users as a study cohort.

2. An investigation of compilation behaviour of users exposed to two different BlueJ versions that feature different compilation and error message presentation mechanisms, but preserving most other environmental factors.

3. An investigation of error resolution timings of students using both environments.

6.1.1 Research Questions

We seek to answer the following research questions:

**RQ6.1:** How do novices spend their programming time with respect to the compilation state of their source code in BlueJ 3 and BlueJ 4?

**RQ6.2:** How long does it take novices to resolve syntactically incorrect source code in BlueJ 3 and BlueJ 4?

6.2 Methodology

6.2.1 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 [4, 73]. 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 [82]. 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:

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 \( (n = 3748) \).\(^1\)

2. We filtered users to retain only those for whom the entirety of their programming activity was in Java 8 (BlueJ 3: \( n = 3278 \), BlueJ 4: \( n = 3744 \)).\(^2\)

3. In [3], 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 has been found to have effects on the interaction of users with the compilation mechanisms in Chapter 5. Users were similarly distributed between ten and fifty hours of programming time using BlueJ for both versions.

### 6.2.2 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

---

\(^1\)The 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.

\(^2\)This was done in order to ensure consistency in compiler output, as different Java versions may have different error messages and may handle code differently [24].
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 6.1.

Table 6.1: All possible consecutive compilation sequences based on success and trigger states for BlueJ 4.

<table>
<thead>
<tr>
<th>Previous Trigger State</th>
<th>Current Trigger State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto</td>
<td>SS        SS        FS        FF</td>
</tr>
<tr>
<td>Manual</td>
<td>SS        SF        SS        FF</td>
</tr>
<tr>
<td></td>
<td>FS        FF        FS        FF</td>
</tr>
<tr>
<td></td>
<td>SS        SF        SS        SF</td>
</tr>
<tr>
<td></td>
<td>FS        FF        FS        FF</td>
</tr>
</tbody>
</table>

In SS sequences, users modify their already syntactically correct code successfully. In SF, they modify correct source code but end up having compile-time errors. In FF, they attempt to fix an error with no success. Finally, in FS, they succeed in correcting an 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 proceeded to labeling 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.

### 6.2.3 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 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 [71]. 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 RQ6.2: (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.

### 6.3 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 6.2 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 6.1. Additionally, Figure 6.2 displays the distribution of duration of all compilation sequences and Figure 6.3 shows a breakdown of the timings of consecutive compilations based both on success and trigger states for BlueJ 4.

#### 6.3.1 RQ6.1: Compilation Behaviour

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?*
Table 6.2: 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.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Trigger</th>
<th>Time Proportion</th>
<th>Occur. Proportion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BlueJ 3</td>
<td>BlueJ 4</td>
</tr>
<tr>
<td>FF</td>
<td>AA</td>
<td>26%</td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>AM</td>
<td>2%</td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>MA</td>
<td>2%</td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>MM</td>
<td>22%</td>
<td>1%</td>
</tr>
<tr>
<td>Total FF</td>
<td></td>
<td>22%</td>
<td>31%</td>
</tr>
<tr>
<td>FS</td>
<td>AA</td>
<td>6%</td>
<td></td>
</tr>
<tr>
<td>FS</td>
<td>AM</td>
<td>1%</td>
<td></td>
</tr>
<tr>
<td>FS</td>
<td>MA</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>FS</td>
<td>MM</td>
<td>5%</td>
<td>0%</td>
</tr>
<tr>
<td>Total FS</td>
<td></td>
<td>5%</td>
<td>7%</td>
</tr>
<tr>
<td>SF</td>
<td>AA</td>
<td>7%</td>
<td></td>
</tr>
<tr>
<td>SF</td>
<td>AM</td>
<td>1%</td>
<td></td>
</tr>
<tr>
<td>SF</td>
<td>MA</td>
<td>5%</td>
<td></td>
</tr>
<tr>
<td>SF</td>
<td>MM</td>
<td>25%</td>
<td>1%</td>
</tr>
<tr>
<td>Total SF</td>
<td></td>
<td>25%</td>
<td>14%</td>
</tr>
<tr>
<td>SS</td>
<td>AA</td>
<td>19%</td>
<td></td>
</tr>
<tr>
<td>SS</td>
<td>AM</td>
<td>7%</td>
<td></td>
</tr>
<tr>
<td>SS</td>
<td>MA</td>
<td>14%</td>
<td></td>
</tr>
<tr>
<td>SS</td>
<td>MM</td>
<td>48%</td>
<td>8%</td>
</tr>
<tr>
<td>Total SS</td>
<td></td>
<td>48%</td>
<td>48%</td>
</tr>
</tbody>
</table>

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
CHAPTER 6. COMPILATION BEHAVIOUR AND ERROR RESOLUTION TIME OF NOVICES EXPOSED TO DIFFERENT COMPILATION MECHANISMS

Figure 6.1: Programming time allocation on consecutive compilation and trigger states (top: BlueJ 3, bottom: BlueJ 4). Key to compilation states and triggers: $SF = \text{Success} \rightarrow \text{Failure}$, etc. $MA = \text{Manual} \rightarrow \text{Automatic}$, etc.

... lasts longer in BlueJ 3 ($\mu = 78s, SD = 705$) than in BlueJ 4 ($\mu = 25s, SD = 475$).

Failed $\rightarrow$ 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 $\rightarrow$ 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 behaviour conducted in 2005 [71]. 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 6.2). 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 behaviour can also be
observed in our case by looking at Figure 6.2 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 6.2 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 6.3). 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 most likely trigger a compilation (compilations occur when users change code lines).

6.3.2 RQ6.2: 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 6.2). 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 intermittently be correct. Measuring TSC using all compilations captures these intermittently 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 6.2 and Figure 6.1) 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.

6.4 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 [24]), 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.

6.5 Discussion

In this study we investigated the compilation behaviour 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 mechanisms 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 behaviour 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 [71]. 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 the next Chapter, we provide a detailed view on the interaction between novices and compiler error messages when the compilation mechanisms vary. We also perform additional analyses that reveal in which compilation sequences users actually see the errors generated in BlueJ 4. 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.
INTERACTION BETWEEN NOVICES AND COMPILER ERROR MESSAGES

Abstract

In the present study, we explore the effects of compilation mechanisms, specifically error message presentation on programmer behaviour by providing a synthesis of findings regarding the interaction between novice programmers and compiler error messages. The experiments described in this work utilize BlueJ users’ programming process data mined from the Blackbox database. In total, the behaviour of over 2000 users is explored. Multiple quantitative approaches to analyze programming process data taken from Blackbox are used and described. Specifically, we focus on compiler error messages, the trigger of each compilation, and the location of error indications in the source code. We identify several connections between error message presentation and novice programmer behaviour. Our findings allow us to emphasize the importance of design decisions during the development of programming environments as these can substantially influence the programming behaviour of programmers, in this case novices. Additionally, when choosing programming environments for use by students, these mechanisms should be taken into consideration as they have several effects on the programming activity of users.
CHAPTER 7. INTERACTION BETWEEN NOVICES AND COMPILER ERROR MESSAGES

7.1 Introduction

In this study, we present a thorough exploration of how novices engage with compiler error messages when the compilation mechanisms present are differentiated within a programming environment. Our aim is to establish a better understanding of novices’ interaction with a programming environment and to what degree changing the compilation mechanisms alter the interaction between the novice programmer and the system.

7.1.1 Objectives and Research Question

The present work focuses on contributing new findings regarding error message related interaction. We conduct our study by investigating error messages; how novices interact with them when the error message presentation varies from an “enforced first error message” (BlueJ 3) to an “on-demand any/all error messages” (BlueJ 4) manner. We are guided the following high-level research question:

RQ7.1: How does the manner in which novices interact with error messages change when the compilation mechanisms present only the first error message automatically versus when it shows any/all error messages on-demand?

7.2 Methodology

All the data used in the present study are in fact mined from the users described in Chapter 6 (see Section 6.2.1). We briefly outline the process of finding those users in the following section.

7.2.1 Blackbox Cohort

Many studies utilizing the Blackbox database target specific attributes such as compilations, specific programming sessions, and compiler error messages as a starting point in identifying the required data for further analysis [4, 73]. In this work, all studies feature the BlueJ user as a starting point. This allows drawing
conclusions by taking into account the entire programming activity of each user and not relying on individual sessions and/or interaction events that may be biased by the method of selection. Although the Blackbox database allows for big data analysis in favor of generalizability of the results, the present work favors quality over quantity, while still retaining a significant amount of users to allow for reasonably representative and therefore generalizable conclusions.

The cohort utilized in the present analysis comprises users that were selected from the Blackbox database with the constraint that the entirety of their programming activity occurred between September and December 2016 for BlueJ 3\(^1\) and September to December 2018 for BlueJ 4.\(^2\) These months’ ranges were used in order to proxy an entire academic semester. Prior studies have found that Blackbox activity levels do correlate with traditional academic semesters such as September-December and January/February-April/May [4]. These users have also been exposed to a single BlueJ version. The initial number of users retrieved were 3673 for BlueJ 3 and 3748 for BlueJ 4.

### 7.2.2 Data Pre-processing

Since the main aspects under analysis are compilation states and the generated outputs from the compiler, the data were reduced to only users who their programming activity was associated with Java 8\(^3\) as it is known that Java error messages may vary across different versions [24]. Additionally, only users who had more than ten and less than fifty hours of programming recorded were included in analysis. This was used as a heuristic as it has been suggested that twenty to thirty hours is a realistic amount of programming for novices during an entire semester [3]. The final number of users in this cohort was 1159 for BlueJ 3 and 1134 for BlueJ 4.

\(^1\)2016 was the last year before BlueJ 4 was released.
\(^2\)This range is a year after BlueJ 4 released, allowing for a substantial number of users to be included in analysis.
\(^3\)Blackbox records the Java version that is installed on the users’ machines in every programming session.
7.2.3 Most Common Error Messages

The first aspect of analyzing the interaction of users with error messages was to identify the most common displayed error messages present in each BlueJ version. First, the top 20 error messages with the highest frequency of occurrence\(^4\) were identified in four different groups of messages: (1) all generated error messages in BlueJ 3, (2) displayed (i.e., first) error messages in BlueJ 3, (3) all generated error messages in BlueJ 4, and (4) displayed error messages in BlueJ 4.\(^5\) Subsequently, the messages which appear in all four groups were selected. This resulted in a list of the top 17 most common displayed error messages present in each BlueJ version along with their frequency of occurrence.

7.2.4 Identifying Specific Compilations that Result in Displayed Error Messages

Although in BlueJ 3 it is automatically inferred that error messages can only result from manual failed compilations, this is not the case in BlueJ 4. Novices programming in BlueJ 4 have the opportunity to see error messages in multiple compilation instances. In response, the compilation sequences in which novices actually see error messages and the percentage of shown messages corresponding to each sequence were identified in BlueJ 4.

7.2.5 Viewing Order of Error Messages

One of the major user-experience differences between the two BlueJ versions is that BlueJ 4 allows users to interact with all underlined locations of their source code, should they wish to see the corresponding error message. To gain insight into how users interact with these error messages, the relationship between the location of displayed error messages and the chronological order in which they were seen by users was investigated.

\(^4\)The frequency of messages after the 20th most frequent was less than 0.004.
\(^5\)Blackbox flags error messages that pop up when users request their display in BlueJ 4. In BlueJ 3, we consider first error messages as displayed. This is noted as a potential threat to validity.
Each error location in Blackbox is described by four fields: (1) start line, (2) end line, (3) start column, (4) end column. The start and end line fields refer to the corresponding source code line(s) that the error was detected and the start and end column signify the horizontal location of the detected error in the source code at a character level of granularity. These are actually the coordinates that the environment uses to highlight error locations with wavy red underlines ("error squiggles", as mentioned in Microsoft documentation [113]). Each error present in each source code file was ranked by its location relative to the others. An error is located first in the source code when its starting line and column numbers are minima in the set of coordinates of all errors present in a single compilation. If multiple errors are present in a single line, then the error with the lowest start column is regarded as the first on that line. Intuitively, the higher up and the more left an error is located in the source code, the earlier its position in the mapping order is.

In addition to identifying each error’s mapping order in the source code, its chronological order was also calculated; the earlier an error message corresponding to an error location was displayed, the earlier in chronological order it was.

The pairwise agreement between the two orders of all displayed error messages in BlueJ 4 was subsequently calculated using Kendall’s Tau [86], but only for source code files and compilations that resulted in more than one error.

### 7.3 Results

Our research question was: “How does the manner in which novices interact with error messages change when the programming editor presents only the first error message automatically versus when it shows any/all error messages on-demand?”

Since BlueJ 3 operates in an enforced first-error-message-only manner, it can be immediately inferred that the number of error messages shown to the user is exactly equivalent to the number of times that their code failed to compile (of course this does not necessarily mean that users always read or pay attention to them). In BlueJ 4, the interaction is inherently more complicated since a failed compilation does not necessarily result in a single displayed error message (if any
at all). In this section, results regarding the following aspects of interaction with error messages are discussed:

- The most common displayed error messages in each BlueJ version.
- The percentage of compilations that result in a displayed error message in BlueJ 4.
- The specific compilations that result in error messages being displayed in each BlueJ version.
- The distributions of the number of error messages per failed compilation in BlueJ 4.
- The order in which the error messages are displayed in BlueJ 4.

### 7.3.1 Most Common Displayed Error Messages

After identifying the top 17 most common displayed error messages\(^6\) in each BlueJ version and ranking them, we explored how individual messages' frequency fluctuates across BlueJ versions. In Table 7.1 and Figure 7.1, the frequencies of the 17 most commonly shown error messages are displayed in each BlueJ version along with their respective rankings. Intuitively, this comparison presents the frequency of first messages in BlueJ 3 against the frequency of messages that users most commonly see in BlueJ 4. Note that in BlueJ 3 the display of first error messages is enforced by design, while in BlueJ 4, users voluntarily choose for these error messages to appear.

All messages show a variation in their display frequency between BlueJ 3 and BlueJ 4, with the biggest difference being 6% (for `<;` expected>). This comparison indicates a major change in user experience between programming using the two BlueJ versions. In Figure 7.1, the rankings of messages based on the display frequency of each BlueJ version are indicated. An interesting observation is that five out of eight error messages that show a decrease in display frequency from BlueJ 3 to BlueJ 4, although emerging at compile-time, they do not strictly violate

---

\(^6\)In Section 7.2.4 we explain that there were 17 messages common to the 20 most frequent messages generated by BlueJ 3 and BlueJ 4.
the syntactical rules of Java\(^7\) (as a missing semicolon would for example). Although, “syntax” and “compile-time” errors are terms that have been used interchangeably for a long time [27]. Nonetheless, these errors result in an unsuccessful compile. These messages are:

- cannot find symbol - class
- cannot find symbol - method
- cannot find symbol - variable
- incompatible types
- missing return statement

These five messages can be caused by (1) an omitted declaration or spelling mistakes in the names of classes, methods or variables, (2) errors regarding the declared types of objects, and (3) omitting any required “return” statements for a method. Since the display frequency decreased in BlueJ 4 for all of these, a possible interpretation is that users do not deem it as necessary to diagnose such errors as much as “hard” compiler errors, either consciously or subconsciously. A possible explanation for this is that error indication alone is enough for users to correct these errors while editing their code without the need to see the error messages. In other words, in BlueJ 3 users are forced to see the messages resulting from these errors while in BlueJ 4 they are not — and often in BlueJ 4 users choose not to. However, the author considers that the skill of identifying the cause of an error only by relying on its indication as one that more experienced programmers would possess. This brings into question what knowledge is required to demonstrate this behaviour. Furthermore, it leads to the hypothesis that if the experiments are repeated with more experienced users, the observed frequency at which these messages are voluntarily displayed and read (thus deemed required for a successful fix) would decrease, and for less experienced users, would increase. This provides a fruitful avenue for future work.

\(^7\)https://docs.oracle.com/javase/specs/jls/se7/html/jls-2.html#jls-2.3
Apart from the five messages mentioned, \textless;'\textgreater; \textbackslash expected>, \textless'invalid method declaration; return type required\textgreater;, and \textless'.class' expected\textgreater; showed a decreased in display frequency in BlueJ 4, while the rest of the messages in the list showed increased frequency. This increase may be attributed to the following reasons:

1. Users face particular difficulty with a specific error location and they choose to see the error message to help resolve this.

2. Syntax errors tend to occur more persistently in incomplete programs. Students may type code and get errors for their incomplete code during automatic compilations, which happen during code edits. The error reporting in these instances could be considered redundant, since the error would not exist in the first place if students completed their code. These false positive errors not only consume time unnecessarily, it is possible that they can also shift the users’ attention and reduce their focus on the actual programming task.

### 7.3.2 Proportion of Compilations that Result in Displayed Error Messages

The proportion of failed compilations that result in error messages being displayed in BlueJ 3 can easily be inferred as being 100%, since there is an inherent one-to-one relationship between a failed compilation and a displayed error message. In BlueJ 4 however, the proportion of failed compilations that resulted in displayed error messages was only 27%. This could lead to one or more of the following conclusions: (1) that some automatic compilations are redundant to a degree, as users do not utilize them while they are composing to actually view the error messages 73% of the time and/or (2) that the red error squiggles that signal the position of errors allow students to fix them without seeing the error messages. The second case seems somewhat less probable in general, as compilers are known to be unable to reliably detect both the cause of an error, including the actual location. Furthermore, novices are unlikely to possess advanced debugging skills that would allow them to correct their mistakes by just looking at the (correct) location of an error, unless the error is trivial. Most of these cases of failed compilation with no
Table 7.1: Frequencies of the most common displayed compiler error messages in the two BlueJ versions. The negative sign in “Diff.” (Difference) indicates if there was a decrease in frequency from BlueJ 3 to BlueJ 4. The messages are sorted based on the absolute difference in frequencies between BlueJ 3 and 4 in descending order.

<table>
<thead>
<tr>
<th>Compiler Error Message</th>
<th>BlueJ 3</th>
<th>BlueJ 4</th>
<th>Diff. (3→4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>';' expected</td>
<td>0.104</td>
<td>0.164</td>
<td>0.06</td>
</tr>
<tr>
<td>reached end of file while parsing</td>
<td>0.033</td>
<td>0.073</td>
<td>0.04</td>
</tr>
<tr>
<td>cannot find symbol - method</td>
<td>0.076</td>
<td>0.042</td>
<td>-0.034</td>
</tr>
<tr>
<td>cannot find symbol - variable</td>
<td>0.182</td>
<td>0.153</td>
<td>-0.029</td>
</tr>
<tr>
<td>illegal start of expression</td>
<td>0.042</td>
<td>0.065</td>
<td>0.023</td>
</tr>
<tr>
<td>incompatible types</td>
<td>0.069</td>
<td>0.047</td>
<td>-0.022</td>
</tr>
<tr>
<td>cannot find symbol - class</td>
<td>0.054</td>
<td>0.036</td>
<td>-0.018</td>
</tr>
<tr>
<td>&lt;identifier&gt;expected</td>
<td>0.031</td>
<td>0.044</td>
<td>0.013</td>
</tr>
<tr>
<td>illegal start of type</td>
<td>0.01</td>
<td>0.021</td>
<td>0.011</td>
</tr>
<tr>
<td>not a statement</td>
<td>0.034</td>
<td>0.043</td>
<td>0.009</td>
</tr>
<tr>
<td>illegal character</td>
<td>0.007</td>
<td>0.012</td>
<td>0.005</td>
</tr>
<tr>
<td>missing return statement</td>
<td>0.022</td>
<td>0.0175</td>
<td>-0.0045</td>
</tr>
<tr>
<td>')' expected</td>
<td>0.048</td>
<td>0.044</td>
<td>-0.004</td>
</tr>
<tr>
<td>invalid method declaration; return type required</td>
<td>0.014</td>
<td>0.01</td>
<td>-0.004</td>
</tr>
<tr>
<td>'.class' expected</td>
<td>0.011</td>
<td>0.008</td>
<td>-0.003</td>
</tr>
<tr>
<td>class, interface, or enum expected</td>
<td>0.02</td>
<td>0.023</td>
<td>0.003</td>
</tr>
<tr>
<td>unclosed string literal</td>
<td>0.006</td>
<td>0.008</td>
<td>0.002</td>
</tr>
<tr>
<td>Total percentage:</td>
<td>0.762</td>
<td>0.81</td>
<td>-</td>
</tr>
</tbody>
</table>

shown error message arise from editing the source code and changing code lines that users experience as red error squiggles on their code.

### 7.3.3 Specific Compilations that Result in Displayed Error Messages

In Figure 7.2, the displayed error messages’ proportion for each compilation sequence is listed. For both BlueJ versions, a similar trend is observed: the majority of error messages are shown during Failed—Failed (FF) compilation sequences (71-75%), around a quarter of messages are shown to users during Failed—Successful (FS) sequences (21-25%), and a very small number of messages are shown in terminating failed compilations for a given session (3.7-3.8%).
Figure 7.1: Rank variations of the most common shown compiler error messages based on frequency in the two BlueJ versions. The figure was created with the aid of LineUp [59].
In Figure 7.3, the proportions of displayed error messages are listed based on the trigger status of compilations for BlueJ 4. The majority of error messages are displayed during Automatic→Automatic (AA) compilation sequences (78%). 18.1% of displayed error messages are divided almost equally to Manual→Manual (MM, 6.5%), MA (6.3%), and AM (5.3%) sequences. A smaller proportion of displayed messages are shown after terminating automatic compilations (3%), and an even smaller proportion is shown after terminating manual compilations (0.8%).

![Figure 7.2: Proportion of displayed compiler error messages corresponding to different consecutive compilation sequences based on syntax states for BlueJ 3 & 4. “-t” denotes a terminating compilation for a given session.](image)

### 7.3.4 Distributions of Displayed Error Messages per Failed Compilation

In BlueJ 3, every failed compilation results in exactly one error message displayed to the user. This is not the case in BlueJ 4. In Table 7.2, the distributions of all generated compiler error messages and displayed error messages present per failed compilation are presented for BlueJ 4. It is observed that 50% of compilations contain less than two error messages, with none actually shown to the user. A minority of failed compilations (less than 25%) in BlueJ 4 result in more than one
displayed error message. This supports the hypothesis that a considerable amount of compilations (and therefore the amount of error indications) are redundant to BlueJ users and are not being utilized by users to receive feedback via compiler error messages.

Table 7.2: Distributions of the number of compiler error messages per failed compilation in BlueJ 4.

<table>
<thead>
<tr>
<th></th>
<th>All Error Messages Failed Compilation</th>
<th>Shown Error Messages Failed Compilation</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>682581</td>
<td>682581</td>
</tr>
<tr>
<td>μ</td>
<td>3.5</td>
<td>0.3</td>
</tr>
<tr>
<td>SD</td>
<td>6.9</td>
<td>0.6</td>
</tr>
<tr>
<td>min</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>25%</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>median</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>75%</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>max</td>
<td>200</td>
<td>85</td>
</tr>
</tbody>
</table>
7.3.5 Order in Which Error Messages Are Displayed

Performing a Kendall’s Tau test [86] between mapping-on-the-source-code and chronological order ranks for each displayed error message in compilations that contained more than one displayed error message in BlueJ 4 yielded a t-statistic of 0.94 with p<0.01 (see also Table 7.3 for the distribution of the quantity of displayed messages in mapping and viewing order according to their rank), indicating a strong agreement between the two ranking methods. Therefore, it can be deduced that in almost all cases, BlueJ 4 users view the compiler error messages in the left-to-right, top-to-bottom order that these messages are mapped on the source code. The higher up and more left they are located in the code, the earlier users will move their mouse to hover over and view them. Additionally, the three first displayed error messages both in terms of location and in chronological order account for around 90% of the total number of displayed error messages (see Table 7.3), meaning that users either do not bother viewing error messages of lower ranks. It could be the case that some students are aware that focusing on the first errors often fixes all the subsequent and eliminates further messages (a behaviour known to exist [18]) or that their source code in general rarely produces a large number of error messages (something that was generally observed in Section 7.3.4).

7.4 Threats to Validity

Since all work described in this study relies on anonymous Blackbox data, it is important to acknowledge lack of any contextual information such as whether users are actually novice programmers, what their intent is, and if they program in a practical session, practicing at home, etc. Unfortunately, this is a limitation present in all Blackbox studies [23]. However, given the simplicity of BlueJ and the limitations it poses to experienced programmers, as well as the substantial time users spent programming in BlueJ after applying constraints in the data pre-processing steps, large numbers of experienced programmers in the data seem unlikely.

Sometimes in BlueJ 4, fixing a mistake present in a code area that the compiler identified as the source of the error may trigger a shown error message to be
CHAPTER 7. INTERACTION BETWEEN NOVICES AND COMPILER ERROR MESSAGES

Table 7.3: Distributions of displayed error message quantity in mapping order in the source code (the higher and more left an error message is in the source – the lower its ranking number) and viewing order in terms of time (max rank=85).

<table>
<thead>
<tr>
<th>Rank</th>
<th>Mapping Order</th>
<th>Viewing Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.401</td>
<td>0.399</td>
</tr>
<tr>
<td>2</td>
<td>0.399</td>
<td>0.399</td>
</tr>
<tr>
<td>3</td>
<td>0.100</td>
<td>0.102</td>
</tr>
<tr>
<td>4</td>
<td>0.041</td>
<td>0.042</td>
</tr>
<tr>
<td>5</td>
<td>0.021</td>
<td>0.021</td>
</tr>
<tr>
<td>6</td>
<td>0.012</td>
<td>0.012</td>
</tr>
<tr>
<td>7</td>
<td>0.007</td>
<td>0.007</td>
</tr>
<tr>
<td>8</td>
<td>0.005</td>
<td>0.005</td>
</tr>
<tr>
<td>9</td>
<td>0.003</td>
<td>0.003</td>
</tr>
<tr>
<td>10</td>
<td>0.002</td>
<td>0.002</td>
</tr>
<tr>
<td>11</td>
<td>0.002</td>
<td>0.002</td>
</tr>
<tr>
<td>12</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>13</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>14</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>15</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>16</td>
<td>0.001</td>
<td>0.001</td>
</tr>
</tbody>
</table>

This may inject distortions in the results present in this work and we aim to perform observation studies that will lead to more accurate estimations in the future.

Blackbox records a displayed error message only the first time it is shown in each compilation. If the user triggers the error message to appear again without performing a new compilation, then this is not reflected in the data. If however, a new compilation occurs and the error that the message is associated with is still present in the source code, then triggering its display will be recorded in the Blackbox database. This caveat possibly negates the previous threat mentioned to a degree.

Additionally, it is assumed that users truly interpret the feedback provided from BlueJ. However, there is no way to be certain that this is the case with all users. Some may not pay attention to error indicators or error messages despite them being displayed/shown. A special case of programming activity that was
recently noticed while using BlueJ is that if the programmer is more experienced and types code quickly, then the red error squiggles appear momentarily and disappear during edits. Further investigation of this using in-person or recorded programming sessions is warranted.

7.5 Discussion

In this work, the interaction of novice programmers with their programming environment was explored. We investigated the use of two different versions of the BlueJ programming environment that have fundamental differences their compilation mechanisms and specifically in: (1) the way compilation is invoked; (2) how many error messages are accessible to users; and (3) the manner in which error messages are presented to the user. All other user features of the programming environment are consistent between versions.

In terms of the interaction between novices and error messages in the two BlueJ versions, findings from Part II of this thesis showed that users see more compiler error messages over time in BlueJ 4 than they do in BlueJ 3. This is to be expected, as BlueJ 3 always shows only the first error message, compared to BlueJ 4 which indicates the location of all error messages produced by the compiler. We also found substantial differences in the frequency of individual high-frequency error messages that users see in each BlueJ version. In BlueJ 3, only the first error message is displayed when a compilation fails, but in BlueJ 4 users choose which error messages to see as pop-ups, thus allowing the identification of errors that users want to receive more detail about. A particularly interesting finding is that five of the highest-frequency error messages that are seen less by users in BlueJ 4 compared to BlueJ 3 are often caused by misspellings of the names of classes, methods or variables, by missing return statements in methods, and by errors in the declared type of variables. Assuming that the decrease in display frequency of these messages in BlueJ 4 is attributed to users correcting the offending code only by looking at the underlined locations, it is likely that these errors are more trivial for users to rectify with the assistance of the error message presentation

---

8 Although the studies in Part II utilize different study cohorts from the present study, Chapters 4 and 5 include cohorts of users that were exposed to a single BlueJ version as in this Chapter.
present in BlueJ 4. However, it has been suggested in the past that users might dismiss simple syntactical errors, based on the notion that the compiler will always identify them [98].

As there is a one-to-one relationship between failed compilations and displayed error messages in BlueJ 3, users are always shown a single compiler error message when a compilation fails. In BlueJ 4, it was found that only in around a quarter (27%) of the occasions that a failed compilation occurs users request error messages to be shown to them from the environment. This naturally leads to the question of whether the error indication in the form of red error squiggles is effectively contributing to the identification of errors by novices or is another feature in the environment that often distorts users’ sensory information [123] during the programming process. Although we do not possess evidence supporting or refuting this, it is unlikely that novices have debugging skills that allow them to fix errors only by their location. Additionally, compilers are notoriously inaccurate [13], and often performing edits on an indicated code area can complicate the issue even further. In contrast, the majority of displayed error messages are shown during repeatedly failed automatic compilations in BlueJ 4. This leads to conclude that users utilize the automatic compilations to a degree, but in specific cases that they are actively trying to resolve issues. In most cases, automatic compilation is either utilized in terms of error indication (without the text that occurs when a pop-up is requested) or is completely ignored and the error indicators are seen as redundant. Further, a small percentage of error messages (13%) are displayed after manual compilations in BlueJ 4, instances that seem more suitable for users to consciously seek feedback, since they explicitly request these compilations from the environment.

Finally, we investigated the chronological order in which error messages are read by the user, and their physical location in the code. It was found that users tend to read multiple error messages present in a file starting from the top left to the bottom right. This is interesting as BlueJ 4 does not pose any restriction on the order in which multiple error messages can be shown. Yet, users tend to follow an order as if they were reading an English-language text. Although unsurprising from this point of view, this is often code that the user recently wrote, and it would be reasonable to hypothesize that users may, for instance, start with the error
closest to the line of sight, or the middle of the screen, etc. However, this is not the case for the average user in our study whom we have no reason to believe is not representative of the Blackbox database in general.

The present study concludes Part III of the present thesis. We provided an overview of the differences in compilation behaviour of users; the relationship between the syntax state of the source code and the trigger states of compilation, the error resolution, and the interaction with error messages. We identified several variations and common patterns present in the interaction between novice programmers and different compilation mechanisms. In the next Part, we turn our focus to evaluating the compilation mechanisms based on programmers experiences by conducting an online survey (Chapter 8), and perform an observational study with postgraduate students programming in BlueJ (Chapter 9).
Part IV

Usability Testing
EVALUATION OF COMPILATION MECHANISMS: AN ONLINE SURVEY OF PROGRAMMERS

Abstract

In this work we investigate programmers’ views on the compilation mechanisms featured in the BlueJ pedagogical programming environment. We conduct a survey in which we demonstrate how compilation, error indication and error message presentation mechanisms operate in BlueJ versions 3 and 4. These two versions encapsulate fundamentally different approaches to these mechanisms, allowing us to examine to what degree programmers find them effective for novices. The survey participants were 305 programmers with different levels of experience who provided rating scores for the individual compilation mechanisms mentioned. Additionally, participants provided suggestions on how these features should be facilitated to assist novices. The present findings serve as evidence regarding the effectiveness and usability of different mechanisms featured within a programming environment. These findings can assist designers of pedagogical programming environments in making evidence-based decisions about their products and facilitate the development of environments that can achieve greater efficacy for novices in their first steps of learning.
CHAPTER 8. EVALUATION OF COMPILATION MECHANISMS: AN ONLINE SURVEY OF PROGRAMMERS

8.1 Introduction

Large-scale analysis of programming process data from BlueJ users in the Blackbox database [24, 23] has shown substantial differences in the interaction between novices and BlueJ versions 3 and 4 in the studies conducted in Part II of the present thesis. Specifically, it was found that the number of manual compilations over time decrease when users program in BlueJ 4 compared to BlueJ 3, and although it is not a prerequisite to press the compile button, users still manually compile. Additionally, it was found that the number of displayed compiler error messages is higher over time in BlueJ 4 than in BlueJ 3, since the environment allows access to all generated error messages from the compiler. Moreover, it was found that manual compilations are more often successful in BlueJ 4 than in BlueJ 3, making the authors hypothesize that manual compilations may be psychologically driven in BlueJ 4. In Part III, an overview of how much time users spend on different compilation sequences when programming in each BlueJ version was provided, highlighting multiple similarities and differences between them with respect to the syntax state of the source code and the trigger states of the compilations. In the same work, a lower error resolution time in BlueJ 4 than in BlueJ 3 was found, with the authors attributing it to the early feedback provided by the environment due to the dynamic error reporting. Finally, multiple findings regarding how novices interact with compiler error messages in each BlueJ version were presented.

8.1.1 Objectives and Research Question

In this work we explore how the individual compilation-related mechanisms employed by BlueJ 3 and 4 are perceived by programmers regarding their usefulness during the programming process. We conduct a user study with programmers from different experience levels as participants. This allows for more detailed evidence about the individual compilation mechanisms that are different in the two BlueJ versions, which were previously studied as a whole (in Parts II and III). We measure the efficiency scores assigned by participants regarding the compilation, error indication, and error message presentation features of the two BlueJ versions. In addition, we also ask programmers to express their preferences regarding the following aspects: (a) usefulness of manual compilation when auto-compile is present, (b) reasons for compiling manually even when automatic compilation is present, (c) usefulness of indicating existing errors on the code while editing, (d) strategies on addressing multiple compiler error messages, (e) which error messages in the stack list should be displayed, (f) the location of error messages within the programming environment, and (g) the manner in which error messages should be displayed (manually/automatically). We frame the present research using the following research question:
RQ8.1: How are the compilation mechanisms present in BlueJ 3 and BlueJ 4 perceived by programmers and what are their suggestions for improving their contribution in the user-experience while programming?

8.2 Methodology

8.2.1 Participant Recruitment

We used the Prolific platform for recruiting participants and Google Forms for conducting the survey.\(^1\)\(^2\) Prolific allows setting up research studies and defining pre-screening criteria that filter participants according to a study’s requirements. For instance, survey conductors can request that their survey be sent to different users based on attributes such as ‘first-year undergraduate computer science student’, etc. Once set up, Prolific publishes the study to people who are already signed up in the platform as potential participants in research studies, and redirects those interested to the actual survey. In our case, participants were compensated with 5 GBP credit in their Prolific account after completing a survey.

8.2.2 Participant Screening

Although the target group of this study is novice programmers, the study’s participation was expanded to include more experienced programmers in order to capture more informed and diverse feedback. To that front, three different cohorts were chosen as the study’s participants and the pre-screening criteria of Prolific were set up accordingly. There were no age, ethnicity, or gender restrictions imposed during the screening process. All participants were fluent in the English language in order to ensure that they were able to comprehend the questions. In terms of Prolific’s pre-screening process, all participants were asked “Which of the following languages are you fluent in?”. Additionally, all participants had computer programming skills which was ensured by Prolific’s pre-screening process, during which participants were asked “Do you have computer programming skills?”. Finally, participants were able to complete the study only by using a PC or a laptop, as the surveys contained pictures and animations that would not be clearly visible in smaller screens. The three groups of participants are described below:

- **Undergraduate students in CS related subjects.** This group included undergraduate students of every level (there was no restriction on the year

---

\(^1\)https://www.prolific.co/

\(^2\)This study has been approved by the University College Dublin ethics committee as being exempt from a full ethical review. Application number: LS-E-21-258-Karvelas-Becker. The data collection was conducted in February 2022.
of study) who were at the time of the survey studying for a Computer Science or a Computing (IT) degree.

- **Postgraduate students in CS related subjects.** This group included postgraduate students that were studying for a Graduate (MA/MSc/MPhil) or a Doctorate (PhD) degree in Computer Science or Computing (IT).

- **Professional programmers working in CS related positions in industry.** This group included full-time working professionals in the Information Technology sector.

In total, 305 participants were recruited, almost equally distributed among the three participant groups described, and allocated equally to two different surveys, each targeting mechanisms present in a different BlueJ version (3 and 4). We published each survey (out of six, two for each participant group) on Prolific incrementally while excluding past participants to ensure that no one had either seen or completed the surveys for both BlueJ versions. In a pilot study, we recorded a completion time range of 5-15 minutes for each survey and the studies were advertised on Prolific as lasting 10 minutes.

### 8.2.3 Survey Materials

Two different surveys were designed and imported in Google Forms, each targeting the compilation mechanism, error indication, and error message presentation present in either BlueJ 3 or BlueJ 4. The surveys’ first page contained a participant information sheet that followed standard survey/questionnaire practices. Participants were explicitly informed that the study was part of a broader research frame aimed at identifying mechanisms in programming environments that are effective for novices. The questionnaires are fully presented in Appendix B, Section B.4.

The survey was divided into four parts: (1) General Questions, (2) Compilation, (3) Error Indication, and (4) Error Message Presentation. In part 1, participants were asked to answer questions regarding their computer programming background and preferences while programming. In parts 2, 3, and 4, participants were shown a picture or an animation of the compilation functionality, error indication, and error message presentation respectively. The shown mechanisms were dependent on the BlueJ version that the survey was written for. After seeing the mechanism in a specific section, participants were asked additional questions, targeting the aspects that were mentioned in Section 8.1.1. As a rule, quantitative responses were captured using a seven-point Likert scale in a negative to positive range, with 4 being a neutral response.
8.3 Results

8.3.1 Demographics

In total, 305 participants between 19-58 years of age completed the survey. We report a cohort comprising of 77% male and 22% female participants (the remaining 1% had not provided any information). Undergraduate student participants were distributed across all years of studies, with the highest proportion being in the 3rd year (35.5%). Additionally, 83.5% of the professionals reported being undergraduate degree holders, 72% of whom reported having a CS-related degree.

8.3.2 Compilation

Rating  After participants were shown an animation of how compilation operates in the BlueJ version discussed in their survey, they were asked the following question: “How helpful do you find the way compilation is operated within the editor?” We measured a mean response score of 5.5 ($SD = 1.2$) for BlueJ 3 and 5.6 ($SD = 1$) for BlueJ 4. A Mann-Whitney U test [107] did not reveal any statistically significant difference between the sets of responses for each BlueJ version. This pattern was also present across comparisons of the scores assigned by individual cohorts (see Figure 8.1).

In the end of the survey’s part about compilation, participants were asked to provide suggestions about how the compilation should work. We coded all valid responses to this question for each BlueJ version’s survey independently. There were 72 valid responses for BlueJ 3 and 63 for BlueJ 4. For BlueJ 3, the most recurring emerged codes within the responses were: (a) keyboard shortcut (16/72), (b) automatic compilation (14/72), and (c) suggestions on the compile button location (6/72). The rest of the responses were unique in terms of their content ($\leq 3/72$ for all remaining codes). For BlueJ 4, the most recurring emerged code was automatic compilation at a different level of granularity (27/63). These responses included suggestions that can be summarized as “at character level”, “when file is saved”, “every few minutes/seconds”, “when closing brackets in a method/class”, and “after user inactivity”. Two additional codes were identified in multiple responses: the importance of the automatic compilation mechanism being present in the editor (12/63), and concerns about the speed of the programming process due to the automatic compilation (10/63). The rest of the of responses were unique in terms of content ($\leq 4/63$ for all remaining codes).

Manual Compilation Frequency  Participants provided the frequency in which they would press the Compile button in a scale from 1 (Never) to 7 (Very often). We measured a mean response score of 5.2 ($SD = 1.5$) for BlueJ 3 and 3.4 ($SD = 1.5$) for
BlueJ 4. A Mann-Whitney U test revealed a statistic of 18202 ($p < 0.01$), indicating statistically significant difference between the responses for each BlueJ version. This pattern was present across comparisons of the scores assigned by individual cohorts.

Additionally, participants who completed the survey for BlueJ 4 responded moderately positively ($\mu=5.7$, $SD = 1.4$) to the presence of manual compilation when automatic compilation is featured in the programming environment. Participants in the BlueJ 4 survey who replied on the positive scale of manual compilation frequency (i.e., >4), were asked to provide a textual justification on the reasoning behind manually compiling ($n = 36$). After coding the responses, we identified four distinct explanations: (a) habit (12/36), (b) confidence (10/36), (c) doubting the accuracy of automatic compilation (9/36), and (d) precaution (8/36).

### 8.3.3 Error Indication

**Rating** After participants were shown a picture of how error indication is facilitated within the BlueJ version that their survey was written for, they were asked the following: “*How helpful do you think the way errors are indicated on the code is?*” We measured a mean of 5.4 ($SD = 1.5$) in BlueJ 3 and 5.6 ($SD = 1.3$) in BlueJ 4 and no statistically significant difference between the sets of responses using a Mann-Whitney U test. This pattern was present across comparisons of the scores.
assigned by individual cohorts (see Figure 8.2).

Participants who rated the BlueJ 3 error indication negatively (i.e., <4) provided textual responses for their reasoning \((n = 44)\). After coding the responses, we identified the following as consistent among replies: (a) more detail on error messages is needed \((10/44)\), (b) highlighting more than the first error \((8/44)\), (c) a better choice of the color palette in order for the indication to be more distinct \((7/44)\), and (d) more precision regarding the indicated error location \((5/44)\). The rest of the emerged codes had less than 4 responses under them.

![Figure 8.2: Participant ratings for the error indication featured in each BlueJ version (rectangle: mean, horizontal line: median).](image)

In the end of the survey’s part on error indication, some participants provided suggestions on how it should be facilitated within the editor. We report 113 valid responses for BlueJ 3 and 55 for BlueJ 4. For BlueJ 3, the most recurring emerged codes were: (a) line indication should be present \((23/113)\), (b) different colouring palette, as the error indication should be more vivid and not blended with the colours of other elements in the editor \((20/113)\), (c) more detail in terms of error messages \((17/113)\), (d) suggestions on how to fix the errors \((16/113)\), (e) messages should appear on code \((11/113)\), (f) more accurate error indication \((10/113)\), (g) and show all errors present \((9/113)\). For BlueJ 4, there was significant variation among the replies. The notable codes were (a) indication should be more visible \((12/55)\), and (b) suggestions on fixing an error should be present \((6/55)\).
Error Squiggles Rating  Participants in BlueJ 4 also responded moderately positively when asked about the usefulness of the presence of error squiggles when changing lines in the code ($\mu = 5.6, SD = 1.3$). Some participants that provided a negative rating (i.e. $< 4, n = 11$), provided textual reasoning which is summarized to “red squiggles being intrusive”, that “there should be more detail provided for the errors”, and that “hints on how to solve errors should be present”.

8.3.4 Error Message Presentation

Rating  After presenting an animation of how error messages are presented, participants were asked the following: “How helpful do you find the way error messages are presented in the editor?”. We measured a mean of 5 ($SD = 1.6$) for responses in BlueJ 3 and 6.3 ($SD = 1$) in BlueJ 4. A Mann-Whitney U test provided a statistic of 6023 ($p<0.01$), indicating statistically significant difference between the two. This is consistent with the comparisons of scores across all participant cohorts (see Figure 8.3).

Figure 8.3: Participant ratings for the error message presentation featured in each BlueJ version (rectangle: mean, horizontal line: median).

Addressing Multiple Compiler Error Messages  Regarding which error messages programmers address when presented with an error message log, the majority of participants (56.4%) responded that they address all error messages.
significant proportion of participants (38.4%) replied that they address only the first error message. A much smaller percentage (3.9%) responded “other”, signifying an alternative approach. These participants provided textual responses \( (n=12) \) that described their strategy, which can be summarized to “fixing the first, compiling, and repeating until all errors eliminated”, and “identifying and eliminating the most crucial/easiest to fix/easiest to search online error message”. Finally, 0.3% of respondents replied that they do not address any of the error messages from the compiler log, while 0.1% replied that they had never seen more than one error message at a time.

**Which Error Messages Should Be Shown to Users** The majority of participants (83.3%) replied that all error messages should be shown to users in a programming environment. A much smaller percentage (14.1%) replied that only the first error message should be displayed, in case there are multiple errors in a compilation. Finally, 0.07% replied that no error messages should be shown to users. The remaining 2% of participants did not provide any response.

**Error Message Display Location** On a holistic basis, 46.6% of participants responded that error messages should be displayed on the code area of the editor. Another, equally large percentage (46.2%) replied that they prefer the messages being shown under the editor in a dedicated area. The rest of the participants (7.2%) replied “elsewhere”. We observed variations in these preferences across different participant groups and depending on which survey was being completed (see Table 8.1). The variation in the preference of professionals is possibly due to their prolonged exposure to environments that show messages under the programming editor. The variations across the two different BlueJ version surveys suggest the presence of bias induced by the error message presentation present in the survey. Participants who replied “elsewhere” provided textual reasoning for their choice, and most of their responses referred to allowing the user to see messages with both approaches, as it is convenient to use one over the other in different cases.

**Error Message Display Manner** From a holistic perspective, 65.2% of participants responded that they prefer the messages being shown to the user automatically by the programming environment. A smaller number of participants (32.1%) responded that they would prefer the display to be on-demand, while the remaining 2.6% responded “other”. This pattern is consistent across all participant groups. However, we observed variations in the percentage allocations of these preferences depending on which BlueJ version users had seen, suggesting an induced bias. Participants that selected “other” provided textual reasoning \( (n = 9) \), highlighting that the option to have both approaches to displaying messages is desirable, while
Table 8.1: Percentages of participants’ preferences regarding the proper location of error messages within a programming environment, grouped by cohort and survey version.

<table>
<thead>
<tr>
<th>Cohort</th>
<th>On Code</th>
<th>Under Editor</th>
<th>Elsewhere</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undergraduates</td>
<td>54.5</td>
<td>40.6</td>
<td>5</td>
</tr>
<tr>
<td>Postgraduates</td>
<td>50.5</td>
<td>43.6</td>
<td>5.9</td>
</tr>
<tr>
<td>Professionals</td>
<td>35</td>
<td>54.4</td>
<td>10.7</td>
</tr>
<tr>
<td>BlueJ 3 survey</td>
<td>31.6</td>
<td>61.9</td>
<td>6.5</td>
</tr>
<tr>
<td>BlueJ 4 survey</td>
<td>62</td>
<td>30</td>
<td>8</td>
</tr>
<tr>
<td>Total</td>
<td>46.6</td>
<td>46.2</td>
<td>7.2</td>
</tr>
</tbody>
</table>

also stating that simple explanations for errors should be shown automatically, and if users wish to see more detail, they could request it manually from the environment.

8.4 Threats to Validity

We acknowledge that demonstrating visually how the investigated mechanisms operate does not constitute a thorough demonstration of the programming environment, and a study with participants using the environment instead would have been a more suitable option, something that we aim to perform in future work. In addition, since this work utilizes an online survey, it is difficult to establish the level of participants’ comprehension regarding what was asked. Although we attempted to separate the mechanisms featured within BlueJ (i.e., compilation, error indication, error message presentation), we cannot guarantee that participants perceive these mechanisms as independent and as a result, their responses may be influenced by this (e.g., when asked about the error indication in BlueJ 4, some participants noted that no error messages were present, even though that part was being examined at a later section in the survey). Furthermore, although participants were informed in the beginning of the survey that the mechanisms involved target novice programmers, it is unclear to what degree their responses are expressed from a novice’s point of view, a fact that could be reflected in the results presented. Finally, our results include responses to open questions, the analysis of which may be biased by the authors’ point of view. The major variations across replies for the open questions in our survey does not allow for a thorough systematic analysis. Although we do not draw any hard conclusions based on these responses, we present the most common ones, as they can be highly informative in identifying potential needs of programmers when it comes to programming environments.
8.5 Discussion

In this work, we conducted a survey that addressed how programmers perceive the compilation mechanism, error indication, and error message presentation that are featured in two versions of the BlueJ introductory programming environment, versions 3 and 4. Previous studies conducted in Parts II and III of this thesis have provided evidence that the programming behaviour of novices is altered depending on which version is used. By conducting this study, we investigated how these features are perceived by programmers on an individual basis instead of evaluating them holistically, bundled together in a BlueJ version.

Regarding compilation, we did not find any significant difference between manual and automatic with respect to how these two approaches are perceived by programmers. However, text responses revealed that participants are in favor of automatic compilation, albeit at different levels of granularity, which provides an opportunity for future investigation on the instances in which users need a compilation to occur. A “smart” plugin that performs compilations has been suggested as a solution to this before in Chapter 6. Participants expressed that the presence of manual compilation is important even when the environment performs automatic compilations, as in some cases users want to consciously perform a compilation. In the survey, participants justified the need to manually compile primarily due to already being used to doing so, and because they do not trust the automatic compilations enough, while manually compiling provides them with more confidence about the state of their code. This has been suggested before in Chapters 3 and 4 after analyzing the programming activity of novices. An interesting finding is that participants expressed that they would compile more often in BlueJ 3 than in BlueJ 4 with a statistically significant difference between the responses, a fact that has been shown to be true in Part II of the present thesis.

Regarding error indication, we did not find any difference in programmers’ responses when asked to rate the manner in which errors are indicated in BlueJ 3 and BlueJ 4. Some participants expressed that indicating the line in which an error occurred is important, that more detailed and accurate error messages are required, a fact that provides further evidence on the importance of research on improving compiler error messages [13]. The coloring of the editor was also raised as an important aspect by programmers, especially the fact that it should not blend with the coloring palette used for the code and the error indication (see also the Web Content Accessibility Guidelines 2.1). Some participants also noted that all error messages should be accessible to users as well as suggestions on how to fix the errors. In addition, participants responded positively to the presence of error squiggles when an automatic compilation is triggered, something that has been

---

111

---

3https://www.w3.org/TR/WCAG21/
suggested as being potentially distracting to programmers, due to false positive errors appearing, caused by incomplete code implementation, in Part III.

Regarding the manner in which error messages are presented to the user, we found a statistically significant difference in participants’ responses when asked to rate the presentation featured in BlueJ 3 (enforced first error message) and BlueJ 4 (on-demand all error messages), with the latter being perceived as more useful. When asked which error messages participants address when presented with a list containing multiple error messages, the majority of participants (56.4%) responded all error messages, while a significant proportion (38.4%) reported addressing only the first in the list. In extension, most of the participants showed a preference of all error messages being accessible in a programming environment. In terms of the messages’ location, opinions show a division between showing the error messages under the editor in a dedicated area and showing them on the code area. We argue that display location is probably a matter of personal preference and dependent on the exposure that a programmer has to programming environments, although this does not apply to novices who have minimal experience in programming environments. Finally, most participants responded that they would prefer automatic display of error messages, without the need of performing an action within the environment.

Through this work, we add further evidence regarding the investigation of the differences between the compilation mechanisms employed by BlueJ 3 and 4. Furthermore, we provide multiple programming environment features that programmers with various experience levels perceive as useful for novices during the programming process. This is of interest to programming environment designers and to the computing education research community, as it is a step towards improving the effectiveness of programming environments, and in extent improving the programming experience of novices during their first steps of learning.

The next Chapter of this thesis concludes Part IV, and is in fact the last empirical study. We present a series of interviews that were conducted with postgraduate students in University College Dublin interacting with the compilation mechanisms present in BlueJ 3 and 4 while programming. Through this research, we perform an empirical evaluation of the behavioural findings so far, as well as the results from the online survey that was presented in this study. We finally provide a high-level description of the programming process when the different compilation mechanisms are present.
OBSERVATIONS ON THE PROGRAMMING BEHAVIOUR OF STUDENTS USING DIFFERENT COMPILATION MECHANISMS

Abstract

In the present work we conduct a user study of programming sessions with postgraduate students using BlueJ versions 3 and 4. We investigate the divergence in the programming process with respect to the different compilation mechanisms present in each BlueJ version using mixed methods. We evaluate our previous findings from analyses of non-contextual Blackbox programming process data regarding the displayed compiler error messages, manual compilations and their success rates generated by students using BlueJ, this time in a known context. Additionally, we perform an evaluation of our previous findings from the online survey that targeted how useful the individual compilation mechanisms present in BlueJ are perceived by programmers after only having the compilation mechanisms visually demonstrated to them, by having postgraduate students respond to the survey after performing the programming task. We find consistent patterns with multiple of our previous findings and we outline some observed differences regarding the survey responses, and the interaction between users and compilation mechanisms. Finally, we provide a higher level description of the behaviour of students while programming using different compilation mechanism setups.
CHAPTER 9. OBSERVATIONS ON THE PROGRAMMING BEHAVIOUR OF STUDENTS USING DIFFERENT COMPILATION MECHANISMS

9.1 Introduction

So far, the work described in this thesis has uncovered (1) differences in the interaction between novices and compilation mechanisms present in BlueJ 3 (manual compilation with enforced first error message presentation) and BlueJ 4 (automatic and manual compilation with on-demand any/all error messages presentation), (2) differences in compilation behaviour with respect to the syntax state of the source code, (3) differences in error resolution timings, with BlueJ 4 providing quicker reach of syntactically correct source code, and (4) differences in the interaction with compiler error messages. These differences were observed after analyzing context-agnostic programming process data generated by novice programmers, from the Blackbox database. In addition, in Chapter 8, we performed a survey which targeted the perceived usefulness of the individual mechanisms employed by each BlueJ version, while participants - programmers with multiple experience levels - provided a few suggestion and considerations on improving those. Two important caveats from analysis of the Blackbox data in the conducted research are (1) the fact that we have no contextual information about the BlueJ users (who they are, the code they are writing, the task setting, etc.), (2) the insights that are generated from the analysis are based on a limited range of information from the programming process, which is an obstacle in inferring more contextualized insights about programming behaviour. Further, in the online survey that was conducted, participants were demonstrated how the compilation mechanisms operate with the aid of pictures and animations. In the present study, we aim to evaluate some of our previous findings, but also to perform an observation study on programmers while they are interacting with the compilation mechanisms, but framing this interaction with a known programming context. We conduct and present a user-study with postgraduate students performing a programming task using BlueJ 3 and 4. We explore how the interaction between the user and the compilation mechanisms emerges and we evaluate it under the prism of insights that were formulated through analyses using the Blackbox programming process data described in previous chapters.

9.1.1 Aims and Objectives

This user study serves to provide the following:

• An evaluation of the findings in Part II of this thesis, regarding the quantity of displayed error messages, manual compilations, and percentage of manual compilation success.

• Insights on the reasons behind manually compiling, and how programmers
CHAPTER 9. OBSERVATIONS ON THE PROGRAMMING BEHAVIOUR OF STUDENTS USING DIFFERENT COMPILATION MECHANISMS

utilize error messages in situations where the compilation mechanisms operate differently.

• Further and more general insights on how programmers interact with the different compilation mechanisms and a more transparent and contextualized view of how programming behaviour is affected (as well as the degree in which this effect takes place), since the findings from the previous work using Blackbox data, although allow for generalizability due to large samples, they do not derive from any controlled experiments with live programming sessions.

• An evaluation of the results from the online survey in Chapter 8, which focused on the impression of programmers on the effectiveness of the compilation mechanisms featured in BlueJ 3 and 4, as participants were asked to rate the compilation mechanisms based on a brief demonstration with pictures and animations. The interviews described in the present study featured a more controlled cohort of participants, since their programming background was provided to the researcher.

9.2 Methodology

9.2.1 Participant Recruitment

The interviews were advertised via internal email (see Appendix B.1) that was sent to all the postgraduate students enrolled in Advanced Data Structures in Java (COMP47500) in University College Dublin (UCD) during the Spring Trimester of the 2021/2022 academic year.\(^1\) COMP47500 is an elective course that is primarily targeted towards students in either MSc Computer Science (Negotiated Learning) or MSc Computer Science (Conversion) in UCD. In order to enroll, students need to have completed a Java programming course in UCD (COMP20300 - Java Programming for Negotiated Learning students, COMP30820 – Java Programming (Conv) for Conversion students) or provide proof of prior learning in Java.\(^2\) The course covers advanced data structures and algorithms using the Java programming language. As such, enrolled students are already proficient in the Java syntax and exposed to at least some of its basic functionalities. We acknowledge that this target group is not the ideal for the purposes of this research study, since BlueJ is designed for novice programmers still in learning programming and Java.

\(^1\)This study has been fully approved by the UCD Human Research Ethics Committee – Sciences after a full ethical review (application number: LS-21-102-Karvelas-Becker).

\(^2\)More information can be found in: https://hub.ucd.ie/usis/!W_HU_MENU.P_PUBLISH?p_tag=MODULE&MODULE=COMP47500 (last accessed June 20, 2022).

115
However, given the time and the COVID-19 pandemic restrictions partially in place, targeting students that the primary researcher had direct communication with as a Teaching Assistant seemed the best possible strategy at the given time. Students were informed in the email about the purpose of the study, while it was explicitly stated that participation was voluntary and disjointed from their assessment in COMP47500. In return of their participation, each student was provided a 21 GBP (25.5 EUR) Amazon eGift Voucher. Students were asked to sign a consent declaration that was provided in the study’s announcement email (see Appendix B.2). In total, 16 students signed up over the academic trimester, and we split them equally in two groups, each being assigned a different BlueJ version to use during the programming task. The assignment of each student to a group was done by iteratively switching between groups while students contacted the researcher for participating.

9.2.2 Interview Materials

The interviews were conducted over Zoom and were structured in three parts. In the first part, participants were asked general questions about their programming background and some programming behaviour-related aspects. In the second part, participants were provided with a programming task and were asked to use a BlueJ version while implementing their solution (see Appendix B.3). Briefly, the programming task required to build two classes along with their associated attributes. Then, participants were asked to instantiate objects of these classes and sort them in an array based on one of the attributes. We considered the development of the classes to be an easy task for the participants, and the main method as the more difficult part, given the participants’ experience in Java programming. This allows to observe the programming behaviour of users in varying difficulty scenarios and identify differences in how they interact with compilation and error messages in each case. Participants were asked to complete the programming task on a single source code file within BlueJ. This was done for three reasons: (1) to ensure that the screen capture was uninterrupted during the Zoom session, (2) because this would simulate scenarios that novices are very often faced with (usually novices tend to use one source code file), and (3) because using a single source file that contains multiple classes and a main method is not a common practice, which the researcher hoped to create confusion and raise errors in the students’ code. We recorded the BlueJ programming window of participants using Zoom to compile the transcripts for later investigation. While programming, the researcher sometimes intervened to ask questions related to the observed behaviour. The task was expected to last 30-40 minutes, but most participants did not manage to finish the task on time.

3 A lot of participants were in fact not aware how to resolve this issue.
(mainly because of the final part being more advanced), and the programming session was interrupted by the interviewer. Posterior to part two, participants were asked to answer an online questionnaire that included questions related to their impression of using the programming environment. The materials used on the survey were identical to those used in the online survey described in Chapter 8 (see Appendix, Section B.4).

9.2.3 Quantifying the Interaction Between Participants and BlueJ

Each participant’s programming session recording was examined posterior to the interviews. The following information was extracted from each programming session:

1. The number of displayed compiler error messages
2. The number of manual compilations and their success state
3. The time point that participants performed their first manual compilation
4. The total session time

Shortly after examining the first sessions of participants working on BlueJ 4, we noticed that a large number of displayed compiler error messages are triggered inadvertently by the participants, while editing offending code areas (remember that interacting with an error location in BlueJ 4 triggers the message to pop up on the editor). We attempted to isolate these instances by keeping a record of the number of compiler error messages that each user triggered without intending to view an error message. Although there is no strict criterion for deeming a displayed error message as being unintentional and it would not be practical to ask the participants each time this was observed during the interview, the heuristic used can be described as either (1) the message appeared for a very brief amount of time, excluding the possibility of being consciously intercepted by the participant, (2) the trigger was the result of an edit in the offending code area that was intended before viewing the error message, or (3) the trigger was the result of the mouse cursor being moved from one point on the editor to another, but the user did not stop at the offending error location. Although these instances were examined multiple times to ascertain the correct number of unintentional triggers, this is noted as a potential threat to validity. Using the aforementioned information, the three interaction metrics discussed in Part II of this thesis were constructed for each participant:
1. Displayed Compiler Error Messages per Hour (DCEMpH). We used this metric to describe (1) all displayed error messages, (2) only the messages that appeared deliberately by the user

2. Manual Compilations per Hour (CpH)

3. Percentage of Successful manual Compilations (PSC)

9.3 Results

9.3.1 Demographics

In total, 16 students enrolled in COMP47500 signed up as participants in the interview. Participants were asked to provide general information about their computing background in the beginning of each interview. All students reported that they were studying for a MSc Computer Science (Negotiated Learning) degree. Almost all participants had already been exposed to advanced CS modules in UCD (e.g., Big Data Programming, Distributed Systems, Multi-Agent Systems, Machine Learning, etc.). In addition, all participants reported having knowledge of more than one programming language and professional programming experience which ranged from less than a year to several years prior to their MSc studies in UCD. Other demographic information about the participants were not collected.

9.3.2 Interaction with BlueJ

9.3.2.1 Quantitative Analysis

In terms of the interaction metrics, we observed similar patterns as in previous studies that relied solely on Blackbox programming process data. For DCEMpH, we measured a mean of 14.4 ($\text{SD} = 5.3$) for BlueJ 3 and 43.9 ($\text{SD} = 22.9$) for BlueJ 4. However, as mentioned in Section 9.2.3, we observed participants triggering the display of compiler error messages inadvertently in high frequency. Thus, after isolating these instances, we measured a reformed DCEMpH for participants in BlueJ 4 at a mean of 17.2 ($\text{SD} = 10.3$), which is still higher than BlueJ 3, albeit at a reduced magnitude (see Table 9.1 for the full distribution description and Figure 9.1).

In terms of manual compilations, participants exhibit a mean of 17.4 ($\text{SD} = 5.9$) in BlueJ 3 and a much lower rate ($\mu = 5.8, \text{SD} = 4$) in BlueJ 4 (see also Table 9.1 and Figure 9.2). Another observation is that participants in BlueJ 3 compile for the first time at a later timepoint during the programming session (see Table 9.2 and Figure 9.3).
CHAPTER 9. OBSERVATIONS ON THE PROGRAMMING BEHAVIOUR OF STUDENTS USING DIFFERENT COMPILATION MECHANISMS

Table 9.1: Distributions of the interaction metrics for the participants performing the programming task. The DCEMpH values under 4* represent the deliberate appearance of compiler error messages.

<table>
<thead>
<tr>
<th></th>
<th>DCEMpH</th>
<th>CpH</th>
<th>PSC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>4*</td>
</tr>
<tr>
<td>n</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>µ</td>
<td>14.4</td>
<td>43.9</td>
<td>17.2</td>
</tr>
<tr>
<td>SD</td>
<td>5.3</td>
<td>22.9</td>
<td>10.3</td>
</tr>
<tr>
<td>min</td>
<td>6</td>
<td>1.7</td>
<td>1.7</td>
</tr>
<tr>
<td>25%</td>
<td>10.9</td>
<td>32.8</td>
<td>11</td>
</tr>
<tr>
<td>median</td>
<td>15.3</td>
<td>49.1</td>
<td>18</td>
</tr>
<tr>
<td>75%</td>
<td>17.9</td>
<td>57</td>
<td>24.5</td>
</tr>
<tr>
<td>max</td>
<td>21.6</td>
<td>75</td>
<td>31.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>µ</td>
<td>14.4</td>
<td>5.8</td>
</tr>
<tr>
<td>SD</td>
<td>5.3</td>
<td>4</td>
</tr>
<tr>
<td>min</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>25%</td>
<td>10.9</td>
<td>8.3</td>
</tr>
<tr>
<td>median</td>
<td>15.3</td>
<td>18.3</td>
</tr>
<tr>
<td>75%</td>
<td>17.9</td>
<td>27.1</td>
</tr>
<tr>
<td>max</td>
<td>21.6</td>
<td>40</td>
</tr>
</tbody>
</table>

Finally, we measured a very low mean score for PSC in BlueJ 3 (\(\mu = 18.2, SD = 14.6\)), whereas in BlueJ 4 this was measured at 69.6 (\(SD = 22.7\)). The distributions are described more thoroughly in Table 9.1 and Figure 9.4.

9.3.2.2 Qualitative Analysis

The recorded sessions were viewed multiple times with the aim of identifying observable patterns of users while they were programming in BlueJ 3 and BlueJ 4. Initially, the sessions were observed in real time during the interviews, while keeping a record of some observed actions. After all interviews were completed, the recordings of the programming sessions were viewed and each participant was pro-
filed with a textual description (this process was conducted sequentially for the two BlueJ versions, starting with version 3). After examining the participants’ profiles, we tried to identify common patterns across their programming sessions. In the following paragraphs, the main observations that derived from the aforementioned process are presented.

**BlueJ 3** Participants exhibited a consistent pattern while programming in BlueJ 3. They spent at least the first half of the programming session without compiling at all. Towards the end, after having completed the code components for all three tasks of the programming exercise, participants switched their programming flow to “compile-edit-compile” cycles, until that were either halted by the researcher or until all compile-time errors were eliminated. It is of notable interest the fact that students exhibited an apparent change in their programming mode, a fact reflected in the changes in their programming pace. In almost all instances that participants clicked compile, they exhibited the following actions:

1. They read the error at the bottom of the BlueJ window.
2. They searched for the error indicator on the code.
3. They performed a potential fix.
4. They clicked the compile button to evaluate their fix.

There were two participants that demonstrated a different behaviour. Instead of leaving the debugging process for the very end, they started performing mini compile-edit-compile cycles during the editing process earlier on, usually initiating them after adding a significant functionality in the code (e.g., a complicated statement that utilized a comparator to compare string attributes of a couple of class instances). One student also seemed insecure at some point during programming and compiled a few times, possibly to reassure themselves and test what was the correct way to instantiate an object of one of the classes.

Evidently, the metrics that were used in previous analyses to describe the interaction between a user and the programming environment (DCEMpH, CpH, PSC) are highly volatile during the programming activity while using BlueJ 3. There are large time intervals during which users never compile, only edit their code. There are other instances during which the metrics show spikes – this is where the bulk of the metrics is formed. For example, one participant exhibited a DCEMpH of 15.7, a CpH of 18.261 and a PSC of 14.3%. However, their first compilation happened after 17 minutes from the start of the session, while the whole session lasted a total of 23 minutes. This means that the metrics describe only small ranges of the programming activity, and they are not uniformly distributed across the entire timeline of the programming session.
BlueJ 4 Participants exhibited a more complicated interaction behaviour with BlueJ 4, rendering it less tractable to describe (this was one of the insights formed in Chapter 4). One observation is that participants were exposed to compiler error messages much earlier during their programming sessions compared to BlueJ 3. However, there was a large number of instances where the messages were triggered to appear inadvertently (a quantification of this phenomenon is also reflected in Table 9.1). These unintentional appearances of messages manifested when users were editing offending code areas, without intending to see the messages. A large number of unintentional messages surfaced as a result of copying and pasting code from other code areas (e.g., when creating accessor and modifier methods for the Book class, users copied the code they had written from the Author class). These messages did not seem to disrupt the participants while they were programming, as there was not a noticeable decrease in their code editing momentum. This is a very important aspect that needs to be addressed in the manner in which Blackbox collects data, as it can potentially distort the insights derived from analysing large volumes of Blackbox data (or magnify their significance). Some of these instances are experienced by users, depending on the duration of the message’s appearance (which in turn highly depends on the momentum of the user's cursor movement over an error location). Other instances most certainly go unnoticed by the programmer, since the messages’ appearance lasts very briefly (these instances were successfully intercepted by the researcher due to the low playback speed used.
CHAPTER 9. OBSERVATIONS ON THE PROGRAMMING BEHAVIOUR OF
STUDENTS USING DIFFERENT COMPILATION MECHANISMS

MANUAL COMPILATIONS WERE A VERY RARE PHENOMENON IN BLUEJ 4. IN THE INSTANCES
WHERE PARTICIPANTS MANUALLY COMPILED (1) THEY PERFORMED A MAJOR EDIT (WHICH
POSSIBLY ADDRESSED A CHALLENGING ASPECT ACCORDING TO THE PARTICIPANT) IN THEIR CODE
RIGHT BEFORE CLICKING COMPILE, (2) THEY FINISHED UP THEIR EDITING, AND MANUALLY
COMPILED BECAUSE BLUEJ WAS NOT EXPLICITLY NOTIFYING THE USER THAT THE CODE WAS
DEVOID OF COMPILE-TIME ERRORS AFTER AN AUTOMATIC COMPILATION, (3) THEY COMPILED
EITHER AS A LAST VERIFICATION STEP BEFORE ATTEMPTING TO EXECUTE THEIR CODE, OR (4) THEY
ASSUMED THAT CLICKING THE COMPILE BUTTON WOULD ALSO EXECUTE THE CODE (PARTICIPANTS
WERE NOT DEBRIEFED ABOUT HOW THE ENVIRONMENT OPERATED BEFORE THE PROGRAMMING
TASK PRIOR TO THEM USING IT, AS WE WANTED TO OBSERVE HOW THEIR BEHAVIOUR WOULD
ORGANICALLY MANIFEST, WITHOUT INTERVENTION). IN ADDITION, ONE PARTICIPANT WAS OBSERVED
CLICKING THE COMPILE BUTTON THREE TIMES IN A ROW WITHOUT EDITING THE CODE IN BETWEEN.
WHEN ASKED TO JUSTIFY THIS ACTION, THE RESPONSE WAS “FORCE OF HABIT, I GUESS”.
THESE OBSERVATIONS PROVIDE A JUSTIFICATION OF THE HIGH PSC RATES OBSERVED
IN BLUEJ 4 (SEE TABLE 9.1).

IN GENERAL, PARTICIPANTS DID NOT EXHIBIT A DISTINGUISHABLE SWITCH IN THEIR
PROGRAMMING FLOW WHILE ATTEMPTING TO SOLVE THE PROGRAMMING EXERCISE, AS WAS THE
CASE WITH BLUEJ 3. INSTEAD, CORRECTING ERRORS OCCURRED IN TANDEM WITH BUILDING
THE CODE. IN MANY INSTANCES, PARTICIPANTS FIXED ERRORS WITHOUT UTILIZING ANY
MESSAGES. THE INSTANCES THAT PARTICIPANTS VOLUNTARILY SAW ERROR MESSAGES OCCURRED
VERY SHORTLY AFTER WRITING THE OVERTANDING CODE AND CHANGING CODE LINES. IN THESE

Figure 9.4: Percentage of Successful Manual Compilations (PSC) boxplots of participants
during the programming session (rectangle: mean, horizontal line: median).
instances, participants moved their mouse very quickly over the error indication to see the error message. A hypothesis derived from this observation is that if users do not expect an error to emerge and they notice an error indication, then they immediately try to rectify the mistake, instead of moving on and addressing it later.

9.3.3 Survey Questions

After completing the programming task, participants were asked a series of questions which aimed to measure their impression of using the two BlueJ versions, as well as to capture their preferences regarding the compilation mechanisms employed by a programming environment. The questionnaire followed the same structure and format as the one used in the previous study presented in Chapter 8 with online participants (the questionnaire can be seen in Appendix B.4). Tables 9.3 to 9.7 present the results for the mechanism ratings and the preferences provided by participants analytically. In general, the results conformed to the results present in the online survey conducted in Chapter 8. We provide the most notable differences between the responses to the survey of the present study and the responses from the postgraduate participants in the online survey below:

- Participants provided an even lower mean score for the compilation mechanism in BlueJ 3 ($\mu = 4.6, SD = 1.6$) compared to the online survey ($\mu = 5.5, SD = 1.2$), while their ratings for the compilation mechanism in BlueJ 4 was similar ($\mu = 5.4, SD = 0.9$) to the online survey ($\mu = 5.5, SD = 1.1$).

- In the previous study 45.5% of participants responded that they address all error messages when presented with a list of multiple error messages, 49.5% of participants responded that they address only the first, and 3% responded “Other”. In the present study, the preferences were formed as $5/16=31.3\%$, $8/16=50\%$, and $3/16=18.7\%$ respectively.

- In the previous study 50.5% of participants responded that they prefer the error messages to be shown on the code area of the programming environment, 43.6% of participants responded that they prefer them to be shown under the editor on a dedicated area, and 5.9% responded “Elsewhere”. In the present study, the preferences were formed as $5/16=31.3\%$, $11/16=68.7\%$, and $0/16=0\%$ respectively.

- In the previous study 60.4% of participants responded that they prefer the error messages to be presented automatically by the environment, 38.6% of participants responded that prefer them on-demand, and 1% responded “Other”. In the present study, the preferences were formed as $7/16=43.8\%$, $9/16=56.2\%$, and $0/16=0\%$ respectively.
CHAPTER 9. OBSERVATIONS ON THE PROGRAMMING BEHAVIOUR OF STUDENTS USING DIFFERENT COMPILATION MECHANISMS

Table 9.3: Breakdown of mean rating values provided by participants regarding compilation and error message presentation-related mechanisms present in the two BlueJ versions. Numbers inside the parentheses represent the measured standard deviations.

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>BlueJ 3</th>
<th>BlueJ 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilation Mechanism</td>
<td>4.6 (1.9)</td>
<td>5.4 (0.9)</td>
</tr>
<tr>
<td>Error Squiggles Triggered by Automatic Compilation</td>
<td>-</td>
<td>6.3 (1.1)</td>
</tr>
<tr>
<td>Compile Button Presence</td>
<td>-</td>
<td>5.6 (1.1)</td>
</tr>
<tr>
<td>Manual Compilation Frequency</td>
<td>4.9 (1.2)</td>
<td>2.8 (1.4)</td>
</tr>
<tr>
<td>Error Indication Mechanism</td>
<td>6 (0.7)</td>
<td>5.8 (1.3)</td>
</tr>
<tr>
<td>Error Squiggles Triggered by Changing Lines</td>
<td>-</td>
<td>5.6 (1.5)</td>
</tr>
<tr>
<td>Error Message Presentation Mechanism</td>
<td>4.5 (1.8)</td>
<td>5.6 (1.6)</td>
</tr>
</tbody>
</table>

Table 9.4: Participant responses when asked about their strategy of addressing errors.

<table>
<thead>
<tr>
<th>When presented with a list of multiple compiler error messages, which error message(s) do you address?</th>
<th>All</th>
<th>First</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlueJ 3</td>
<td>3/8</td>
<td>4/8</td>
<td>1/8</td>
</tr>
<tr>
<td>BlueJ 4</td>
<td>2/8</td>
<td>4/8</td>
<td>2/8</td>
</tr>
<tr>
<td>Total</td>
<td>5/16</td>
<td>8/16</td>
<td>3/16</td>
</tr>
</tbody>
</table>

Table 9.5: Participant preferences on the desired presentation of error messages.

<table>
<thead>
<tr>
<th>Which error messages would you like the editor to show?</th>
<th>All</th>
<th>First</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlueJ 3</td>
<td>6/8</td>
<td>2/8</td>
<td>0/8</td>
</tr>
<tr>
<td>BlueJ 4</td>
<td>7/8</td>
<td>1/8</td>
<td>0/8</td>
</tr>
<tr>
<td>Total</td>
<td>13/16</td>
<td>3/16</td>
<td>0/16</td>
</tr>
</tbody>
</table>

9.4 Threats to Validity

The present study utilized postgraduate students with multiple levels of experience, who already possessed knowledge of advanced programming concepts and were intermediate users of the Java programming language. As a result, any insights
Table 9.6: Participant preferences on the desired location display of error messages.

<table>
<thead>
<tr>
<th>Where would you prefer the error messages to be shown on an editor?</th>
<th>Under Editor</th>
<th>On code</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlueJ 3</td>
<td>5/8</td>
<td>3/8</td>
</tr>
<tr>
<td>BlueJ 4</td>
<td>6/8</td>
<td>2/8</td>
</tr>
<tr>
<td>Total</td>
<td>11/16</td>
<td>5/16</td>
</tr>
</tbody>
</table>

Table 9.7: Participant preferences on the desired manner of display of error messages.

<table>
<thead>
<tr>
<th>How would you prefer the messages be shown to you?</th>
<th>Automatically</th>
<th>On-demand</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlueJ 3</td>
<td>5/8</td>
<td>3/8</td>
<td>0/8</td>
</tr>
<tr>
<td>BlueJ 4</td>
<td>2/8</td>
<td>6/8</td>
<td>0/8</td>
</tr>
<tr>
<td>Total</td>
<td>7/16</td>
<td>9/16</td>
<td>0/16</td>
</tr>
</tbody>
</table>

derived from this study are not guaranteed to apply to novices. However, given the fact that the quantitative assessment of the participants’ programming interaction with BlueJ aligns with results from past studies that utilized Blackbox data, we can cautiously assume that further behavioural observations would also apply to novices. A similar study that would use novices as participants could provide an evaluation of this.

Since the interviewer was also the Teaching Assistant of COMP47500 during the trimester that the study was conducted, some participants could be psychologically impacted from being observed while programming, which could in turn influence their programming habits during the sessions and distort the results and insights derived from them. Performing the same task on their own (e.g., while practicing at home) could potentially provide different insights from those presently inferred.

Finally, the measurement of displayed compiler error messages did not exclude repeatedly viewed messages. In contrast, the Blackbox database records a message as being displayed only the first time it is triggered to appear in a given compilation. If another compilation occurs and the error that the message is associated with persists, then there is a new recording of its display in the database. This could explain the difference in the measurements of DCEMpH in the present study and the measurements of BlueJ users in Part II of this thesis.
CHAPTER 9. OBSERVATIONS ON THE PROGRAMMING BEHAVIOUR OF STUDENTS USING DIFFERENT COMPILATION MECHANISMS

9.5 Discussion

In this work, we performed a user-study featuring interviews with postgraduate students undertaking a programming exercise using BlueJ 3 and BlueJ 4, and responding to a questionnaire targeting the usefulness of each version’s compilation mechanisms according to their impression. This study serves as a qualitative evaluation of insights from the studies described in previous chapters that relied solely on large volumes of programming process data retrieved from the Blackbox database. This study also allowed us to observe the interaction between users and BlueJ in a known context. It also serves as an evaluation of the findings derived from the online survey conducted in Chapter 8 by investigating potential differences between the responses of the postgraduate students who participated in it and the postgraduate students who served as a study cohort in the present study.

Part of the analysis focused on quantifying the interaction between participants and each BlueJ version by utilizing the metrics used in previous chapters. We quantified the number of compiler error messages over time, the number of manual compilations over time, as well as the percentage of successful manual compilations. We noticed a very high number of displayed compiler error messages over time in BlueJ 4 (much higher than in studies presented in Chapters 3 to 5 using Blackbox data) - a very large percentage of which were triggered inadvertently by users while editing their code. These messages appeared briefly during the programming process, but users did not utilize them to correct their mistakes. To address this issue, we created a new metric to quantify the compiler error messages that appeared deliberately by users. The numbers of messages dropped significantly, although it still remained higher than the number of messages that users see in BlueJ 3. However, if this phenomenon occurs in the general population of novice programmers, then this would mean that novices see closely similar numbers of messages over time in both BlueJ versions. Although, as mentioned in Section 9.4, Blackbox’s technical specification with respect to recording error messages could potentially mitigate this issue to a degree. In contrast, if more experienced programmers interact with the environment at an increased pace compared to novices, then this would mean that non-deliberate triggers of messages would be more common than it would for novice programmers. A replication study with novices would clarify this issue further.

Manual compilations were found to be much less frequent in BlueJ 4 than in BlueJ 3, and they occurred when users added an advanced code snippet that they wanted to immediately check if it was correctly structured, when they were not receiving a warning from the environment that their code was devoid of compile-time errors, and when they wanted to perform a “final check” on their program right before executing it. In BlueJ 3, participants started performing manual
compilations later than they did in BlueJ 4, in order to check the correctness of their program while performing sequential “compile-edit-compile” cycles, until their program was devoid of errors - in which case they moved on to executing their code - or until they were notified to stop the programming task due to time restrictions.

The increased success rate in manual compilation in BlueJ 4 seems to be the after effect of the fact that error correction moves in tandem with building the code. BlueJ 4 offers intermediate feedback that users sometimes utilize in order to correct their errors, and in turn, the code rarely contain errors when users manually compile. In contrast, since BlueJ 3 does not provide any feedback on compile-time errors during the editing process, users leave the debugging for later stages of the programming sessions and as a result a lot of these compilations contain errors. Another factor in the observed high success rates of compilations in BlueJ 4 is that some of them occur since the environment does not explicitly inform the programmer that their code does not contain any errors while editing, urging the user to compile manually to ascertain the situation.

The qualitative analysis through observing the programming sessions of participants also revealed a significant differentiation in the manner that users tackled the error correction of their code. In BlueJ 3, participants spent more than the first half of the programming session to write the biggest part of their code, and left the debugging for later, towards the end. The debugging was structured in sequential actions of manual compilation, reading the error message, identifying the error location, performing a potential fix, and compiling again. This procedure occurred multiple times until either all errors were eliminated or the session was stopped by the researcher. In contrast, participants who were using BlueJ 4 performed the debugging occasionally while editing the code, sometimes by utilizing error messages and in other instances by spotting and identifying the errors themselves. In their final steps during the session, participants performed manual compilations right before moving to executing and testing their code. As is evident, depending on the compilation mechanisms present in the environment users alter their approach to correcting their errors, while adapting to the feedback they receive from the environment during the programming process.

Replicating the survey that was described in Chapter 8, by having participants use the environment instead of visually demonstrating its individual compilation functionalities also allowed to ascertain the validity of the study apart from a few deviations that are noted in Section 9.3.3.

The present study concludes Part IV of this thesis and marks the end of the conducted empirical studies. In the final Part, the overarching discussion of the findings is presented, along with implications, and suggestions for future work.
Part V

Discussion
10.1 Programming Environments: What Do We Want From Them?

Before proceeding to the discussion of findings that stemmed out of the research described in this thesis, we will first reflect back on the role of programming environments in novice programmers’ learning process. Programming environments, and in extent their embedded mechanisms, are tools that developers use to program. In this sense, a baseline for their functionality is a set of mechanisms that provide all the necessary tools required for software development: a code editor to write the source code, a compiler and interpreter for compiling and executing the source code, a console to receive feedback for the input code, as well as a user interface that allows the interaction between the programmer and all the embedded tools. Features that aim to assist developers on top of just providing functional use need to address established difficulties that emerge during the programming process or optimize the solutions that existing mechanisms provide.

In the case of novices, we first need to determine exactly what we want the environment to be effective in and explore its role in the life of a student in an introductory programming course. Is minimizing the time spent on writing code the objective? If that’s the case, large language models such as GitHub’s Copilot [106] could very well do the heavy work, without the programmer being required to act besides providing a description of the problem that the output code will address. However, this would not provide the student exposure to writing
their own code on top of other pedagogical implications that are in place [53]. Is minimizing the number of compile-time errors during the programming process the goal? Then a frame-based editor would solve this problem by eliminating the possibility of having compile-time errors in the first place, achieved with providing more controlled code structures to students. However, this would not allow students to hone their debugging skills. These are exaggerated examples of approaches that address specific objectives but eliminate other educational aspects from the learning experience of a programming student, such as learning how to deal with errors, and working with a programming language and a programming environment: tools that are widely used in professional contexts. Establishing the expectations from the programming environment is a task that is intertwined with the expectations we have from novice programmers, something that is not consistently established, especially on a large scale [14].

The fundamental idea behind improving the effectiveness of programming environments for novices can be summarized to the following statement:

“Programming environments should ease the programming process while allowing students to focus on the fundamental notions of programming.”

By endorsing this goal it is inferred that language syntax, compiler error messages, and programming environments are byproducts of the learning process which students have to deal with. Although, it has been recently found that exposing students to programming language syntax exercises has short-term benefits to their performance [105]. A different argument could be that students need the exposure to a “real” programming language, “real” problems that arise during programming and that they also need to learn how to use programming tools. Thus, we can refactor the aim of the programming environment to the following:

“Programming environments should maximize the quality of learning in the shortest amount of time possible, expose the novice to realistic aspects of the programming process while, minimizing the novice’s effort and difficulties that arise.”

This is a very complex undertaking. The present study does not examine the effectiveness of any of the compilation mechanisms with respect to a response variable, such as grades or learning outcomes. Variables, outcomes, and factors evolve [25], but the manner in which users interact with compilation mechanisms while programming is not something susceptible to change on similar magnitudes, at least regarding the features that are investigated in the present work. We have explored how design decisions for compilation mechanisms affect novice programming behaviour, since the intent of their design was to ease the programming process for the user in the first place.
10.2 Programming Process Data Repositories: What Do We Want From Them?

Programming process data (PPD) are an important source of information for advancing the knowledge of the Computing Education Research (CER) community. These data usually capture interaction events and source code of students, as well as logs from programming environments (e.g., compiler error messages, warnings, and hints that emerge while programming). Access to these details enables researchers to draw inferences based on (1) data that are often organically generated, which are valuable as they pertain to the natural context, (2) more detailed investigations that would be very difficult to conduct otherwise, due to the complex and multifaceted nature of the programming process.

Establishing standardized formats for PPD has been raised as an actionable direction towards data producers [128], since it could limit the barriers of collaboration among researchers. Furthermore, standardized data formats could assist in addressing the need for more replication studies, which is a major concern in CER [63]. ProgSnap [69] and ProgSnap2 [128] are initiatives that attempt to standardize the collection, storing, and availability of PPD by proposing a set of important information that need to be included in the data collection. Moreover, they provide abstract and flexible hierarchical structures for storing this information. The majority of the work in this thesis has been conducted using Blackbox data, which follow a similar format to the proposed in ProgSnap2 (although some features of Blackbox have been characterized as being problematic [23], making it complicated to use, see also Appendix A). In the following section, we provide suggestions related to the data collection that specifically target the BlueJ and Blackbox context, based on deficiencies that we identified through working with them and concerns that resulted from this.

10.2.1 Blackbox Implications

One technical aspect that is important to be noted is that from BlueJ 4 and onwards, Blackbox records all error messages that appear to the user, superficially because the user requested them. However, based on our study this is not the case. A very large number of messages appeared because users wanted to edit the offending code area without seeing an error message (although this could potentially be mitigated to a degree by the fact that Blackbox records a displayed error message only the first time it appears to the user in each compilation). This creates a complication and potentially leads to misleading inferences from the data. Although we found that the number of shown messages indeed increases when BlueJ offers on-demand error message presentation, the number of recorded
shown messages in Blackbox is much higher. One solution to this is to redesign the recording mechanism to store “shown_error_message” events based on a different heuristic, such as the following:

**Input:** lower_time_limit

```java
while user_cursor_location = error_location and user ≠ editing do
count_time()
end while
if counted_time ≥ lower_time_limit then
    recorded_event ← shown_error_message
else
    recorded_event ← early_shown_error_message
end if
```

We can set the “lower_time_limit” to an arbitrary value (or an average error message display time, although this would require a priori experimentation with novices as participants). Furthermore, this solution has two technical requirements: (1) that Blackbox records the movement of the user’s cursor, and (2) that Blackbox identifies an editing event even when the edit is inline (so far, edits are recorded when users change lines). These changes would most likely increase the demand in the server’s storage capacity substantially, unless the monitoring is designed to operate completely locally on BlueJ (which will in turn burden the user’s machine unnecessarily from the user’s perspective).

An additional suggestion based on our experience of working with Blackbox data is that it would be highly beneficial to future researchers if the database is modified to record the location of the edits that occur on the source code (so far, Blackbox records edits in the form of differences between source code snapshots and the researcher needs to reconstruct the source code for either manual or automatic inspection of the programming process).

### 10.3 Thesis Overview

We explored the role of compilation mechanisms in the programming behaviour of novices using the BlueJ programming environment to implement code in Java. This exploration covered three aspects. First, we examined the quantitative aspects of the core interaction between novices and compilation mechanisms: (1) the number of displayed compiler error messages that emerge over time, (2) the number of manual compilations that novices request from the environment over time, and (3) the percentage of manual compilation success. Second, we examined the interaction of novices with compiler error messages and their compilation behaviour: how much time they spend on sequential source code states based on the soundness of their code with respect to compile-time errors, the triggers of compilation that
occur during the programming process, as well as the error resolution time. Third, we investigated whether individual components of the compilation mechanisms (compilation functionality, error indication, and error message presentation) are perceived differently in terms of their usefulness by programmers with different experience levels. Since the behavioural findings of our work derived from context-agnostic programming process data mined from the Blackbox database, we also conducted a user study that aimed to evaluate our findings and put them in context of a specific programming task with participants whose background was known. We formalized our inquiry using the following research questions:

**RQ1:** How is the quantified interaction between novice programmers and programming environments affected by changes in the way compilation mechanisms operate within a programming environment and how does transitioning between mechanisms affect this interaction?

**RQ2:** How is the compilation behaviour and interaction with compiler error messages altered by changes in the way compilation mechanisms operate within a programming environment?

**RQ3:** How do programmers from different experience levels perceive these mechanisms in terms of their usefulness and what are their suggestions on improving them?

**RQ4:** How do the interaction and behavioural findings of this research emerge while observing programmers using the compilation mechanisms during the programming task?

We will proceed to discussing the overarching findings for each research question in the sections that follow.

### 10.4 RQ1. Quantified Interaction and Transitioning Between Mechanisms

We presented evidence on the difference in the interaction between novices and compilation mechanisms, when the latter varies in terms of functionality. On-demand error message presentation allows students to explore the feedback provided by the compiler with more freedom as opposed to being restricted by enforced first error message presentation. Students choose to view more error messages over time than when they are programming using an enforced first error message presentation. Although error messages are considered cryptic and highly confusing for students, they are being utilized during the programming process. However, this
does not mean that error messages directly contribute to the resolution of errors by students. Since compilers are often the only source of feedback for students who are performing a programming task, students could be utilizing the error messages as means of searching for a solution online [156], especially after considering the magnitude of the availability of online sources (e.g., StackOverflow) for solutions on programming-related issues during the past decade [116] and the number of online questions related to compiler error messages [151]. Thus, in this sense, messages could simply be series of keywords without any directly contributing value to the novice apart from being used for an online search. In terms of which approach of error message presentation is more efficient, there are many questions that need to be explored by the research community to provide an answer. Do we want students to get exposed to more compiler error messages, as they are a fundamental aspect of the programming process? Do we want novices in CS1 to learn how to program, while avoiding getting distracted by any technicalities, such as language syntax, error messages, working with a programming environment, etc.? Both approaches have their merits.

The problem of granting freedom within the interaction space between users and programming environments over a guided restriction (which is posed by the enforced first error message presentation) can be reduced (or expanded) to the debate over two major teaching approaches that stemmed out of notions of constructivism by Piaget and Vygotsky [54] and guided learning [20] theories. From a practical perspective, Stride, a frame-based editing language [90, 91, 45] eliminates the problem of exposure to error messages by making it impossible to the user to make syntax errors in the first place (a restriction of the user), while BlueJ 4 and BlueJ 5 allow users to explore the feedback provided by the environment at their own pace (users are granted freedom). A possible alternative to these approaches could be a tool that provides instructional scaffolding to novices, incrementally exposing them to more complicated structures and aspects of the programming process, since novices utilize subsets of the programming languages [25] (e.g., Hedy [56] provides a gradual programming approach to novices, as they are increasingly getting introduced to more advanced programming concepts and programming language syntax structures). If compiler error messages are the main venue through which developers receive (and will continue to receive in the foreseeable future) feedback on their code and it is beneficial for students to be exposed to them and better prepare themselves for more advanced programming contexts, then compiler design and compiler error messages are the fundamental aspects that need to be improved, since programming environment feedback relies heavily on those. Although recent developments of large language models that are able to automatically generate code from text instructions may cause complications in the way introductory programming is taught [53, 61], alter the expectations we
have from programmers, and in extent shift the functionalities of programming environments.

The presence of automatic compilation while editing the code makes students compile manually less frequently than they do over time in a manual compilation context. This is, from a user-experience perspective, an improvement in terms of the quantity of required actions from the user, as they are no longer required to repeatedly compile themselves to check for errors. However, automatic compilation provides an increase in the error indications on the code (which is traditionally achieved through red wavy underlines – “error squiggles”). The success states of automatic compilation while editing are highly volatile due to the incomplete code segments that very often result to errors. These errors are false positives, but users experience them as error squiggles, rendering automatic compilations a redundant feature to a degree if they are not being utilized. False positive error messages have been reported by developers as an obstacle to comprehension [8, 75], further supporting the claim that they negatively impact the programming process, especially in the case of novices. A future study that explores the degree that error squiggles are distracting for users while programming would provide more evidence on the suitability of this error indication mechanism. In the present thesis, although we advocate against it, we do not provide any hard evidence to support or refute it.

In addition, manual compilations are more often successful when the environment offers automatic compilation than when only manual compilation is present, as we found that users utilize compilation partially for reassurance, since the environment does not inform about the correct syntax state of the program (providing positive feedback has been suggested to have a positive effect on student engagement with the programming task [108]). In addition, early feedback provided by the automatic compilation aids users in correcting their errors earlier in the programming process (which could support the suggestion that proactive feedback assists in eliminating misconceptions formulated while programming [100]), and manual compilations are the last steps before source code execution or the evaluating agents of complex code structures that users are not confident about. In contrast, when only manual compilation is present, the role of compilation is mostly functional: users invoke the compiler to check for errors.

The fundamental aim of the exploration of the quantified interaction was to investigate whether alterations in the compilation mechanisms affect novice programming behaviour. We found that not only the environment restrictions through compilation mechanisms affect this interaction, but whether users have been exposed to one mechanism over multiple plays a substantial role in the magnitude of difference in the interaction with different environments, as students adapt their interaction habits accordingly (conforming to environment functionalities has
been observed even in professional developers [100]). We also explored if the order in which novices are exposed to compilation mechanisms contributes to this, but we did not find any major variations in the behaviour of students. Nevertheless, compilation mechanism design decisions have a visible effect on the novice’s programming process. Future work could explore the implications of this behaviour variations on learning, a topic that could become another dissertation by itself at the very least.

10.5 RQ2. Compilation Behaviour and Interaction with Compiler Error Messages

We found that users still manually compile when automatic compilation is present in the programming environment, despite this not being required to compose and debug programs, as code is automatically compiled during editing and any lingering errors are indicated. These users compile at a reduced frequency compared to those using manual compilation, which is the only way to reveal error messages in order to diagnose errors. An important finding is that manual compilations in an automatic compilation context are successful around 76% of the time, as opposed to 52% when only manual compilation is present. This supports the hypothesis that some manual compilations in automatic compilation contexts may be psychologically-driven. A possible cause for this is that users are not entirely confident that their code is correct only by relying on the fact that no errors are indicated by the editor. In turn, they manually compile to double check that their code is indeed syntactically correct. This is further supported by our observation of the programming process of postgraduate students in Chapter 9. An interesting observation is that some users also tend to double or triple-click on the compile button unnecessarily (with no change to the source code). This could be a sign of stress caused by cognitive dissonance, where users’ perception of their code’s state does not align with the feedback they receive from the environment [52, 64], although further work is required to evaluate this.

With respect to compilation activity based on syntax states when novices use the two examined compilation mechanism setups, multiple similarities and differences are observed. In both setups, for approximately half of the total programming time users are working on repeatedly compiler error-free source code. For a smaller yet substantial amount of time, users work on repeatedly erroneous source code. A less substantial portion of total time is spent on editing previously syntactically correct code and then introducing new errors. Finally, the smallest amount of total programming time is spent on fixing (or removing) erroneous source code successfully. This is an important finding – overall these novices spend the least
time successfully fixing errors, more time introducing errors into previously syntactically correct code, even more time unsuccessfully attempting to correct errors (or editing their source while experiencing red error squiggles by the constant failing of their automatic compilations), and the most time editing error-free code without introducing more errors. Although the fact that about half of programming time is spent editing error-free code without introducing new compiler errors can be viewed as positive, the fact is that the bulk of the other half of programming time is spent dealing with errors, often unsuccessfully. In this error-ridden programming time, these novices are introducing new errors or dealing with existing ones unsuccessfully. Unfortunately, they spend the least time of all successfully fixing errors that they are dealing with for a substantial portion of their programming time.

Differences in the compilation activity between the two compilation mechanisms are observed in the total time spent between failed compilations, with automatic compilation users spending more time in this state compared to manual compilation users. However, the majority of these instances occur between automatic compilations where the compiler may report errors in incomplete code that would not be errors in the near future if the remaining code is successfully entered. This is also supported by the fact that users seem to be more quick in introducing errors to their syntactically correct source code in automatic compilation context compared to manual. Finally, more total time is spent to perform the final edit(s) in order to fix previously erroneous source code in an automatic compilation context.

Regarding the individual timings of compilation sequences, BlueJ 4 users experience less time between compilations as automatic compilations occur whenever users edit their code or change lines. The duration of all compilation sequences’ (grouped by syntax states) are longer in BlueJ 3 than in BlueJ 4. After further investigating the compilation sequences in BlueJ 4 by classifying them based on the trigger states (automatic or manual), we observed that the duration of compilation sequences in which the previous compilation was successful and manual is substantially increased compared to all other sequence durations. This could be explained by assuming that these compilations serve as internal milestones for users. These users might have reached a satisfactory state of source code and spent some time considering their next steps after compiling. This is even more reinforced when considering the fact that compilation sequences in which the compilation status changed from successful/manual to failed/automatic last the longest of all possible sequence permutations. In order to perform an automatic compilation, users have to edit their code – meaning that their first line change in the source code was much later than the previous compilation.

Finally, after considering the error resolution time in each BlueJ version to be the time between two successful (non-sequential) compilations and considering the
total time spent on syntactically incorrect code, we found that users spend more
time on syntactically correct code in BlueJ 4 than they do in BlueJ 3. This is likely
due to BlueJ 4 providing intermediate feedback during code editing, resulting in
users reaching syntactically successful source code states faster.

We explored how novices interact with compiler error messages when there
is enforced error message presentation versus when there is on-demand any/all.
We identified fundamental differences in multiple aspects of this interaction and
provided details into how users utilize error messages when they are free to ex-
plore them on their own. Inherently, when users have access only to the first error
message present in a programming environment, the interaction is restricted to
focusing on the error that the message is associated with on the code (if the indi-
cated location is correct). When all error messages are accessible, the interaction
is more complicated both in terms of how and when the messages are accessed,
but also of which messages are presented to the user. A very important finding
was that the majority of compilations when automatic compilation is present are
not utilized to see any error messages. This directly nullifies the importance of
error indications dynamically emerging while editing the code, although some
error indications may be utilized to identify and resolve more trivial errors without
seeing their associated message. The majority of error messages (around 75%) is
viewed between repeatedly failed compilations, which is supported also by the
study in Chapter 9, where students were viewing messages while building their
code, instead of entering a “debugging” mode to correct their error as is the case in
BlueJ 3. The rest of messages are mostly viewed right before performing edits that
make the code free of compile-time errors. In addition, we found that most error
messages (around 80%) are presented during repeated automatic compilations.
In every failed compilation there is an average of 3-4 error messages, while the
number of viewed messages is on average less than 1 (remember that most failed
compilations do not result in a shown error message). In cases where more than
one error message is viewed by a user in a failed compilation, users as a rule, read
those from a top left to bottom right order (as reading an English language text).
Finally, we identified a lot of variations in the display frequency of messages, when
the first is enforcedly shown versus when all messages are accessible on-demand.
Specifically, we found that 8 out of the top 17 most viewed error messages in BlueJ
are displayed less frequently in BlueJ 4. The rest of messages in the list of 17 most
common showed an increase of display in BlueJ 4. One explanation is that syntax
errors are more likely to occur in incomplete code structures, as the compiler does
not identify the expected tokens and in between compilations, novices see these
messages, possibly unnecessarily, but they would not be aware of this. However, as
we observed in our observational study in Chapter 9, a lot of shown error messages
are triggered inadvertently by users.
10.6 RQ3. Programmers’ Perception of Compilation Mechanisms

One additional aspect that was explored in this thesis, was the perceived usefulness of the examined compilation mechanisms based on programmers’ views. Participants showed no difference regarding how useful they find compilation and error indication present in BlueJ 3 and BlueJ 4. However, they showed a stronger preference on the manner that BlueJ 4 presents error messages over the one in BlueJ 3. Participants also showed positive response towards manual compilation when automatic compilation is embedded in the environment. Most participants also expressed that they prefer having access to all error messages in an environment. Regarding the location and manner of message presentation, opinions were divided (although most participants expressed that they want messages to be presented automatically, probably because this saves them time and effort when compared to performing additional actions to receive feedback from the environment), indicating that personal experience is a significant factor in user preference. Providing customization options in an IDE could potentially address more users’ preferences. Participants were also positive on the error squiggles presence, with very few of them negatively rating them as a mechanism. Participants who responded that they would often compile manually, explained that the reason for their action is reassurance that their code is correct. This supports our claim that the compiler’s role has psychological implications to a degree. Although compilation was not found to have a difference with respect to participants’ perception, many participants expressed that a keyboard binding for manual compilation and the presence of automatic compilation are important factors to them. A lot of participants also expressed that automatic compilation should be at a different level of granularity instead of when changing lines. One additional suggestion was to indicate the line numbers that an error occurred within the editor.

An overlooked aspect in the studies in Chapters 8 and 9 was to make an inquiry on the past experiences of participants with compilation mechanisms. Having more information about the mechanisms that participants have been mostly exposed to, could be highly informative and provide the means to explore past experiences with users’ perception of the examined mechanisms. This is a fruitful avenue for future work on this domain.

10.7 RQ4. Changes in the Programming Process

The observed findings from the studies utilizing Blackbox programming process data (Part II and III) allowed for inferences regarding the programming process of students on a large scale. We were able to draw insights from purely context-
agnostic information, without knowing the aim of the programming tasks, level of
difficulty, student experience, etc. In Chapter 9, we performed an observational
study with postgraduate students performing a programming task. The study
allowed us to situate our previous findings from Part II at context and draw
higher level inferences on the changes that the studied compilation mechanisms
drive in the programming process. Although the participants were not from a
CS1 context, our observations aligned with previous assumptions. We provided a
summary of the programming activity, focusing on how participants approached
the error resolution. We found that participants that used “manual compilation
with enforced first error message presentation” approached the programming task
by writing the majority of their code first, and towards the end they switched to
“compile-edit-compile” interaction cycles in order to address the compile-time errors
that were present. In contrast, participants exposed to “automatic and manual
compilation with on-demand any/all error message presentation” were resolving
the emerged errors in tandem with implementing their code. Participants were
utilizing error messages by interacting with error indicators primarily right after
implementing programming statements that were more complicated. Their manual
compilation occurred rarely and revolved around instances where there was no
information provided by the programming environment about the correctness
of their code, after challenging edits of the source, and as last verification steps
before proceeding to executing their programs. In these instances, their manual
compilations were very often successful, justifying the high success rates of manual
compilation that we observed in Blackbox data (Part II). Future studies should
be focused on replicating this experiment with more inexperienced programming
students. Although the qualitative analysis of our interviews aligned with most
of our prior assumptions, a replication study would provide even more robust
inferences.

10.8 Suggestions on Designing Compilation
Mechanisms

Based on the findings of the described work and the author’s insights while re-
searching this domain, a set of guidelines for designing compilation mechanisms
(and programming environments) is adduced below:

1. Automatic compilation should be present, although its frequency and relation-
ship with error indication should be further explored. “Just-in-time” [47] static
analysis concepts, or “idle time” compilations could persevere the
benefits of automatic compilation without disrupting the programmers’ work-
flows [75].
2. Manual compilation should be present, as users are utilizing it before execution and after implementing more challenging code segments. Also include a key-bind for compilation and make it known to the user, as it potentially saves time if utilized by the programmer. Toolbar buttons are easy to learn, but keyboard shortcuts are more efficient [95]. Making keyboard bindings more accessible by conveying their identity via toolbar buttons to assist users in remembering them through visual recognition [55] is very useful in this case.

3. Include salient positive feedback [108, 142] from the compiler when compilations are correct, especially in automatic compilations. Positive feedback has been identified as an important part of learning, even in programming contexts [108]. There is also evidence that positive feedback may have positive effects on novices’ performance in learning simple textual programming languages [96]. Considering that the feedback provided in programming environments is inherently negative (e.g., errors and error messages), it is important that environments adopt positive responses that are delivered in a clear manner to the user. In this work, it was found that some users compile to make sure that their code is devoid of errors when the environment is not providing information about it.

4. Error indication should be distinct and not blend with other colours present in the code editor. Since the purpose of this feature is to warn users about errors, visual representation should be vivid. However, this does not necessarily apply to cases where the indication occurs while the user is potentially preoccupied with a different task within the editor. In these cases, distraction should be avoided, as developers do not favor mechanisms that disrupt their workflows [75]. Colorgorical [58] and the contrast rules of the Web Content Accessibility Guide\(^1\) can be utilized to make elements of the editor more accessible to users.

5. Allow access to more than just the first error message. Users utilize messages in a non-deterministic manner than simply utilizing the first. Provide an automatic manner of showing the messages to users, and include a redirecting feature that maps each message to the offending code area.

6. Make the user interface aesthetically modern looking and pleasing to the user. Novices are attracted to industrial-grade IDEs, partially because of their slick appearance.

\(^1\)https://www.w3.org/TR/WCAG21/
7. Provide the option of customizing the environment, but at the same time, do not over-encumber the main window. The editor should contain a minimal set of information.

8. Perform observational studies with participants from the target group that the IDE is aimed for. In this thesis, we explored the impact of the environment on the behaviour by automatically analyzing the programming activity of thousands of users, but we found that our observational study in Chapter 9 was highly informative, even with a limited number of participants. Investigate the degree that users respond to the mechanisms with respect to their design’s intent. Furthermore, re-iterate and question whether the design decision is appropriate based on observations of their usage.

10.9 Suggestions to the Research Community

Based on the inquiry of this thesis, we present high-level suggestions on the research directions that the research community should focus in future work:

1. Investigate different levels of automatic compilation granularity, their ties to performance trade-offs (in CS1 contexts, performance is not a significant factor, due to the code simplicity), and their effect on users. Smart features that perform timely compilations could rid the programming process of unnecessary compilations. Machine learning could provide a solution for this feature.

2. Invest on compiler design and compiler error message research. The present work has indicated that compilation mechanisms rely heavily on the accuracy of the compiler, and possibly on their explainability. The environment is a filter, but the compiler is the main agent in the feedback process.

3. Invest on identifying accurate and scalable response variables for evaluating programming environments (e.g., speed, learning outcomes, etc.) and provide experimental materials/exercises that researchers can use to evaluate their developed mechanisms on. This field is very challenging in identifying the desired outcomes and more effort is needed towards establishing them.

10.10 The Verdict

Compilation mechanisms are an integral part of the interaction between programmers and programming environments. These mechanisms have a number of implications in the programming process, ranging from the interaction habits
with the environment, specifically towards compilation and error messages, to higher level error resolution approaches. Thus, design decisions play a major role in shaping how users - especially novice programmers - behave while programming. It is reasonable to assume that these changes in behaviour have implicit effects on the learning process. However, more studies are required to investigate this further. In terms of how these mechanisms are perceived, we identified that programmers - regardless of their experience levels - have stronger preferences towards the “automatic and manual compilation with on-demand any/all error message presentation” mechanisms over the “manual compilation with enforced first error message presentation”, mainly due to the manner that error messages are presented within the programming environment. In addition, programmers provided highly diverse suggestions on improving compilation mechanisms present in BlueJ. This diversity could be the effect of the diverse exposure to past experiences with programming, languages, used mechanisms and environments. Programming tool designers should be thorough in the design process, as their products have an important role in the programming behaviour, while also taking into account that the majority of programmers already have preferences shaped by prolonged exposure to existing tools. In the case of novices, the task of designing effective and at the same time aesthetically appealing environments is a major challenge, but one that would attract students to using truly beneficial environments while they learn.

***
“He’s dreaming now,” said Tweedledee: “and what do you think he’s dreaming about?”

Alice said “Nobody can guess that.”

“Why, about you!” Tweedledee exclaimed, clapping his hands triumphantly.

“And if he left off dreaming about you, where do you suppose you’d be?”

“Where I am now, of course,” said Alice.

“Not you!” Tweedledee retorted contemptuously.

“You’d be nowhere. Why, you’re only a sort of thing in his dream!”

* Source: Through the Looking-Glass, and What Alice Found There (1871)

Chapter 4: Tweedledum And Tweedledee
Part VI

Appendices
THE BEAST IN BLACK: FROM BLACKBOX TO DATA

A.1 Author’s Note

The major source of data that was utilized in the present thesis was mined from Blackbox, a repository of programming process data that are generated by BlueJ users since 2013. Blackbox is essentially a MySQL database with a large schema which comprises dozens of interconnected tables that facilitate an extensive snapshot of the programming process of BlueJ users. Blackbox holds a tremendous amount of information (e.g. Terabytes of textual data, the exact size of which in 2018 can be found in [23]). The database’s size, combined with the fact that a major portion of the stored data are organically generated by novice programmers interacting with a programming environment to write Java programs makes Blackbox a truly valuable source of information for research. However, working with a repository of this size encapsulates a lot of challenges and caveats.

A.2 Challenges of Working with Blackbox

1. First and foremost, dealing with a complicated schema such as Blackbox’s is - from a technical perspective - challenging by itself. The researcher needs to
spend a substantial load of work to start comprehending how to conduct their experiments: which database tables to use, how to connect them correctly, and how to extract and transform the retrieved data to meaningful datasets that can be later processed during the data analysis.

2. Second, SQL is inherently difficult when working with databases of this size, since queries can contain semantic mistakes that are easy to miss, leading to overly long response times from the Blackbox server.  

3. Third, if there is a semantic mistake present in a SQL query, then there is a high probability that the mistake will be discovered during the analysis of the extracted information (e.g. while processing the final datasets), which will in turn create a feedback loop to challenge 2. Fun times.

4. Fourth, even if all SQL queries are masterfully written, there will always be a bit of information that the researcher forgot to include in their original SELECT statements. Then, if goddess Tyche is on their side, the left out information can easily be retrieved by writing a simple query and appending the extra information to the originally retrieved data. The author will allow the reader the privilege of imagining what the next steps are in the opposite case.

5. Finally, since Blackbox data are inherently non-contextual, the researcher needs to spend time to think how to deal with noise, outliers, and undesired elements in big data analysis in general. In the present thesis, identifying suitable BlueJ users was always the starting point of working with Blackbox.

**A.3 Blackbox Data in the Present Thesis**

In this section, all the Blackbox database fields that were used in the present thesis are briefly outlined and explained (see Table A.1). The order of data retrieval, the transformation to datasets, and the specific analyses that followed were highly

---

1This challenge provided the author of this thesis ample opportunity to discover a few creative and innovative - albeit cruel - ways to get notified whenever a query was complete. Hint: the secret is always in traditional folk music with insufferably loud bagpipes in an office full of PhD students. 

148
volatile processes that kept evolving as the author kept progressing through his PhD, making a consistent “low-level” technical approach impossible to describe in the present thesis. However, we mention all the database information that was used throughout this thesis, with the hope that it will aid future researchers in narrowing down their search in Blackbox, should their work be related to our scope.

Table A.1: Blackbox database fields that were used through the work described in the present thesis. The labels in the horizontal parallel lines signify the database table that the subsequent fields are located in Blackbox.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>users</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>numeric</td>
<td>The unique identifier of a user</td>
</tr>
<tr>
<td>created_at</td>
<td>datetime</td>
<td>The timestamp that the user first appeared in Blackbox</td>
</tr>
<tr>
<td><strong>sessions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>numeric</td>
<td>The unique identifier of a session</td>
</tr>
<tr>
<td>user_id</td>
<td>numeric</td>
<td>Identifier that links sessions with users</td>
</tr>
<tr>
<td>created_at</td>
<td>datetime</td>
<td>The timestamp that the session was created</td>
</tr>
<tr>
<td>installation_details_id</td>
<td>string</td>
<td>Identifier that links sessions with installation_details</td>
</tr>
<tr>
<td><strong>installation_details</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>numeric</td>
<td>The identifier of installation_details</td>
</tr>
<tr>
<td>interface_language</td>
<td>string</td>
<td>The language that the BlueJ installation interface is in</td>
</tr>
<tr>
<td>operating_system</td>
<td>string</td>
<td>The operating system of a user’s machine</td>
</tr>
<tr>
<td>bluej_version</td>
<td>string</td>
<td>The BlueJ version of an installation</td>
</tr>
<tr>
<td>java_version</td>
<td>string</td>
<td>The java version of a BlueJ installation</td>
</tr>
<tr>
<td><strong>packages</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>numeric</td>
<td>Identifier for a package</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>Package name</td>
</tr>
<tr>
<td>project_id</td>
<td>numeric</td>
<td>Identifier that links the package with the project</td>
</tr>
</tbody>
</table>
### APPENDIX A. THE BEAST IN BLACK: FROM BLACKBOX TO DATA

#### Continuation of Table A.1

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>master_events</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>numeric</td>
<td>Identifier of a master event.</td>
</tr>
<tr>
<td>user_id</td>
<td>numeric</td>
<td>The user identifier that the master event is associated with</td>
</tr>
<tr>
<td>project_id</td>
<td>numeric</td>
<td>The project identifier that the master event is associated with</td>
</tr>
<tr>
<td>event_id</td>
<td>numeric</td>
<td>Identifier that links the master events table with specific event tables (e.g., compilations)</td>
</tr>
<tr>
<td>event_type</td>
<td>string</td>
<td>The name of the event in uppercase camel-case, non-underscored format</td>
</tr>
<tr>
<td>created_at</td>
<td>datetime</td>
<td>The timestamp of the master event</td>
</tr>
<tr>
<td>source_time</td>
<td>datetime</td>
<td>The timestamp of the master event (user’s machine time)</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the master event (e.g., compile, edit, etc.)</td>
</tr>
<tr>
<td>session_id</td>
<td>numeric</td>
<td>The session identifier the master event is associated with</td>
</tr>
<tr>
<td>package_id</td>
<td>numeric</td>
<td>The package identifier that the master event is associated with</td>
</tr>
<tr>
<td><strong>compile_inputs</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>numeric</td>
<td>An entry’s identifier. The role of this table is to connect compile events with the source files that they are associated with</td>
</tr>
<tr>
<td>compile_event_id</td>
<td>numeric</td>
<td>The compile event identifier</td>
</tr>
<tr>
<td>source_file_id</td>
<td>numeric</td>
<td>The source file identifier that the compile event is associated with</td>
</tr>
<tr>
<td><strong>compile_events</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>numeric</td>
<td>The identifier of a compile event.</td>
</tr>
<tr>
<td>success</td>
<td>boolean</td>
<td>The success state of a compilation (1-success, 0-fail)</td>
</tr>
</tbody>
</table>
## Continuation of Table A.1

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason</td>
<td>string</td>
<td>The trigger of a compile event (e.g., “user” for manual compilation)</td>
</tr>
<tr>
<td>session_id</td>
<td>numeric</td>
<td>The identifier of the session that the compile event is associated with</td>
</tr>
<tr>
<td>compile_outputs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>numeric</td>
<td>The identifier of a compile output</td>
</tr>
<tr>
<td>is_error</td>
<td>boolean</td>
<td>The error state of the output (1-error, 0-warning)</td>
</tr>
<tr>
<td>compile_event_id</td>
<td>numeric</td>
<td>The identifier of the compile event that the compile output is associated</td>
</tr>
<tr>
<td>start_line</td>
<td>numeric</td>
<td>The line number that was the compiler identified as start of an offending</td>
</tr>
<tr>
<td>end_line</td>
<td>numeric</td>
<td>The line number that was the compiler identified as end of an offending</td>
</tr>
<tr>
<td>start_column</td>
<td>numeric</td>
<td>The column number that was the compiler identified as start of an offending</td>
</tr>
<tr>
<td>end_column</td>
<td>numeric</td>
<td>The column number that was the compiler identified as end of an offending</td>
</tr>
<tr>
<td>message</td>
<td>string</td>
<td>The message that was relayed as an output of the error/warning</td>
</tr>
<tr>
<td>source_file_id</td>
<td>numeric</td>
<td>The identifier of the source code file that the output is associated with</td>
</tr>
<tr>
<td>shown</td>
<td>boolean</td>
<td>The display state of the output (1-shown, 0-not shown)</td>
</tr>
<tr>
<td>session_id</td>
<td>numeric</td>
<td>The session identifier that the compile output is associated with</td>
</tr>
<tr>
<td>source_files</td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>numeric</td>
<td>The identifier of the source code file</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the source code file</td>
</tr>
<tr>
<td>project_id</td>
<td>numeric</td>
<td>The project identifier that the source file is associated with</td>
</tr>
</tbody>
</table>
### Continuation of Table A.1

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shown_error_indicators</td>
<td></td>
<td></td>
</tr>
<tr>
<td>master_event_id</td>
<td>numeric</td>
<td>The master event identifier for an error indicator</td>
</tr>
<tr>
<td>compile_output_id</td>
<td>numeric</td>
<td>The compile output identifier that the indicator is associated with</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>projects</td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>numeric</td>
<td>The identifier of a project</td>
</tr>
<tr>
<td>user_id</td>
<td>numeric</td>
<td>The user identifier that the project is associated with</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the project</td>
</tr>
<tr>
<td>source_histories</td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>numeric</td>
<td>The identifier of an entry in source_histories</td>
</tr>
<tr>
<td>master_event_id</td>
<td>numeric</td>
<td>The master event identifier that the entry is associated with</td>
</tr>
<tr>
<td>source_file_id</td>
<td>numeric</td>
<td>The source file identifier that the entry is associated with</td>
</tr>
</tbody>
</table>
B.1 Participant Recruitment Advertisement

Dear COMP47500 students,

Ioannis Karvelas, the module’s Teaching Assistant is doing a research study as part of his PhD during this Semester, and he is looking for participants that match your background in computing knowledge.

The study involves interviews over Zoom, during which you will be asked some general questions about your programming background and you will write a small program using a specific programming environment. After using it, you will be asked some questions related to the environment’s effectiveness according to your impression. There is no assessment of your ability to code during the interview, rather an observation of your programming patterns and how you are using the environment during your programming task. The study is fully compliant with UCD’s ethics considerations and has been approved by the university’s ethics committee.

Participation is totally voluntary, and it will not affect your grades in the module in any way. As a reward, each participant will receive a 25€ One4all voucher after completing the interview. You will also be actively contributing
to research on programming environments and helping future programmers by informing the field of Computing Education research with your input.

Students interested in participating or in learning more about this research should contact the module’s TA, Ioannis Karvelas at email_address. Ioannis will provide you with any further information. Attached you will find a pdf with more details on the study.
**1. KEY INFORMATION ABOUT THE RESEARCHER AND THIS STUDY**

**Researcher:** Ioannis Karvelas, PhD Student  
**Affiliation:** School of Computer Science, University College Dublin  
**Topics of Study:** Computing Education, Programming Behavior, Programming Environments, Human-Computer Interaction  
**Title:** Programming Behavior-Related Patterns while Using Different Compilation Mechanisms

**1.1. What is this research about?**

The current study is part of a broader research project that aims to identify differences in the programming behavior of novices in Java programming when they are exposed to different feedback mechanisms in programming environments. Specifically, this research targets the manner in which the code is compiled (checked for syntax errors) and errors and error messages are shown to users. Research on programming environment features is scarce and identifying beneficial ways to assist novice programmers in their first steps of learning is a step towards developing truly assisting programming environments for future novice programmers. At this stage of the project, we would like the input of people who already have some experience with Java programming in assessing specific compilation and error message presentation mechanisms.

This study involves interviews with people with a background in computer programming, ideally using the Java programming language. The interviews for each participant will be on Zoom (around 45 minutes to 1 hour) with the study’s researcher and will involve:

- A questionnaire/survey of programming-related questions (no quiz involved)
- A programming task using a specific programming environment that will be recorded over Zoom.
- Questions related to the impression made by using the environment

The data generated by the interviews will be pseudonymized and will be used for quantitative and qualitative analysis. You can find more detailed information about the study in the following sections.

**2. PURPOSE OF THIS STUDY**

**2.1. Why am I doing this research?**

Programming environments and their effectiveness in terms of assisting programmers in their tasks are highly under-studied in research. At the same time, computer programming has been identified as a highly demanding task in terms of cognitive load. Beginners in programming have to deal with multiple aspects of learning how to code, such as learning the fundamentals of programming and working on actual coding tasks using a programming language and an editor. The current research focuses on establishing the effectiveness of different compilation and feedback mechanisms within programming environments by identifying their strengths and weaknesses in assisting novice programmers. This way, future programming environment
designers will be able to make informed decisions about their products leading to making programming more accessible to beginners.

3. WHO CAN PARTICIPATE IN THIS STUDY

3.1. Why have you been invited to take part?
We want to get the opinions of people with a computing background, and particularly those who have been exposed to the Java programming language. The demographic profile of students in COMP47500 (Advanced Data Structures in Java) matches the study criteria, so your participation will be highly valuable to our research. Participants of all ages and genders are welcome and there is no limit on the number of participants that we require.

4. INFORMATION ABOUT STUDY PARTICIPATION & PRIVACY

4.1. How will your data be used?
During the study you will be redirected by the interviewer to a Google Forms survey in which you will be answering questions related to your programming background and with respect to your opinion on a specific programming environment. Additionally, you will be asked to complete a small programming task, while your interviewer will be recording your programming session (only the programming window - neither audio nor a profile picture will be recorded) and possibly asking questions regarding your interaction with the environment. The data retrieved from the programming session (Google Forms replies and video recordings on Zoom) will be stored in an encrypted laptop provided by UCD in a pseudonymized manner. The data will be used for quantitative and qualitative analysis and after a period of evaluation, will be archived and eventually deleted.

4.2. What will happen if you decide to take part in this research study?
Once you read this information sheet and accept to take part in the study, you can contact the researcher responsible for this study in order to book an appointment on Zoom. You will be asked to download a specific programming environment prior to the interview, but if you have trouble installing it, this can happen in the beginning of the Zoom interview with the researcher’s assistance. During the interview, the interviewer will provide you with a link to access a survey in Google Forms in which you will be asked to answer some questions related to your exposure to programming. At some point, the interviewer will start recording the Zoom session and you will be asked to share the programming environment’s window. You will be asked to perform a programming task using this environment, while your interviewer will possibly intervene by asking you to justify your interaction with the programming environment.

4.3. How will your privacy be protected?
You will not be asked to provide any personal information at any point during the interview. We will not require you to provide your name, email address or any sort of information that could directly or indirectly lead to your identification during the interview. You will be asked to create a unique alphanumeric word that you only will know. The data provided via Google Forms along with the screen recording during your programming session will be pseudonymized using this keyword and downloaded in encrypted devices for a specific amount of time required for analysis and evaluation, then archived for a period of time, and eventually deleted permanently.
5. INFORMATION ABOUT STUDY RISKS AND BENEFITS

5.1. What are the benefits of taking part in this research study?
You will be providing valuable information that the researchers responsible for this study will use to improve the field of designing effective software for programmers. In addition, you will be given a 25€ One4all voucher as a token of appreciation for participating in the study. Participation does not provide any benefits regarding your grade in COMP47500.

5.2. What are the risks of taking part in this research study?
There are no risks involved in taking part in this study. This study will involve answering questions related to your experience with programming and a small programming task during which you may be asked to justify some of your programming related actions. There is no assessment of any kind involved and any data collected from this interview will be pseudonymized. There will be no assessment of your programming skills during the interview and your participation will be completely disjointed from your grades in COMP47500. Your grade will not be positively or negatively affected from respectively participating or refraining from taking part in the study.

6. ENDING THE STUDY

6.1. Can you change your mind at any stage and withdraw from the study?
If you decide at any point during the interview that you wish to withdraw from the study, you can notify your interviewer. Any recordings of the Zoom session will be ceased immediately and any submitted answers to Google Forms will be deleted. If you decide that you wish to withdraw from the study after the interview takes place, you can contact the researcher indicated at the end of this document via email. The researcher will ask you to provide the alphanumeric key that you created at the start of the interview and proceed to delete any information associated with that key.

6.2. How will you find out what happens with this project?
Should you wish to learn about the outcomes of this study, you should contact the researcher indicated at the end of this document.

CONTACT DETAILS

Ioannis Karvelas (he/him)
Ph.D. Student (IRC Scholar)
A0.12, School of Computer Science
University College Dublin
Belfield, Dublin 4, Ireland
Email: ioannis.karvelas@ucdconnect.ie
Website: https://ucdcs-research.ucd.ie/phd-student/ioannis-karvelas/
Academic Supervisor: Dr. Brett Becker
CONSENT DECLARATION

I have read this information sheet and have had time to consider whether to take part in this study. I understand that my participation is voluntary (it is my choice) and that I am free to withdraw from the research at any time without disadvantage. I agree to take part in this research.

I understand that, as part of this research project I will be asked to take part in a Zoom interview with the study’s researcher that will involve answering programming related questions on Google Forms and performing a programming task while being observed by the researcher.

I understand that my name will not be identified in any way and that the data generated by me will be appropriately pseudonymized during the interview.

I am voluntarily agreeing to have my data collected through Google Forms and stored in encrypted laptop devices for the duration of the study.

I am voluntarily agreeing to have my screen recorded on Zoom during the programming task set to me by the researcher at the time of the interview and later stored in encrypted laptop devices for the duration of the study.

I agree that the data can be used in the publication of higher degrees and scientific publications.

___________ I consent

___________ I do not consent

Date:
B.3 Programming Task

The Book and Author Exercise

You are asked to write a Java program for a Bookstore management system (called Bookstore.java). You should create a Java file that contains the Bookstore, Author, and Book classes (explained below). The Bookstore class should contain the main() method that tests the implementation. The steps of the required implementation are described below.

(1) Write the Author class that contains the following:

- two private instance variables: name (String), email (String)
- a constructor that initializes the instance variables with the given values like the following: public Author(String name, String email) {...}
- public accessor methods: getName(), getEmail()
- a toString() method that returns “author-name at email” (e.g. Alan Turing at a.turing@princeton.co.uk)

(2) Write the Book class that contains the following:

- four private instance variables: name (String), author (Author), price (double), qtyInStock (int)
- a constructor that initializes the instance variables with the given values like the following: public Book(String name, Author author, double price, int qtyInStock) {...}
- public accessor methods: getName(), getAuthor, getPrice(), getQtyInStock()
- a toString() that returns “book-name by author-name at email” (note that the Author’s toString() returns “author-name at email”)

(3) Write a main() that tests the class Book (which uses the Author class). The main method should create an array to store the following Book instances:

<table>
<thead>
<tr>
<th>Book Name</th>
<th>Author Name</th>
<th>Author email</th>
<th>Price</th>
<th>Quantity in Stock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computational Awareness</td>
<td>Michael Drew</td>
<td>m.drew@books</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>Dominating Chaotic Attractors</td>
<td>Rosa Matthews</td>
<td>r.matthews@books</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>Arming the Integers</td>
<td>Hugh Rose</td>
<td>h.rose@books</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>Depending on Infinity</td>
<td>Cathy Londe</td>
<td>c.londe@books</td>
<td>20</td>
<td>5</td>
</tr>
</tbody>
</table>

For your convenience, I include the necessary data input stored in Author and Book instances:

Author a1 = new Author("Michael Drew", "m.drew@books");
Author a2 = new Author("Rosa Matthews", "r.matthews@books");
Author a3 = new Author("Hugh Rose", "h.rose@books");
Author a4 = new Author("Cathy Londe", "c.londe@books");

Book b1 = new Book("Computational Awareness", a1, 15, 10);
Book b2 = new Book("Dominating Chaotic Attractors", a2, 15, 10);
Book b3 = new Book("Arming the Integers", a3, 20, 5);
Book b4 = new Book("Depending on Infinity", a4, 20, 5);

Note that you first have to create the author instances first. Sort the elements of the array based on the authors' last name (in either ascending or descending order). Print the array elements before and after the sorting. A UML diagram of the two classes can be seen below:
B.4 Online Survey and Interview Questionnaires

B.4.1 BlueJ 3 Survey

Participant Information Sheet

Researcher: Ioannis Karvelis – Ph.D. Student
Affiliation: School of Computer Science, University College Dublin, Ireland
Topics of Study: Computing Education, Programming Behavior, Programming Environments, Human-Computer Interaction
Title: Novice Programming Behavior and Programming Environments

1. What is this research about?
The current study is part of a broader research project that aims to identify differences in programming behavior of beginners in Java programming when they are exposed to different feedback mechanisms in programming environments.

2. Why am I doing this research?
Programming environments and their effectiveness in terms of assisting programmers in their tasks are highly under-studied in research. By identifying strengths and weaknesses in different programming mechanisms, future programming environment designers will be able to make informed decisions about their products and programming will become more accessible.

3. Why have you been invited to take part?
We want to get the opinions of people with a computing background, whether that is higher level education or professionals in computing related positions. Based on Prolific, your demographic profile matches our criteria, and we would really appreciate your contribution to this research.

4. How will your data be used?
The data generated by this Google Form will be used for analysis by the researchers involved in the present study. Your data will be completely anonymous and will be stored in encrypted storage devices provided by the researchers’ institution. After the completion of the study, the data will be archived for a period and will eventually be deleted.

5. What will happen if you decide to take part in this research study?
Once you read this information sheet and accept to take part in the study, Google Forms will proceed to the actual survey. During the survey you will be shown some pictures and animations of how a specific programming editor operates and you will be asked a few simple questions. After you complete the survey, you will be provided with a specific key that you can use in Prolific in order to get paid for your participation.

6. How will your privacy be protected?
You will not be asked to provide any sensitive information regarding your person at any point during the survey. We will not require you to provide your name, email address or any sort of information that could directly or indirectly lead to your identification. The data generated by this Google Form will be downloaded in encrypted devices for a specific amount of time required for analysis and evaluation, then archived for a period, and
eventually deleted permanently.

7. What are the benefits of taking part in this research study?
Apart from the monetary reward that Prolific provides you for taking part in this survey, you will be providing valuable information that the researchers responsible for this study will use to improve the field of designing effective software for programmers. Think of it as a good deed for future programmers!

8. What are the risks of taking part in this research study?
There are no risks involved in taking part in this study. This study will involve answering questions related to your experience with programming and a few simple questions regarding your opinion on how a specific programming environment operates. Any data collected from this survey will be completely anonymous.

9. Can you change your mind at any stage and withdraw from the study?
If you decide at any point during the survey that you wish to withdraw from the study, you can simply exit the browser tab. If you decide that you wish to withdraw from the study after completing the survey, you can contact the researcher indicated at the end of this document via email. The researcher will proceed in deleting any information associated with you.

10. How will you find out what happens with this project?
Should you wish to learn about the outcomes of this study, you should contact the researcher indicated at the end of this document. However, research sometimes takes a long time to get published, so any findings derived from the current study could be seen in different scientific venues many months after the study takes place.

11. Contact details for further information
Ioannis Karvelas (he/him)
Email: ioannis.karvelas@ucdconnect.ie

*Required

1. Declaration *

Tick all that apply.

☐ I have read this information sheet and have had time to consider whether to take part in this study. I understand that my participation is voluntary (it is my choice) and that I am free to withdraw from the research at any time without disadvantage. I agree to take part in this research.
Survey Description

In this survey, you will be shown how a specific programming editor operates in terms of compilation (the manner of checking the syntax of the code for errors) and error message presentation (the manner of indicating the errors in the code and displaying the relevant compiler error messages). This editor is designed for beginners in Java programming and the aim of this study is to investigate its effectiveness in terms of assisting students, in order to inform future designers on how to develop better editors for novice programmers.

General Questions

2. What is your Prolific ID? *

3. Do you currently hold an undergraduate (bachelor’s) degree? *
   
   Mark only one oval.
   
   Yes [ ] No [ ]

4. If you currently hold an undergraduate (bachelor's) degree, is it Computer Science related (e.g. Informatics / Software Engineering / Computing etc)?

   Mark only one oval.
   
   Yes [ ] No [ ]

5. Q1. What is your programming level? *

   Mark only one oval.
   
   1  2  3  4  5  6  7
   
   No experience [ ] [ ] [ ] [ ] [ ] [ ] [ ] Very experienced [ ] [ ] [ ] [ ] [ ] [ ] [ ]
6. Q2. What is your level of familiarity with Java? *

Mark only one oval.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>No experience</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very experienced</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7. Q3. What is your preferred programming language? *


8. Q4. What is your preferred programming editor (please provide version number if available)? *


9. How often do your compilations fail? *

Tick all that apply.

- [ ] (Always) - In every compilation
- [ ] (Very often) - In every other compilation
- [ ] (Often) - Once in every 3 compilations
- [ ] (Not often) - Once in every 4 compilations and less
- [ ] (Never) - All my compilations are correct
- [ ] I don't know
10. Q6. How helpful do you think compiler error messages are in terms of assisting you while programming?
   If you have no opinion, then please proceed to the next question.

   *Mark only one oval.*

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
   Not helpful at all       Very helpful

11. Q7. How frustrating do you find compiler error messages while programming?

   *Mark only one oval.*

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
   Not frustrating at all    Very frustrating

12. Q8. When presented with a list of multiple compiler error messages, which error message(s) do you address?

   *Tick all that apply.*

   - a. All error messages
   - b. The first (top) error message only
   - c. No error message at all
   - d. I have never seen more than one error message at a time
   - e. Other. Please specify in the Question below.

13. Q8.1. If you selected "Other" in the previous question, please specify your answer below:

   [Blank space]

   Section 2: Compilation
The animation below shows how the editor compiles the code. By clicking the Compile button on the top left corner of the editor, a compilation is initiated by the user.

14. Q1. How helpful do you find the way compilation is operated within the editor? *
   
   *Mark only one oval.*

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very unhelpful</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very helpful</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

15. Q2. How often do you think you would hit the Compile button? *

   *Mark only one oval.*

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very often</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
16. Q3. Do you have any suggestions about the way compilation should work instead? *

Section 3: Error Indicators

The picture below shows how errors are indicated within the editor. The line containing the first error in the code is highlighted in yellow and the location of only the first error is highlighted in red.

17. Q1. How helpful do you think the way errors are indicated on the code is? *

Mark only one oval.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Very helpful</td>
</tr>
</tbody>
</table>
18. Q2. In case you did not highly rate the previous question, could you pin down what it was that you didn’t like?

19. Q3. Are there any improvements/suggestions you think would make the error indication more effective?

Section 4: Error Message Presentation

The following animation displays how error messages are shown in the editor. After you click the Compile button on the top left of the editor, if errors are present, the message of only the first error appears at the bottom of the window in a dedicated output area.
20. Q1. How helpful do you find the way errors are presented in the editor? *
   
   Mark only one oval.
   
   1  2  3  4  5  6  7
   
   Very unhelpful ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very helpful

21. Q2. Which error messages would you like the editor to show: *

   Tick all that apply.
   
   ☐ a. All the error messages, even if there are multiple errors
   ☐ b. Only the first error message (as shown)
   ☐ c. No error messages, even if the syntax is incorrect. The indicators are enough without the text of the messages.

22. Q3. Where would you prefer the error messages to be shown on an editor? *

   Mark only one oval.
   
   ☐ a. On the code as a popup
   ☐ b. Under the editor window in a dedicated error/output area
   ☐ c. Elsewhere (please specify in the Question below)

23. Q3.1. If you answered "Elsewhere" in the previous question, please specify your answer.

   ______________________________________________________________
   ______________________________________________________________
   ______________________________________________________________
24. Q4. How would you prefer the messages be shown to you? *
   
   Mark only one oval.
   
   a. On-demand (you must perform a certain action for the messages to appear)
   b. Automatically (in case any errors exist)
   c. Other. Please specify in the Question below.

25. Q4.1. If you answered “Other” in the previous question, please specify your answer.

   
   

26. Q5. How usable do you think a programming editor that shows neither error indication nor error messages would be to you? *

   Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
</table>
   Very unusable | | | | | | | Very usable

27. Q6. How frustrating would you find a programming editor that shows neither error indication nor error messages? *

   Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
</table>
   Not frustrating at all | | | | | | | Very frustrating
B.4.2 BlueJ 4 Survey

Participant Information Sheet

Researcher: Ioannis Karvelas – Ph.D. Student
Affiliation: School of Computer Science, University College Dublin, Ireland
Topics of Study: Computing Education, Programming Behavior, Programming Environments, Human-Computer Interaction
Title: Novice Programming Behavior and Programming Environments

1. What is this research about?
The current study is part of a broader research project that aims to identify differences in programming behavior of beginners in Java programming when they are exposed to different feedback mechanisms in programming environments.

2. Why am I doing this research?
Programming environments and their effectiveness in terms of assisting programmers in their tasks are highly under-studied in research. By identifying strengths and weaknesses in different programming mechanisms, future programming environment designers will be able to make informed decisions about their products and programming will become more accessible.

3. Why have you been invited to take part?
We want to get the opinions of people with a computing background, whether that is higher level education or professionals in computing related positions. Based on Prolific, your demographic profile matches our criteria, and we would really appreciate your contribution to this research.

4. How will your data be used?
The data generated by this Google Form will be used for analysis by the researchers involved in the present study. Your data will be completely anonymous and will be stored in encrypted storage devices provided by the researchers’ institution. After the completion of the study, the data will be archived for a period and will eventually be deleted.

5. What will happen if you decide to take part in this research study?
Once you read this information sheet and accept to take part in the study, Google Forms will proceed to the actual survey. During the survey you will be shown some pictures and animations of how a specific programming editor operates and you will be asked a few simple questions. After you complete the survey, you will be provided with a specific key that you can use in Prolific in order to get paid for your participation.

6. How will your privacy be protected?
You will not be asked to provide any sensitive information regarding your person at any point during the survey. We will not require you to provide your name, email address or any sort of information that could directly or indirectly lead to your identification. The data generated by this Google Form will be downloaded in encrypted devices for a specific amount of time required for analysis and evaluation, then archived for a period, and
eventually deleted permanently.

7. What are the benefits of taking part in this research study?
Apart from the monetary reward that Prolific provides you for taking part in this survey, you
will be providing valuable information that the researchers responsible for this study will
use to improve the field of designing effective software for programmers. Think of it as a
good deed for future programmers!

8. What are the risks of taking part in this research study?
There are no risks involved in taking part in this study. This study will involve answering
questions related to your experience with programming and a few simple questions
regarding your opinion on how a specific programming environment operates. Any data
collected from this survey will be completely anonymous.

9. Can you change your mind at any stage and withdraw from the study?
If you decide at any point during the survey that you wish to withdraw from the study, you
can simply exit the browser tab. If you decide that you wish to withdraw from the study
after completing the survey, you can contact the researcher indicated at the end of this
document via email. The researcher will proceed in deleting any information associated
with you.

10. How will you find out what happens with this project?
Should you wish to learn about the outcomes of this study, you should contact the
researcher indicated at the end of this document. However, research sometimes takes a
long time to get published, so any findings derived from the current study could be seen in
different scientific venues many months after the study takes place.

11. Contact details for further information
Ioannis Karvelas (he/him)
Email: ioannis.karvelas@ucdconnect.ie

*Required

1. Declaration *

   Tick all that apply.

   ☐ I have read this information sheet and have had time to consider whether to take part
in this study. I understand that my participation is voluntary (it is my choice) and that I am
free to withdraw from the research at any time without disadvantage. I agree to take part in
this research.
In this survey, you will be shown how a specific programming editor operates in terms of compilation (the manner of checking the syntax of the code for errors) and error message presentation (the manner of indicating the errors in the code and displaying the relevant compiler error messages). This editor is designed for beginners in Java programming and the aim of this study is to investigate its effectiveness in terms of assisting students, in order to inform future designers on how to develop better editors for novice programmers.

General Questions

2. What is your Prolific ID? *

3. Do you currently hold an undergraduate (bachelor’s) degree? *
   Mark only one oval.
   ☐ Yes
   ☐ No

4. If you currently hold an undergraduate (bachelor’s) degree, is it Computer Science related (e.g. Informatics / Software Engineering / Computing etc)?
   Mark only one oval.
   ☐ Yes
   ☐ No

5. Q1. What is your programming level? *
   Mark only one oval.
   
<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>No experience</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Very experienced</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>
6. Q2. What is your level of familiarity with Java? *

Mark only one oval.

1  2  3  4  5  6  7

No experience  ○ ○ ○ ○ ○ ○ ○ Very experienced

7. Q3. What is your preferred programming language? *

________________________________________

8. Q4. What is your preferred programming editor (please provide version number if available)? *

________________________________________

9. How often do your compilations fail? *

Tick all that apply.

☐ (Always) - In every compilation
☐ (Very often) - In every other compilation
☐ (Often) - Once in every 3 compilations
☐ (Not often) - Once in every 4 compilations and less
☐ (Never) - All my compilations are correct
☐ I don't know
10. Q6. How helpful do you think compiler error messages are in terms of assisting you while programming?  
If you have no opinion, then please proceed to the next question.

* Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not helpful at all</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Very helpful</td>
</tr>
</tbody>
</table>

11. Q7. How frustrating do you find compiler error messages while programming?  

* Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not frustrating at all</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Very frustrating</td>
</tr>
</tbody>
</table>

12. Q8. When presented with a list of multiple compiler error messages, which error message(s) do you address?

Tick all that apply.

a. All error messages
b. The first (top) error message only
c. No error message at all
d. I have never seen more than one error message at a time
e. Other. Please specify in the Question below.

13. Q8.1. If you selected "Other" in the previous question, please specify your answer below:

_________________________________________________________________________________________

Section 2: Compilation
The animation below shows how the editor compiles the code. While writing, when the user changes lines (e.g. by hitting Enter) the editor performs a syntax check on already written code. Additionally, users can click the compile button on the top left corner to perform a manual check at any time.

14. Q1. How helpful do you find the way compilation is operated within the editor? *

Mark only one oval.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Very unhelpful | Very helpful
15. Q2. Compilation is performed every time you change a line while typing. So, every time there is an automatic compilation (i.e. every time you change a line), if syntax errors are present, red squiggly lines are drawn in the editor where the error is detected. How helpful do you think this is while you are writing code?

*Mark only one oval.*

1 2 3 4 5 6 7

Very unhelpful ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very helpful

16. Q3. Compilation can also be performed if you click the Compile button on the top left corner of the window. How helpful do you think having this feature is?

*Mark only one oval.*

1 2 3 4 5 6 7

Very unhelpful ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very helpful

17. Q4. How often do you think you would hit the Compile button, even with automatic error checking present?

*Mark only one oval.*

1 2 3 4 5 6 7

Never ☐ ☐ ☐ ☐ ☐ ☐ ☐ Very often
18. Q4.1. If you replied often (>4) in the previous question, could you explain why you would still use the Compile button even though the editor will compile for you automatically?

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

19. Q5. Do you have any suggestions about the way compilation should work instead? *

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

Section 3: Error Indicators
The picture below shows how errors are indicated within the editor. A red squiggly line is shown under the offending code in each error location and the sidebar also becomes red for each line containing an error. At the bottom right the total number of errors is also shown.

20. Q1. How helpful do you think the way errors are indicated on the code is? *

   Mark only one oval.

   |   |   |   |   |   |   |   |
   | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
   | Very unhelpful | | | | | | Very helpful |

21. Q2. If errors are present, the red indicators are shown whenever you change a line while writing code. How helpful do you think this is?

   Mark only one oval.

   |   |   |   |   |   |   |   |
   | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
   | Very unhelpful | | | | | | Very helpful |
22. Q2.1. In case you did not highly rate the previous question (<4), could you pin down what it was that you didn't like?


23. Q3. Are there any improvements/suggestions you think would make the error indication more effective?


Section 4: Error Message Presentation
The following animation displays how error messages are shown in the editor. To see an error message, you have to move the cursor to a specific underlined area using the mouse or the keyboard. A black pop-up window containing the error message will appear. Alternatively, you can click the compile button multiple times, in which case the errors start appearing from first to last and cycle over again.

24. Q1. How helpful do you find the way error messages are presented in the editor? *

Mark only one oval:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Very unhelpful

25. Q2. Which error messages would you like the editor to show: *

Tick all that apply.

- a. All the error messages, even if there are multiple errors
- b. Only the first error message (in the case of multiple errors)
- c. No error messages, even if the syntax is incorrect. The indicators are enough without the text of the messages.
26. Q3. Where would you prefer the error messages to be shown on an editor? *

Mark only one oval.

☐ a. On the code as a popup
☐ b. Under the editor window in a dedicated error/output area
☐ c. Elsewhere (please specify in the Question below)

27. Q3.1. If you answered "Elsewhere" in the previous question, please specify your answer.

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

28. Q4. How would you prefer the messages be shown to you? *

Mark only one oval.

☐ a. On-demand (you must perform a certain action for the messages to appear)
☐ b. Automatically (in case any errors exist)
☐ c. Other. Please specify in the Question below.

29. Q4.1. If you answered "Other" in the previous question, please specify your answer.

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
30. Q5. How usable do you think a programming editor that shows neither error indication nor error messages would be to you?

* Mark only one oval.

1  2  3  4  5  6  7

Very unusable  □ □ □ □ □ □ □ Very usable

31. Q6. How frustrating would you find a programming editor that shows neither error indication nor error messages?

* Mark only one oval.

1  2  3  4  5  6  7

Not frustrating at all  □ □ □ □ □ □ □ Very frustrating

This content is neither created nor endorsed by Google.

Google Forms


search (ICER ’14). ACM, Glasgow, Scotland, United Kingdom, 43–50. DOI: 10.1145/2632320.2632343.


[107] Henry B. Mann and Donald R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. The Annals of Mathematical Statistics, 18, 1, 50–60. DOI: 10.1214/aoms/1177730491.


