

Two Roads Diverge: Mapping the Path of Learning for Novice Programmers Through Large Scale Interaction Data and Neural Network Classifiers

by

Natalie Culligan BSc Comp. Sci. (Hons.)



**Maynooth
University**
National University
of Ireland Maynooth

**Dissertation submitted in partial fulfilment requirements for candidate
for the degree of**

Doctor of Philosophy

Department of Computer Science

Maynooth University, Maynooth, Ireland

Supervisor: Dr. Kevin Casey

Head of Department: Dr. Joseph Timoney

February 2021

Table of Contents

List of Figures	v
List of Tables	v
Acronyms Used Throughout Thesis	viii
Acknowledgments.....	ix
Abstract.....	xiii
1. Introduction.....	1
1.1 Problem Statement.....	1
1.2 Research Questions.....	2
1.3 Contributions.....	2
1.4 Publications.....	3
1.5 Chapter Overview	4
1.6 Chapter Conclusion.....	6
2. Related Research.....	7
2.1 MULE as a Pedagogical Tool.....	7
2.1.1 VPL.....	8
2.1.2 Examples of Modifications to VPL.....	13
2.1.3 Alternatives to VPL	13
2.1.4 Pedagogical Coding Environments Conclusions	15
2.2 MULE as a Research Tool.....	16
3. MULE and the Collection of Data	22
3.1 Motivation.....	22
3.2 Paper: Building an Authentic Novice Programming Lab Environment	22
3.3 Overview of MULE	33
3.4 Features of MULE	34
3.4.1 Workbook	34
3.4.2 Analytics	39
3.5 Use of MULE in Introduction to Programming Modules.....	40
3.6 GDPR and Ethical Collection of Data	40
3.7 Chapter Summary	41
4. Overview of Experiments and Data.....	42
4.1 Description of Data Sets	42
4.1.1 Data Set 1.....	42
4.1.2 Data Set 2.....	43

4.2 Description of Data Types	47
4.2.1 Mouse Movements	47
4.2.2 Compile, Run, Evaluate Actions	47
4.2.3 Complexity of Code Submitted.....	47
4.3 Methods of Analysis	47
4.3.1 Wilcox Rank Sum Test	48
4.3.2 Linear Regression	48
4.3.3 Neural Network Binary Classification.....	48
4.4 The HOG Classifier	49
4.5 Overview of Experiments	51
4.5.1 Experiment 1: Mouse Movements (MM)	52
4.5.2 Experiment 2: Compile, Run, Evaluate Movements (CRE)	52
4.5.3 Experiment 3: Code Complexity (COMPLEX).....	52
4.5.4 Final Experiment: HOG	52
4.6 Chapter Summary	52
5. Experiment 1: Mouse Movements	53
5.1 Introduction to Mouse Movement Experiment.....	53
5.2 Paper: What the Mouse Said: How Mouse Movements Can Relate to Student Stress and Success.....	54
5.3 Mouse Movement Features.....	66
5.4 Mouse Movements Analysis and Neural Network	67
5.4.1 Wilcox Rank Sum Test	68
5.4.2 Linear Regression	71
5.4.3 Neural Networks	71
5.5 Mouse Movements Week-by-Week.....	72
5.6 Mouse Movements Conclusions	75
6. Experiment 2: Compile, Run, and Evaluate (CRE) Movements	76
6.1 Introduction to CRE Experiment	76
6.2 Paper: Exploring the Coding Behaviour of Successful Students in Programming by Employing Neo-Piagetian Theory.....	76
6.3 CRE Features	88
6.4 CRE Movements Analysis.....	89
6.4.1 Wilcox Rank Sum Test	89
6.4.2 Linear Regression	91
6.5 Neural Network Classifiers.....	94
6.5.1 Comparison of CA Classifiers	94

6.5.2 Comparison of Written Exam Classifiers	95
6.6 CRE Week-by-Week	95
6.7 CRE Conclusions	99
7. Experiment 3: Complexity of Student Code	100
7.1 Introduction to Code Complexity	100
7.2 Features	101
7.3 Code Complexity Analysis	102
7.3.1 Wilcoxon Rank Sum Test: File Size	103
7.3.2 Wilcoxon Rank Test: Nodes	105
7.3.3 Linear Regression: File Size	107
7.3.4 Linear Regression Test: Node Data	109
7.4 Neural Network Classifiers	111
7.4.1 Comparison of CA Classifiers	111
7.4.2 Comparison of Written Exam Classifiers	113
7.5 Code Complexity Week-by-Week	114
7.6 Discussion of Results	118
8. Experiment 4: The HOG Classifier	120
8.1 Introduction to HOG	120
8.2 Features	120
8.3 Neural Network Classifiers	120
8.3.1 Comparison of Continuous Assessment Classifiers	121
8.3.2 Comparison of Written Exam Classifiers	122
8.5 Conclusions	123
9. Conclusions	125
9.1 The Research Instruments	125
9.1.1 MULE	125
9.1.2 HOG	125
9.2 Discussion of Student Behaviour	130
9.2.1 Week 1	131
9.2.2 Week 2	131
9.2.3 Week 3	132
9.2.4 Week 4	132
9.2.5 Week 5	133
9.2.6 Week 6	134
9.2.7 Week 7	135
9.2.8 Week 8	136

9.2.9 Week 9	137
9.2.10 Week 10	137
9.3 The Research Questions.....	138
9.4 Future Work	140
9.5 Conclusion	141
10. Appendix.....	143
10.1 Consent Form.....	143
10.2 Information Sheet.....	144
10.3 Wilcox Rank Sum Test Results MM	146
10.3.1 CA.....	146
10.3.2 Written Exam	149
10.4 Linear Regression Results MM.....	152
10.4.1 CA.....	152
10.4.2 Written Exam	155
10.5 Full Classifier Results MM.....	158
10.5.1 CA	158
10.5.2 Written Exam	159
10.6 Full Classifier Results CRE Movements	161
10.6.1 CA.....	161
10.6.2 Written Exam	163
10.7 Full Classifier Results COMPLEX.....	165
10.7.1 CA.....	165
10.7.2 Written Exam	169
10.8 Full Classifier Results HOG	172
10.8.1 CA.....	172
10.8.2 Written Exam	176
10.9 Sample VPL and MULE Scripts.....	179
10.9.1 vpl_run.sh.....	179
10.9.2 vpl_evaluate.sh.....	179
10.9.3: vpl_compile.sh.....	183
10.9.4: metadata.json	183
10.9.5: description.html	183
Bibliography.....	185

List of Figures

Figure 2-1: VPL Student View	8
Figure 2-2: Example of the Execution Files in VPL.....	9
Figure 2-3: Diagram of VPL Components.....	10
Figure 3-1: MULE Student View Layout	33
Figure 3-2: MULE Student Lab Selection Menu.....	35
Figure 3-3: Assignment Metadata File in MULE	35
Figure 3-4: Description Panel in Workbook with CA Grade and Personal Best	36
Figure 3-5: MULE Grade Storage	37
Figure 3-6: Workbook Admin View.....	38
Figure 3-7: MULE Data Collection	39
Figure 4-1: HOG Classifier Workflow	50
Figure 6-1: CRE Patterns	89
Figure 7-1: Selection.java Sample Code.....	101
Figure 7-2: Selection.java with Comments Removed.....	101
Figure 7-3: Parse Tree Generated from Selection.java	102
Figure 10-1: Sample vpl_run.sh.....	179
Figure 10-2: Sample vpl_evaluate.sh.....	182
Figure 10-3: Sample vpl_compile.sh	183
Figure 10-4: Sample metadata.json.....	183
Figure 10-5: Sample description.html.....	183
Figure 10-6: Sample description.html in MULE	184
Figure 10-7: Workbook display of description.html.....	184

List of Tables

Table 1-1: Contributions to Published Paper 1	3
Table 1-2: Contributions to Published Paper 2	4
Table 1-3: Contributions to Published Paper 3	4
Table 5-1: MM Wilcox Rank Sum Test CA for Week 1 to 4.....	68
Table 5-2: MM Wilcox Rank Sum Test CA for Week 5 to 7	69
Table 5-3: MM Wilcox Rank Sum Test CA for Week 8 to 10.....	69
Table 5-4: MM Wilcox Rank Sum Test Written Exam for Week 1 to 3	70
Table 5-5: MM Wilcox Rank Sum Test Written Exam for Week 4 to 6.....	70
Table 5-6: MM Wilcox Rank Sum Test Written Exam for Week 7 to 10	71
Table 6-1: CRE Features.....	88
Table 6-2: CRE Wilcox Rank Sum Test CA for Weeks 1 to 5.....	90
Table 6-3: CRE Wilcox Rank Sum Test CA for Weeks 6 to 10.....	90
Table 6-4: CRE Wilcox Rank Sum Test Written Exam for Weeks 1 to 5.....	91
Table 6-5: CRE Wilcox Rank Sum Test Written Exam for Weeks 6 to 10.....	91
Table 6-6: CRE Linear Regression CA for Weeks 1 to 5	92
Table 6-7: CRE Linear Regression CA for Weeks 6 to 10.....	92
Table 6-8: CRE Linear Regression Written Exam for Weeks 1 to 5	93
Table 6-9: CRE Linear Regression Written Exam for Weeks 6 to 10	93
Table 6-10: CRE CA Classifier Accuracy	94

Table 6-11: CRE CA Classifier False Passes.....	94
Table 6-12: CRE Written Exam Classifier Accuracy	95
Table 6-13: CRE Written Exam Classifier False Passes.....	95
Table 7-1: COMPLEX File Size Wilcox Rank Sum Test CA for Weeks 1 to 5 ..	103
Table 7-2: COMPLEX File Size Wilcox Rank Sum Test CA for Weeks 6 to 10	103
Table 7-3: COMPLEX File Size Wilcox Rank Sum Test Written Exam for Weeks 1 to 5	104
Table 7-4: COMPLEX File Size Wilcox Rank Sum Test Written Exam for Weeks 6 to 10	104
Table 7-5: COMPLEX Nodes Wilcox Rank Sum Test CA for Weeks 1 to 5	105
Table 7-6: COMPLEX Nodes Wilcox Rank Sum Test CA for Weeks 6 to 10	106
Table 7-7: COMPLEX Nodes Wilcox Rank Sum Test Written Exam for Weeks 1 to 5	106
Table 7-8: COMPLEX Nodes Wilcox Rank Sum Test Written Exam for Weeks 6 to 10	106
Table 7-9: COMPLEX File Size Linear Regression CA for Weeks 1 to 5.....	108
Table 7-10: COMPLEX File Size Linear Regression CA for Weeks 6 to 10.....	108
Table 7-11: COMPLEX File Size Linear Regression Written Exam for Weeks 1 to 5	109
Table 7-12: COMPLEX File Size Linear Regression Written Exam for Weeks 6 to 10	109
Table 7-13: Nodes Linear Regression CA	110
Table 7-14: Nodes Linear Regression CA	110
Table 7-15: Nodes Linear Regression Written Exam	111
Table 7-16: Nodes Linear Regression Written Exam	111
Table 7-17: COMPLEX CA Classifier Accuracy.....	112
Table 7-18: COMPLEX CA Classifier False Passes	112
Table 7-19: Compare Early Semester CA Classifiers.....	112
Table 7-20: Compare Late Semester CA Classifiers	112
Table 7-21: COMPLEX Written Exam Classifier Accuracy.....	113
Table 7-22: COMPLEX Written Exam Classifier False Passes	113
Table 7-23: Comparing Early Semester Written Exam Classifiers	113
Table 7-24: Compare Late Semester Written Exam Classifiers	114
Table 8-1: HOG CA Classifier Accuracy	121
Table 8-2: HOG CA Classifier False Passes.....	121
Table 8-3: Averages of Early Semester Classifiers for CA	121
Table 8-4: Averages of Late Semester Classifiers for CA.....	122
Table 8-5: HOG Written Exam Classifier Accuracy	122
Table 8-6: HOG Written Exam Classifier False Passes.....	122
Table 8-7: Averages of Early Semester Classifiers for Written Exams	123
Table 8-8: Averages of Late Semester Classifiers for Written Exams	123
Table 9-1: Comparing Early Semester CA Classifier Accuracy.....	126
Table 9-2: Comparing Early Semester CA Classifier False Passes	126
Table 9-3: Comparing Early Semester CA Classifier Average Accuracy and False Passes	126
Table 9-4: Comparing Late Semester CA Classifier Accuracy	127
Table 9-5: Comparing Late Semester CA Classifier False Passes.....	127

Table 9-6: Comparing Late Semester CA Classifier Average Accuracy and False Passes	127
Table 9-7: Comparing Early Semester Written Exam Classifier Accuracy	128
Table 9-8: Comparing Early Semester Written Exam Classifier False Passes	128
Table 9-9: Comparing Early Semester Written Exam Classifier Average Accuracy and False Passes	128
Table 9-10: Comparing Late Semester Written Exam Classifier Accuracy	129
Table 9-11: Comparing Late Semester Written Exam Classifier False Passes	129
Table 9-12: Comparing Late Semester Written Exam Classifier Average Accuracy and False Passes	130
Table 10-1: MM Wilcox Rank Sum Test CA for Weeks 1 to 3	146
Table 10-2: MM Wilcox Rank Sum Test CA for Weeks 4 to 7	147
Table 10-3: MM Wilcox Rank Sum Test CA for Weeks 8 to 10	148
Table 10-4: MM Wilcox Rank Sum Test Written Exam for Weeks 1 to 3	149
Table 10-5: MM Wilcox Rank Sum Test Written Exam for Weeks 1 to 3	150
Table 10-6: MM Wilcox Rank Sum Test Written Exam for Weeks 1 to 3	151
Table 10-7: MM Linear Regression CA for Weeks 1 to 3	152
Table 10-8: MM Linear Regression CA for Weeks 4 to 7	153
Table 10-9: MM Linear Regression CA for Weeks 8 to 10	154
Table 10-10: MM Linear Regression Written Exam for Weeks 1 to 3	155
Table 10-11: MM Linear Regression Written Exam for Weeks 4 to 7	156
Table 10-12: MM Linear Regression Written Exam for Weeks 8 to 10	157
Table 10-13: MM CA Threshold -0.1 Classifier	158
Table 10-14: MM CA Threshold -0.1 Classifier	158
Table 10-15: MM CA Threshold-0.1 Classifier	159
Table 10-16: MM Written Exam Threshold-0.1 Classifier	159
Table 10-17: MM Written Exam Threshold-0.1 Classifier	160
Table 10-18: MM Written Exam Threshold-0.1 Classifier	160
Table 10-19: CRE CA Threshold-0 Classifier	161
Table 10-20: CRE CA Threshold-0 Classifier	161
Table 10-21: CRE CA Threshold-0.1 Classifier	162
Table 10-22: CRE CA Threshold-0.1 Classifier	162
Table 10-23: CRE CA Threshold-0.15 Classifier	163
Table 10-24: CRE Written Exam Threshold-0 Classifier	163
Table 10-25: CRE Written Exam Threshold-0 Classifier	164
Table 10-26: CRE Written Exam Threshold-0.1 Classifier	164
Table 10-27: CRE Written Exam Threshold-0.15 Classifier	165
Table 10-28: COMPLEX CA Threshold-0 Classifier	165
Table 10-29: COMPLEX CA Threshold-0 Classifier	166
Table 10-30: COMPLEX CA Threshold-0 Classifier	166
Table 10-31: COMPLEX CA Threshold-0.1 Classifier	167
Table 10-32: COMPLEX CA Threshold-0.1 Classifier	167
Table 10-33: COMPLEX CA Threshold-0.1 Classifier	168
Table 10-34: COMPLEX CA Threshold-0.15 Classifier	168
Table 10-35: COMPLEX CA Threshold-0.15 Classifier	169
Table 10-36: COMPLEX Written Exam Threshold-0 Classifier	169
Table 10-37: COMPLEX Written Exam Threshold-0 Classifier	170
Table 10-38: COMPLEX Written Exam Threshold-0.1 Classifier	170

Table 10-39: COMPLEX Written Exam Threshold-0.1 Classifier.....	171
Table 10-40: COMPLEX Written Exam Threshold-0.15 Classifier.....	171
Table 10-41: HOG CA Threshold-0 Classifier	172
Table 10-42: HOG CA Threshold-0 Classifier	172
Table 10-43: HOG CA Threshold-0.1 Classifier	173
Table 10-44: HOG CA Threshold-0.1 Classifier	173
Table 10-45: HOG CA Threshold-0.1 Classifier	174
Table 10-46: HOG CA Threshold-0.15 Classifier	174
Table 10-47: HOG CA Threshold-0.15 Classifier	175
Table 10-48: HOG CA Threshold-0.15 Classifier	175
Table 10-49: HOG Written Exam Threshold-0 Classifier	176
Table 10-50: HOG Written Exam Threshold-0 Classifier	176
Table 10-51: HOG Written Exam Threshold-0.1 Classifier	177
Table 10-52: HOG Written Exam Threshold-0.1 Classifier	177
Table 10-53: HOG Written Exam Threshold-0.1 Classifier	178
Table 10-54: HOG Written Exam Threshold-0.15 Classifier	178
Table 10-55: HOG Written Exam Threshold-0.15 Classifier	179

Acronyms Used Throughout Thesis

CA: Continuous Assessment

COMPLEX: Code Complexity

CRE: Compile, Run, Evaluate

CS1: Computer Science 1

LMS: Learning Management System

MM: Mouse Movements

MULE: Maynooth University Learning Environment

VIF: Variance Inflation Factor

VPL: Virtual Programming Lab

Acknowledgments

None of this would have been possible without my fiancé Jon. I am so lucky to have you in my life, an incredible person with so much kindness, intelligence, and humour. Thank you for always having a different perspective and opening my eyes to possibilities and ideas I never would have considered before. Thank you for believing in me when I did not. Thank you for the endless support, encouragement, the proof reading, and the grammar checks. Thank you for listening to me ramble about data and experiments and papers and ideas for future work and so on. Thank you for reminding me to eat and sleep and rest. I can never repay you for all of the love and support throughout this PhD.

Thank you to my closest friends, Brian, Liz and Gillian, and their support throughout the PhD.

Thank you, Liz for always being a source of honesty, common sense, and style in my life. Thank you for the updates on Poppy and Dexter, my wonderful godchildren, it always makes my day. Thank you for always reaching out when I would forget to do anything other than work. I cannot wait to be able to visit you and catch up after the pandemic.

Thank you, Gillian, for being the embodiment of sunshine and making me laugh so much. Thank you for joining in the celebrations of each of the milestones along the way throughout this process, even from across oceans.

Thank you, Brian, for your sense of humour and insight. Thank you for being my friend for so many years, even though we are so different from when we first met. We've been through so many changes as people, and you helped me grow as a person and see perspectives I would never have seen otherwise. Thank you for driving me to many much-needed coffee breaks and Tesco trips, and of course for the wonderful cakes and macarons.

Thank you to my family, the Culligans and the McConvilles. Thank you to my grandad who gave me a love of reading and learning that has stayed with me my entire life. Thank you to my Great-Aunt Nancy, for my name and for always believing in me. Thank you to Uncle Liam – the beautiful Cross pens you gave me are now proudly displayed in my office. Thank you, dad, for cycling to Newry (from Drogheda!) way back in 1989 to collect a Commadore 64, and bringing it all the way back home, and changing my entire life. Thank you for all of the support and help. I don't know what I would have done without you bringing me out for

drives to relax during this process. Thank you, mum, for giving me the gift of determination and unreasonable stubbornness – it has been essential in completing this thesis. Thank you for inspiring me through your own career, and filling my head with notions from an early age, you allowed me to believe that I could achieve anything I wanted. Thank you both for being proud of me. Thank you to my brother Stephen for all your help and advice throughout the years with all things computer related. Thank you to my brother James, for all your wonderful art throughout the years.

Thank you to the McVeighs, for all of your support. Thank you for your love and concern. Thank you, Anne, for always being so kind and welcoming, and for your prayers - they have never let me down. Thank you, Rory, for being so reliable – always ready to help anyone, no matter how much you already have on your plate.

Thank you to my supervisor Kevin for all of your guidance and support, and for the many late nights and early mornings working with me on MULE to get it running in time for the labs the next morning. At the time it was awful, but now it's one of the best memories of my PhD!

Thank you to Susan, for seeing potential in me, and starting me on this journey. Thank you to Aidan, Emlyn and Misha for all your work, support and patience with us on the MULE project. Thank you to Brett Becker for adopting the MULE system in Beijing. Thank you to Kylie for your support and help – you've been an incredible teacher and mentor to both Jon and I, and we will never forget that. Thanks of course to my lab partner through the Covid-19 shutdown – Bunny the cat.

Thank you to all my teachers over the years who inspired me, from Mrs O'Callaghan in St. Oliver's, to Larry in BCFE, to Susan in Maynooth.

Finally, thank you to the wonderful children in my life. Thank you to my incredible godchildren, Poppy and Dexter. Poppy, I hope you continue to love coding and we can teach Dexter together when this pandemic is over. Thank you to Dervla, and Iris, to Ali, Freya and of course to Ollie, my fellow teacher and computer enthusiast. I'm so excited to see where life will bring all of you.

Thank you to everyone who has supported me in this journey. I hope I can continue to rely on it.

Every day I am working to make you all proud of me.



Circa 1993: Already prepared for a life of computers and writing

For Jon, mum, and dad

Abstract

Learning to program is a fundamental part of Computer Science education. To become a proficient programmer, one must become competent at both code comprehension and code production. Research shows that the most effective way to teach programming to students is through practical exercises. However, the increasing numbers of students in Computer Science classes means it is difficult to correct assignments and provide timely feedback. This can result in fewer practical assignments and/or less useful feedback for each student. Automated grading tools, and understanding of how novice programmers learn to code, is essential for these growing numbers of students. The Maynooth University Learning Environment, or MULE, was built to address this challenge. MULE is a cloud-based learning environment built from the ground up with the goal of teaching introductory programming courses in an authentic manner while facilitating the collection of large-scale behavioural data to support Learning Analytics. In this thesis, behavioural interaction data and code written by students in MULE is used to investigate the differences between successful and unsuccessful programming student behaviour, with the use of data analysis and Neural Network classifiers. The result is a method of classification that predicts early on if a student is likely to be in the top or bottom 50% of grades in the class with up to 87% accuracy, and a model of the path of learning for successful students, including key times, assignments, and topics during the introduction to programming module when the higher and lower achieving students diverge in behaviour.

1. Introduction

In this chapter the problem statement and the research questions for this thesis will be outlined. The contributions of this thesis and publications produced as part of this thesis will be presented, and finally a chapter overview will describe the contents of the thesis.

1.1 Problem Statement

It has often been stated that it is difficult to learn to program. It may be more useful to say that it is difficult to teach programming. Computer Science has one of the highest dropout and failure rates in third level education [1, 2, 3, 4, 5]. As third level Computer Science courses become more popular, and class sizes become larger, it becomes harder to provide feedback and support to students. It is difficult to even identify the students who need support when teaching classes that may have hundreds of students.

To address this, this research investigates the behaviours of students as they learn how to code and compares the behaviour of “successful” and “unsuccessful” students, where a student is deemed to be successful if they are in the top 50% of grades in the class and unsuccessful if in the bottom 50% of grades. Data is passively collected from students as they write their first programs, and we investigate how the data reflects the students’ eventual outcome in the module, in terms of grades. Through this research, we hope to find key labs, topics, and assignments that are indicators of a student’s success. From these keys, we can 1) identify students at risk, and 2) advise on curriculum changes that may revisit key topics and assignments that students may need repetition to fully understand.

The way that programming is taught in introductory programming modules means that all lab exercises are based on the lessons from the previous session. So, once a student has a bad day or week, they are at a disadvantage for the next lab, which compounds into a larger disadvantage for the next lab. Computer science may or may not be unique in this, but it is not difficult to imagine that a heavy importance on the first few weeks of a first-year university course could be detrimental to the success of the students, when many, if not most students are adapting to a new way of life, and their first taste of freedom as adults. They may not yet have the maturity to be as diligent in their studies as is necessary, and then find only a few short weeks into the semester, that while they are able to catch up to their other courses, they are entirely lost and confused in their introduction to programming courses.

If this is the problem, that missing out on a fundamental of programming, such as loops or string manipulation, leaves the student unable to keep up with the rest of the material, then the solution is to provide ways that the student can revisit or reaffirm the previous lessons. In our university, there are drop-in help clinics that provide this help, but it is also necessary to provide advice and guidance to these students, when we as educators realise, the students are failing to keep up with the course.

It is the goal of this project to explore the use of easily observed non-intrusive data to find students in need of extra help and to improve problem areas in our curriculum. In the bigger picture, this data can also be used to give us insight into what learning to program looks like, and to improve the way it is taught.

1.2 Research Questions

These are the research questions to be answered in this thesis:

1. *How can we observe student behaviour as they learn to code in a non-intrusive way?*
2. *Are there divergences in the observed student behaviour between the highest and lowest achieving students?*
3. *How early in the semester can students be classified as higher or lower achieving, to allow for interventions?*

1.3 Contributions

The first contribution of this work is the creation of MULE – Maynooth University Learning Environment - a pedagogical coding environment that collects data on student behaviour as they complete coding assignments [6].

The second contribution is insight into the process of a novice programmer learning to code. By investigating points of divergence between the higher and lower achieving students, this project will identify the parts of an introductory programming course that may be pitfalls or stumbling blocks for novice programmers. This research gives us insight into what topics and concepts need to be introduced later, and which need to be covered in more depth and with more repetition, allowing for increased potential for success of our students.

This research has been in three different behavioural data types:

- Mouse movement and its relation to student stress and success.

- Coding behaviour, specifically patterns of compilation, running, and evaluation and how it correlates to student understanding and success.
- Code complexity and its relation to student success and understanding.

The third contribution to the field is the creation of the HOG classifier, a data processing and Neural Network binary classifier system that predicts with up to 87% accuracy the likelihood of a student being in the higher or lower 50% of the grades in their introduction to Computer Science module.

1.4 Publications

These publications are each printed in edited form as a part of this thesis. *Building an Authentic Novice Programming Lab Environment* is in Section 3.3, *What the Mouse Said: How Mouse Movements Can Relate to Student Stress and Success* is in Section 5.2, and *Exploring the Coding Behaviour of Successful Students in Programming by Employing Neo-Piagetian Theory* is in Section 6.2. Permission has been granted for the reproduction of these papers. In this section, the paper details will be presented, along with tables detailing my contribution to each paper.

1) Building an Authentic Novice Programming Lab Environment

N. Culligan and K. Casey, “Building an Authentic Novice Programming Lab Environment,” in *International Conference on Engaging Pedagogy*, Dublin, Ireland, 2018.

In this paper, the creation of MULE and the motivation behind the system from a pedagogical perspective is described. The author contributions to this paper can be seen in Table 1-1.

	<i>My Contribution</i>	<i>Other’s Contribution</i>
<i>Software</i>	70%	30%
<i>Writing</i>	90%	10%
<i>Running Experiment</i>	80%	20%

Table 1-1: Contributions to Published Paper 1

2) What the Mouse Said: How Mouse Movements Can Relate to Student Stress and Success

N. Culligan and K. Casey, “*What the Mouse Said: How Mouse Movements Can Relate to Student Stress and Success*,” in Psychology of Programming Interest Group, Toronto, Canada, 2020.

In this paper, the first exploration of the Mouse Movement data collected by MULE from second semester data is described, along with the construction of a stress classifier, and a pass/fail classifier that would become the basis of HOG. The author contributions to this paper can be seen in Table 1-2.

	<i>My Contribution</i>	<i>Other's Contribution</i>
<i>Software</i>	70%	30%
<i>Writing</i>	80%	20%
<i>Running Experiment</i>	70%	30%

Table 1-2: Contributions to Published Paper 2

3) Exploring the Coding Behaviour of Successful Students in Programming by Employing Neo-Piagetian Theory

N. Culligan and K. Casey, “Exploring the Coding Behaviour of Successful Students in Programming by Employing Neo-Piagetian Theory” in *Psychology of Programming Interest Group*, Toronto, Canada, 2020.

This paper discusses the exploration of a second data type, Compile-Run-Evaluate movements, using the first semester data that would become the main data set for this thesis. The author contributions to this paper can be seen in Table 1-3.

	<i>My Contribution</i>	<i>Other's Contribution</i>
<i>Software</i>	80%	20%
<i>Writing</i>	85%	15%
<i>Running Experiment</i>	80%	20%

Table 1-3: Contributions to Published Paper 3

1.5 Chapter Overview

In this section, an overview of each chapter in this thesis will be presented.

Chapter 2: Related Research

In this chapter, the related research and alternative tools for this project are examined and discussed. This literature is in two sections: a review of pedagogical coding environments, and a review of literature on examining student behavioural data. In the review of alternative pedagogical environments, different pedagogical coding environments are examined, in terms of usefulness as teaching tools, and as data collection tools. In the review of other work that examines novice programmer behavioural data, the goals and tools used in this thesis are explained.

Chapter 3: MULE and the Collection of Data

In this section, the published conference paper on the MULE system is reprinted. MULE and its features are described, as well as a description of the modules and universities the system has been used in. The data collected is also outlined, as well as an explanation of how the data was collected ethically.

Chapter 4: Overview of Experiments and Data

In this section, the data sets gathered by MULE during this thesis are listed and described. The assignments completed by students while this data was collected is listed here, along with a summary of the assignments topics. The different data types collected by MULE are listed and explained. The methods used to examine the collected data are explained here, including the HOG classifier, a method for cleaning MULE data, and using it in Neural Network classifiers. Finally, the four experiments in this thesis are named and described.

Chapter 5: Experiment 1: Mouse Movements

In this chapter, the MM paper is reprinted, which outlines the pilot study using the MM data gathered by MULE to build Neural Network classifier to classify sequences of MM as being from an exam session or a regular lab session. This chapter also explains how the data was used with the HOG classifier, and explains why these two experiments had different results, and why the HOG classifier may have been less successful.

Chapter 6: Experiment 2: Compile, Run, and Evaluate

This chapter outlines the second experiment, using the CRE data gathered by MULE. This chapter includes a reprint of a published pilot study, using a prototypical version of HOG with the CRE data, and then goes on to run the CRE data with the version of HOG used in the four experiments in this thesis, to allow for accurate comparisons of the results to the results from Chapter 5 and Chapter 7.

Chapter 7: Experiment 3: Complexity of Student Code

This chapter outlines the third experiment, using the COMPLEX data gathered by MULE. COMPLEX uses compressed code size and the number of nodes in a parse tree to compare complexity of student code. The methods used to process the COMPLEX data is outlined, and the results of running this data with the HOG classifier are presented and discussed.

Chapter 8: Experiment 4: The HOG Classifier

This chapter describes the final experiment. In this experiment, the HOG classifier is used with all three of the data types from the previous three chapters: MM, CRE, and COMPLEX. This chapter examines how successful the classifiers are when using this data.

Chapter 9: Conclusions

In this chapter, the findings and results of the four experiments will be compared and discussed. The Research Questions outlined in Section 1.2 are examined in regard to how they are answered by the thesis. Finally, there is a section outlining the findings of the experiments and the thesis.

1.6 Chapter Conclusion

In this chapter, the problem that this thesis aims to investigate is outlined and the Research Questions are stated. The contributions and papers published from this research are listed, and summaries of the chapters in the thesis are presented.

2. Related Research

In this chapter, the potential tools, technology, and methods for this thesis will be discussed, by examining similar literature, and discussing the tools and techniques used by these studies.

The goal of this project is to investigate student behaviours that would inform us on:

1. An individual student's likelihood of performing well in the module according to their grades at the end of the semester.
2. The key points in divergence between the highest and lowest achieving students.

We hope that by learning more about these two things, we can first, identify students in danger of failing and intervene in time for the student to catch up with their peers. Secondly, we hope that finding early signs of failure may indicate certain times or topics in a student's first year of learning to code that are key in their eventual success. If a student is likely to fail because they don't understand a topic on the first try, this may indicate it is a topic that needs more time and attention in the curriculum, or in other support facilities.

This project aims to investigate the passively observed behaviour of programmers, using learning analytics gathering software in conjunction with a system for novice programmers to write, compile, and run their code as they learn to program.

In this chapter, we will discuss the related literature in the following areas:

1. Research for building the pedagogical coding environment MULE.
2. Research for collecting and processing the data from MULE, and building the HOG classifier.

2.1 MULE as a Pedagogical Tool

The first research question asks, "*How can we observe student behaviour as they learn to code in a non-intrusive way?*". When this research project began, VPL [7], the Virtual Programming Lab plugin for Moodle, was the system used to deliver assignments to students in Maynooth University first year Introduction to Programming labs, and to provide automated grading and feedback, so an investigation of VPL took place for three reasons:

- 1) To investigate if VPL could be modified to collect data for the purposes of this project
- 2) To investigate alternatives to VPL, and if they could be modified
- 3) To investigate what advantages VPL has as a pedagogical tool, so that any replacement built would not have a negative impact on the students.

In this section, VPL and its alternatives are discussed. The Kitchenham method [8] was originally used to carry out the literature review, but as the direction of the project changed, and various parts were removed and added in, and so the literature review is no longer in this format.

2.1.1 VPL

2.1.1.1 Overview of VPL

VPL is an auto-grading plug-in for Moodle, a SCORM [9] 1.2 compliant, open-source Learning Management System, or LMS. VPL provides a simple, online development environment for programming assignments that can be configured to give students instant feedback on their code. It is an open-source tool and can be freely used and modified. VPL has a large range of languages it supports, automatic and semiautomatic grading, password restrictions, black box testing or test cases, plagiarism detection, and offers configurable features for each individual assignment [7]. The student view can be seen in Figure 2-1.

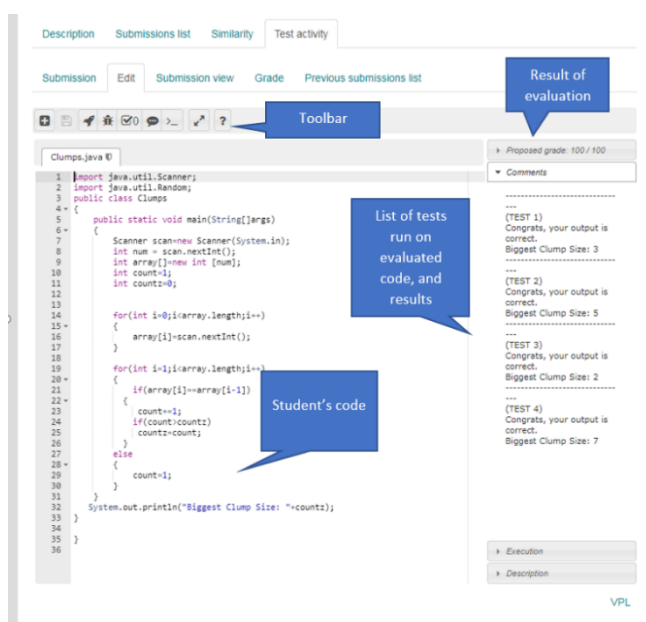
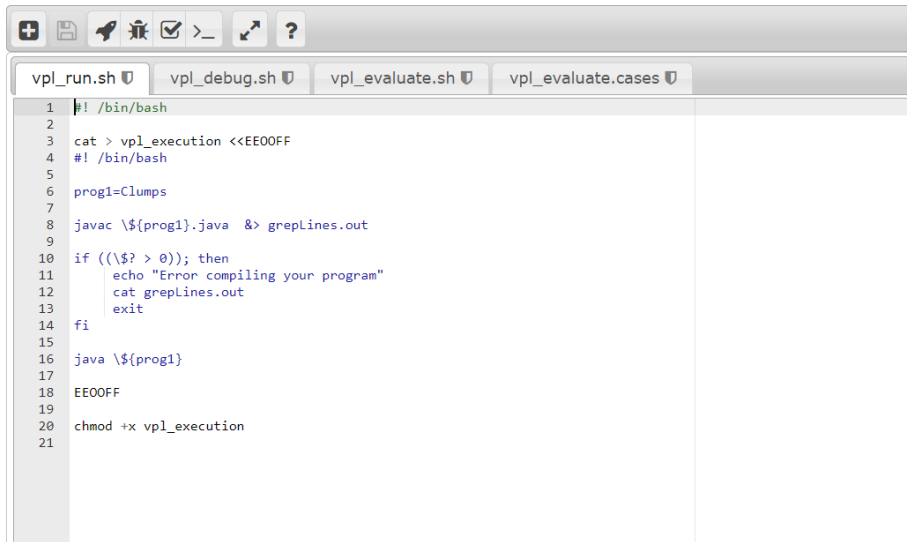


Figure 2-1: VPL Student View

When a student compiles, runs, or evaluates their code in VPL, a shell script provided by the course co-ordinator is used to run the students code in a jail server.

Execution files: Clumps



```
1 #!/bin/bash
2
3 cat > vpl_execution <<EEO0FF
4 #!/bin/bash
5
6 prog1=Clumps
7
8 javac ${prog1}.java &> greplines.out
9
10 if (($? > 0)); then
11     echo "Error compiling your program"
12     cat greplines.out
13     exit
14 fi
15
16 java ${prog1}
17
18 EEO0FF
19
20 chmod +x vpl_execution
21
```

Figure 2-2: Example of the Execution Files in VPL

These scripts are called execution scripts and the execution scripts view can be seen in Figure 2-2. The execution files are:

- vpl_run.sh
 - This script is used to run the user's code on the Jail Sever. In the screenshot in Figure 2-2, the script runs the commands:

```
javac Clumps.java
```

```
java Clumps.java
```

To compile the user's code and then run the code and return the output to the user.

- vpl_debug.sh
 - Used in running the debugger. Not used in our course, so left blank.
- vpl_evaluate.sh

- In the evaluate scripts, the code submitted by the user is run against a test, or multiple tests, each of which awards marks to the user if the test is passed.
- vpl_evaluate.cases
 - Some of the tests in vpl_evaluate.sh may use different input to test the user's code. The test input and the expected output can be stored in vlp_evaluate.cases.

Examples of each of these, other than vpl_debug.sh can be seen in Appendix Section 10.8.

The architecture of VPL consists of 3 components, as can be seen in Figure 2-3:

1. A plug-in module for Moodle
2. An in-browser code-editor
3. A jail server

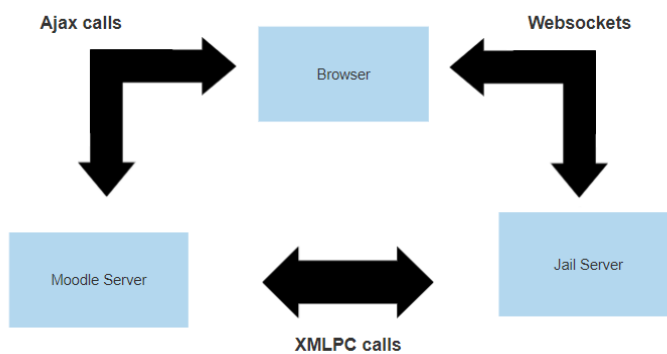


Figure 2-3: Diagram of VPL Components

The components communicate through Ajax calls, XMLRPC calls, and WebSockets [7]. The jail server is used to execute the test scripts on code submitted by students. The jail server is separate from Moodle, so if the jail server were to crash, or experience some issues, Moodle is unaffected. It is very unlikely that a student program could crash the jail server due to the restraints placed on the code run in the jail server that ensure that submitted code when run can only use a

set number of resources, as defined by the administrator, and is terminated if still running after a specified amount of time.

When an assignment is evaluated by VPL, the student's code and the instructor's test script is packaged into an XMLRPC message that is sent to the jail server. The student's code is tested with the script, and the output is sent back to Moodle. The output may contain feedback along with a grade, depending on how the assignment is set up by the course co-ordinator.

2.1.1.2 Advantages of VPL

VPL is flexible and robust and allows for diverse and sophisticated ways of testing student code [10]. The administrator has the option to:

- Control how the student code is graded
- Define the rubric used to grade the code
- Make the grade visible/invisible to the student
- Limit the number of submissions for an assignment
- Control the resources to be used by the jail server
- Allow programs to be submitted by individual students, or a group of students
- Access can be restricted by IP address
- Disable copy/paste of code in VPL

One of the advantages of VPL is the opportunity to provide students with instant feedback when they are programming. It has been reported that students do not read feedback when the feedback is not provided quickly [11]. When students can receive instant feedback on their code as they write, they are given the opportunity to assess their progress, and use the feedback given to improve their work and their understanding. Feedback is one of the most powerful influences on learning and achievement [12] [13].

2.1.1.3 Criticisms of VPL

VPL can help instructors to save time correcting assignments, but there is still a significant time commitment in setting up a VPL assignment, as they need to prepare a question, write a test script for evaluating student submissions, test that the script works well, and consider the many possible ways of solving a problem that a student may come up with, as well as providing useful feedback for issues students may encounter.

VPL is also a scaffolded learning environment, meaning it provides support to students as they learn a new skill [15]. Scaffolding is useful, but it is important that students can perform learned skills once the scaffolding is removed [15], and so it is important that the scaffolded environment is as close to the kind of authentic coding environment the students will “graduate” to as possible.

In VPL, it is difficult for users to browse through their already written code and use as a starting point for a new assignment but doing this may be indicative of a novice programmer who is doing well [16]. It requires multiple instances of a browser or switching between browser tabs, whereas in an authentic coding environment there would usually be an option for various editor windows.

VPL only allows for one version of an assignment to be saved in the system. This could discourage students from continuing work on an assignment once they have a solution that awards them a grade that they are happy with. There is evidence to suggest that “tinkering” with code is a strategy employed by successful novice programmers [18], so it is important to facilitate this behaviour. There are varying definitions of tinkering, as listed by *Berland et al.* [18], but for our purposes, the act of tinkering is playful experimentation.

While VPL is a useful solution to the problems associated with growing numbers of students enrolling in programming modules, the automation of the grading process results in the loss of an important connection between instructors and students, which may result in students that are struggling or in need of intervention not receiving the assistance that they need, and eventually failing or dropping out of the course [10].

2.1.2 Examples of Modifications to VPL

VPL is open source, and many modifications and extensions have been created. Existing modifications were examined to see if any fit the needs of this project and to investigate if it was possible to modify VPL to satisfy our requirements.

2.1.2.1 *Ante*

Ante is a framework that works with Moodle and VPL to encourage students to use test-driven software development. Ante requires that students submit test cases *before* they submit their coding assignment. The students are not allowed to submit their assignment until they achieve a perfect score in their test cases [18].

2.1.2.2 *Grading Process Management Module*

In the paper “*Architecture to Support Automatic Grading Processes in Programming Teaching*”, the Grading Process Management Module is described. At the time that the paper was published, the VPL system had only basic grading functionality, such as compilation and functional correctness, and there was no easy way to create customisable assessment. The grading submodule framework allows for additional assessment relating to structure, indentation, variable names, etc. The implementation of this system required changes in data infrastructure, directory system, and database. [20]

2.1.3 Alternatives to VPL

Other existing pedagogical coding tools were also investigated as potential tools for this project and will be examined in this section.

2.1.3.1 *BlueJ*

BlueJ is an IDE (Integrated Development Environment) designed by researchers in Kent University [21]. It is a pedagogical tool for teaching object-oriented programming concepts, designed with 3 specific goals in mind:

- To make the environment truly object-oriented by including a visualisation of the objects and classes.
- To encourage experimentation with individual objects to allow for better understanding of Java.

- To provide a simplified IDE to allow for students to focus on learning concepts instead of getting to grips with a complicated IDE.

BlueJ is based on the language Blue [21], which was designed to teach object-oriented programming. BlueJ uses a similar environment and visualisation to Blue, but uses the Java programming language [22]. While BlueJ is beneficial for introducing object-oriented concepts, unlike VPL, it does not include automatic assessment, and so was not appropriate for this project.

2.1.3.2 *iVprog*

VPL is compared to *iVprog* in the paper “*Programming Web-Course Analysis: How to Introduce Computer Programming*” [23]. *iVprog* is a virtual lab for visual programming languages, a way of introducing students to Computer Science concepts, such as loops and iteration, before introducing them to a formal programming language. Automated assessment in *iVprog* is based on test cases – the system compares expected outputs to the actual outputs produced by the student’s code. In a number of papers examining automated assessment of code [19, 24, 25], the authors noted the diversity of grading criteria, and roughly divided the criteria into two categories: static and dynamic. Static examines the users’ code, and dynamic examines the output of the code when it is run. For example, if a course co-ordinator wants to award marks based on the use of the “*System.out.print*”, that would be static criteria, however, if the course co-ordinator wants to award marks if the program prints the words “*Hello World*”, that would be dynamic criteria. *iVprog* can only grade based on dynamic criteria, unlike VPL, which can examine both.

In the study, two online courses teaching introduction to Computer Science were run simultaneously, with one class using VPL and the other using *iVprog*. Sixteen subjects finished the module, and the NASA TLX protocol [26] was used to test the mental demand and frustration of the users. They found that the users that used *iVprog* felt more frustrated than those using VPL, although the VPL subjects experienced more mental demand and effort to carry out tasks [23].

2.1.3.3 *Assignment Manager*

Xue Bai et al., 2016 [14] describe an assignment manager and a structure for auto-grading programming assignments. This web-based tool includes the following features:

- Editing tool
- Malicious code checking
- Runtime environment
- Auto-grading and feedback component

This system has similarities to VPL. It is a web-based tool that uses a remote application server to test student's programming assignments. The remote server contains a Java runtime environment, compiler, database, external files, services, and external APIs to create the programming environment.

In this paper, an experiment is described in which two sections of the same course, taught by the same professor, were run with and without the system. It was found that the number of errors per assignment per student was lower when using the system, and the average grade was higher when using the system. It was also found that the professor spent around 15 hours less on assignments with the system, despite giving 10 extra assignments. However, the scope of this experiment was limited, and further study needs to be done on this subject.

2.1.4 Pedagogical Coding Environments Conclusions

VPL is a useful Moodle-based plugin for teaching and assessing programming concepts. However, it is tied to Moodle, which may not be ideal for all institutions, and testing and teaching in this environment is not an authentic programming environment. The fact that it is web-based is an advantage – students can log in anywhere from any computer and access their work, course materials, and write and compile programs. As previously mentioned, although VPL is a useful solution to the problems associated with growing numbers of students enrolling in programming modules [14], the automation of the grading process results in the loss of an important connection with students for the instructor. Without this connection, it may be hard to tell if the students have a good grasp of the material [10]. VPL can help instructors to save time on correcting assignments, but there is still a significant time commitment in setting up a VPL assignment, as they need to prepare a question, write a test script for evaluating student submissions, test that the script works well, and consider the many possible ways of solving a problem a student may come up with, as well as anticipating and providing useful feedback for issues students may encounter. VPL is also only

equipped to deal with small pieces of code and is not yet capable of handling larger programming projects.

From this study into VPL and its alternatives, it was decided that it was necessary to custom build a system for this research, to allow for non-intrusive collection of data from an authentic coding environment, or as close to authentic as a pedagogical coding environment could be. The system would need to include the following:

- An emphasis on an “authentic” coding environment
- Passive collection of data
- Content delivery within the system
- Support the same large range of programming languages as VPL
- Automatic and semi-automatic grading
- Configurable features for every individual assignment, including disabling copy/paste, restrictions on IP address, time, and individual usernames
- A secure jail server that ensures that submitted code when run can only use a set number of resources, as defined by the administrator, and is terminated if still running after a specified amount of time
- The instructors at Maynooth University have already spent a significant amount of time setting up VPL assignments by preparing questions, writing test scripts for evaluating student submissions, testing the scripts, and writing useful automated feedback for issues students may encounter. For this reason, it was necessary for the new system to be able to use the same scripts as VPL

2.2 MULE as a Research Tool

When planning the system as a research tool, research into student programming behaviour and the data gathered was examined, to inform on how to collect the most potentially successful data. We also wanted to collect data passively, and never interrupt the students normal learning-to-program experience.

In the papers describing experiments involving the BlueJ [24, 28] system by Jadud, the papers describe the data collected by the system, which include:

- Student code
- Username
- Number of compilations so far

- Compilation result
- Filename being compiled
- When compile was initiated
- When the server received the information
- IP-address
- Hostname
- OS name
- Snapshots of code every time the compiler was used

The data was collected passively and was used to examine correlations between “error quotient”, a metric for how well students deal with errors, and module outcome, using simple Linear Regression. In the BlueJ paper “*A First Look at Novice Compilation Behaviour using BlueJ*” the authors discuss “extreme movers” (a reference to the paper “*Conditions of Learning in Novice Programmers*” [29]), which they describe as “tinkerers”, and how these students would sometimes allow their experimental code to accumulate, causing their code to become increasingly complex and, eventually, incomprehensible. The authors found that students tend to program in large blocks, then spend time writing and compiling code in small bursts in order to fix syntax errors. Accordingly, multiple compilations may indicate a large number of syntactic problems.

The paper “*Using Keystroke Analytics to Improve Pass-Fail Classifiers*” [30] uses keystroke analytics to predict a student’s success in a programming module, and notes that keystrokes are useful for improving accuracy in early semester predictions, when interventions are likely to have better impact. The authors note that although most of the data gathered over the course of a semester is gathered at the end, there is more than enough keystroke data to assist in predicting student outcome early on.

In the paper “*Analysis of Source Code Snapshot Granularity Levels*” [31], the authors examine three different data types from novice programmers of various ages from 12 to 76:

1. Submissions
2. Snapshots (save, compile, run, and test events)
3. Keystroke-events

Submissions are final versions of a program submitted for correction/grading, provided by a plugin for NetBeans that provides feedback and

grading to the student. Using Wilcoxon Rank Sum test, as the data was not assumed to be normally distributed, the authors found statistically significant differences in the amount of work from participants to reach assignment goals between those with programming experience and those without. This difference continued to be visible throughout the course, although the behaviour of the participants was more alike in the final weeks of the course, perhaps implying that these behaviours are indicators of programming proficiency.

In the paper “*Evaluating Neural Networks as a Method for Identifying Students in Need of Assistance*” [32], the authors use a measurement called “steps” when building a system for identifying students in need of assistance, where “Steps” were calculated as the number of submission events recorded for each coding exercise. The paper also examined time spent and error counts. In this paper, the authors do not include absence of submissions as a feature, as they are focusing on identifying at-risk students who are actively participating in the course, and to better match the work by *Ahadi et al.* [33] that they were reproducing. The paper also explores the efficacy of Neural Networks in identifying students who need assistance as early as possible in an introduction to programming course. When building the Neural Network, the data was vectorised but not normalised. The Neural Network had rectified linear units in the internal layers, a cross entropy cost function, 200 units per layer, fully connected layers and randomly initialised parameters. The Neural Network was tested with 1,2, and 3 hidden layers, and the Neural Network with 1 layer was most successful. The conclusions of the paper found that Naïve Bayesian, Random Forest, and Neural Networks all performed well in classifying students in danger of failing, but found that the Neural Network was “pessimistic”, was more accurate in classifying failing students, and was more likely to classify passing students as failing than the other way around.

In “*Programming: Factors that Influence Success*” [34] Bergin and Reilly examined 15 factors in predicting if a student is likely to pass or fail, using Pearson correlations. The strongest correlation to success was the student self-perception of their understanding of the module, and one of the most statistically significant factors in predicting success was comfort level, in relation to how the student felt about the course. This was measured through cumulative responses to questions about the students’ understanding and difficulty completing lab assignments. A regression model was built that was able to account for 79% of the variance in programming performance results.

“The Normalized Programming State Model: Predicting Student Performance in Computing Courses Based on Programming Behavior” [35] introduces the Normalized Programming State Model (NPSM). This model is comprised of 11 different states that the students move through as they work on assignments, such as “Editing without debug”, “Editing Syntactically Correct, Last Debug Successful”, “Editing Syntactically Correct, Last Debug Unsuccessful”, data points on students as they code. This data is used to predict students’ performance in a programming module with 36% – 67% accuracy.

In the paper *“MouStress: Detecting Stress from Mouse Motion”* [36] Sun *et al.* constructed a Mass Spring Damper model for the human arm - essentially a model for approximating arm motion and stiffness which could be fed with data from mouse movements. Using arm stiffness as a proxy for stress in the user, the authors report that their method was tested across a variety of prescribed stress tasks and the stress detection was still strong when generalised across these different tasks. Student t-tests were used to examine the correlation between the measures and the subjects state of stress. The classifier worked when generalised but was more effective when trained and tested separately for each user. The final model has an accuracy of around 70%.

In the paper *“Mouse Trajectories and State Anxiety: Feature Selection with Random Forest”* [37] , Yamauchi claims there is both psychological and neurological evidence to suggest that mouse trajectories can be used to assess affective states, such as anxiety. The results of their study show that temporal features, such as speed of mouse movement, and spatial features, such as direction change, were both indicative of the user’s state of anxiety, and a classifier was built using these metrics and applying Random Forest.

In the paper *“When high-powered people fail: Working memory and “choking under pressure” in math”* [38], Beilock and Carr discuss the connection between anxiety and a loss in academic performance and suggest that situation-related worries – such as examination stress or anxiety – can result in a loss of focus on task at hand as the working memory is occupied. Alternatively, it has also been suggested that over-attending to performance, overthinking tasks usually performed automatically, can lead to underperforming in an uncomfortable or stressful situation. Similarly, in the paper *“On the causal mechanisms of stereotype threat: Can skills that don't rely heavily on working memory still be threatened?”* [39] Beilock *et al.* discuss how a more stressful or anxious state can also affect

tasks that are usually performed in an automated fashion, without the subject consciously thinking about it.

In the paper “*Automatic Prediction of Frustration*” [40] Kapoor *et al.* use a specialised pressure mouse with additional sensors to detect frustration in subjects as they attempt to complete a Towers of Hanoi puzzle computer game. The game includes an “I’m frustrated” button for the users, which is used to associate behaviour with frustrated state. The resulting classifier can predict frustration at an accuracy of 79%, outperforming the random classifier (58%).

In “*Neo-Piagetian Theory and the Novice Programmer*” [41], Teague found that the development of programming skills is both “sequential and cumulative”, and that behaviours associated with sensorimotor and preoperational reasoning are evident from very early in the semester. Teague also reports that there is evidence of students beginning to struggle at a very early stage, before non-trivial concepts are introduced.

In “*Concrete and other Neo-Piagetian forms of Reasoning in the Novice Programmer*” [42], Lister discusses the reasoning behind the use of Neo-Piagetian and non-classical Piagetian theory. Classical Piagetian theory considers the progress through different stages of learning to be a consequence of a biological maturing of the brain. Neo-Piagetian theory, on the other hand, considers this instead a result of gaining experience, and in particular, the ability to “chunk” knowledge within a certain knowledge domain. The paper reports that students in their CS1 classroom exhibited three broad forms of Neo-Piagetian reasoning – Formal Operational Reasoning, Preoperational Reasoning, and Concrete Operational Reasoning.

The authors of “*Mired in the Web: Vignettes from Charlotte and Other Novice Programmers*” [43] ask if a student can have different levels of ability for different tasks which test similar programming concepts – if a student can trace and understand code, can they also write that code? They also ask why some students do not seem to be able to understand code with abstractions and instead rely on tracing code with specific values. The study found that students who were still operating at the sensorimotor level in Week 2 were often still operating the same way in Week 5 and were lagging behind students who were operating at the preoperational level in Week 2. They defined students in the preoperational stages by certain behaviours which they observed using think-aloud data from students. Preoperational behaviours were guessing, a fragile grasp of semantics, confused

use of nomenclature, an inability to trace simple code, as well as general misconceptions. Errors due to cognitive overload and reluctance to trace were considered behaviours associated with both sensorimotor and preoperational. The ability to trace but not explain code, as well as a reliance on specific values, were signs of the preoperational stage.

In the paper “*Problem Solving and the Development of Abstract Categories in Programming Languages*” [44] Adelson found that expert programmers’ memory chunks tended to be semantically or functionally related, while novices typically chunked by syntax. Semantic knowledge consists of programming concepts that are generalised, and independent of programming language, whereas syntactic knowledge is more precise and rooted in exact representations of concepts in specific programming languages

The paper “*Utilizing Student Activity Patterns to Predict Performance*” [45] uses “*data such as the number of successful and failed compilations, on-campus vs. off-campus connections, time spent on the platform, material covered*” to create a pass-fail classifier for programming students, and to gain insights into how students are using key concepts. By the end of the semester, the pass-fail classifier works with just under an 85% accuracy. The most successful technique used by the researchers is compression of student code (with comments removed) and measuring the resulting compressed file to measure code complexity, which was the most successful feature in the classifier.

In conclusion, this research led us to focus on three main areas of data:

1. Low level behavioural data: Mouse movements, as a possible indicator of student stress [36, 37, 38, 39, 34]
2. Medium level behavioural data: Compile, run, and evaluate actions [24, 28, 35, 32, 45]
3. High level behavioural data: Code submitted for assignments [45]

In this chapter, the related research and alternative tools for this project are examined and discussed. This literature is in two sections: a review of pedagogical coding environments, and a review of literature on examining student behavioural data. In the review of alternative pedagogical environments, different pedagogical coding environments are examined in terms of usefulness as teaching tools, and as data collection tools. In the review of other work that examines novice programmer behavioural data, the goals and tools used in this thesis are explained.

3. MULE and the Collection of Data

In this chapter, the motivation for the design choices for MULE and for the HOG classifier are explained and related back to some of the literature explored in Chapter 2.

3.1 Motivation

From the literature review, it became clear there was no existing software that would collect the range of data that was required for this study and satisfy the pedagogical requirements of the first-year introduction to programming course.

As there was no existing software to satisfy the requirements, the online programming education tool, MULE, was created. It is not an entirely “authentic” programming environment, in that this IDE will not be used in software development work environments, it is purely pedagogical. However, in “*A Quantitative Analysis of a Virtual Programming Lab*” [46], a study is described in which a learning group is divided into students who use web-based tools and a control group that used a traditional setup - there was no significant difference between the results of the two groups, suggesting that the use of web-based tools does not negatively impact on a student’s progress. The paper also states that “*Students typically spend too much time to install such tools and get acquainted with them, just to be able to perform their homework assignments*”, suggesting that a system that can simply be logged into from any browser can be beneficial – it removes a significant amount of the initial learning curve that may be intimidating to students, possibly contributing to the high dropout rate of students in introductory Computer Science courses [47].

3.2 Paper: Building an Authentic Novice Programming Lab Environment

Building an Authentic Novice Programming Lab Environment

Natalie Culligan and Kevin Casey
natalie.culligan, kcasey { @mu.ie }
Faculty of Computer Science
Maynooth University
Maynooth, Co Kildare, Ireland

Abstract

As computer science becomes increasingly popular and classes become larger, there is an ever-increasing demand on course coordinators' time. As well as teaching classes, running labs, preparing exams, and providing feedback to students on their work throughout the year, course coordinators are required to keep their courses updated in order to prepare their students in a rapidly changing and evolving industry. As computer scientists, and as programmers, automation stands out as a potential solution. Automating the correction of labs and exams would free the course coordinators' time, allowing them to focus on improving the course in other ways. VPL, or Virtual Programming Lab, is a plugin for a Learning Management System, such Moodle, that provides automation of this nature, by using shell scripts to assess student code and provide automated feedback. The VPL system includes a web-based editor embedded in Moodle that students use to write their code. Our concern is that VPL does not provide a sufficiently authentic programming experience. With this in mind, we have created MULE, a browser-based desktop environment in which students can view course assignments, write, compile and run their code, while maintaining the advantages provided by VPL such as instant feedback.

Keywords

Computer Science Education, Programming, Virtual Coding Environment, Automatic Assessment, Computer Science Pedagogy, Online Programming

1. Introduction and Motivation

It has been claimed that the most effective way to teach programming to students is through practical exercises [1]. However, the increasing number of students in software engineering classes makes it harder to correct and provide feedback to these students in a timely manner. This can result in fewer practical assignments and/or less useful feedback for each student. Automated grading tools that can provide useful feedback to help the student understand any issues with their code is essential to cope with these growing numbers of students [6]. It has been reported that students do not read feedback unless the feedback is provided quickly [7]. When students can receive instant feedback on their code as they write, they are given the opportunity to assess their progress, and use the feedback given to improve their work and their understanding [10]. Feedback is one of the most powerful influences on learning and achievement [8]. VPL, or Virtual Programming Lab is a Learning Management System (such as Moodle) based system that provides instant feedback to students as they perform programming assignments. The feedback can be tailored to the assignment and to the level of the class by the course coordinator.

VPL is a scaffolded coding environment. Scaffolding in education refers to support provided to students [9], in this case through an interface. Scaffolding is useful in educational settings, but it is important that when scaffolding is removed, the student can perform the learned tasks competently without the scaffolding [13]. VPL is different from an “authentic” coding environment in many ways and our concern is that students may encounter problems when “graduating” from VPL to a traditional coding environment. This research is focused on creating a programming environment that is as authentic as possible, while also providing tools for course coordinators to provide instant feedback to their students.

2. VPL

VPL, or Virtual Programming Lab, is an auto-grading plugin for Moodle,

or other SCORM compliant LMS [12]. VPL provides a simple, online development environment for writing programming assignments within the LMS. VPL has a large range of languages it supports and can be modified. It has automatic and semiautomatic grading, plagiarism detection, and offers configurable features for every assignment and allows for diverse and sophisticated ways of testing student code [5,12,14]. For example, the course coordinator has the option to define the rubric used to grade the code, make the grade visible or invisible to the student, restrict access by IP address or disable copy/paste in the VPL code editor. One of the key advantages of VPL is the opportunity to provide students with instant feedback when they are programming.

3. MULE

MULE is an online desktop-like environment that students can log into from anywhere and view their previous work and continue work on assignments, or practice coding. MULE imports or recreates the advantages of VPL and adds new features. It allows students to open windowed applications, emulating a traditional desktop. The course assignments are delivered through an application in the MULE desktop called “Workbook”. From the application the user can browse assignments to be completed, assignments already completed and all their submitted code. From any assignment page in the workbook, the user can open the code editor to write their code.

From the editor, a student can write, compile, run and evaluate their code. When a student runs their code, it does not run on their local computer. Instead we use the VPL jail server – an external server that runs a student's code and returns the output.

Every time a student makes an “attempt” on an assignment – when they save their work, or when they submit it for evaluation – the attempt is recorded by the system. We hope that this will allow cautious students to experiment with their code, without fear of doing irreparable damage to their final grade, as our system always saves the students highest grade and, at any point in time, a student can always return to a previous attempt.

The system uses two rating systems - “Grade” and “Personal Grade”.

“Grade” is the grade that contributes to the students’ continuous assessment and final grade. “Personal Grade” is to allow the students to go back and retry assignments they have completed, to complete them in a different way, or to try to achieve a higher score. Again, the idea behind this is that we want to encourage students to continue to use the system, rewrite code and revisit previous work after the labs have ended.

4.VPL and MULE

While VPL is an excellent solution to the problems associated with growing numbers of students enrolling in programming modules, the environment is not an authentic programming environment. VPL provides an in-browser code editor from which students can compile, run and evaluate their code but differs from a traditional coding environment in the following ways:

1. Users are not able to easily open multiple windowed instances of the code editor and compare their current assignment to previous ones.
2. Users can only save one version of their code for an assignment in the system.
3. The system is not designed for students to create their own code independent of assignments
4. VPL provides a “scaffolding” for students to work within. It is not an authentic coding environment, but a pedagogical one.

These differences are discussed below.

1) Some of the behaviors that are indicative of a student who is doing well in their introduction to programming module, such as reading through their old code, using pieces of old code as boiler plate or rewriting old code [4] is difficult to do in the VPL interface – it requires multiple instances of a browser or switching between browser tabs. In MULE, the users can open multiple instances of the code editor within the desktop environment, so students can quickly and easily compare their successful code with code which may be returning errors, for example, or use successful code as a template to write new code.

2) VPL only allows one version of an assignment to be saved within the

system. This could discourage students from continuing work on an assignment once they have a “good enough” solution, and also from continuing to “play” with the code after an assignment is completed. There is evidence to suggest that “tinkering” with code is a strategy employed by successful novice programmers [3], so it is important to facilitate this behavior.

3) When using VPL and Moodle, a course coordinator creates an assignment that the student is to complete. The assignment opens an editor, usually with the files to be submitted already created for the student to write their code into. While it is technically possible for an instructor to provide a “free” assignment where students can write whatever code they want, there is no easily usable filesystem for saving their code. In MULE, students can use both “Workbook mode” and “Editor mode”. Workbook mode is used for assignments set by the instructor – students need to write code in pre-named files, which they can then compile, run and evaluate from their editor. In Editor mode, students can use the editor as a normal editor - they can create new files in a traditional filesystem, save them under any name, compile and run them.

4) We are concerned that students may exclusively use VPL when writing code, and never write code outside of their given assignments. Novices often find the initial set-up involved with programming to be intimidating and frustrating. By removing barriers to entry such as the installation of editors, IDEs and compilers, students are free to focus on learning to write code [11,17] and may be more successful in their studies. However, this also presents a problem – if students are intimidated and frustrated by the installation of these programming tools, they may avoid them altogether, meaning students may exclusively use the learning environment. This may result in students who “graduate” from the learning environment to IDEs, like those used in industry, being disadvantaged, and unable to make the leap from “scaffolded” coding environments to traditional environments. For this reason, MULE simulates a regular desktop, but removes the initial barriers to entry by being browser-based.

5. MULE Feedback

As of semester one of the 2018/2019 academic year, we have begun using

MULE in our 1st year labs, with around 300 first year students, instead of VPL. The demonstrators, many of whom have experience with VPL, were asked to fill out a short survey on how VPL and MULE compare.

While it

would be ideal to survey students on their experiences with the system, unfortunately the students have not used VPL. Of the 10 demonstrators who have previous experience with the system, 5 thought MULE was more intuitive, 2 thought VPL was more intuitive, 3 that they were equivalent. The results of the survey showed that most of the demonstrators felt that MULE made it easier to review work from previous assignments(Q7) and that MULE was a “natural” or more authentic programming environment for students(Q8).

	Strongly agree	Agree	Neutral	Disagree	Strongly Disagree
Q1 VPL is intuitive to use	0.00%	72.73%	9.09%	18.18%	0.00%
Q2 MULE is intuitive to use	16.67%	58.33%	8.33%	16.67%	0.00%

	None	A little	Intermediate	A lot	Too much
Q3 Rate the level of intervention needed for MULE	0.00%	33.33%	58.33%	8.33%	0.00%
Q4 Rate the level of intervention needed for VPL	0.00%	33.33%	44.44%	22.22%	0.00%

	Definitely MULE	MULE mostly	About the same	VPL mostly	Definitely VPL
Q5 Which is quicker for novice students to get started with?	70.00%	20.00%	0.00%	10.00%	0.00%
Q6 Which do you feel students would more likely use outside lab time	30.00%	60.00%	0.00%	10.00%	0.00%
Q7 Which makes it easier to review previous work from labs	80.00%	10.00%	0.00%	10.00%	0.00%

	Definitely Browser /MULE	Browser /MULE mostly	About the same	Moodle /VPL mostly	Definitely Moodle /VPL
Q8 Which UI do you feel is most natural for students?	36.36%	45.45%	9.09%	9.09%	0.00%

Table 2.1 – Results of Demonstrator survey

6. Conclusions

MULE is an online programming education tool that retains of the advantages of VPL and has some significant improvements. Foremost among these is that it is a realistic browser-based representation of an authentic programming environment that students will encounter later in their courses, and ultimately in industry. The browser-based nature of the tool has a compelling advantage in that there is a low barrier of entry for students – they do not need to install any special software as they would normally be required to do. In “A Quantitative Analysis of a Virtual Programming Lab” [15], the authors state that “*Students typically spend too much time to install such tools and get acquainted with them, just to be able to perform their homework assignments*”. This suggests that a system that can simply be logged into from any browser can be beneficial

– it removes a significant amount of the initial learning curve that may be intimidating to students, possibly contributing to the high dropout rate of students in introductory Computer Science courses.

7. Future Work

There is a myriad of feature requests from the first-year students and their demonstrators, evidence of the high degree of engagement we have had over the first semester where MULE has been used. One of the more promising features to be provided for in the near future is enhanced error message reporting. There is evidence to suggest that clearer error messages for novice programmers may improve student success. In the paper “An Exploration Of The Effects Of Enhanced Compiler Error Messages For Computer Programming Novices” [2] the use of enhanced compiler error messages was tested, and the results were examined. The results showed that the use of the Decaf editor resulted in fewer signs of struggling students in comparison to a control group, who saw standard error messages. Integrating the Enhanced Compiler Error Messages into MULE would provide an opportunity to study how differently students behave when given clearer error messages.

References

- [1] Bai, Xue, Ade Ola, and Somasheker Akkaladevi. "ENHANCING THE LEARNING PROCESS IN PROGRAMMING COURSES THROUGH AN AUTOMATED FEEDBACK AND ASSIGNMENT MANAGEMENT SYSTEM."
- [2] Becker, Brett A. "An exploration of the effects of enhanced compiler error messages for computer programming novices." (2015).
- [3] Berland, M., Martin, T., Benton, T., Petrick Smith, C., & Davis, D. (2013). Using learning analytics to understand the learning pathways of novice programmers. *Journal of the Learning Sciences*, 22(4), 564-599.
- [4] Blikstein, Paulo. "Using learning analytics to assess students' behavior in open-ended programming tasks." Proceedings of the 1st international conference on learning analytics and knowledge. ACM, 2011.
- [5] Caiza, Julio C., and José María del Álamo Ramiro. "Programming assignments automatic grading: review of tools and implementations." (2013): 5691-5700.

- [6] Cheang, Brenda, et al. "On automated grading of programming assignments in an academic institution." *Computers & Education* 41.2 (2003): 121-131.
- [7] Duncan, Neil. "'Feed-forward': improving students' use of tutors' comments." *Assessment & Evaluation in Higher Education* 32.3 (2007): 271-283.
- [8] Getzlaf, Beverley, et al. "Effective instructor feedback: Perceptions of online Graduate students." *Journal of Educators Online* 6.2 (2009): n2.
- [9] Jackson, S. L., Stratford, S. J., Krajcik, J. S., & Soloway, E. (1995). *Model-It: A case study of learner-centered design software for supporting model building*. In Proc. from the Working Conference on Applications of Technology in the Science Classroom.
- [10] Kitaya, Hiroki, and Ushio Inoue. "An online automated scoring system for Java programming assignments." *International Journal of Information and Education Technology* 6.4 (2016): 275.
- [11] Richter, Thomas, et al. "ViPLab: a virtual programming laboratory for mathematics and engineering." *Interactive Technology and Smart Education* 9.4 (2012): 246-262
- [12] Rodríguez-del-Pino, Juan C., Enrique Rubio-Royo, and Zenón J. Hernández-Figueroa. "A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features." *Proceedings of the International Conference on e-Learning, e-Business, Enterprise Information Systems, and e-Government (EEE). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012*.
- [13] Sharma, P., & Hannafin, M. J. (2007). Scaffolding in technology-enhanced learning environments. *Interactive learning environments*, 15(1), 27-46.
- [14] Thiébaud, Dominique. "Automatic evaluation of computer programs using Moodle's virtual programming lab (VPL) plug-in." *Journal of Computing Sciences in Colleges* 30.6 (2015): 145-151.
- [15] Vanvinkenroye, Jan, et al. "A quantitative analysis of a virtual programming lab." *Multimedia (ISM), 2013 IEEE International Symposium on. IEEE, 2013*.

3.3 Overview of MULE

MULE is a desktop-like, browser-based pedagogical coding environment built for use in introductory programming modules, using the OS.js system [48]. An example of a layout of the system can be seen in Figure 3-1, showing a student view including the Workbook application, the code editor, and the terminal, with the output from code that has been run.

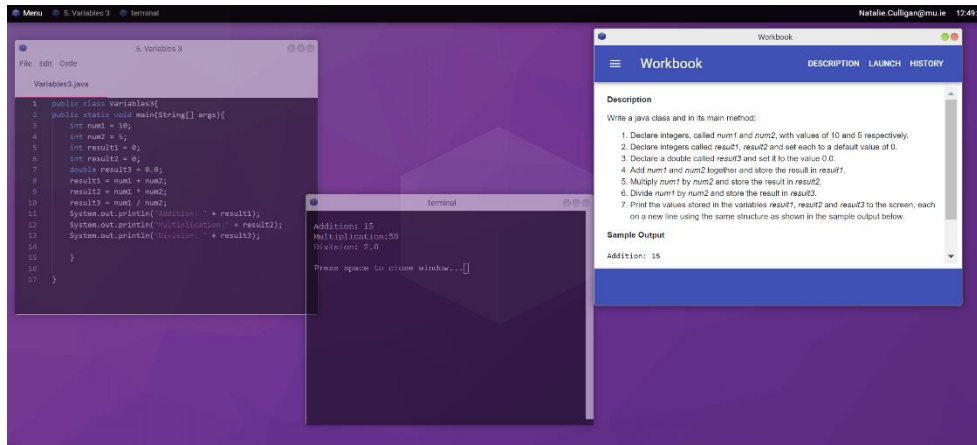


Figure 3-1: MULE Student View Layout

As the students use the system and work on their assignments, the system collects behavioural data such as assignments compiled, run, and evaluated, code written, errors returned, and mouse and keyboard interaction data. The data is stored as the user works, and it is periodically sent to the MULE server and saved in the database. The data is also sent whenever a user closes the MULE webpage.

MULE can be accessed through a browser, using Moodle authentication (or any other SCORM compliant authentication) to log in. MULE has advantages as a pedagogical tool over traditional environments, such as:

- Users can log in to the system from any internet browser and continue or review their work.
- Users do not need to spend time setting up their personal computers with editors or compilers. Instead, the student can potentially go from no coding experience to their first “Hello World” program in a few minutes.
- Users can access and complete their assignments entirely through the “Workbook” application. From this application, students can view the assignment instructions, write, run, evaluate, and submit their code. The Workbook application is described more in Section 3.4.1.

- Students can record notes directly onto their MULE workspace and access them from any browser. We did not anticipate this but found that many students choose to use MULE this way.
- Every attempt a student makes on a question is recorded and can be accessed from the “History” tab in the workbook, which is described in Section 3.4.2.
- Course co-ordinators can restrict access to certain questions or aspects of MULE according to admin-defined rules that check time, user id, user role (student or lecturer) and/or IP-address. For example, this feature was used to restrict access to students’ previously completed work during in-lab exams. By checking the IP-address of the user, students not currently taking the exam were able to use the system as normal and could not see the exam questions.
- Course co-ordinators can also choose to not allow copy and pasting within the MULE code editor, to discourage plagiarism.
- When a student evaluates their work, their code is assessed by an evaluation script, which can be customised by the course co-ordinator. The evaluation function is used to provide instant feedback to the student on their work.

3.4 Features of MULE

After the publication of the paper in Section 3.2, development continued on MULE, using the feedback from students and from course co-ordinators. The overall layout and additional features added to MULE are described below.

3.4.1 Workbook

Within MULE, we use the application “Workbook” to deliver assignments to students. The Workbook is a windowed application with a sidebar and tabs. The sidebar, which can be seen in Figure 3-2, is used to navigate through the different weekly labs and assignments. Once an assignment is selected, the user can select from the “Description”, “Launch”, and “History” tabs. There is also a fourth tab, “Admin”, only accessible by administrators.

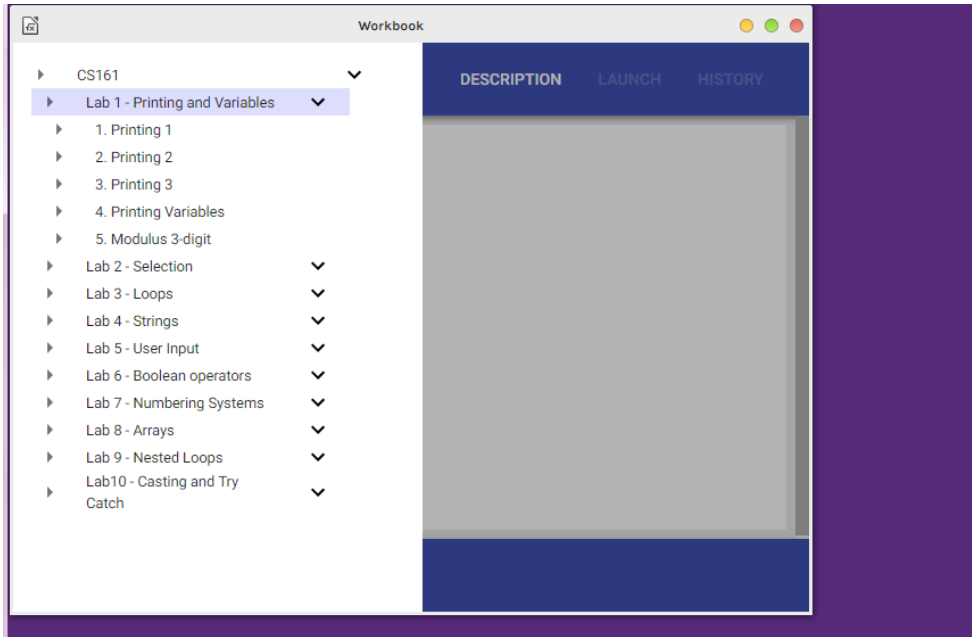


Figure 3-2: MULE Student Lab Selection Menu

All the content for the Workbook is stored in JSON files, including the execution files for the assignments and metadata files, as shown in Figure 3-3. Examples of these files can be seen in Appendix Section 10.9.

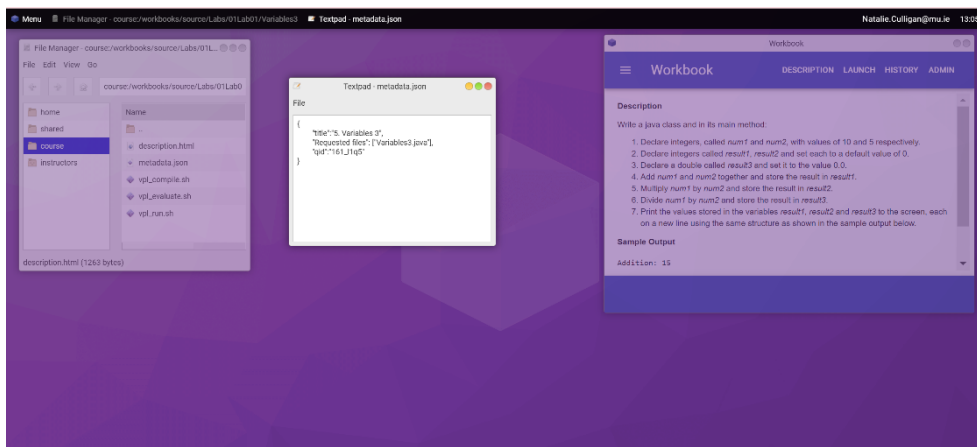


Figure 3-3: Assignment Metadata File in MULE

3.4.1.1 Description

This section in the Workbook displays instructions on how to complete the assignment for the student. The description is written by the course co-ordinator in the form of a HTML file, as can be seen in Appendix Section 10.9, in Figure 10-5, Figure 10-6 and Figure 10-7.

3.4.1.2 Launch

The Launch feature opens the code editor for the student to write their code. The course co-ordinator assigns a filename and has the option to provide default code, for example, if the purpose of the assignment is to edit or fix provided code. If the student has made previous attempts on the question, the most recent code will be opened. From here, the student can compile, run, and evaluate their code.

By evaluating their code, the student can receive instant feedback. There is evidence that students do not read feedback unless it is provided quickly [11], and when they do, they are more able to use it to improve their work and understanding [13].

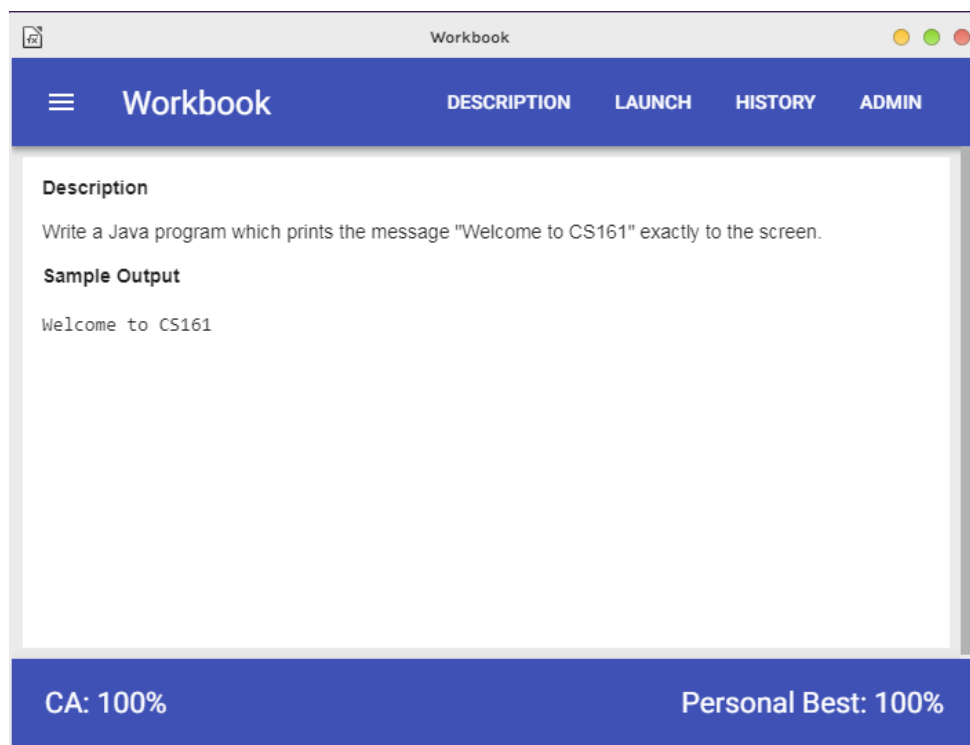


Figure 3-4: Description Panel in Workbook with CA Grade and Personal Best

3.4.1.3 History

It is mentioned in Section 2.1.1.3. that VPL would only allow one version of an assignment to be saved within the system. In MULE, the History tab shows every attempt made by the student on the question. We define “attempt” to be any code that has been saved, run, or evaluated. The attempts are displayed as a list, and includes the time, the grade given, and the grade type. The goal of this is to allow a student to experiment and change their code without fear of losing the code

that earned the highest marks, as there is evidence to suggest that playing/tinkering with code is an indication of student success [40].

3.4.1.4 CA Grade and Personal Grade

As shown in Figure 3-4, in MULE’s Workbook, a student can view two grades: their CA grade and their Personal Best. When using MULE in the first-year labs for the data sets used in this thesis, we chose to only allow for code to be evaluated for CA grades within the designated lab times by checking IP-addresses and time rules when a student evaluated. Note that the IP-address checking is optional, and it was not used during the Covid-19 lockdown when students were working on their labs from home.

As seen in the Figure 3-5, the MULE web application sends a request to the MULE server which checks the users’ permissions, and if they are permitted to run or evaluate the assignment (according to IP-addresses and current time) and if the result is a Personal Best or a CA grade. The evaluation is run on the jail server (using the VPL jail server described in Section 2.1.1.1). The result is returned to the MULE application via a websocket and the grades are stored in a database.

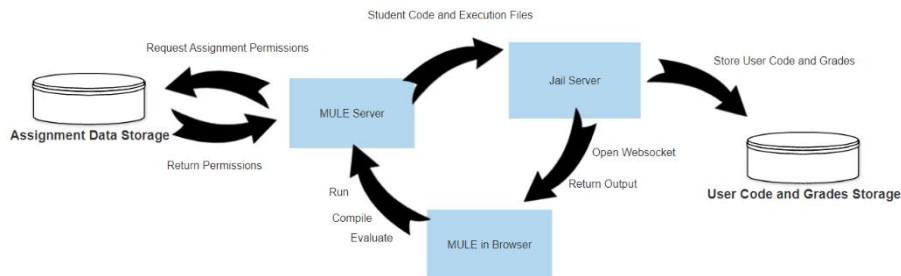


Figure 3-5: MULE Grade Storage

Once the weekly labs had been completed, the students were permitted to evaluate their code outside of the lab but would only be awarded “Personal Best” (Personal Grade) marks that did not count towards their final grade. With this system, the staff were able to both encourage students to continue to work on labs they did not achieve 100% on after the labs had completed, and ensure they attended the labs in person, where they can receive help and guidance from the lab demonstrators.

Using this system, if a student is struggling with labs, a demonstrator or lecturer can easily check if a student has completed the previous labs, and if not, can recommend the student familiarise themselves with the previous material. We also hope that by being able to record their highest mark, students will be more inclined to revisit previous labs and challenge themselves to complete labs in a different way.

3.4.1.5 Admin Mode

If a user logs in as an administrator (or lecturer), they are given access to an extra panel in the Workbook, which can be seen in Figure 3-6. In this panel, there are 5 options:

Compile Workbook: When changes are made to the workbook by an administrator, they must rebuild the Workbook with this option for the changes to take effect.

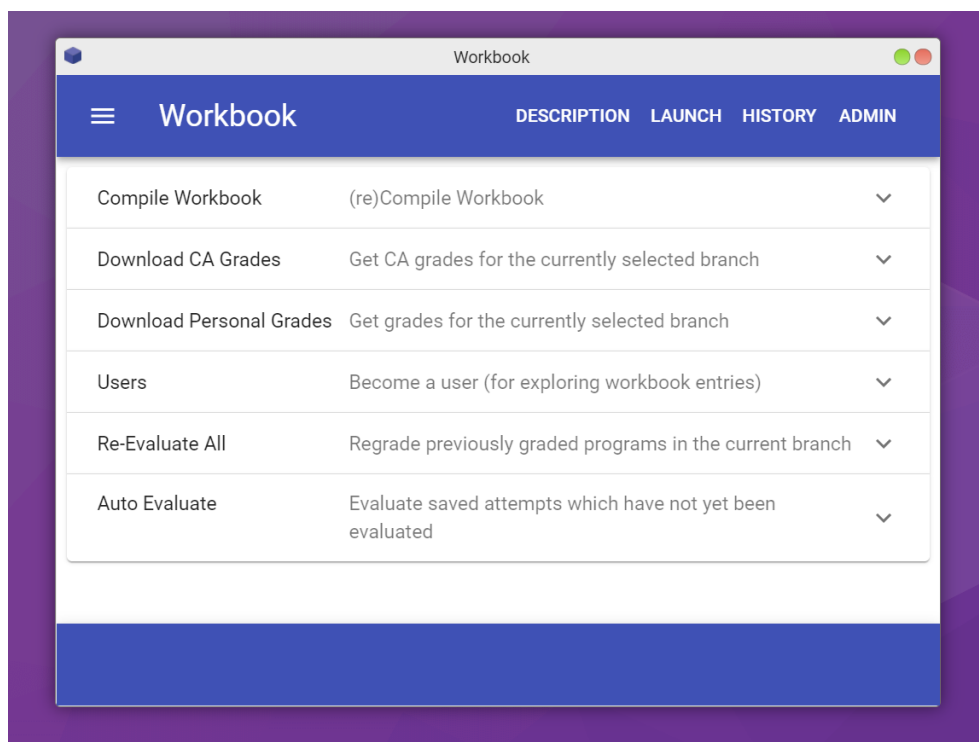


Figure 3-6: Workbook Admin View

Download CA Grades: This downloads a .csv file with the CA grades for the selected branch. If the user has selected Lab 4, all grades for all questions in Lab 4 will be downloaded. If Lab 4 Question 2 is selected, the .csv file will only contain the grades for Lab 4 Question 2.

Download Personal Grades: This option is the same as “Download CA Grades”, but downloads Personal Grades instead of CA.

Users: Here an administrator can select a student name and view the workbook as that student. They can view and edit all of the student’s attempts and grades.

Re-Evaluate All: This allows an administrator to re-run the evaluation option on all previously evaluated attempts between two user-specified times. This is useful if there is a problem with a script for example – the script can be rewritten and re-run.

Auto-evaluate: Allows the administrator to evaluate all the last saved attempts between two specified times, in the current branch. In our labs, we ran this option 5 minutes after each lab closed, allowing students who just missed the time limit to still get marks for their work.

3.4.2 Analytics

If the student gives consent when they first log into the system (or if they decide to give consent later), MULE will gather behavioural data as the student uses the system. MULE sends the data in a JSON format to the server periodically, after a set amount of time, or whenever the student logs out, as illustrated in Figure 3-7.

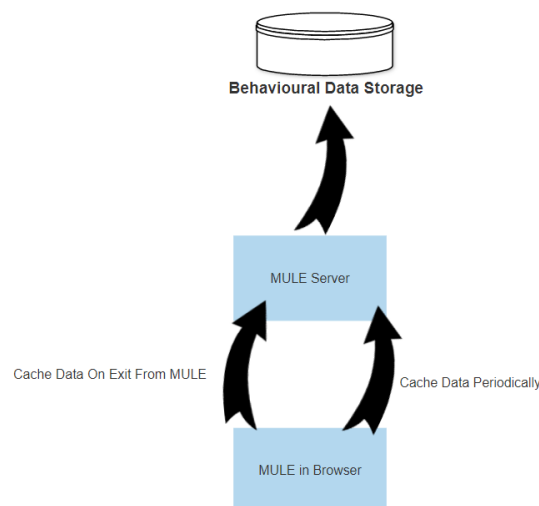


Figure 3-7: MULE Data Collection

The data collected is as follows:

- Mouse movements
- Code compiled/saved/run/evaluated
- Windows opened (such as compile/run/evaluate terminals)

3.5 Use of MULE in Introduction to Programming Modules

The MULE system has been used in first year Introduction to Programming modules in Maynooth University since September 2018. The system has been able to collect data from this module from February 2019. The system has also been used in the first-year Computer Systems module to teach the Prolog programming language.

MULE has been used in the Beijing University of Technology to teach the C++ programming language in connection with the University College Dublin - Beijing University of Technology collaboration. The system has also been used in the Fuzhou University in teaching the programming language Prolog. No data was gathered from these modules, as ethical clearance for data collection was only requested for the Irish Introduction to Programming through Java module.

Altogether, MULE has been used in teaching Introduction to Programming modules to over 1000 students. It has been of particular value during the Covid-19 shutdown, allowing students to continue to participate in labs and receive automated feedback and grades.

3.6 GDPR and Ethical Collection of Data

Ethical approval was applied for from the Maynooth University Ethics Committee and granted on the 7th of June 2018. When students first sign into the MULE system, they are asked to digitally sign a consent form which explains how their data is used and gives them the option to opt out of the research entirely. They are also asked to read an information sheet that explains clearly what they are agreeing to by signing the consent form. The information sheet and consent form can be seen in Section 10.1 and 10.2.

Due to the potentially sensitive nature of the data that we gather from students, storing and anonymising the data is of the utmost importance. Once we collect data from a user, the data is stored and associated with a hash of their login name or email.

3.7 Chapter Summary

In this chapter, the design requirements and the motivation behind MULE and the experimental design behind the experiments are discussed. The use of MULE in university courses is listed, and the ethical permission acquired for the data collection is outlined.

4. Overview of Experiments and Data

In this chapter, the data sets gathered by MULE during this project, and the assignments that the participants worked on while the data was collected, are explained on a week-to-week basis, in terms of the topics and programming concepts required to complete the assignments. The methods of analysis are explained, including the HOG classifier, a system built for this thesis that processes the MULE data, and runs Neural Network binary classifiers using that data. Finally, there is an overview of the four experiments described in this thesis.

4.1 Description of Data Sets

The MULE system has been used in the first year Introduction to Programming modules in Maynooth University since September 2018 and was able to collect data from February 2019. We have two data sets from the MULE system, semester 2 from the academic year 2018/2019, and semester 1 from the academic year 2019/2020.

The collection of data was then interrupted by the Covid-19 pandemic that required the closure of the lab facilities. While the students continued to use the system from home, it was unclear if the results from the data gathered from this semester would be valid. There were concerns that the results would not reflect the students' progress, but instead would be indicative of the students' home learning environment. In particular, the experiments that examined stress could potentially be influenced by the stress of living through a pandemic, so it was decided that this data would not be used.

4.1.1 Data Set 1

This data was collected from 196 out of 250 first year students in their second semester of Introduction to Programming, the academic year 2018/2019, from February to May of 2019. Of the 250 students, 54 were removed from the data set for one or more of the following reasons:

1. Student did not take both in-lab examinations
2. Student did not complete the course
3. Student participated in less than two lab sessions

This set is used in the MM paper in Section 5.2 only.

4.1.2 Data Set 2

In the first semester of the 2019/2020 academic year, data was collected as the students completed their first Introduction to Programming module, and as they learned to use the MULE system. The resulting data set is over 200GB.

The goal of this study was to focus on identifying at-risk students who were actively participating in the course, so data from students who did not complete most of their lab assignments and participate in both lab exams was removed. The four experiments in this thesis in Chapters 5, 6, 7, and 8 use 255 of the class of 300 from data set 2.

Participants were removed for the following reasons:

1. Student did not participate in both lab exams
2. Student did not participate in the final written exam
3. Student did not participate in more than 4 of the 10 lab sessions
4. Student requested to have their data removed.

4.1.2.1 Description of Weekly Assignments

The data is a set of behavioural data from the 255 students as they complete a total of 54 assignments across ten labs. A brief description of the topics for each lab is below, with a description of some of the more significant individual assignments.

Week 1:

The four assignment questions in this week's lab cover the concepts of:

- Print statements
- Assigning variables
- Basic mathematical operations
- Storing the results
- Printing variables

Week 2:

The six assignment questions in this week's lab require the use of:

- Mathematical operators
- Combinations of mathematical operators
- If-else statements
- Switch statements
- Ternary operators

In these questions the students are told explicitly which variable types and techniques to use (e.g., "Store in an integer variable" or "Use a switch statement").

Week 3:

The seven assignment questions in this week's lab cover the concepts of:

- While loops
- For loops
- Do-while loops
- Strings
- Numeric operations

In most of these assignments, the students are again told which techniques to use, but in Question 6, the students must decide themselves which kind of loop, or what kind of variable to use.

Week 4:

The six assignment questions in this week's lab cover the concepts of:

- String Manipulation
- Concatenation
- Generating substrings
- Finding the length of string
- Finding a character in string
- Convert to lower case or upper case
- Reverse strings
- Compare strings
- Selection statements

Students are given some guidance on how to solve problems, for example they might be told "use a loop to do x", but not be told which kind of loop to use.

Week 5:

The five assignment questions in this week's lab cover the concepts of:

- User input
- String manipulation as described in Week 4
- Mathematical operators
- Loops
- Selection statements

This is the first week that students are given almost no direct instructions in which techniques and variable types to use. This represents an important step in learning to program: not just how to use the techniques but knowing when to apply them to solve problems.

Week 6:

This week includes a lab exam and a normal lab session. The exam contains three questions and the normal lab session for this week contains three assignment questions requiring:

- User input
- Finding characters in strings
- String indexes
- Conditional statements

The exam consists of three exam questions covering the concepts of:

- Numerical operations
- Conditional statements
- For-loops
- While-loops
- User input

For both the normal lab questions and the exam questions in general the students are not explicitly told to use certain techniques.

Week 7:

The four assignment questions in this week's lab cover the concepts of:

- User input
- Character position in strings

- Conditional statements
- Loops

The questions do not explicitly ask the students to use certain techniques.

Week 8:

The six assignment questions in this week's lab cover the concepts of:

- Declaring arrays
- Manipulating arrays
- Updating arrays
- Reading from arrays

The questions do not explicitly ask the students to use certain techniques.

Week 9:

The five assignment questions in this week's lab cover the concepts of:

- Nested loops
- traversing 2D arrays

The students are told what type of loops to use for some of these assignments.

Week 10:

This week includes a lab exam and a normal lab session. The exam contains two questions, and the normal lab session for this week consists of three assignments requiring:

- Casting strings into other values
- User input
- Use of a try catch statement

The exam questions covered the concepts of:

- Numerical operators
- Conditional statements
- 2D arrays
- Nested loops
- String comparisons

The questions do not explicitly ask the students to use certain techniques.

4.2 Description of Data Types

Within the collected data sets, we collected three types of data which will be examined in this thesis. The data was divided into three tiers:

- Low level: Mouse movements
- Medium level: Compile, run, and evaluate patterns
- High Level: Complexity of submitted code assignments

These tiers will be discussed in Section 4.2.1, Section 4.2.2, and Section 4.2.3.

4.2.1 Mouse Movements

As the students use the MULE system, they use their mouse to navigate the environment, to open assignments, and to save, compile, run, and evaluate their code from drop down menus. This data is then used to generate metrics on student behaviour, which are explained in more detail in Section 5.2 and Section 5.3.

4.2.2 Compile, Run, Evaluate Actions

As the students complete their assignments, they use the system to compile, run, and evaluate their work. With this data, we examine if there are connections between how often a student “moves” from one of these actions to another, and if this is related to their CA and Written Exam outcome. This is explained in more detail in Section 6.3.

4.2.3 Complexity of Code Submitted

When a student evaluates their code, the code is sent to the jail server where a shell script runs a series of checks on the code, and on the output of the code, and assigns a grade according to how well the code fulfils the assignment criteria. The student code is stored using a RethinkDB database. Both the submitted code and the grade is stored for use in this study. The code complexity is measured using the size of the code when compressed, and the number of nodes in a parse tree generated from the code. This is explained in more detail in Section 7.2.

4.3 Methods of Analysis

The following methods are used to investigate the differences between the higher and lower achieving groups, as identified by their performance in their CA

grades and their end of year Written Exam, to select the best input data for classifiers and to build classifiers using the three data tiers.

4.3.1 Wilcoxon Rank Sum Test

The Wilcoxon Rank Sum Test is a non-parametric test used to test if there is a difference between two groups [31]. It is used here to examine the differences in Mouse Movement, CRE behaviour, and Code Complexity between the highest and lowest achieving students on a week-to-week basis, to give insight into when the two groups of students diverge in behaviour.

The features in the data sets for this project range in distribution – some are normally distributed, and some are not, and as the Wilcoxon Rank Sum Test does not assume known distributions, we can use it to examine if there are any significant differences between the two classifications, and to examine if there are any clear stages in the semester when the behaviour of students diverges. In this thesis, any answer of <0.05 is considered a significant result and implies that the two groups being compared are significantly different. The code written to process the data and carry out the Wilcoxon Rank Sum Test uses the Python library SciPy [49].

4.3.2 Linear Regression

Linear Regression is used to find how useful an individual metric is for predicting student outcome. The results of these tests are used to:

- 1) Examine which metrics are key in the divergence of behaviour of the higher and lower achieving students
- 2) Select which of the metrics from each data set are used as input features for the classifiers

The code written to carry out these tests uses the Sklearn [50] Python library to create the Linear Regression tests.

4.3.3 Neural Network Binary Classification

The input features selected by the Linear Regression tests are used to build Neural Network classifiers that predict if a participant is likely to be in the highest or lowest 50% of grades in the class in terms of CA or written exam. The code written to create and run the Neural Networks use the Tensorflow Python library [51]. Neural Networks were chosen as the paper “*Evaluating Neural Networks as a*

Method for Identifying Students in Need of Assistance” [32] found that Neural Networks performed as well as other classification methods but were more likely to classify passing students as failing than the other way around.

4.4 The HOG Classifier

Throughout the four experiments in this thesis, the HOG classifier, a companion for MULE, was written to generate Neural Network classifiers that classify students as being in the top 50%, or the bottom 50% of the class grades, according to the CA grades and the Written Exam grades. The paper in Section 6.2 uses a prototypical version of this classifier. The HOG classifier process can be seen in Figure 4.1.

1. Each of the four experiments in Chapters 5, 6, 7, and 8 have data sets that are divided into 10 subsections. These subsections are the 10 weeks of in-person labs during the semester. Although MULE collects data whenever the students use the system, the data sets only include data from the in-person labs, as we don't know what environment the students are in when working outside of these lab times. The data sets are stored in a database using SQLite. Python is used to generate metrics of the students' behaviour that will be used in the classifier, referred to as “features”.
2. For the midpoint of each grade category (CA and written exam), the data is divided into the top and bottom 50%. The top participants are marked as 1s and the bottom as 0s. These two groups are referred to as “Passes” and “Fails”, although it's important to note that not all the students in the “Fails” group failed the module.
3. The features from the given data types most correlated to the relevant grade category are selected using Linear Regression. The three data types had different ranges of correlation to outcome, so we used different correlation cut-off points (called “thresholds” throughout this thesis) and tested them with every datatype and both grade types. Any feature with a correlation of more than this threshold according to the Linear Regression tests is used in the classifier. The thresholds are: 0, 0.1 and 0.15. For one experiment -0.1 was also used, due to the low number of correlated features.

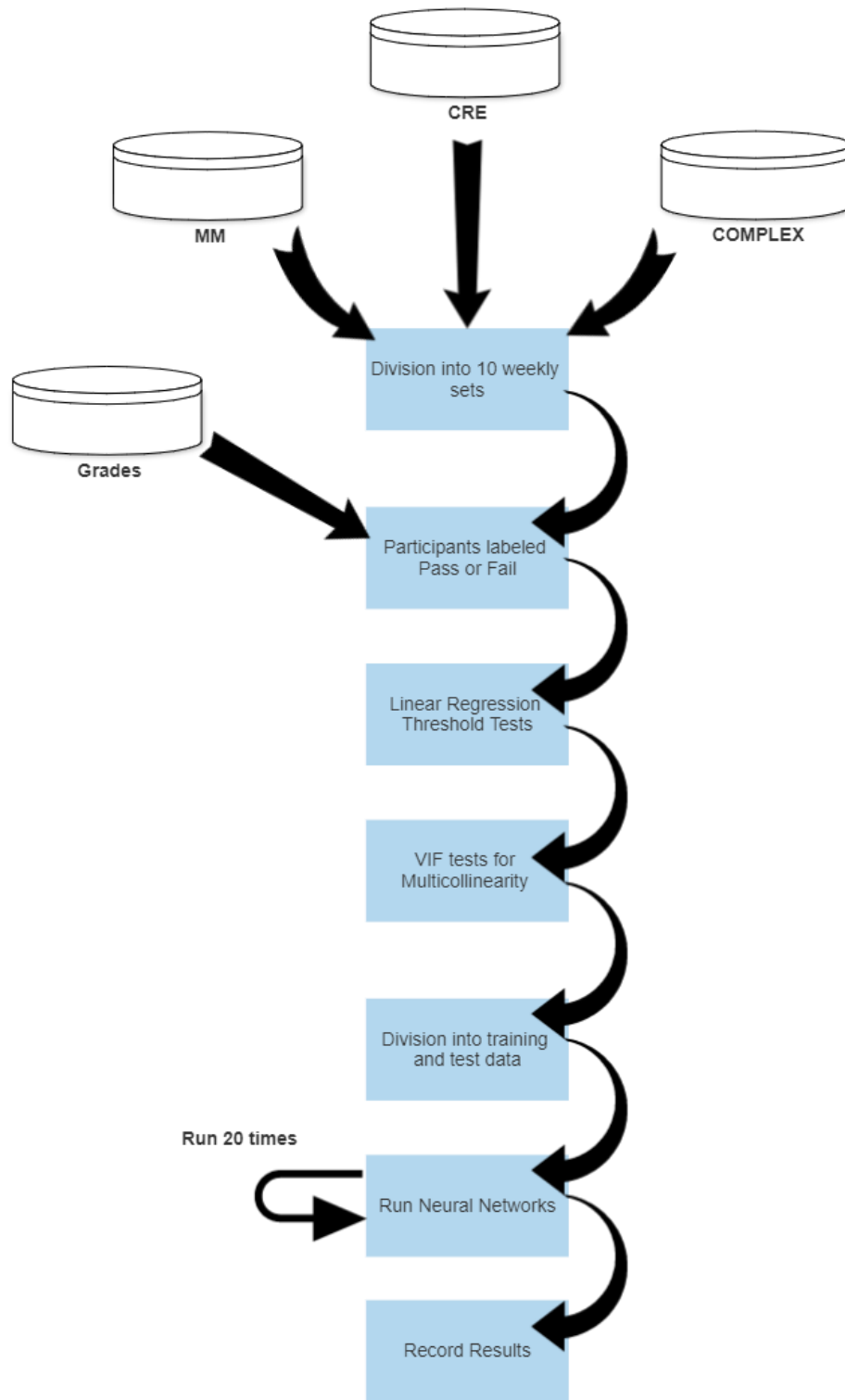


Figure 4-1: HOG Classifier Workflow

4. Multicollinearity can cause issues with Neural Networks, so Variance Inflation Factor VIF [52] is used to determine if these features are too highly linearly related, and if so, removes the feature most linearly related. In the paper “Exploring the Coding Behavior of Successful Students in Programming by Employment Neo-Piagetian Theory” in Section 6.2 we used a max VIF of 5, but for HOG we set the max VIF to 10, as is acceptable according to other research [53].
5. The data from the remaining features are then divided into training and testing data, where testing data is from 50 participants, and the remaining are training data. The testing data is randomly selected, but the training and testing only takes place if the test data has an even number of pass and fail participants (25 pass and 25 fail), to avoid reporting of a lucky/unlucky data set.
6. The Neural Network is trained on the training data, and then tested on the test data at least twenty times, with different divisions of training and testing data. Twenty times was chosen due to the work in the paper in Section 5.2., which found that there was little difference in the variance of the results of the classifiers after running ten times and after running 60 times.
7. This process is carried out for each of the ten data sets for each experiment. Every one of these data sets also includes the data of the previous lab sessions, as an example, the experiment for Week 4 also uses the data from Week 1, 2 and 3.
8. For each experiment, the most successful classifier is selected from the different thresholds, for both Continuous Assessment and Written Exam groups. The most successful classifiers are selected according to
 - i. Highest Accuracy
 - ii. Lowest False PassesFalse passes are when the classifier classifies a student as being in the top 50% but is in the bottom 50%.

4.5 Overview of Experiments

In this section, each of the four experiments in this thesis will be summarised.

4.5.1 Experiment 1: Mouse Movements (MM)

The first section of data explored is the Mouse Movements of students as they learn to code. The data recorded was used to generate metrics on the students' behaviour in short mouse movement sequences during the students' scheduled lab sessions, including lab exams. These metrics recorded attributes such as speed, time to click the mouse button, and distance travelled. This experiment is described in Chapter 5.

4.5.2 Experiment 2: Compile, Run, Evaluate Movements (CRE)

The second section of data explored is the compile, run, and evaluate patterns of students as they learn to code. The metrics used in this section are "movements" a student takes from a compile to a run, or a run to an evaluate. These movements are recorded as a percentage of the movements a student took per lab and are used to explore the different behaviour of the higher and lower achieving students. This experiment is described in Chapter 6.

4.5.3 Experiment 3: Code Complexity (COMPLEX)

The third section of data explored is the complexity of code submitted by students for evaluation. The complexity is measured first by removing the comments in the code, compressing the files, and recording the size of the result, and secondly by generating parse trees for the code and recording the number of nodes in the parse tree. This experiment is described in Chapter 7.

4.5.4 Final Experiment: HOG

In the final experiment, classifiers are built using all of the data from the previous three chapters to test if a combination of data produces a more successful classifier than any of the first three experiments. This experiment is described in Chapter 8.

4.6 Chapter Summary

In this chapter, the data sets gathered and used in this thesis were described, the analysis methods were outlined, and the HOG classifier was described. The four experiments that will follow in Chapters 5,6,7, and 8 were summarised.

5. Experiment 1: Mouse Movements

In this section, the MM data gathered by the MULE system will be explored using the methods outlined in Section 4.3 and 4.4. The first exploration into the MM data is described in the published paper in Section 5.2, and the subsequent study using the HOG classifier system is described in Sections 5.3, 5.4, and 5.5.

5.1 Introduction to Mouse Movement Experiment

There is existing work to suggest certain mouse movement behaviour is correlated to stress and mood [36, 37, 52], and that students' experience of stress [53] or comfort level [5, 54] can be an indication of performance. Using mouse movements to detect students in need of intervention has potential as a non-invasive method for detecting students in danger of failing their Introduction to Programming module.

The first experiment on the MM data was run using data set 1 (as described in Section 4.1.1). This data is from the second semester of the academic year 2018/2019, and the published paper from this experiment is in Section 5.2. [57]. In this paper, a classifier was built to classify sequences of MM as being from stressful (in formal lab exam sessions) or less stressful (in regular, less formal weekly lab sessions) environments. The resulting classifier was moderately successful, with an accuracy of 62.9%. Interestingly, the classifier worked better on students who did poorly in their exams, so a second classifier was built to use MM to classify students as passing or failing the module. This classifier was more successful, with an accuracy of 69% in predicting CA grades. This experiment was a success in that it found a connection between MM data and stressful lab situations (lab exams), and between MM and student outcome, but the resulting classifiers could not be used to find students in danger of failing early in the first semester.

The focus of this thesis is to investigate the behaviour of novice programmers, so the experiment in Section 5.3 to Section 5.5 uses the data set 2 from semester 1 in the year 2019/2020, and the HOG classifier described in Section 4.4. This experiment differed from the one described in the paper in Section 5.2, in that it ran on weekly data sets, not on the full semester data set, because the goal was to find early indicators of student outcome. This experiment was less successful than the first experiment in Section 5.2 and did not achieve classifier results of higher than

57%, nor did the Wilcoxon Rank Sum Test find a consistent significant difference in any feature throughout the semester.

5.2 Paper: What the Mouse Said: How Mouse Movements Can Relate to Student Stress and Success

What the Mouse Said:

How Mouse Movements Can Relate to Student Stress and Success

Natalie Culligan

Department of Computer Science
Maynooth University
natalie.culligan@mu.ie

Kevin Casey

Department of Computer Science
Maynooth University
kevin.casey@mu.ie

Abstract

Stress in students may be a useful indication for when a student is struggling and in need of academic intervention. Investigating differences in student behaviour in stressful and comparatively less stressful environments could be helpful in understanding the processes involved in learning to code, and combatting the high levels of drop-out and failure in undergraduate computer science. In this paper we will discuss the mouse movement data gathered from Maynooth University Learning Environment (MULE), our in-house, browser-based pedagogical environment for novice programmers, during the time period February to May of 2019. This included 5 supervised, scheduled lab sessions and two in-lab examinations. The data was used to examine 21 different measurements of student behaviour, for example, by measuring efficiency of the mouse path, or the time between mouse click-down and mouse click-up. These features were used to build a Deep Neural Net that classifies sequences of mouse movements as being either from a more stressful environment or a less stressful one by training the classifier on data from examination situations and regular weekly lab situations, with the goal of comparing how students behave in environments with different levels of student comfort. The classifiers had an average accuracy of 61.9% but was more successful with students who performed poorly in their lab examinations. To further examine this connection between mouse movement, stress and student outcome, a second classifier was built to classify students as being in the high or low 50% of lab-exam grades in the module, with an accuracy of 69%.

1. Introduction

In this study we use data collected by Maynooth University Learning Environment, or MULE (Culligan, Casey 2018). MULE is an online, browser-based pedagogical desktop environment which has been used in multiple first-year coding modules. We received clearance from the University Ethics Committee to collect mouse movements from students as they learn to code from the 29th of February until the 3rd of May in the Introduction to Programming II module (taught in Java) with 250 students completing the module. The students were informed about the use of their data and were asked to consent at the beginning of the semester. All students who completed the module chose to participate in the study.

Using the mouse movement data collected by MULE, a Deep Neural Net (DNN) binary classifier was built to detect if a sequence of mouse movements is from a stressful (in-examination) or less stressful environment (in-lab). The classifier is not universal. It needs to be trained on a student's own data and does not work on all students. This was expected, as stress and comfort are subjective and not all students will experience stress in the same way during an examination. Students may also have different mouse use "styles", which makes it harder to generalise mouse behaviour caused by stress. We must also consider that some students are not stressed during an examination and may even be less stressed

than in a normal lab situation.

The classifier works very well for some students and poorly for others, with an average increase of 11%-12% over the accuracy baseline of 50%, an average accuracy of 61.9% for classifying both in-lab and in-examination sequences. The classifier was moderately successful but interestingly the classifier was more successful for students who did poorly in the module. To further explore this, we built a second DNN to investigate if the mouse movement data could be used to classify students as being in the top or bottom 50% of module grades. This classifier was more successful than the stress classifier, classifying students as being in the top or bottom 50% of the module Continuous Assessment grades with an accuracy of 69%, over an accuracy baseline of 50%.

In this paper, the following questions will be explored in relation to the gathered mouse movement data.

1. *Are there differences in mouse movement behaviour of students between lab and exam situations, and can this be a first step in a classifier for stressed students?*
2. *Are there differences in student mouse behaviour and stress in students in CSI between students who perform well in-lab examinations and written exams, and those who perform poorly?*

The null hypothesis for these questions are as follows:

1. *The results from the Deep Neural Net for classifying sequences of mouse movements sequences as being from stressful or not stressful environments performed no better, or not significantly better than random chance.*
2. *The results from the Deep Neural Net for classifying individuals as being in the top or bottom performing 50% of students performed no better, or not significantly better than random chance.*

2. Motivation and Related Research

Stress in students may be a useful indication for when a student is struggling and in need of academic intervention. Intervention for students experiencing unusual amounts of stress could be helpful in combatting the high levels of drop out and failure in undergraduate computer science (Beaubouef et al, Biggers et al, Giannakos et al, Hembree et al, Kinnunen et al). This is the first of our studies into student behaviour as they learn to code, and in this study we focus on mouse movement. There are studies that suggest that mouse movement is linked to stress and mood (Sun *et al.*, Wahlström, *et al.*, Yamauchi). In this paper, we are interested in examining student mouse movement in stressful and less stressful environments to try and gain insight into behaviours that indicate stress, and investigate if this is related to student performance.

2.1. Stress Levels in Students

Computer science courses have been reported to have low levels of retention in comparison to other subjects (Giannakos *et al.*, Kinnunen, *et al.*). Research suggests that student comfort is a useful signifier of student success and retention (McCracken *et al.*, Tenenber, *et al.*, Wilson and Shrock), and that stressful situations such as examinations can cause a student to perform below their ability (Beilock and Carr).

Beilock and Carr discuss the connection between anxiety and a loss in academic performance, and suggest that situation-related worries – such as examination stress or anxiety – can result in a loss of focus on task at hand as the

working memory is occupied. Alternatively, it has also been suggested that over-attending to performance, overthinking tasks usually performed automatically, can lead to underperforming in an uncomfortable or stressful situation. Beilock *et al.* discuss how a more stressful or anxious state can also affect tasks that are usually performed in an automated fashion, without the subject thinking too much about it – their paper mentions soccer players’ dribbling. We propose that mouse movement could be considered in a similar manner.

Connolly *et al.* found that in their study of 86 computing undergraduate students, 44.4% reported not feeling relaxed when using computers, suggesting that research into this area would be beneficial to a significant portion of the student population.

Bergin and Reilly examined 15 factors in predicting if a student is likely to pass or fail. One of the most statistically significant factors in predicting success was comfort level, in relation to how the student felt about the course. This was measured through cumulative responses to questions about the students’ understanding and difficulty completing lab assignments.

2.2. Mouse Movement and Stress

There is prior evidence of a link between student stress and comfort level and their mouse movements. Sun *et al.* constructed a Mass Spring Damper model for the human arm - essentially a model for approximating arm motion and stiffness which could be fed with data from mouse movements. Using arm stiffness as a proxy for stress in the user, the authors report that their method was tested across a variety of prescribed stress tasks. The classifier worked when generalised but was more effective when trained and tested separately for each user.

Yamauchi claims there is both psychological and neurological evidence to suggest that mouse trajectories can be used to assess affective states, such as anxiety. The results of their study show that temporal features, such as speed of mouse movement, and spatial features such as direction change were both indicative of the user’s state of anxiety. The researchers in this paper ran a separate analysis for male and female users and found different indications of state anxiety, with female subjects being more inclined to use a less efficient mouse path when anxious, and male subjects being more likely to change their mouse velocity.

Kapoor *et al.* use a specialised pressure mouse with additional sensors to detect frustration in subjects as they attempt to complete a towers of Hanoi puzzle computer game. The game includes an “I’m frustrated” button for the users, which is used to associate behaviour with frustrated state. The resulting classifier can predict frustration at an accuracy of 79%, outperforming the random classifier (58%).

3. Research Design

The goal of this study was to examine the relationship between student mouse behaviour, student outcome, and comfort level in students in CS1, an introduction to programming module. Using the data from MULE, we constructed a Deep Neural Net binary classifier to classify sequences of mouse movements as being from a stressful environment or a less stressful one.

MULE was used to collect mouse movement data from students as they learned to code in an authentic learning environment. To use the system, the students sign in through their Moodle accounts from any internet browser on any machine, they do not need to be in the university computer labs. The system is a

desktop-like environment simulated within the browser, where they can view assignments from a designated application, use a text editor to write code for the assignments, and compile, run and automatically evaluate their code, receiving a grade and automated feedback if their code has errors. The students use the mouse to navigate the system, to open assignments, open the code editor, and to save, compile, run and evaluate from drop down menus. As the student works, the system automatically stores their mouse movements, along with a timestamp and an anonymised user key to allow for cross session comparisons. Stored mouse movements are sent to the database every 30 seconds, or as soon as the user tries to log out or close the system tab. The system collects mouse movement data as shown in Table 1. Anonymised data on students’ performance in the module was also collected, specifically how they performed in the written examination, in weekly labs and in-lab examinations. The total number of students who completed the second semester was 250, of which 196 are included in this study. We removed data from students who did not participate enough for their data to be used in the study, including:

1. *Students who did not take both in-lab examinations*
2. *Students who did not complete the course*
3. *Students who participated in less than two lab sessions*

Data Type	Description
userID	The anonymous ID assigned to the student
dumpID	The ID of the dump from student session to the database
sessionID	An ID assigned to the session when a student logs in until they log out
Time	Timestamp of when the event took place, not when it was stored
Type	Mousemove, mouseup or mousedown
X	X co-ordinates of the mouse’s current position
y	Y co-ordinates of the mouse’s current position

Table 1: Mouse movement data features

Students have labs for 3 hours once a week for 12 weeks per semester. The students began using the system in the first semester of the academic year 2018/2019 and used the system for the rest of the academic year. The mouse movement data set we are examining in this paper is from the second semester, from the 29th of February until the 3rd of May. This time period includes 5 regular weekly labs and 2 in-lab examinations. We compare mouse data from students in a regular lab situation versus mouse data from an examination situation, to examine the differences between coding when in situations with different levels of comfort. Both situations are in the same physical space, but with different rules. The students are not allowed to speak to each other, ask for help from demonstrators or look back at their previous work during the examination situation, but are encouraged to do so during regular labs. One of the authors worked as a demonstrator in the labs where this research took place to ensure the coding environment was working correctly, and to assist the students.

We recorded mouse data from students as they worked in scheduled labs, scheduled examinations, and outside of these times. The data from outside of the lab is not discussed in this paper. Data outside scheduled labs and examinations may be the result of users other than the signed-in student and/or very different mouse set up (touch screen, touch pad, or different desk size, for example). Students may also be working in very different situations due to environmental

noise, distractions, or caretaking responsibilities, for example.

The mouse data from each student is divided into sequences to be assessed by the classifier. Each sequence begins with any mouse movement and ends with a mouse click-up, and any sequence that is longer than 1450ms is rejected to avoid evaluating sequences from when the student is idle. This time limit was chosen through trial and error, and found the classifier worked best with sequences under this time limit. Tests are run on each sequence to find various metrics for the users' behaviours. Metrics include SequenceSpeed, ClickTime and Efficiency. Each sequence also has an identifier, as in-lab, in-examination or out-lab. Once we have the metrics for each of the sequences, they are used to train and test the Deep Neural Net.

We used a total of 21 different features in our classifier.

Features

1. *AngleVariance1:*

Finds all the different angle changes from one movement to the next (with precision of 2 digits) within a sequence and returns the total number of unique angles.

2. *AngleVariance2*

Same as above, but the total number of angles returned.

3. *AngleVariance3*

The ratio of total angles to unique angles.

4. *VarianceDistance1*

Finds the optimal distance between every set of two mouse movements to 1 decimal place and returns the number of all unique distances.

5. *VarianceDistance2*

Same as above but returns the number of all distances.

6. *VarianceDistance3*

The ratio of all unique distances and all distances in the sequence.

7. *Overshoot-x*

Measures how far a user "overshoots" with the mouse in the direction they are moving the mouse in, along the X axis. If a user moves from point a to point b within a small window of time, point b being where they click the mouse, if at some point during this journey they move further along the x-axis than where they ended, this is recorded as an *Overshoot-x*.

8. *Overshoot-y*

Same as *Overshoot-x*, but along the y axis.

9. *Overshoot*

The square root of *Overshoot-x* and *Overshoot-y* squared and added.

10. *OvershootDirectionAngle*

Finds the angle of the overshoot.

11. *SequenceSpeed*

The total distance travelled divided by the total time.

12. *SequenceDuration*

The time duration of the sequence.

13. *DistanceTravelled*

The true distance travelled during the sequence.

14. *OptimalDistance*

The distance in a straight line between the start and end points of the sequence.

15. *Efficiency*

Optimal distance divided by total distance travelled.

16. *Direction*

The direction from the first point in the sequence to the last.

17. *DirectionAngle*

The direction angle between the starting point and the ending point of the sequence.

18. *AngleDifference*

The absolute value of *DirectionAngle* subtracted from *OvershootDirectionAngle*.

19. *ClickTime*

The time between click down and click up.

20. *Hesitate*

The amount of time the mouse stalls before the user clicks.

21. *ClickRatio*

This is *Hesitate* divided by *ClickTime*

Yamauchi's paper '*Mouse Trajectories and State Anxiety: Feature Selection with Random Forest*' found that speed and direction were indicators of a subject's emotional state. Our features are chosen to examine this connection, with features such as *DistanceTravelled* and *ClickTime* relating to speed, and *DirectionAngle* and *OvershootDirectionAngle* relating to direction. The paper also discusses tracking direction change, x-overshoot, y-overshoot, which we replicated in our experiment with features such as *Overshoot-x*, *Overshoot*, *DirectionAngle* and *DirectionAngle*. Beilock et al discuss how a more stressful or anxious state can also affect tasks that are usually preformed in an automated fashion. We investigated this with the features *VarianceDistance1*, *VarianceDistance2*, *VarianceDistance3*, to give us insight into how much the subject changed their speed, and the features *AngleVariance1*, *AngleVariance2* and *AngleVariance3* to investigate how often the subject changed direction, perhaps due to confusion or indecisiveness as a result of stress or discomfort.

As per Sun *et al.*, we trained our classifier per user, instead of building a generalised stress classifier. Our initial experiments involved a general classifier using a large subsection of the data from all students, but this classifier did not perform significantly better than random chance. To build a classifier for a user, we selected all the sequences from in-examination, and then a random selection of sequences of an equal amount from in-lab, or vice-versa, depending on the imbalance of data categorised as in-lab or in-examination. The features we get from the mouse movements of each student are then used to train and test a deep neural net, built in Python using TensorFlow (Abadi, Martín, *et al.*).

For most students, we have much more in-lab data than in-examination, so we take a random sample of the in-lab data equal to the size of the in-examination data. We used TensorFlow's *DNNclassifier* module, with 3 hidden layers of 10 units, a batch size of 5 and 2000 epochs. The classifier outputs a 1 if the mouse movement sequence is classified as in-lab and 0 if the sequence is classified as in-exam. When running the classifier for each student, we wanted to ensure that the results were not due to chance, or a "lucky" selection of test data from the total data set. To combat this, we selected a subsection of the data as test data, and rejected it if it was not 50/50 in-lab and in-exam, again to avoid good results that are just the result of a classifier only choosing one classification, regardless of feature input. To check that the variance for the classifier results was low, and we were not reporting outliers, the classifiers were run in sections of ten, and the variance within results was checked. The variance for all users was 0.05 or less, with one exception that had a larger variance of 0.13. We performed multiple sets of ten, checking the variance on the cumulative results. For each student, the classifier was run 60 times, with a different random division of

training and test data with no increase in variance over 0.016 between the first 10 and the final 60.

4. Discussion of Classifier Performance

The classifier works very well for some students and poorly for others, with an average increase of 11% to 12% over the accuracy baseline and an average accuracy of 62.9% for classifying both in-lab and in-examination sequences. However, for some students that performed poorly in their lab examinations, we found the classifier could work 30% over baseline. On examination of the results, it became apparent that the classifier was more successful with the students who performed poorly in the module than those who performed well. One of the possible reasons for student stress during exams is that they may be unable to use their usual method of solving coding problems. Some students will take previously written code, copy it and rewrite it to complete the given task. During exams the students no longer have access to their previous code. They may panic when they find they cannot use their usual strategy (though they are informed beforehand of the format and rules of the exam), or they may be experiencing additional strain on their working memory. This strain may come from the extra work now being performed by the student. For example, they can't copy a while loop from previous work, so instead they struggle to remember how to write one. The student is not comfortable and familiar with the computer science concepts needed to construct the code to solve the exam question and has been relying on 'tinkering', a technique used by students as described by Perkins *et al.* and Jadud.

4.1. Stress Classifier

When examining the results of the classifiers, differences between the high-performing and low-performing students became apparent. Table 2 shows the average classifier of two groups, the top 50% of grades and bottom 50% of grades. This was done for Continuous Assessment, written exam and total module grade, and repeated with the top and bottom 40%, 30%, 20% and 10%.

	Module High	Module Low	Written Exam High	Written Exam Low	CA High	CA Low
50%	61.875%	62.7913%	61.3518%	62.627%	60.8315%	63.1473%
40%	60.8037%	62.6183%	61.1019%	61.949%	60.7157%	63.8293%
30%	60.1556%	62.6878%	60.9173%	62.6955%	59.7906%	63.9333%
20%	60.0027%	62.5255%	59.8544%	62.8666%	59.7657%	63.4562%
10%	58.8089%	62.8847%	58.2153%	62.643%	59.4642%	65.3041%

Table 2: Comparison of the high and low performing students

In all groups, and with all three grade types, the lower grades group have more successful classifiers, with the difference becoming more pronounced as we look at smaller subsections. We suspect that the reason students in the lower-grade groups are easier to classify is because these students may experience additional strain when writing code, perhaps due to exam anxiety, or a lack of comfort with the material. In the paper "*On the causal mechanisms of stereotype threat: Can skills that don't rely heavily on working memory still be threatened?*", Beilock, *et al.* claim that while overloaded working memory does not directly affect procedural skills because it is not reliant on working memory, over-attention to procedural skills does impact the subject's performance – a

worried student may overthink their behaviour, causing changes in their mouse movement.

4.2. High Low Grade Classifier

We were interested in the possible connection between mouse movements and student grades, from the apparent relation between classifier success and the students' performance in the module shown in Table 2. We suspected that the results indicated a relation between mouse movements, specifically indications of stress in exams, and student grades. There is previous work (Casey) to suggest that low-level keystroke data can be used to improve grade classifiers, so we wanted to examine if mouse movement data could also be used. To investigate this, a trio of DNN classifiers were created to predict the outcome of students in:

1. *Continuous Assessment (coding exercises, and lab exams),*
2. *End of year written exams*
3. *The module overall.*

The DNN uses the same configuration as the stress classifier. We tried other configurations, including increasing the number of hidden units, but found this was the most successful setting. The DNN classifies each student into one of two categories – either the higher or lower 50% of the class, divided by the results in order. For this dataset we calculated the average of each of the features in the table for in-lab and out-lab. We found this gave the best results, possibly because the indicator of a student who does well or poorly is the difference, or the similarity of the behaviour between regular labs and exams, in line with the findings that the students who did poorly were more easily classed by the classifier.

Grade	Higher 50%	Lower 50%	Classifier Results
Written Exam	63% and over	61% and under	0.588333333
Module Total	59% and over	58% and under	0.656666667
Continuous Assessment	54% and over	53% and under	0.693333333

Table 3: Results of classification

Like the previous classifier, the high/low classifier was run 60 times, each time randomly selecting the training set and the testing set. Like the stress classifier, the randomisation was written to ensure that the testing data set would always be 50% from each classification, to avoid misleadingly high or low results from a classifier only choosing one classification.

5. Discussion of Research Questions

1. *Are there differences in mouse movement behaviour of CS1 students between lab and exam situations, and can this provide insight to the different comfort levels experienced by students in these environments?*

The DNN classifier was mildly successful, implying that there is at least a weak link between mouse movement and comfort level. Students may still be stressed in lab situations, but because the classifier was more successful with students who did poorly in their lab examinations, we believe this is evidence that the classifier works as an indicator of stress – we believe that students who are taking examinations that they are not doing well in are more likely to be experiencing stress than others. We can reject the null hypothesis, as the classifier is more successful than a random chance classifier.

2. Are there differences in student mouse behaviour and comfort level in students in CSI between students who perform well in-lab examinations and written exams, and those who perform poorly?

The classifier is more effective with students who perform poorly than those who perform well. We would expect students who do poorly in the module to be more stressed in examinations than students who are comfortable with the material and are performing well. To examine this further we built a second DNN classifier and found that we were able to classify students into high/low performing groups with 69% accuracy. We reject the null hypothesis as the high/low classifier is more successful than random chance.

6. Conclusions and Future Research

In this paper, we have reported on the construction of a moderately successful Deep Neural Net that classifies sequences of mouse movements as being from a stressful or less stressful environment. While other researchers have published work on the connection between mouse movements and stress, to our

knowledge this is the only study of mouse movements and stress that uses mouse data gathered outside of closed experimental environments. From the analysis of the results, we found a connection between mouse movements and a student's grades, especially grades for practical coding assignments.

The classifiers in their current state are not a useful mechanism for detecting stress in students, or for predicting if students will be in the high or low 50% of grades. However, in the construction of these classifiers, we have found mechanisms that will contribute to the construction of models of successful students, and classifiers for students in need of academic intervention. This study is part of a larger project to examine the relationship between student behaviour when learning to code and student success and retention. Our coding environment gathers data beyond mouse movement, including keystrokes, compilation and run results, and returned errors. Other research in this area has used data such as keystrokes to predict student outcome (Casey), and from our work in this paper, which suggests a connection between student success and comfort-level, we believe this data will give further insight to student behaviour in stressful situations. Further work can be done in relation to the mouse analytics performed so far. We are currently refactoring our recording of mouse data so that we can capture additional data in order to attach more meaning to mouse sequences. This would, for example, allow us to distinguish between a mouse sequence that led to a file being saved, versus a mouse sequence that led to a compilation of student code.

We believe there is huge potential for study of this data, which is gathered from an authentic learning environment, as students learn to code. With continued research, we plan to build a larger model of the behaviour of novice programmers as they learn to code, with the potential for an integrated classifier in our coding environment that will alert course coordinators to a student in need of intervention. We hope the construction of a model of successful students will be a useful way to inform and build curriculums that best help students achieve their potential.

7. References

Abadi, Martín, et al. (2016) Tensorflow: A system for large-scale machine learning. 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16).

Beaubouef, Theresa, and John Mason. (2005) Why the high attrition rate for computer science students: some thoughts and observations. ACM SIGCSE Bulletin 37.2 103-106. DOI: <https://doi.org/10.1145/1083431.1083474>

Beilock, Sian L., and Thomas H. Carr. (2005) When high-powered people fail: Working memory and “choking under pressure” in math. Psychological science 16.2 01-105. DOI: <https://doi.org/10.1037/e537052012-380>

Beilock, Sian L., et al. (2006) On the causal mechanisms of stereotype threat: Can skills that don't rely heavily on working memory still be threatened?. Personality and Social Psychology Bulletin 32.8 1059-1071. DOI: <https://doi.org/10.1177/0146167206288489>

Bergin, Susan, and Ronan Reilly. (2005) Programming: factors that influence success. ACM Sigcse Bulletin 37.1 411-415. DOI: <https://doi.org/10.1145/1047344.1047480>

Biggers, Maureen, Anne Brauer, and Tuba Yilmaz. (2008) Student perceptions of computer science: a retention study comparing graduating seniors with cs leavers. ACM SIGCSE Bulletin. Vol. 40. No. 1. ACM. DOI: <https://doi.org/10.1145/1352135.1352274>

Casey, Kevin. (2017) Using keystroke analytics to improve pass-fail classifiers. Journal of Learning Analytics 4.2 189-211. DOI: <https://doi.org/10.18608/jla.2017.42.14>

Connolly, Cornelia, Eamonn Murphy, and Sarah Moore. (2008) Programming Anxiety Amongst Computing Students—A Key in the Retention Debate?. IEEE Transactions on Education 52.1 52-56. DOI: <https://doi.org/10.1109/te.2008.917193>

Culligan, N., & Casey, K. (2018). Building an Authentic Novice Programming Lab Environment. Irish Conference On Engaging Pedagogy

Giannakos, Michail N., *et al.* (2017) Understanding student retention in computer science education: The role of environment, gains, barriers and usefulness. Education and Information Technologies 22.5 2365-2382. DOI: <https://doi.org/10.1007/s10639-016-9538-1>

Hembree, Ray. The nature, effects, and relief of mathematics anxiety. Journal for research in mathematics education (1990): 33-46. DOI: <https://doi.org/10.2307/749455>

Kapoor, Ashish, Winslow Burleson, and Rosalind W. Picard. (2007) Automatic prediction of frustration. International journal of human-computer studies 65.8 724-736. DOI: <https://doi.org/10.1016/j.ijhcs.2007.02.003>

Jadud, M. C. (2006). An exploration of novice compilation behaviour in BlueJ (Doctoral dissertation, University of Kent). DOI: <https://doi.org/10.1080/08993400500056530>

Kinnunen, Päivi, and Lauri Malmi. (2006) Why students drop out CS1 course?. Proceedings of the second international workshop on Computing education research. ACM. DOI: <https://doi.org/10.1145/1151588.1151604>

Lister, Raymond. Concrete and other neo-Piagetian forms of reasoning in the novice programmer. Proceedings of the Thirteenth Australasian Computing Education Conference-Volume 114. Australian Computer Society, Inc., 2011. DOI: <https://doi.org/10.1215/9780822381525-005>

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B. D., ... & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In Working group reports from ITiCSE on Innovation and technology in computer science education (pp. 125-180). ACM. DOI: <https://doi.org/10.1145/572139.572181>

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of learning in novice programmers. *Journal of Educational Computing Research*, 2(1), 37-55. DOI: <https://doi.org/10.2190/gujt-jcbj-q6qu-q9pl>

Sun, D., Paredes, P., & Canny, J. (2014, April). MouStress: detecting stress from mouse motion. In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 61-70). ACM. DOI: <https://doi.org/10.1145/2556288.2557243>

Tenenberg, Josh D., *et al.* (2005) Students Designing Software: a Multi-National, Multi-Institutional Study. *Informatics in Education* 4.1 143-162.

Wahlström, J., *et al.* (2002) Influence of time pressure and verbal provocation on physiological and psychological reactions during work with a computer mouse. *European journal of applied physiology* 87.3 257-263. DOI: <https://doi.org/10.1007/s00421-002-0611-7>

Wilson, B. C., & Shrock, S. (2001). Contributing to success in an introductory computer science course: a study of twelve factors. *Acm sigcse bulletin*, 33(1), 184-188. DOI: <https://doi.org/10.1145/364447.364581>

Yamauchi, Takashi. (2013) Mouse trajectories and state anxiety: feature selection with random forest. 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction. IEEE, 2013. DOI: <https://doi.org/10.1109/acii.2013.72>

In this section, the pilot study using the MM data from data set 1 is reprinted. The rest of this chapter describes the study with this MM data using the data set 2, which is the same data set used in the experiments in Chapter 6,7, and 8.

5.3 Mouse Movement Features

The data set used in the following analysis and classifiers are the MM data from data set 2, described in Section 4.1.2. The data sets consist of ten sets of data for each student, one for each of the weekly labs. For each student, the data set contains the average of each of the metrics described below for that week. This differs from the original paper in Section 5.2, in that:

- 1) The data from the paper is from the second semester of the academic year 2018/2019, so the students in this data set have some university experience. The data from this section is from the first semester of the academic year 2019/2020, so the students are novice programmers.
- 2) The metrics used in the original paper are: 1) the overall average of non-exam lab sessions and 2) the overall average of exam sessions. The metrics from this section are the averages from each of the ten weekly lab sessions.

The MM are used to generate various metrics about the user's behaviour. The full list of metrics can be seen in Section 5.2:3 Research Design. Some of the more relevant metrics, according to the tests in Section 5.4, are as follows:

1. *Overshoot-x*

This metric measures how far a user "overshoots" with the mouse in the direction they are moving the mouse in, along the X axis. If a user moves from point A to point B within a small window of time, point B being where they click the mouse. If at some point during this journey they move further along the x-axis than where they ended, this is recorded as an *Overshoot-x*.

2. *Overshoot-y*

This metric is the same as *Overshoot-x*, but along the y-axis.

3. *Overshoot*

This metric is the square root of *Overshoot-x* and *Overshoot-y* squared and added.

4. *OvershootDirectionAngle*

This metric is the angle of the overshoot.

5. *SequenceSpeed*

This metric measures the total distance travelled divided by the total time.

6. *OptimalDistance*

This metric is the total distance in a straight line between the start and end points of the sequence.

7. *Efficiency*

This metric measures the optimal distance divided by total distance travelled.

8. *Direction*

This metric is the direction from the first point in the sequence to the last.

9. *ClickTime*

This metric is the time between click down and click up.

These metrics are calculated from a mouse movement sequence, which is any sequence of mouse movements that ends with a mouse click-up (when the mouse button is released) and is shorter than 1450ms. The time 1450ms was chosen through trial and error in the paper in Section 5.2. Unlike the stress classifier from the paper in Section 5.2, the goal of this experiment is not to classify individual sequences from throughout the entire semester, but to classify students, on a week-to-week basis. So, like in the pass/fail classifier in the paper in Section 5.2, the metrics for the classifier are the average of all the sequences from a single lab, and a single student. This results in 21 input features for each lab for the Neural Network classifier, a total of 210 input features.

5.4 Mouse Movements Analysis and Neural Network

Each of the three data types in this project, as presented in Section 4.2, are examined using the same HOG classifier methodology. However, when this methodology was used on the MM data, there was a problem - the Linear Regression tests returned very few positive results, implying the MM and student outcome were not correlated. It may be that the success of the classifier in the published paper was due to the difference in behaviour in normal labs and in exam labs, or that there is a larger difference in the MM data between the two student groups in the second semester than in the first semester, perhaps due to increased stress.

The goal of this analysis was to examine when the student's behaviour begins to diverge between the successful and unsuccessful students on a week-to-week basis, and so the original method of processing data does not work for these purposes. To compensate for this, the threshold for correlation was set to -0.1, as

this was the highest threshold that would allow for enough data for the weekly Neural Networks to run.

5.4.1 Wilcoxon Rank Sum Test

In this section, the most relevant results of the Wilcoxon Ranks Sum Test are presented. The metrics are described in Section 5.3, and the full tables of results can be seen in the Appendix Section 10.3. As described in Section 4.3.1, a result of less than 0.05 is considered significant and is in bold. The tests comparing the top 50% and the bottom 50% of the grades for CA are shown in Tables 5-1 and 5-2 for the most significant features.

5.4.1.1 CA

In this section, the results of the Wilcoxon Rank Sum Test for the MM data, comparing the top and bottom 50% of the class according to the CA grades, are presented. In Table 5-1, we see that the relevant features for the first four weeks are DIRECTION, OPTIMAL_DISTANCE, CLICKTIME, HESITATE and CLICKRATIO. Each of these features is only relevant for one of these four weeks, so it may be possible the results are due to chance. The full tables can be seen in the Appendix Section 10.3.1. in Table 10-1, Table 10-2, and Table 10-3.

<i>Lab</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>
<i>DIRECTION</i>	0.029588	0.212553	0.196411	0.017461
<i>OPTIMAL_DISTANCE</i>	0.784908	0.942278	0.929454	0.012592
<i>CLICKTIME</i>	0.280768	0.028114	0.560714	0.626366
<i>HESITATE</i>	0.114143	0.023804	0.073371	0.823879
<i>CLICKRATIO</i>	0.215464	0.003706	0.078136	0.9681

Table 5-1: MM Wilcoxon Rank Sum Test CA for Week 1 to 4

In Table 5-2, there are 14 relevant features, an increase from the five in the four weeks shown in Table 5-1. Some of the features in Table 5-1 are also present in Table 5-2: OPTIMAL_DISTANCE, CLICKTIME, and CLICKRATIO. Week 7 has a total of 10 significant features, implying that by Week 7 there is some divergence in student MM behaviour between the two groups.

The Table 5-3 shows the significant features from Week 8 to Week 10. None of the features present in both Tables 5-1 and 5-2 are significant in this table. However, we do see a large number of relevant features in Week 8, a total of six.

<i>Lab</i>	5	6	7
<i>EFFICIENCY</i>	0.015108	0.740481	0.009588
<i>DIRECTIONANGLE</i>	0.687177	0.03941	0.972124
<i>OVERSHOOTDIRECTIONANGLE</i>	0.961894	0.000429	0.000973
<i>OVERSHOOTY</i>	0.509035	0.292012	0.018673
<i>OVERSHOOT</i>	0.51013	0.285189	0.016298
<i>SEQUENCE_DURATION</i>	0.357729	0.023814	0.769529
<i>DIRECTION</i>	0.652375	0.265013	0.019592
<i>OPTIMAL_DISTANCE</i>	0.012978	0.271141	0.192956
<i>VARIANCE1</i>	0.019584	0.555292	0.033115
<i>VARIANCE2</i>	0.354173	0.267524	0.06174
<i>VARIANCEDIST1</i>	0.025512	0.795275	0.019036
<i>VARIANCEDIST3</i>	0.048165	0.168897	0.972124
<i>CLICKTIME</i>	0.594471	0.13252	0.01131
<i>CLICKRATIO</i>	0.164852	0.383295	0.016139

Table 5-2: MM Wilcox Rank Sum Test CA for Week 5 to 7

<i>Lab</i>	8	9	10
<i>EFFICIENCY</i>	0.01516	0.031693	0.141307
<i>OVERSHOOTX</i>	0.004262	0.113071	0.221358
<i>SEQUENCE_SPEED</i>	0.013254	0.109021	0.10326
<i>OVERSHOOTDIRECTIONANGLE</i>	0.003856	0.746383	0.004045
<i>OVERSHOOTY</i>	0.151661	0.029468	0.821522
<i>OVERSHOOT</i>	0.146422	0.028147	0.844168
<i>ANGLEDIFFERENCE</i>	0.020005	0.045101	0.160813
<i>DIRECTION</i>	0.067285	0.743648	0.043685
<i>VARIANCE1</i>	0.023864	0.991351	0.234068

Table 5-3: MM Wilcox Rank Sum Test CA for Week 8 to 10

The results of the Wilcox Rank Sum Test in this section do not show any clear divergence in MM behaviour in any one feature consistently throughout the semester. However, there are some interesting results, such as the high number of significant features in Week 7.

5.4.1.2 Written Exam

In this section, the results of the Wilcoxon Rank Sum Test for the MM data, comparing the top and bottom 50% of the class according to the Written Exam grades. The full tables can be seen in the Appendix Section 10.3.2 in Table 10-4, Table 10-5, and Table 10-6.

<i>Lab</i>	<i>1</i>	<i>2</i>	<i>3</i>
<i>DIRECTION</i>	0.03914	0.533739	0.310773
<i>CLICKRATIO</i>	0.475342	0.024542	0.150893

Table 5-4: MM Wilcoxon Rank Sum Test Written Exam for Week 1 to 3

In Table 5-4, we can see that there are only two significant features for Week 1 to Week 3: *DIRECTION* and *CLICKRATIO*.

<i>Lab</i>	<i>4</i>	<i>5</i>	<i>6</i>
<i>OVERSHOOTDIRECTIONANGLE</i>	0.610436	0.937438	0.01481
<i>SEQUENCE_DURATION</i>	0.628833	0.263724	0.027804
<i>OPTIMAL_DISTANCE</i>	0.087417	0.006531	0.769646
<i>VARIANCE2</i>	0.223559	0.033498	0.603226
<i>VARIANCEDIST1</i>	0.186354	0.013678	0.938154
<i>VARIANCEDIST2</i>	0.223559	0.033498	0.603226
<i>HESITATE</i>	0.81172	0.07292	0.007423
<i>CLICKRATIO</i>	0.54628	0.025851	0.039092

Table 5-5: MM Wilcoxon Rank Sum Test Written Exam for Week 4 to 6

In Table 5-5, the significant results for Week 4 to Week 6 are presented. There are no significant results in Week 4, but there are a total of five significant results in Week 5, and four in Week 6. *CLICKRATIO* is significant in both Table 5-4 and 5-5, in Week 2, Week 5, and Week 6.

In Table 5-6, there are eight significant features in Week 7, similar to what was seen in Table 5-2, again suggesting there may be some divergence in student behaviour in Week 7 in the two groups. Six of these features are significant in both the CA and Written Exam tests, including *OVERSHOOTDIRECTIONANGLE*, *OVERSHOOTY*, *DIRECTION*, *CLICKTIME* AND *CLICKRATIO*. Week 8 has seven relevant features, some of which are also significant for Week 7, including *OVERSHOOTDIRECTIONANGLE* and *DIRECTION*, both of which were also relevant in Week 7 in the CA results.

<i>Lab</i>	7	8	9	10
<i>EFFICIENCY</i>	0.053057	0.015793	0.243869	0.193793
<i>OVERSHOOTX</i>	0.031453	0.0259	0.466534	0.337519
<i>OVERSHOOTDIRECTION ANGLE</i>	0.005802	0.009798	0.963972	0.257123
<i>OVERSHOOTY</i>	0.042232	0.094216	0.020249	0.726642
<i>OVERSHOOT</i>	0.038224	0.094216	0.01911	0.704971
<i>ANGLEDIFFERENCE</i>	0.391185	0.029862	0.205317	0.254914
<i>DIRECTION</i>	0.010417	0.015633	0.554644	0.315411
<i>VARIANCEI</i>	0.011661	0.008744	0.919408	0.773659
<i>VARIANCEDISTI</i>	0.019036	0.030427	0.955334	0.206768
<i>CLICKTIME</i>	0.031453	0.387216	0.574179	0.875254
<i>HESITATE</i>	0.289976	0.382132	0.022279	0.138475
<i>CLICKRATIO</i>	0.002758	0.067007	0.189006	0.099411

Table 5-6: MM Wilcox Rank Sum Test Written Exam for Week 7 to 10

In the tables in this section, it can be seen that there are some differences between the top and bottom student groups in some of the features, implying a difference in MM between the students who did well and those who did not, although there is not enough consistency to state which features are where the divergence happens.

5.4.2 Linear Regression

Linear Regression was used to check for correlations between MM features and student outcome, both CA results and Written Exam results. Almost no individual feature returned a positive regression score, showing a lack of correlation between MM and student outcome. The full table of Linear Regression scores can be seen in the Appendix Section 10.4.

5.4.3 Neural Networks

The HOG classifier does not work well with the MM data. Though there was some success with predicting student outcome from MM data in the paper in Section 5.2, this does not translate to the HOG classifier. In the experiment described in the paper, the best results came from using both in-lab and in-exam data, and it may be that the success of the classifier was due to the difference or lack of difference between those features, but this classifier does not replicate that.

Very little of the MM data has a Linear Regression coefficient of over 0 in the Linear Regression tests as explained in Section 5.4.2, so the HOG method of selecting features meant that very few of the metrics were used in the classifiers, with no metrics selected for larger thresholds. In an attempt to find some data to use with the HOG classifier, the Linear Regression threshold was reduced to -0.1, but the resulting classifier never achieved more than an average of 57% for any week. The results of the -0.1-threshold classifier can be seen in the Appendix Section 10.5. In summary, the Neural Networks using data set 2 were unsuccessful, due to a lack of high coefficients of over 0, and the attempt to use features with coefficients of over -0.1 did not create a useful Neural Network classifier.

5.5 Mouse Movements Week-by-Week

Week 1

There are already signs of differing behaviours in MM from the first week of labs, as the Wilcoxon Rank Sum Test shows significant differences in the DIRECTION feature of the higher and lower achieving groups, for both CA and Written Exams. As this is the first time the students used the system, this might simply reflect that some students are more comfortable adapting to the use of new technology than others.

Week 2

By Week 2, the number of significant features from the Wilcoxon Rank Sum Test increases to three, and the differences are found in CLICKTIME, HESITATE, and CLICKRATIO in CA groups, and only CLICKRATIO in the Written Exams. This week is the one of only three that had any Linear Regression coefficients of over 0 in the Written Exam results, the three features are CLICKTIME, HESITATE and CLICKRATIO, the same three features with significant differences in the Wilcoxon Rank Sum Test this week. The classifier accuracy result for CA this week is 50.8%, with 16 False Passes out of 50 students, 25 of which were passes and 25 were fails. A classifier with around 50% accuracy works as well as a random classifier, and so is unsuccessful. The classifier accuracy for Written Exam is 50%, and it has 15 False Passes.

Week 3

There are no significant differences in the Wilcoxon Rank Sum Test for Week 3 for CA or Written Exams. The CA classifier has an accuracy of 56%, and 12.4 False Passes. The Written Exam classifier has an accuracy of 49%, and 11.6 False Passes.

Week 4

Like in Week 1, there is a significant difference in CA in the average direction of the two groups, but also in the OPTIMAL_DISTANCE, but there are no differences in the Written Exam results. The CA classifier had an accuracy of 55% and 15.3 False Passes, and the Written Exam result has an accuracy of 46% and 14.5 False Passes.

Week 5

In Week 5, there is an increase in the number of features with significant differences between the groups. In this week, EFFICIENCY, VARIANCE1, VARIANCEDIST3, are significant for CA, while VARIANCE2, VARIANCEDIST1 and CLICKRATIO are significant for Written Exam. OPTIMAL_DISTANCE, (like in Week 4 for CA) and VARIANCEDIST1 are significant for both. The classifier has an accuracy of 54.5% and 13.55 False passes, and the Written Exam classifier has an accuracy of 47% and 13.75 False Passes.

Week 6

In Week 6, only three features are significant in CA groups and four in the Written Exam, despite it being an exam week, when we expected to see differences in student behaviour and therefore more significant features in the Wilcoxon Rank Sum Test. The differences are in DIRECTIONANGLE in CA, and HESITATE, CLICKRATIO (like Week 5) in Written Exam, and OVERSHOOTDIRECTIONANGLE and SEQUENCE_DURATION in both. OVERSHOOTDIRECTIONANGLE also had a positive coefficient in the Linear Regression tests this week. The classifier has an accuracy of 55%, and 11.1 False Passes, and the Written Exam classifier has an accuracy of 46% and 14.8 False Passes.

Week 7

Week 7 has the most significant differences of all the weekly Wilcoxon Rank Sum Tests, a total of 10. The differences are in EFFICIENCY (like in Week 5) OVERSHOOTDIRECTIONANGLE, OVERSHOOTY, OVERSHOOT,

DIRECTION (like in Week 1 and 4), in VARIANCE1 (like Week 5), VARIANCE2, VARIANCEDIST1 (like Week 5), CLICKTIME (like Week 2), and CLICKRATIO (like Week 2). The Written Exam results have eight relevant features, the most of any week in the Written Exam tests. The features are: OVERSHOOTX, OVERSHOOTY, OVERSHOOTDIRECTIONANGLE, DIRECTION, VARIANCE1, VARIANCEDIST1, CLICKTIME, and CLICKRATIO. CLICKRATIO also had a very slight positive coefficient this week of 0.015701 in the Linear Regression tests. This CA classifier has an accuracy of 56%, and a False Pass rate of 13.2, and the Written Exam classifier has an accuracy of 49% and 13.45 False Passes.

Week 8

Week 8 has six significant features, efficiency (like in Week 5 and 7), OVERSHOOTX, SEQUENCE_SPEED, OVERSHOOTDIRECTIONANGLE (like Week 6 and 7), ANGLEDIFFERENCE, and VARIANCE1 (like Week 5 and 7). This week has seven significant features for Written Exam: EFFICIENCY, OVERSHOOTX, OVERSHOOTDIRECTIONANGLE, ANGLEDIFFERENCE, DIRECTION, VARIANCE1 and VARIANCEDIST1. The CA classifier has an accuracy of 51%, and 21.8 False Passes and the Written Exam has an accuracy of 46.8% and 16.75 False Passes.

Week 9

Week 9 has four significant features, EFFICIENCY (like Week 5, 7, and 8), OVERSHOOTY (like Week 7), OVERSHOOT (like Week 7) and ANGLEDIFFERENCE (like Week 8). The Written Exam groups have only three significant features: OVERSHOOTY, OVERSHOOT and HESITATE. This week's CA classifier has an accuracy of 49%, and a full 25 False Passes, meaning every Fail is classified as a pass. The Written Exam classifier has an accuracy of 48.6% and has 20.85 False Passes.

Week 10

This week only has two significant features: OVERSHOOTDIRECTIONANGLE (like Week 6,7, and 8) and DIRECTION (like in Week 1, 4, and 7), and no significant features for the Written Exam groups. This week's CA classifier again has an accuracy 49%, and a full 25 False Passes out of 25 Fails, meaning every Fail is classified as a pass. The Written Exam classifier has an accuracy of 48.5% and has 20.3 False Passes.

5.6 Mouse Movements Conclusions

While correlations seem to exist between MM data and student outcome in semester two, when looking at the average of student behaviour in ordinary lab sessions and exam sessions, these correlations are not apparent when using the HOG classifier to examine the data on a week-to-week basis. This means it was not possible to draw any conclusions on the divergence of student behaviour from the results of the classifiers. It is unclear why the semester 2 classifier in Section 5.2 was successful and the semester 1 classifier was not. It may be because the students do not show signs of stress until the second semester. It seems more likely that the reason is that the original semester 2 classifier used the average of the values of features from all in-lab and in-exam sequences across the semester, while the semester 1 classifier divided the data into weeks. It may be that the resulting success was due to the difference in behaviour between the two states: in-exam and in-lab.

Although the Neural Network was unsuccessful, there is evidence of a divergence of student behaviour, especially in Week 5 and 7 from the Wilcoxon Rank Sum Test, suggesting that these may be key stages in the Introduction to Programming module. However, very few of the features that show significant results are consistent throughout the semester, and so the results may be coincidence. The only features that have any consistency are DIRECTION, EFFICIENCY, OVERSHOOTDIRECTIONANGLE, and CLICKRATIO, which all have significance in four labs for CA or Written Exam tests. CLICKRATIO also has two of the very few positive Linear Regression results. Despite the disappointing results from the data with the HOG classifier, it has been shown that the data has value in building a Pass/Fail classifier from the paper in Section 5.2, and possible value as a stress detector, as it has some success in classifying students as being in a stressful (exam) environment or in a less stressful (normal lab) environment, but this requires more research.

6. Experiment 2: Compile, Run, and Evaluate (CRE) Movements

6.1 Introduction to CRE Experiment

In this chapter, the paper describing the first experiment using the CRE data and a prototypical HOG classifier system is reproduced, in Section 6.2. The remaining sections of the chapter explore the results of using the same methodology used in the Chapters 5 and 7 with the CRE data.

There is previous work that examined and found statistically significant differences between students with and without programming experience in their behaviour, including the number of runs and tests while working on their code [31], tests being similar to MULE's evaluate. They also found that these differences lessened in the final weeks of the semester (of 12 weeks), suggesting the behaviours are signs of programming proficiency.

In the MULE system, the users write code to answer their coding assignments, which they can then compile, run, and evaluate. In this section, we examine the patterns in which students run, compile, and evaluate and how this relates to student exam outcome.

This experiment is similar to the one described in the paper in Section 6.2 [58], but the experiment was rerun to conform to the HOG classifier methodology, and to include the Wilcox Rank Sum Test. The classifiers in this thesis predict student outcome in the final Written Exam and in CA, whereas the classifier in the paper only predicted the outcome of the in-lab formal coding exams. The classifier in the paper achieves a highest accuracy of 78% in Week 8, slightly higher than this experiment's highest accuracy of 76.7%. However, the experiment in Section 6.3 to Section 6.6 has more success in earlier weeks, for example, the classifier achieves an accuracy of 73% in Week 5, slightly higher than the papers highest early semester accuracy of 70%, and the experiment also classifies students by Written Exam outcome and achieved an accuracy of 78% in Week 9.

6.2 Paper: Exploring the Coding Behaviour of Successful Students in Programming by Employing Neo-Piagetian Theory

Exploring the Coding Behaviour of Successful Students in Programming by Employing Neo-Piagetian Theory

Natalie Culligan

Department of Computer Science
Maynooth University
natalie.culligan@mu.ie

Kevin Casey

Department of Computer Science
Maynooth University
kevin.casey@mu.ie

Abstract

We have collected data from approximately 300 students in their third-level first year Introduction to Programming module as they learn to write code using our in-house pedagogical coding environment, MULE. This data includes performance in lab exams and pseudocode questions, and data on code compiled, code run, and code evaluated, which we call CRE data. Evaluations are automatically graded and feedback is provided to students on their code. The student can only evaluate their code in the scheduled lab place and times but can evaluate as many times as they wish without penalty. The pseudocode questions are used to examine the students' understanding of programming concepts, by removing the use of the compiler and comparing their performance in pseudocode questions to CRE data. Using a Neo-Piagetian framework, we examine pseudocode performance, lab exam performance and programmer behaviour in terms of CRE data. We investigate CRE data as signs of a student's progression through the three stages of Piagetian understanding and build a series of Deep Neural Net binary classifiers to test if this passively collected behavioural data can be used to detect students in danger of failing.

1. Introduction

Computer Science has one of the highest failure and dropout rates in 3rd level education (Bennedsen, & Caspersen, Corney *et al.* 2010, Lang *et al.*, Watson & Li). In this paper, we will investigate if students in introductory computer science courses are failing to reach the later stages of Neo-Piagetian understanding, and if we can investigate and observe signs of these stages through passive data collection, and the results of pseudocode tasks in the weekly practical coding labs. The research question for this study is:

- Can we observe signs of progression through the Neo-Piagetian stages of learning by examining passively collected data on students' coding behaviour?

The coding behaviour data we discuss in this paper is the order in which students compile, run, and evaluate their code. Evaluation provides the student with automatic grades and feedback. The students use the pedagogical coding system MULE to complete their weekly coding tasks. In this system, students are unable to run their code until they have successfully compiled and cannot evaluate their code until it has run successfully.

In the doctoral thesis "*Neo-Piagetian Theory and the Novice Programmer*" (Teague), the author states that "*Programming competence requires abstract reasoning skills and learning to program is about the sequential and cumulative development of those abstract reasoning skills in an unfamiliar domain.*" We wanted to introduce pseudocode questions into our first-year curriculum to encourage students to build mental models of

programming concepts by requiring students to predict code output, without relying on the compiler. These pseudocode questions are English language representations of code that cannot be run with a compiler but represent programming concepts such as loops and arrays (Lopez *et al.*). With pseudocode, we can see if the students can abstract the concepts away from Java and apply what they have learned in class in a much more generalised way. This is useful as if the students are able to do so, they are more likely to be able to reuse the skills and apply them in a variety of ways, instead of memorizing and replicating techniques they have used in the past.

In this paper, we will discuss our findings when investigating CRE data in weekly labs as students graduate from random/loosely guided “tinkering” to more intentional code-writing. While previous work has discussed “tinkering” as a viable method of learning programming, we will discuss if this is true throughout the first semester, or if CRE data that implies an over-use of tinkering is in fact an indication that a student is not developing a good mental model of fundamental programming concepts and is therefore in danger of falling behind.

2. Related Research

2.1. Student Behaviour when Learning to Code

There have been numerous studies that investigate novice programmer behaviour such as patterns of compilation and running of code and how it relates to student success.

Perkins *et al.*, investigate the different strategies that novice programmers adopt when learning to code, and describe what they term “stoppers”, “movers”, and “extreme movers”. “Stoppers” are novices who, when faced with a problem without a clear course of action, stop attempting to find a solution to the problem and appear to be unwilling to explore the problem any further. “Movers” are novices who will constantly modify and test their code when faced with a problem. “Extreme Movers” will also constantly modify and test their code but are different from movers in that they do not seem to learn from attempts that previously did not work, and they do not continue to work on solutions that fail the first time so do not end up “homing in” on a working solution. The authors do not specifically speak about how these different patterns relate to compilation and run behaviour, but the below papers do touch on it in direct reference to this study.

Two papers on the programming environment BlueJ (Jadud, 2005, Jadud 2006) discuss the behaviours of the authors’ students, and how similar their students’ behaviours are to those in the above Perkins *et. al.* paper. They discuss their own “extreme movers”, which they describe as “tinkerers”, and how these students would sometimes allow their experimental code to accumulate, causing their code to become increasingly complex and, eventually, incomprehensible. The BlueJ studies found that 24% of all compilation events followed less than 10 seconds after a previous compilation, and half of all compilation events occurred less than 40 seconds after a previous compilation. Students spent more time working on their code after a successful compilation than they did trying to fix a syntax error. The authors found that students tend to program in large blocks, then spend time writing and compiling code in small

bursts in order to fix syntax errors. Accordingly, multiple compilations may indicate a large number of syntactic problems.

In “*Studying the Novice Programmer*” (Soloway & Spohrer) the authors discuss the need for students to build plans. As mentioned above, students who tinker aimlessly create bugs, and without clear goals may fail to progress towards a working solution. The authors used natural language to investigate if students with plans, broken into small tasks, are more successful when programming.

In “*Analysis of Code Source Snapshot Granularity Levels*” (Vihavainen) the author discusses the ratio of “snapshots to submissions”, where a snapshot is a copy of the code taken every time the student saves, compiles, runs, or tests their code. Submissions are final versions of a program submitted for correction/grading, provided by a plugin for NetBeans that provides feedback and grading to the student. Using a Wilcoxon rank sum test, the authors found a statistically significant difference between the number of runs and tests for students with previous programming experience and those without. This difference continued to be visible throughout the course, although the behaviour of the participants was more alike in the final weeks of the course, perhaps implying that these behaviours are indicators of programming proficiency.

One of the research questions in the paper “*Evaluating Neural Networks as a Method for Identifying Students in Need of Assistance*” (Castro-Wunsch) is “*Are neural network (NN) models appropriate for the task of identifying students in need of assistance?*” The authors found that, yes, neural networks predicted at-risk students at least as well as Bayesian and decision tree models, and had the advantage of being “pessimistic”, meaning that the neural networks were more likely to incorrectly classify students as at-risk, rather than incorrectly classify students as not at-risk. From this research, we decided to use neural networks as our classifier.

2.2. Neo-Piagetian Theory and Abstraction in Programming

There are also a number of studies that use Neo-Piagetian theory in examining student behaviour in computer science and discuss abstraction in relation to novice and expert programmers.

In “*Concrete and Other Neo-Piagetian forms of Reasoning in the Novice Programmer*”, (Lister) the author discusses the reasoning behind the use of Neo-Piagetian theory. Classical Piagetian theory considers the progress through different stages of learning to be a consequence of a biological maturing of the brain. Neo-Piagetian theory, on the other hand, considers this instead a result of gaining experience, and in particular, the ability to “chunk” knowledge within a certain knowledge domain.

Corney et. al (2011) describe a study in which almost half of the sample students were unable to answer a simple explain-in-plain-English question in the third week of their introductory programming course, showing that students were encountering problems much sooner than could be detected by traditional programming questions/examinations.

In “*Neo-Piagetian Theory and the Novice Programmer*” (Teague, 2015), the author found that the development of programming skills is both “*sequential and cumulative*”, and that behaviours associated with sensorimotor and preoperational reasoning are evident from very early in the semester.

The authors of “*Mired in the Web: Vignettes from Charlotte and Other Novice Programmers*” (Teague *et al.*) ask if a student can have different levels of ability for different tasks which test similar programming concepts – if a student can trace and understand code, can they also *write* that code? They also ask why some students do not seem to be able to understand code with abstractions and instead rely on tracing code with specific values. The study found that students who were still operating at the sensorimotor level in week 2 were often still operating the same way in week 5, and were lagging behind students who were operating at the preoperational level in week 2. They defined students in the preoperational stages by certain behaviours which they observed using think-aloud data from students. Preoperational behaviours were guessing, a fragile grasp of semantics, confused use of nomenclature, an inability to trace simple code, as well as general misconceptions. Errors due to cognitive overload and reluctance to trace were considered behaviours associated with both sensorimotor and preoperational. The ability to trace but not explain code, as well as a reliance on specific values, were signs of the preoperational stage. The authors note that students may achieve marks for guessed answers, but it is not until they listen to the students speak aloud their thought process that they were able to get a clear picture of the students understanding and ability.

Shneiderman and Mayer found that expert programmers were able to recall more of a program than novices when it was presented to them in normal order, but not when it was scrambled, implying that the experts were able to “chunk” information together when the code made sense. The authors proposed that experienced programmers construct functional representations of computer programs.

Adelson found that expert programmers’ memory chunks tended to be semantically or functionally related, while novices typically chunked by syntax. Semantic knowledge consists of programming concepts that are generalized, and independent of programming language, whereas syntactic knowledge is more precise and rooted in exact representations of concepts in specific programming languages. For example, a novice may think of a loop as a specific for loop in Java, but an expert planning a piece of code may simply think of a loop abstractly, as something that performs a needed function, without thinking about the exact type of loop, the details of the iteration, or the syntax associated with it (Bisant & Groninger, Wiedenbeck).

3. Methodology

For this study, we collected data from around 300 students as they completed their introduction to programming module in Java using MULE, our in-house, browser-based pedagogical coding environment (Culligan & Casey). This system resembles a desktop with both built-in applications for content and assignment delivery, and a code editor for completing, running, and evaluating code for assignments. MULE also includes mechanisms for making sections of the material invisible to some users until some constraints are satisfied such as date/time and IP address – this was used to allow certain assignments to only be accessible in the scheduled lab times and locations. Within MULE, each attempt the student makes on an assignment is recorded, and the student can easily recover any previous attempt, allowing the student to “tinker” and experiment with their code without fear of losing any work. There is evidence to suggest that a certain amount playing/tinkering with code is an indication of student success (Berland *et al.*, Berland & Martin).

For 5 of the 10 mandatory computer lab sessions during the first semester of their computer science course, students were asked to predict the outcome of pseudocode snippets, along with their usual lab consisting of two programming

questions, and some peer-programming tasks. The students were told that they are not awarded any marks towards their continuous assessment for answering the pseudocode questions. The students have access to most of the programming tasks before the lab and can write code, compile, and run it, but not evaluate it for continuous assessment grades. Some of the exercises are only accessible in the labs at the assigned times, so students must write, run, compile, and evaluate the code in the lab.

Although students were able to work on an assignment before assigned lab times, we chose to look exclusively at the data from lab times. Our reasoning is that students outside of labs can be in very different environments – some may have a quiet place to work undisturbed, others may be working in a noisy environment or may be frequently interrupted, so comparisons of their behaviour may be less insightful than those from a formal lab. For most of the semester, the students can only evaluate from inside the lab during the specified lab times, so the data from outside the labs would only have compile and run events.

We did not include data from students who did not participate in the weekly labs (missing more than four), as we wanted to investigate changes in behaviour from week to week and to look at at-risk students who are actively engaging in the course labs on a weekly basis (Castro-Wunsch). After removing students who did not complete 4 or more labs, we were left with 266 subjects. The gathered data is the patterns of student compile, run and evaluate actions:

- Compile: Students cannot run their code until it compiles successfully
- Run: Students cannot evaluate their code until it runs successfully
- Evaluate: The student's code is assigned a grade, and feedback is provided.

This data was used to build Deep Neural Net binary classifiers, that would classify students as being in either the top 50% or the bottom 50% of the class lab exam grades on a week-to-week basis. Each weekly classifier would use the CRE data for each assignment for that week, and from all previous weeks. Below we discuss the results of statistical tests exploring correlations between student behaviour and outcome, and the classifier built to predict student outcome.

4. Analysis

When analysing the data, for every time a student performs a CRE action, we look at that action and the one before and record it as a “movement” - the student moves from a Compile to a Run, is recorded as C2R, or a Run to an Evaluate in R2E for example. When processing this data, we looked at each movement as a percentage of all actions a student took during that lab. From the previous studies on programming and Neo-Piagetian stages, we expected to see the following as signs of progression through the stages:

Sensorimotor Stage: Interacting almost randomly, with little understanding of the outcome, resulting in more C2C movements, less C2R movements and less participation and success with pseudocode questions.

Preoperational Reasoning Stage: The student is beginning to master writing compilable code, and can predict code outcome, resulting in higher amount of C2R movements and R2C movements, fewer C2C movements and more participation and success with pseudocode questions.

Concrete Operational Stage: At this stage, programmers have a good grasp of concepts allowing the programmer to write more complex code, resulting in fewer C2C movements, fewer C2R movements, more R2E movements and more participation and success with pseudocode questions.

The students in the study were divided into two groups: those in the top 50% of the class in lab exam grades, and those in the bottom 50%. The two data sets contain the percentages of total movements per week for each student. A sample of the student data for a week would look like the following:

C2C	C2R	R2C	R2R	R2E	E2C	E2R	E2E
0.33997	0.254913	0.127186	0.01639	0.130208	0.111902	0.003655	0.004159

Table 1: Example of an average sample of student weekly data

The following tests were then run on the two data sets:

- To examine if the differences between the two groups were significant, t-tests were used.
- Linear regression was used to find which movements were most related to lab exam outcome, on a week-to-week basis, to select which movement data would be used in the classifier.
- Finally, the data from the most significant movements each week are used to create a Deep Neural Net binary classifier, to classify each student as being in the top or bottom 50% of the class.

5. Results

To find if there were significant differences between the top and bottom 50% of the students, t-tests were used, the results of which are considered significant differences between the two groups if the result is less than 0.05. These results are in bold. The p-value results of the groups according to lab exam results are in Table 2, and the results of the groups divided by pseudocode performance are in Table 3. Lab 6 and lab 10 included lab exams, during which the students could not look at their previously written code from earlier labs.

	C2C	C2R	R2C	R2R	R2E	E2C	E2R	E2E
1	0.001214	0.470967	0.539369	0.448587	0.090213	0.070492	0.650004	0.24305
2	0.004757	0.02424	0.665045	0.674198	0.005787	0.060203	0.388143	0.260525
3	4.80E-06	5.04E-05	0.112107	0.585846	0.031448	0.498212	0.120838	0.280715
4	1.50E-06	3.16E-08	4.04E-06	0.032048	0.305453	0.031562	0.951828	0.748698
5	4.03E-10	1.47E-08	0.008756	0.100132	0.000341	0.004397	0.015933	0.10881
6	9.00E-14	7.94E-15	5.90E-11	0.019153	0.064127	0.122571	0.026066	0.940655
7	2.05E-06	6.28E-09	6.96E-07	0.561189	0.111728	0.140013	0.924634	0.579504
8	5.73E-13	2.60E-09	0.00853	0.00715	3.36E-05	2.50E-05	0.501948	0.04282
9	0.002878	0.187736	0.187655	0.386204	0.008422	0.0083	0.234538	0.389353
10	6.05E-06	4.70E-07	0.012479	0.683429	0.034407	0.082467	0.609155	0.758995

Table 2: Results of t-test on groups divided by lab exam results

There were significant differences between the two groups found in C2C every week, C2R most weeks, and R2E and R2C in 8 of the 10 weeks. In Table 3, we see that the results are similar results to the lab exam t-tests, the main difference being that the R2E movements are almost never significant.

	C2C	C2R	R2C	R2R	R2E	E2C	E2R	E2E
1	0.656744	0.010296	0.167739	0.096637	0.368741	0.27848	0.979841	0.062224
2	0.007839	0.01318	0.698311	0.79891	0.005272	0.518315	0.074192	0.384717
3	0.001236	0.000551	0.049986	0.141391	0.498473	0.615511	0.872643	0.817136
4	0.00078	0.000378	0.015029	0.089415	0.768301	0.095344	0.72989	0.085109
5	0.000306	0.001537	0.268965	0.043827	0.051354	0.050773	0.147545	0.185538
6	0.000562	0.001477	0.038343	0.018253	0.07908	0.088534	0.328674	0.182885
7	0.014912	0.00068	0.001243	0.371309	0.114922	0.140932	0.328346	0.52572
8	0.014178	0.279716	0.265897	0.414656	0.768084	0.718394	0.434119	0.501809
9	0.043973	0.184825	0.768122	0.165121	0.268245	0.31005	0.380506	0.77323
10	0.027376	0.009383	0.042855	0.350375	0.601622	0.888391	0.139526	0.910241

Table 3: Results of t-test on groups divided by pseudocode results

From our predicted behaviour of the Neo-Piagetian stages outlined at the start of the analysis section we expected to see students who did poorly in the exams displaying different behaviour in the C2C, C2R and R2C movements as more successful students moved onto preoperational reasoning stages. Higher achieving students have a consistently lower average percentage of C2C when groups are divided by lab exam results. The difference in C2C movements gets steadily larger from week 1 until week 7, when it slightly reduces. This is also true for the pseudocode results, with smaller margins of difference. The difference is smaller in the last weeks of the module, which may indicate that our students who do not do well are moving through the Neo-Piagetian stages but are not moving quickly enough for the course.

Higher achieving students have a consistently higher average percentage of C2R when divided by lab exam results. This difference peaks in week 7, for both lab exam and pseudocode results. Both groups have a similar percentage of R2C when divided by lab exam results, but the difference peaks in weeks 6 and 7, when the higher achieving students have a higher average percentage of R2C movements. This may be the point where successful students have reached preoperational reasoning, as an increase in R2C movements indicate the student is in the “tinkering” stage as described by Perkins *et al.*

Week	Average Classifier Success Rate
1	0.62
2	0.6
3	0.68
4	0.7
5	0.62
6	0.6
7	0.72
8	0.76

Table 4: Classifier results

Using the dataset containing the CRE percentages for each student for each week, Deep Neural Net binary classifiers were trained to classify students as being in the top 50% or the bottom 50% of grades for the lab exams. Linear Regression tests were used to compare the CRE actions and their relation to student performance in lab exams. This was used to select movement data to be

used in the Deep Neural Nets. Multicollinearity can be an issue for DNN, so a check was run on the features (where each movement was a feature) and we removed the most highly correlated CRE data and tested again. This was repeated until the remaining data was sufficiently nonlinearly related, when all features had a variance inflation factor (a test for correlation between independent variables) of less than 5. The resulting data set was used to train and test our DNN classifier. A classifier was built for each week of the semester, using the CRE data from that week, and from all previous weeks. The results are shown in Table 4. The results of week 9 and 10 are identical to week 8, as it uses the same CRE data after the multicollinearity tests.

6. Discussion

Other studies have referred to lab 4/week 4 (Teague) as the time around which students who are in danger of failing begin to perform badly or separate in behaviour from the other students. Of course, what takes place at this point varies across different institutions and courses. Nonetheless we see that in line with this estimated timescale, the differing behaviour among students becomes more pronounced around lab 4, and at this point the classifier has a success of 70%. At this point, if students are consistently compiling without progressing to run, this is a sign the student is in danger. This is not hugely surprising. It implies that the student is failing to write compilable code, and we would expect that a student who cannot write compilable code would be in danger.

The percentage of R2C becomes more significant around week 4. A student who compiles code, then runs, but then goes back to compile, is most likely working on a semantic issue, rather than a syntactic one, as mentioned in the BlueJ papers (Jadud 2005, Jadud 2006). We suspect that the reason it becomes relevant to the students' overall performance in lab exam results is because week 4 is when most students should be beginning to master syntax and to abstract solutions, allowing them to construct more complex programs using multiple concepts together. The result is that we see successful students compiling successfully and rewriting their code until they reach a solution, causing successful students to have more C2R movements and fewer C2C movements. Students who are still struggling to write semantically correct code will have even more C2C movements as the assignments get more difficult.

Research Question: Can we observe signs of progression through the Neo-Piagetian stages of learning by examining passively collected data on students coding behaviour?

Yes, we have described the expected signs in CRE movements of progression through the stages of Neo-Piagetian learning, observed these signs in novice programmers and found these signs relate to student success. From our analysis section, we see that the CRE movements that are associated with success change as the semester progresses. Using a Neo-Piagetian framework, we examine these differences.

The three Neo-Piagetian stages in learning to program (Lister, du Boulay, Teague, Teague *et al.*):

(1) Sensorimotor Stage - interacting almost randomly, with little understanding of the outcome

A high percentage of C2C movements may indicate that a student is tinkering almost randomly with their code and is unable to write compilable code. From our analysis, we see that a lower amount of C2C

movements, and a higher amount of C2R movements is associated with better performance in lab exams. This is similar to the findings in the paper (Vihavainen) which found a statistically significant difference between the number of runs and tests for students with previous programming experience and those without.

(2) Preoperational Stage – beginning to master syntax, deeper understanding and being able to predict behaviour from interactions

Students with a higher amount of C2R movements may be in this stage, as they become able to write compilable code, but are still be unable to predict the outcome of their code. As a result of this, the student will repeatedly “tinker” with their code, resulting in increased R2C movements. We found R2C movements became significant from week 4, indicating that students should reach this stage by week 4 if they are to be successful in the module lab exams.

(3) Concrete Operational Stage – can “chunk” (Shneiderman & Mayer) programming concepts and abstractions of the code’s behaviour, allowing the programmer to write more complex code.

At this point, students should be able to write compilable code and successfully predict their code’s outcome. Students at this stage should have fewer C2C movements, fewer C2R movements, and a higher percentage of R2E movements. This indicates that they have a good grasp of semantics and are able to predict code behaviour with less tinkering and playing with code. We would expect to see a higher correlation between outcome and R2E movements as students reach this stage, and while R2E is related to success at some points in the semester, the average difference between the two groups is consistently low. We strongly suspect that most students do not reach concrete operational stage until after their first semester (Teague).

7. Conclusions

We have found that C2C and C2R movements are important indicators of student performance in their first semester of programming. While the highest classifier success of 76% used data from throughout the 8 weeks, we had success with the week 4 classifier which had a success percentage of 70%, showing there is evidence of a student success or failure as early as week 4. This version of the classifier used all 4 weeks C2C percentages as the input data in predicting the student outcome. In future work, it would be worth looking at which specific assignments and topics are key clues in a student’s eventual outcome.

We would expect that student coding behaviour would correlate to lab exam performance and pseudocode performance, if the coding behaviour in question indicates progress through the Neo-Piagetian stages of learning. We have seen that patterns of student behaviour contain indications from an early stage if they are likely to perform well in lab exams. We have discussed how this relates to previous work done in the area of Neo-Piagetian theory in the context of students learning to program. We have established a strong case for the connection between students’ programming behaviour and their stage of Neo-Piagetian learning by showing the correlation between student CRE movements, and their lab exam outcomes, and we discussed the reasons behind those behaviours and how they relate to Neo-Piagetian theory.

Introduction to programming modules that emphasize only how to write code, and grade based primarily on written code may be problematic. Results of

pseudocode assignments in a programming module can help us as researchers and educators to identify students who have not developed mental models of programming concepts, and are instead relying on “hacking”, where students attempt to complete a coding assignment by writing code and testing input/output without planning and predicting their code’s behaviour. Students who are “hacking” may still perform reasonably well in their weekly labs, and so may believe that they are keeping up and do not need to continue to work on their grasp of fundamental coding concepts. These students will then progress to more difficult modules without the programming basics required to engage with the material. This may be a scenario unique to computer science and a significant contributory factor as to why computer science failure rates are so high. In future work, we will examine how a novice programmer’s pseudocode results and patterns of behaviour may relate to code complexity, as a reflection of their ability to “chunk” programming concepts, in order to combine them to create solutions for programming problems.

In conclusion, the most significant findings from this study are, firstly, that the divergence in behaviour between high and low achieving students takes place in week 4. Students who are not displaying signs of progression to the preoperational stage of Neo-Piagetian learning do not do well in their lab exams at the end of the semester. Secondly, we found that these differences in behaviour are less pronounced later in the semester – implying that the students who were behind in week 4 are capable of progression to preoperational stage, but crucially, not at the pace dictated by the module.

8. References

- Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory & cognition*, 9(4), 422-433.
- Bennedson, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *AcM SIGcSE Bulletin*, 39(2), 32-36. J. Bennedson and M. E. Caspersen. Failure rates in introductory programming. *ACM SIGCSE Bulletin*, 39(2):32–36, 2007.
- Berland, M., Martin, T., Benton, T., Petrick Smith, C., & Davis, D. (2013). Using learning analytics to understand the learning pathways of novice programmers. *Journal of the Learning Sciences*, 22(4), 564-599.
- Berland, M., & Martin, T. (2011). Clusters and patterns of novice programmers. In *The meeting of the American Educational Research Association*. New Orleans, LA.
- Bisant, D. B., & Groninger, L. (1993). Cognitive processes in software fault detection: a review and synthesis. *International Journal of Human-Computer Interaction*, 5(2), 189-206.
- du Boulay, B., O'Shea, T., & Monk, J. (1981). The black box inside the glass box: presenting computing concepts to novices. *International Journal of man-machine studies*, 14(3), 237-249.
- Castro-Wunsch, K., Ahadi, A., & Petersen, A. (2017, March). Evaluating neural networks as a method for identifying students in need of assistance. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 111-116).

- Corney, M. W., Lister, R., & Teague, D. M. (2011, January). Early relational reasoning and the novice programmer: Swapping as the “Hello World” of relational reasoning. In *Conferences in Research and Practice in Information Technology (CRPIT)* (Vol. 114, pp. 95-104). Australian Computer Society, Inc..
- Corney, M. W., Teague, D. M., & Thomas, R. N. (2010, January). Engaging students in programming. In *Conferences in Research and Practice in Information Technology*, Vol. 103. Tony Clear and John Hamer, Eds. (Vol. 103, pp. 63-72). Australian Computer Society, Inc..
- Culligan, N., & Casey, K. (2018). Building an Authentic Novice Programming Lab Environment. *Irish Conference On Engaging Pedagogy*
- Jadud, M. C. (2005). A first look at novice compilation behaviour using BlueJ. *Computer Science Education*, 15(1), 25-40.
- Jadud, M. C. (2006). An exploration of novice compilation behaviour in BlueJ (Doctoral dissertation, University of Kent).
- Lang, C., McKay, J., & Lewis, S. (2007). Seven factors that influence ICT student achievement. *ACM SIGCSE Bulletin*, 39(3), 221-225.
- Lister, R. (2011, December). Concrete and other neo-Piagetian forms of reasoning in the novice programmer. In *Conferences in Research and Practice in Information Technology Series*.
- Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008, September). Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the fourth international workshop on computing education research* (pp. 101-112).
- Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of learning in novice programmers. *Journal of Educational Computing Research*, 2(1), 37-55.
- Shneiderman, B., & Mayer, R. (1979). Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International Journal of Computer & Information Sciences*, 8(3), 219-238.
- Soloway, E., & Spohrer, J. C. (2013). *Studying the novice programmer*. Psychology Press.
- Teague, D. (2015). *Neo-Piagetian theory and the novice programmer* (Doctoral dissertation, Queensland University of Technology).
- Teague, D., Lister, R., & Ahadi, A. (2015, January). Mired in the Web: Vignettes from Charlotte and Other Novice Programmers. In *ACE* (pp. 165-174).
- Vihavainen, A., Luukkainen, M., & Ihanola, P. (2014, October). Analysis of source code snapshot granularity levels. In *Proceedings of the 15th Annual Conference on Information technology education* (pp. 21-26).
- Watson, C., & Li, F. W. (2014, June). Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 39-44).
- Wiedenbeck, S. (1985). Novice/expert differences in programming skills. *International Journal of Man-Machine Studies*, 23(4), 383-390.

The paper reproduced here describes a successful attempt to build a classifier that predicts student outcome from student CRE data. The following sections will use this same data using the methods outlined in Section 4.3 and 4.4.

6.3 CRE Features

The data set used in the following experiment is data set 2, as described in Section 4.1.2. In the following tests, the analysis and classifier use eight different features for each of the 10 weeks from this data set, resulting in a total of 80 features. These features are measurements of how often a student Compiled, Ran, or Evaluated, and followed with another Compile, Run, or Evaluate, as mentioned in Section 6.2. If a student Compiles their code, then Compiles again, that is recorded as a C2C. If the student Runs and then Evaluates, that is recorded as a R2E, etc. The eight features are explained in Table 6-1. There is no Compile to Evaluate feature as the MULE system does not allow the user to Evaluate after Compiling without a Run action.

From	To	Recorded as
Compile	Compile	C2C
Compile	Run	C2R
Run	Compile	R2C
Run	Run	R2R
Run	Evaluate	R2E
Evaluate	Compile	E2C
Evaluate	Run	E2R
Evaluate	Evaluate	E2E

Table 6-1: CRE Features

Each of these features is recorded for each of the student's weekly labs and is a percentage of the total CRE actions that the student takes in that lab. C2E movements are not shown in this experiment as the system did not allow students to Evaluate their work until it had been Run.

In the Figure 6-1, a possible reason for each feature is explained. For example, a C2C movement may indicate that a student has not successfully written compilable code, and so must rewrite and then Compile again. A C2R feature means the student has successfully written compilable code. A R2R feature may mean that the student is testing their code, but it could mean that the student's code is not running as expected, or that it is not running as expected. A R2E movement

means that the student has finished testing their code and is checking their grade. E2R may mean that the student is unsatisfied with the evaluation grade and is testing their code to find the problem. E2C can also mean the student is unsatisfied with their grade, and has rewritten their code, though it may also mean the student has moved on to the next question. For many of these features, what is important is how often a student performs an action. A small number of C2C actions is to be expected, even an expert programmer will not write compilable code without making errors every time. However, a larger percentage of C2C actions may imply that the student is unable to write compilable code and is in danger of failing the module.

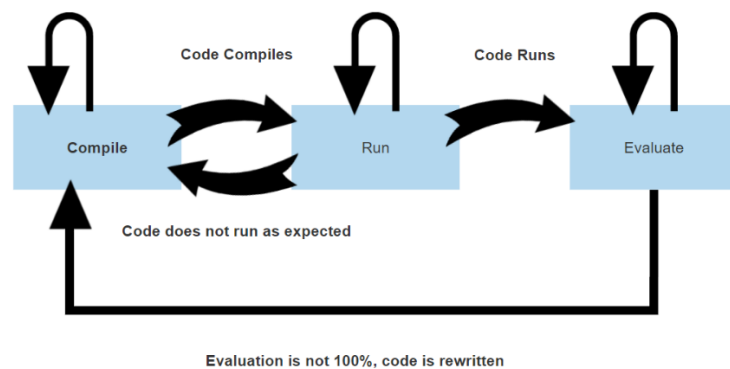


Figure 6-1: CRE Patterns

6.4 CRE Movements Analysis

In this section, the CRE movements are examined. The CRE movements are the patterns of Compile, Run, and Evaluate behaviour as the students complete their coding assignments in the MULE system. The behaviours are examined using the Wilcox Ranks Sum Test, and Linear Regression.

6.4.1 Wilcox Rank Sum Test

This section contains the results of the Wilcox Ranks Sum Test, comparing the top 50% and the bottom 50% of the grades for CA and the Written Exam for each of the eight CRE features. Any result of less than 0.05 is considered significant and is in bold.

6.4.1.1 CA

The results of the Wilcox Rank Sum Tests comparing the groups divided according to the CA results are presented in Table 6-2 and Table 6-3. While there are significant differences between the two groups throughout the semester, when

looking at the CA results, it can be seen in Table 6-2 that there is a spike in Week 5, when every one of the features shows significant difference in the two groups.

<i>Lab</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
<i>C2C</i>	0.003304046	0.000696743	8.34E-11	1.60E-09	1.39E-10
<i>C2R</i>	0.029707699	0.000338183	5.23E-12	4.68E-14	4.76E-09
<i>R2C</i>	0.405981696	0.629870344	0.000500886	1.51E-08	0.000377349
<i>R2R</i>	0.346761087	0.05292788	0.146292951	0.036721567	0.003007271
<i>R2E</i>	0.012478407	0.000756833	0.083116735	0.234028299	0.014072256
<i>E2C</i>	0.074870296	0.177169647	0.864372077	0.000757408	0.016553229
<i>E2R</i>	0.989306277	0.740945091	0.397346141	0.826922363	0.039563534
<i>E2E</i>	0.481646241	0.59367972	0.328548797	0.690327639	0.003405348

Table 6-2: CRE Wilcox Rank Sum Test CA for Weeks 1 to 5

This continues to a lesser degree in Week 6 and Week 7, as shown in Table 6-3. Week 8 has only significant results, C2C and C2R, and Week 9 and Week 10 each have four significant results. C2C and C2R are significant throughout the semester.

<i>Lab</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>C2C</i>	3.17E-20	1.17E-06	2.77E-07	1.64E-07	3.19E-05
<i>C2R</i>	9.81E-21	3.90E-11	0.012137	0.000283	1.05E-07
<i>R2C</i>	4.12E-14	5.93E-10	0.177591	0.53325	1.80E-07
<i>R2R</i>	0.00162738	0.004453	0.064039	0.713808	0.030687
<i>R2E</i>	0.003336881	0.001807	0.228163	0.000993	0.51366
<i>E2C</i>	0.002678636	0.007413	0.201059	0.002447	0.5613
<i>E2R</i>	0.396740295	0.092942	0.205742	0.965096	0.907747
<i>E2E</i>	0.386475328	0.45643	0.053136	0.511572	0.985591

Table 6-3: CRE Wilcox Rank Sum Test CA for Weeks 6 to 10

6.4.1.2 Written Exam

The results of the Wilcox Rank Sum Tests comparing the groups divided according to the Written Exam results are presented in Table 6-4 and Table 6-5. The results are similar to the results of the CA groups Wilcox Rank Sum Tests.

Similar to what was found in the results of the paper in Section 6.2, the most consistently significant features throughout the semester are C2C and C2R, which have significant results for every week in the semester, as shown in Table 6-

4 and Table 6-5. R2C is the next most consistent, with significant results for six of the semester's 10 lab sessions.

<i>Lab</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
<i>C2C</i>	0.001391	0.000537	6.53E-12	7.51E-09	7.75E-08
<i>C2R</i>	0.04382	0.00166	1.16E-11	2.14E-11	2.36E-07
<i>R2C</i>	0.80832	0.861828	0.005912	2.25E-06	0.001155
<i>R2R</i>	0.809358	0.591865	0.673095	0.014489	0.005465
<i>R2E</i>	0.011304	0.010363	0.004539	0.370142	0.187853
<i>E2C</i>	0.042985	0.142691	0.08873	0.003129	0.040557
<i>E2R</i>	0.931642	0.831694	0.794488	0.409669	0.271693
<i>E2E</i>	0.256277	0.584633	0.905502	0.392704	0.038913

Table 6-4: CRE Wilcox Rank Sum Test Written Exam for Weeks 1 to 5

<i>Lab</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>CC</i>	2.42E-15	1.71E-05	0.000301	1.58E-05	0.002588
<i>CR</i>	5.30E-15	1.83E-08	0.067072	0.000468	1.72E-05
<i>RC</i>	3.17E-10	6.07E-07	0.200396	0.971741	2.32E-05
<i>RR</i>	0.000547	0.016242	0.082213	0.86105	0.165729
<i>RE</i>	0.007743	0.014142	0.850643	0.013938	0.514631
<i>EC</i>	0.012237	0.082588	0.796432	0.033548	0.387675
<i>ER</i>	0.210653	0.023292	0.818307	0.851237	0.861414
<i>EE</i>	0.291765	0.719679	0.100404	0.272137	0.678976

Table 6-5: CRE Wilcox Rank Sum Test Written Exam for Weeks 6 to 10

Similar to the results of the Wilcox Rank Sum Test for the Mouse Movements experiment in Section 5.4.1, it can be seen that Weeks 5 and 7 are key weeks. In Tables 6-2 and 6-4, Week 5 is shown to have a large amount of significant results for both the CA and the Written Exam tests. In this experiment, we also see significant differences in Week 4 and 6, Week 6 being data from the first in-lab programming exam.

6.4.2 Linear Regression

In this section the results of comparing each individual feature to the outcome in CA grades, and in Written Exam grades, on a week-to-week basis using Linear Regression are presented in Table 6-6, Table 6-7, Table 6-8, and Table 6-9. Any coefficient of more than 0 suggests a correlation between the feature and the grade and is in bold.

6.4.2.1 CA

This section presents the results of the Linear Regression tests for the CRE data divided by CA results.

<i>Lab</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
<i>C2C</i>	-0.09416	0.019169	0.123269	-0.03011	0.137087
<i>C2R</i>	-0.12198	0.02078	0.104036	0.030479	0.078994
<i>R2C</i>	-0.17784	-0.06182	-0.06688	-0.03215	-0.03886
<i>R2R</i>	-0.16761	-0.0544	-0.12613	-0.13983	-0.08048
<i>R2E</i>	-0.12605	0.028008	-0.0628	-0.12183	0.005408
<i>E2C</i>	-0.28039	-0.04157	-0.07542	-0.08401	0.005655
<i>E2R</i>	-0.1995	-0.05404	-0.06029	-0.10943	-0.04504
<i>E2E</i>	-0.16783	-0.05461	-0.05506	-0.10792	-0.04073

Table 6-6: CRE Linear Regression CA for Weeks 1 to 5

In Table 6-6, it is shown that C2C and C2R are consistently significantly correlated to CA results from Week 2 onwards (with the exception of C2C in Week 4). There is a spike in results in Week 5, with 4 significant features and the highest coefficient so far, >0.13 for C2C.

<i>Lab</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>C2C</i>	0.323132	0.069366	-0.09044	0.240351	-0.01086
<i>C2R</i>	0.325892	0.140258	-0.2907	0.095463	0.069104
<i>R2C</i>	0.16556	0.082785	-0.14443	-0.01828	0.104909
<i>R2R</i>	-0.01554	-0.04266	-0.08905	-0.01912	-0.1271
<i>R2E</i>	-0.01054	-0.01276	-0.14072	0.074252	-0.14515
<i>E2C</i>	-0.01401	-0.03234	-0.13799	0.061116	-0.14482
<i>E2R</i>	-0.03995	-0.00024	-0.09778	0.00053	-0.16711
<i>E2E</i>	-0.04922	-0.047	-0.10543	0.012922	-0.1654

Table 6-7: CRE Linear Regression CA for Weeks 6 to 10

In Table 6-7, C2C and C2R are significantly correlated to CA results for most weeks. The two highest coefficients are in Week 6, with coefficients of >0.32 for C2C and C2R. There is a spike in results in Week 9, with six significant features and a coefficient of >0.24 for C2C.

6.4.2.2 Written Exam

This section presents the results of the Linear Regression tests for the CRE data divided by Written Exam results.

<i>Lab</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
<i>C2C</i>	-0.04923	0.022117	0.168315	-0.05427	0.079982
<i>C2R</i>	-0.09021	0.02293	0.128905	-0.02908	0.032211
<i>R2C</i>	-0.13122	-0.03645	-0.06711	-0.07859	-0.06211
<i>R2R</i>	-0.13391	-0.03141	-0.11069	-0.13128	-0.06415
<i>R2E</i>	-0.08125	0.011948	-0.01036	-0.15574	-0.05909
<i>E2C</i>	-0.25009	-0.02221	-0.04012	-0.07877	-0.04361
<i>E2R</i>	-0.1076	-0.07904	-0.04046	-0.07044	-0.04374

Table 6-8: CRE Linear Regression Written Exam for Weeks 1 to 5

In Table 6-8, it is shown that C2C and C2R are significantly correlated to Written Exam results in Week 2, Week 3, and Week 5, similar to the results for CA shown in Table 6-2. Unlike Table 6-2, there is no clear spike in significant features in Week 5.

<i>Lab</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>C2C</i>	0.229808	0.072222	-0.02825	0.103773	-0.03633
<i>C2R</i>	0.212262	0.10061	-0.10334	0.0585	0.023019
<i>R2C</i>	0.091506	-0.03114	-0.14139	-0.07464	-0.00572
<i>R2R</i>	0.002575	-0.06386	-0.0602	-0.09711	-0.11559
<i>R2E</i>	0.004218	-0.08525	-0.08295	-0.03595	-0.12196
<i>E2C</i>	-0.0023	-0.09864	-0.0823	-0.03735	-0.11979
<i>E2R</i>	-0.01481	0.016484	-0.08543	-0.06302	-0.14547

Table 6-9: CRE Linear Regression Written Exam for Weeks 6 to 10

C2C and C2R are again mostly significant in Table 6-9. The highest Linear Regression coefficients for the Written Exam tests are in Week 6, which is also what was seen in the CA tests in Table 6-7.

The highest Linear Regression coefficient in relation to CA and Written Exam outcome are from Week 6, when the students have their first in-lab examination. Like in the Wilcoxon Rank Sum Test, the C2C movements and the C2R are the most significant indicators of student outcome according to these tests, the highest being in Week 6 and Week 9. The CRE movements are less related to the students' Written Exam than their CA outcome, but there are similar signs of higher and lower achieving students diverging in Week 6.

6.5 Neural Network Classifiers

In this section, the results of the Neural Net classifiers for the CA results and the Written Exam results will be presented and discussed.

6.5.1 Comparison of CA Classifiers

The CA classifier results are shown according to the week in the semester and the threshold used to select the features to be used as input for the Neural Networks in Table 6-10 and Table 6-11. The tables contain the classifier accuracy and the number of False Passes out of 25 students in the lower 50% of the grades, which would be labelled as Fails in a successful classification.

In the CRE classifiers comparison Tables 6-10 and 6-11, the results for the three thresholds are similar, with the threshold-0 being the most successful in that:

- i. It works from Week 3, while other thresholds do not work until Week 5 or 6, due to not having enough features.
- ii. In Table 6-10, the accuracy is the highest for almost every week, other than being slightly lower than threshold-0.1 in Week 6.
- iii. In Table 6-11, the number of False Passes is lowest in every week other than Week 5, when threshold-0.1 has a lower average False Pass number, but also a lower accuracy rate, and Week 7, when the number of average False Passes is slightly higher than threshold-0.1.

	3	4	5	6	7	8	9	10
0	0.677	0.706	0.73	0.758	0.763	0.766	0.767	0.767
0.1			0.684	0.764	0.763	0.762	0.761	0.756
0.15				0.737	0.728	0.737	0.719	0.728

Table 6-10: CRE CA Classifier Accuracy

The threshold-0 classifier is the most successful for predicting student CA outcome with CRE data. Looking at the averages of both the classifier accuracy and the False Passes in the first half and then the second half the semester, threshold-0 has the highest accuracy and the lowest False Pass rate.

	3	4	5	6	7	8	9	10
0	8.7	8.55	7.7	6.9	6.7	6.8	6.75	6.7
0.1			6.15	7.05	6.55	7.1	6.95	7.1
0.15				7.75	7.3	7.95	7.85	7.8

Table 6-11: CRE CA Classifier False Passes

6.5.2 Comparison of Written Exam Classifiers

The Written Exam classifier results are shown according to the week in the semester and the threshold used to select the features to be used as input for the Neural Networks in Table 6-12 and Table 6-13. In these comparison tables, the results show that, like the CA classifier, the threshold-0 classifier is the most successful for predicting student Written Exam outcome with CRE data:

- i. It works from Week 3, while other thresholds do not work until Week 6.
- ii. The accuracy is the highest for every week.
- iii. The number of False Passes is lowest in every week.

	3	4	5	6	7	8	9	10
0	0.731	0.733	0.762	0.781	0.770	0.777	0.780	0.776
0.1				0.718	0.707	0.706	0.709	0.704
0.15				0.71	0.71	0.713	0.713	0.712

Table 6-12: CRE Written Exam Classifier Accuracy

	3	4	5	6	7	8	9	10
0	11	10.2	11.25	7.5	7.25	7.5	6.8	7.15
0.1				7.9	8.35	8.75	8.35	8.15
0.15				8.15	8.35	8.1	8.05	8.05

Table 6-13: CRE Written Exam Classifier False Passes

The most successful classifiers for both CA and Written Exam results are those that use the Linear Regression threshold of 0. Any feature that has any correlation to outcome according to the Linear Regression tests is used in the Neural Network Classifier. The Appendix Section 10.6 presents more information about the successful classifiers, such as the Area Under Curve (AUC), the loss, and the recall as well as the numbers for True Fails, True Passes, False Fails, and False Passes. These attributes can also be seen for the less successful classifiers.

6.6 CRE Week-by-Week

In this section, the results of the Wilcoxon Rank Sum Test, the Linear Regression tests and the results of the classifiers will be examined on a week-to-week basis. The C2C and C2R movements return significant results from the Wilcoxon Rank Sum Test for every week throughout the semester, so this will not be mentioned explicitly in every weekly section.

Week 1

The Wilcoxon Rank Sum Test found significant differences in the C2C, C2R, and R2E movements in the CA tests, shown in Table 6-2, and significant differences in the C2C, C2R, R2E, and E2C movements in the Written Exam tests, shown in Table 6-4. The Linear Regression tests found no positive coefficients in Week 1 in relation to CA or Written Exams. There are not enough features with significant Linear Regression results to run classifiers at this point for any threshold.

Week 2

Both CA and Written Exam have significant differences between the two groups in R2E and well as C2C and C2R, shown in Tables 6-2 and 6-4. The Linear Regressions tests return very low but positive coefficients for C2C, C2R, and R2E in the CA test, shown in Table 6-6, and the Written Exam test, shown in Table 6-8. Although there are three usable features at this point, they are too highly correlated according to the VIF test (as mentioned in Section 4.4) and so there is no classifier for this set of data.

Week 3

The CA Wilcoxon Ranks Sum Test this week returns a significant result for the C2C, C2R, and R2C movements for CA, shown in Table 6-2, and the C2C, C2R, R2C, and R2E for the Written Exam, shown in Table 6-4. The Linear Regression tests only have positive values for the C2C and C2R movements for both CA and Written Exam, shown in Table 6-6 and Table 6-8. This week has the first classifiers, but only for threshold-0. The classifiers do relatively well for so early in the semester, 67% accuracy for CA as seen in Table 6-10, with 8.7 False Passes out of 25 Fails as shown in Table 6-11. The Written Exam classifier has a higher accuracy of 73% as shown in Table 6-12, but also a higher number of False Passes of 11 out of 25, shown in Table 6-13.

Week 4

This week's data has five significant features in CA and the Written Exam: C2C, C2R, R2C, R2R, and E2C, as shown in Tables 6-2 and 6-4. However, this is not reflected by the Linear Regression tests. Only C2R returns a positive coefficient for CA, shown in Table 6-6, and there are no positive coefficients for

the Written Exam, shown in Table 6-8. There are only threshold-0 classifiers this week, and the classifier for CA is slightly more accurate than the previous week with an accuracy of 70.6%, shown in Table 6-10, but the False Pass Rate is still high at 8.55 out of 25 Fails, shown in Table 6-11. The Written Exam classifier accuracy does not improve significantly in comparison to the previous week, as shown in Table 6-12, but the number of False Passes did improve slightly, reducing from 11 to 10.2, as shown in Table 6-13.

Week 5

There is a large jump in the Wilcox Rank Sum Test this week, with all of the features in the CA test returning a significant result, and six of the features in the Written Exam returning a significant result as shown in Tables 6-2 and 6-4. There is also a leap in Linear Regression results, with four features having a positive coefficient in the CA tests, shown in Table 6-6, and two in the Written Exam tests, shown in Table 6-8. The CA classifier for threshold-0 rises to 73% accuracy, shown in Table 6-10, and the number of False Passes drops to 7.7 out of 25, shown in Table 6-11. The first threshold-0.1 classifier has a lower accuracy of 68%, but also a lower False Pass number of 6.15 out of 25. The Written Exam classifier rises to 76%, shown in Table 6-12, but the False Pass is still high, at 11.25 out of 25, shown in Table 6-13.

Week 6

This week is an exam week, and so it is not surprising that it has a large number of significant results from the Wilcox Rank Sum Test. In both the CA and the Written Exam tests, shown in Table 6-3 and Table 6-5, all features other than ER and EE have significant results. The Linear Regression tests show the highest coefficient values for the semester for the CA and the Written Exam data, both in C2C and C2R of this week, shown in Table 6-7 and Table 6-9. These high values mean the first threshold-0.15 classifier can be run, and it has an accuracy of 73%, with a False Pass rate of 7.75 out of 25 for CA, shown in Table 6-10 and Table 6-11, and 71% accuracy and 8.15 False Passes out of 25 for Written Exams, shown in Table 6-12 and Table 6-13. These are not the most successful classifiers of the week, as the threshold-0 classifier for CA has a higher accuracy of 75.7%, and its number of False Passes drops to 6.9. The threshold-0.1 classifier accuracy rises to 76% but its False Pass number rises to 7.05. The threshold-0 is the most successful of the week, with an accuracy of slightly over 76%, and its False Pass number

dropping from 7.7 to 6.9. For the Written Exam classifiers, the threshold-0.1 is very similar to the threshold-0.15, with an accuracy of 71.8% and a False Pass number of 7.9. The threshold-0 classifier is the most successful, with its accuracy rising to 78%, and its False Pass number dropping from 11.25 to 7.5. The Written Exam classifiers do not improve significantly for the rest of the semester.

Week 7

Similar to week six, this week also has six significant features in the Wilcoxon Rank Sum Test for both CA and Written Exams, as shown in Table 6-3 and Table 6-5. The Linear Regression results show three significant features the CA tests in Table 6-7, and two in the Written Exam results in Table 6-9. The CA threshold-0.1 classifier again has an accuracy of 76%, but the False Pass number drops to 6.55. The CA classifier does not improve significantly after this point in the semester. The Written Exam threshold-0.1 classifier is slightly less successful this week than the previous week, with an accuracy of 70%, and a False Pass number of 8.35. The threshold-0 classifier for CA has a slight increase in accuracy to 76%, and a very slight drop in False Pass number to 6.7.

Week 8

This week the Wilcoxon Rank Sum Test only returns significant results for C2C and C2R movements in both CA and Written Exam results, as seen in Table 6-3 and Table 6-5. There are no positive Linear Regression results, which can be seen in Tables 6-7 and 6-9. As a result, there are no new features for the classifiers, and so the results are similar to the previous week.

Week 9

This week the Wilcoxon Rank Sum Test shows four significant features for the CA and the Written Exam groups, C2C, C2R, R2E, and E2C, shown in Tables 6-3 and 6-5. There are six positive coefficients in the CA Linear Regressions tests, and two in the Written Exam tests, shown in Table 6-7 and Table 6-9. The C2C in these results are both one of the highest coefficients in the data set for each grade type. Despite this, the CA classifiers this week show very little change from the previous week, with the biggest change being a drop in accuracy for the 0.15 CA classifier, which can be seen in Table 6-10. The Written Exam classifiers are also very similar to the previous week, with the biggest change being the drop in False Passes for the threshold-0 classifier from 7.5 to 6.8, shown in Table 6-13.

Week 10

Week 10 has four significant features for CA and three for the Written Exam according to the Wilcoxon Rank Sum Test results shown in Tables 6-3 and 6-5. The Linear Regression tests show two positive coefficients in the CA tests and one in the Written Exam tests, as shown in Table 6-7 and Table 6-9. Again, there are no large changes in the classifiers this week.

6.7 CRE Conclusions

The features C2C and C2R consistently have significant Wilcoxon Rank Sum Test results throughout the semester, implying that these are the key behaviours in student divergences. C2C and C2R movements may indicate if the student is able to write compilable code, and if they are moving on from compile attempts to writing code without syntax errors that can run. It is not surprising that at this stage, this is the key difference between the higher and lower achieving students.

The CA classifier becomes useful around Week 6, when the classifier results reach 76% and the number of False Passes drops to less than 7 out of 25 failing students. In the Wilcoxon Rank Sum Tests, we see that the most significant differences occur in Week 5, Week 6, and Week 7, pointing to this time as a key divergence point for the students.

The Written Exam classifier also becomes useful around Week 6 when the classifier results reach over 78% and the number of False Passes is drops to 7.5 out of 25 failing students. This is reflected in the Wilcoxon Rank Sum Test results, where the weeks with the most significant results are Week 4 to Week 7.

Weeks 5,6, and 7 are key weeks in the divergence of student behaviour, and the classifiers peak in success around this time, and do not significantly improve throughout the rest of the semester. This may be due to the increase in complexity of the assignments in Week 5, and the exam in Week 6.

These results largely replicate what was found in the paper in the Section 6.2, with some minor differences due to CA in this case including regular lab assignments, and not just lab exams. It has also been shown that CRE movements are successful in predicting Written Exam results, and not just coding assignment results. Like in the published paper, we have again found that the C2C and C2R movements are the most significant differences between the two groups.

7. Experiment 3: Complexity of Student Code

In this chapter, the COMPLEX data will be explored using the methods outlined in Section 4.3 and 4.4. The COMPLEX data comprises of two measurements of the complexity of student code, using compressed code size and nodes in parse trees generated from the student code. The results of the tests and HOG classifier are discussed.

7.1 Introduction to Code Complexity

In Neo Piagetian theory, as an individual learns a skill, they begin to learn to “clump” concepts relating to the skill together. In terms of programming, this allows for more complex code to be written, as the programmer becomes able to use different concepts together. In this experiment, we examine if the ability to write more complex code means that the student will ultimately be successful in their CA and Written Exams.

There is existing work that explores the connection between a student’s ability to write complex code and their exam outcome. The paper “*Utilizing student activity patterns to predict performance*” [45] uses the size of students’ code after removing comments and being compressed to predict student performance. In this chapter, we will examine if there is a connection between a student’s ability to write complex code and their exam outcome, and if so, how early this connection becomes apparent.

The MULE system records a student’s code every time they submit their code for automatic grading, resulting in a huge database of student code. For this experiment, only the highest graded submission for each assignment is used. Two techniques were used to generate metrics for the complexity of submitted code:

- 1) Comments were removed from the file, and the files were then compressed using Python’s `gzip` library. The size of the compressed file is used as an indication of the code complexity.
- 2) The Python `javalang` and `NetworkX` [57] libraries are used to create parse trees of the code. The more nodes in the generated parse tree, the more complex the code.

The Cyclomatic Complexity [58] model of examining the complexity of files was considered but it was found that the code written by first year programming students was too simple for this method.

7.2 Features

The data set for this experiment includes the file size from each assignment after comments are removed and the code is compressed, and the number of nodes in a parse tree created from each assignment.

In Figure 7.1 is an example of code a user might write and submit in MULE, and the parse tree generated from this code. The assignment asks the student to write code to check if a given age is old enough to vote or not, and to print an appropriate response to the terminal.

```
public class Selection
{
    public static void main (String args [])
    {
        int age = 10;
        //Check if age is 18 or over
        if (age >= 18)
        {
            System.out.println("Person can vote");
        }
        else
        {
            System.out.println("Person cannot vote");
        }
    }
}
```

Figure 7-1: Selection.java Sample Code

To generate the parse tree and file size data, the comments are removed, as shown in Figure 7-2.

```
public class Selection
{
    public static void main (String args [])
    {
        int age = 10;
        if (age >= 18)
        {
            System.out.println("Person can vote");
        }
        else
        {
            System.out.println("Person cannot vote");
        }
    }
}
```

Figure 7-2: Selection.java with Comments Removed

The generated parse tree is shown below in Figure 7-3. In this parse tree there are 12 nodes, so the node data for this user, for this assignment is recorded as 12.

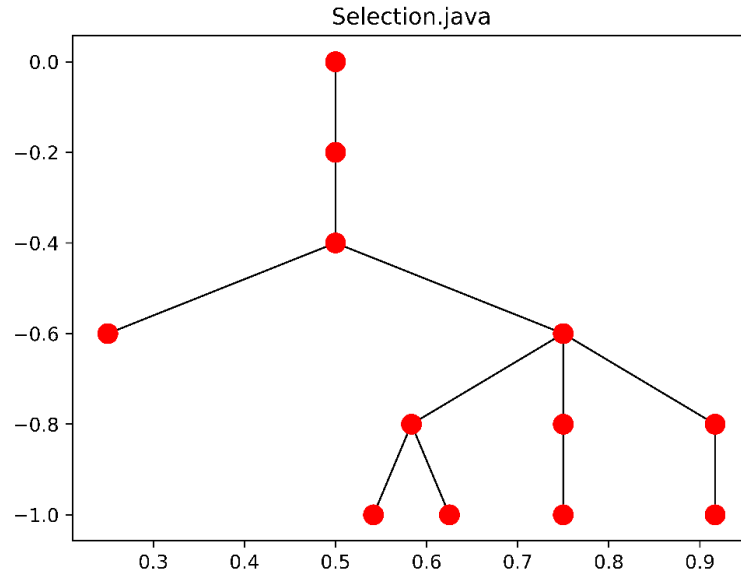


Figure 7-3: Parse Tree Generated from Selection.java

For the file size data, the code without the comment is compressed, and the resulting file size is 207 bytes, so the data for this user, for this assignment is recorded as 207.

For each assignment, the highest graded submission for each student is used. There is a total of 54 assignments consisting of 49 regular lab questions and 5 lab exam questions. The data set also includes the student's outcome in their final CA grade and their end of year Written Exam. A description of the weekly assignments can be found in Section 4.1.2.1.

7.3 Code Complexity Analysis

In this section, the measurements of code complexity are examined. The measurements are:

- 1) The compressed file sizes.
- 2) The number of nodes in generated parse trees of the submitted code.

The measurements are examined using the Wilcoxon Ranks Sum Test, and Linear Regression.

7.3.1 Wilcoxon Rank Sum Test: File Size

This section contains the results of the Wilcoxon Rank Sum Test on the results of the compressed file size with comments removed on each individual assignment during the semester. The two groups being compared are the top 50% and bottom 50% of students in the CA grades, and the Written Exam.

7.3.1.1 Continuous Assessment

In the Table 7-1, it can be seen that there are significant results from the Wilcoxon Ranks Sum test, implying differences in file size between the two student groups in Week 1 in the last question of the week, Question 4. All but one of the questions in Week 2, Question 2, have significant results. Again, only one of the Questions in Week 3 does not have significant results, Question 1. In Week 4, half of the questions, Question 3, 5, and 6 have significant results. Every question in Week 5 has significant results. These results suggest that there are differences in student file size, even in the first week of labs.

<i>Lab</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
<i>Q1</i>	0.604862	0.049804	0.449978	0.319243	5.41E-07
<i>Q2</i>	0.185383	0.954229	0.026081	0.500581	2.03E-09
<i>Q3</i>	0.152753	0.000102	0.001604	0.030253	5.91E-13
<i>Q4</i>	2.24E-06	0.002272	0.000137	0.945491	9.77E-16
<i>Q5</i>	N/A	4.55E-07	5.16E-10	0.016984	1.13E-10
<i>Q6</i>	N/A	4.37E-10	1.98E-08	3.14E-07	N/A
<i>Q7</i>	N/A	N/A	2.16E-08	N/A	N/A

Table 7-1: COMPLEX File Size Wilcoxon Rank Sum Test CA for Weeks 1 to 5

<i>Lab</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>Q1</i>	0.142143	0.232002	0.302718	0.572859	1.52E-05
<i>Q2</i>	0.00241	3.06E-08	7.10E-08	0.075603	1.32E-10
<i>Q3</i>	0.002336	8.94E-10	4.18E-05	0.003301	3.29E-07
<i>Q4</i>	0.978451	1.22E-16	6.82E-16	1.56E-09	2.28E-12
<i>Q5</i>	7.87E-10	N/A	4.95E-07	1.58E-07	2.34E-23
<i>Q6</i>	3.90E-21	N/A	4.56E-16	N/A	N/A

Table 7-2: COMPLEX File Size Wilcoxon Rank Sum Test CA for Weeks 6 to 10

In Table 7-2, it can be seen that most questions from Week 6 to Week 9 have significant results in the Wilcoxon Rank Sum test. In Week 10, every question

has significant results, implying significant differences in file size between two groups throughout the semester.

7.3.1.2 Written Exam

In the Table 7-3, it can be seen that, similar to the CA results, there are significant results from the Wilcoxon Ranks Sum test, implying differences in file size between the two student groups in Week 1 in the last question of the week, Question 4. Four of the questions in Week 2 have significant results and five of the questions in Week 3 have significant results, Question 1. In Week 4, only two of the questions have significant results. Every question in Week 5 has significant results. These results suggest that there are differences in student file size, though there are less significant results than those shown in the CA results in Table 7-1.

<i>Lab</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
<i>Q1</i>	0.665005	0.281836	0.05991	0.5943	0.00028
<i>Q2</i>	0.236985	0.323775	0.794231	0.057319	1.44E-08
<i>Q3</i>	0.39627	0.00072	0.00297	0.101875	2.50E-11
<i>Q4</i>	6.99E-05	0.03272	0.00639	0.819723	7.90E-12
<i>Q5</i>	N/A	0.00013	8.64E-08	0.00684	1.38E-09
<i>Q6</i>	N/A	1.55E-05	7.07E-05	9.27E-05	N/A
<i>Q7</i>	N/A	N/A	5.78E-06	N/A	N/A

Table 7-3: COMPLEX File Size Wilcoxon Rank Sum Test Written Exam for Weeks 1 to 5

In Table 7-4, it can be seen that, similar to the results for CA in Table 7-2, most questions from Week 6 to Week 9 have significant results in the Wilcoxon Rank Sum test. In Week 10, every question has significant results, implying significant differences in file size between the two groups throughout the semester.

<i>Lab</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>Q1</i>	0.674845	0.283349	0.610174	0.962299	0.00354
<i>Q2</i>	0.00748	1.57E-07	2.80E-05	0.03341	1.19E-06
<i>Q3</i>	0.00039	2.40E-06	0.00015	0.13539	0.0023
<i>Q4</i>	0.723586	1.99E-10	8.97E-11	2.72E-06	9.51E-06
<i>Q5</i>	2.42E-06	N/A	0.00036	6.75E-05	8.93E-12
<i>Q6</i>	6.41E-16	N/A	2.25E-10	N/A	N/A

Table 7-4: COMPLEX File Size Wilcoxon Rank Sum Test Written Exam for Weeks 6 to 10

In the file size results, there is a clear difference in the groups for most of the assignments, for both CA and the Written Exam. Lab 5 is notable in that the results for all of the assignments in this lab are significant. This is also true for

Week 10, but earlier occurrences of this are interesting, as they may indicate a divergence point and a key point for possible interventions.

Many of the weeks have significant p-values for the last assignment of the week. These assignments are often not visible until the students are in their assigned lab time, and so only the students with a good grasp of the material, who are able to formulate solutions based on their programming “tool-kit” and are able to work quickly, do well on these assignments. These assignments also tend to be the most challenging, as the questions for each lab increase in difficulty.

7.3.2 Wilcoxon Rank Test: Nodes

In the results of the Wilcoxon Rank Test using the nodes generated from the student code, and the students grades in CA, and the final Written Exam, there are fewer differences in the groups compared to the file size results in both CA and the Written Exam. However, the differences we do see follow the same overall pattern as seen in the file size test: the later assignments in each lab tend to have a significant difference between the two groups, and most of the assignments in Week 5 have significant features.

7.3.2.1 Continuous Assessment

Unlike the results from the file size tests in Table 7-1, there were no significant results in the first week of the labs. Instead, the first significant results were in Week 2, as can be seen in the Table 7-5. In Week 3, five of the questions have significant results, in Week 4, two of the questions have significant results. In Week 5, four of the five questions have significant results.

<i>Lab</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
<i>Q1</i>	0.998653	0.900584	0.086005	0.02603	3.13E-06
<i>Q2</i>	1	0.19393	0.324603	0.204264	2.48E-09
<i>Q3</i>	0.457608	0.00011	0.04159	0.275098	1.52E-09
<i>Q4</i>	0.70783	0.198016	0.00015	0.341463	1.12E-13
<i>Q5</i>	N/A	0.997306	9.10E-08	0.059452	0.826948
<i>Q6</i>	N/A	2.44E-07	3.51E-07	5.27E-06	N/A
<i>Q7</i>	N/A	N/A	9.29E-09	N/A	N/A

Table 7-5: COMPLEX Nodes Wilcoxon Rank Sum Test CA for Weeks 1 to 5

The results of the Wilcoxon Rank Sum test for the node data for the second half of the semester for CA, shown in Table 7-6, are similar to the results of the file

size data results shown in Table 7-2, with the exception of Question 3 in Lab 10, which is not significant in the nodes tests but is significant in the file size tests.

<i>Lab</i>	6	7	8	9	10
<i>Q1</i>	0.337621	0.288302	0.454547	0.052725	4.09E-05
<i>Q2</i>	0.00012	0.0017	6.58E-09	0.074773	6.25E-13
<i>Q3</i>	0.0049	3.48E-08	0.00014	0.00814	0.452005
<i>Q4</i>	0.903257	7.31E-15	4.04E-12	5.92E-09	2.43E-09
<i>Q5</i>	1.63E-18	N/A	2.04E-05	0.03391	1.34E-18
<i>Q6</i>	2.78E-15	N/A	7.41E-16	N/A	N/A

Table 7-6: COMPLEX Nodes Wilcox Rank Sum Test CA for Weeks 6 to 10

7.3.2.2 Written Exam

Like in the CA results in Table 7-5, there were no significant results in the first week of the labs. The first significant results are in Week 2, as can be seen in Table 7-7. In Week 3, five of the questions have significant results, and only one of the questions in Week 4 has a significant result. In Week 5, four of the five questions have significant results.

<i>Lab</i>	1	2	3	4	5
<i>Q1</i>	0.830238	0.322535	0.00169	0.056988	0.00557
<i>Q2</i>	1	0.665618	0.561424	0.770247	4.42E-08
<i>Q3</i>	0.911954	0.02369	0.079865	0.427524	1.10E-06
<i>Q4</i>	0.852682	0.657663	0.00013	0.525592	1.41E-10
<i>Q5</i>	N/A	0.99731	8.65E-06	0.11216	0.622649
<i>Q6</i>	N/A	0.00222	2.64E-05	0.00077	N/A
<i>Q7</i>	N/A	N/A	3.35E-06	N/A	N/A

Table 7-7: COMPLEX Nodes Wilcox Rank Sum Test Written Exam for Weeks 1 to 5

<i>Lab</i>	6	7	8	9	10
<i>Q1</i>	0.582665	0.270668	0.997306	0.387402	0.0247
<i>Q2</i>	0.00141	0.08103	1.36E-05	0.096176	2.61E-06
<i>Q3</i>	0.00131	7.25E-05	0.0011	0.308295	0.8014
<i>Q4</i>	0.914632	1.62E-08	6.27E-07	0.00015	0.00341
<i>Q5</i>	1.99E-13	N/A	0.00416	0.230357	5.75E-10
<i>Q6</i>	1.39E-09	N/A	7.79E-10	N/A	N/A
<i>Q7</i>	N/A	N/A	N/A	N/A	N/A

Table 7-8: COMPLEX Nodes Wilcox Rank Sum Test Written Exam for Weeks 6 to 10

The results of the Wilcoxon Rank Sum test for the node data for the second half of the semester for the Written Exam, shown in Table 7-8, are similar to the results of the file size data results shown in Table 7-2, with the exception of Lab 9, which has only one significant result in the nodes tests but has three significant results in the file size tests.

What is interesting is that Q5 in Week 5, does not return a significant result in CA or in the Written Exam, as shown in Table 7-6 and Table 7-8, despite it being significant according to the file size tests. It may be because the nodes method of measuring complexity considers repetition, for example copy-and-pasted print statements, as increased complexity, whereas the compression method reduces complexity in the event of copy-and-pasted code. If the code written for Week 5 Q5 by the lower achieving students involves a lot of repetitive code from students attempting to solve the problem, this could result in a similar number of nodes, but would have a lower compressed file size. Overall, there are less significant differences between the two groups in the node data tests than in the file size tests, perhaps suggesting that the file size tests are more successful and more useful when examining the differences between the two groups.

7.3.3 Linear Regression: File Size

In this section, the Tables 7-9, 7-10, 7-11, and 7-12 present the results of the Linear Regression tests finding the correlation between the file size data and the CA results, and file size and Written Exam results.

7.3.3.1 *Continuous Assessment*

In the Linear Regression coefficients for the first half of the semester, presented in Table 7-9, it can be seen that the first significant coefficients are in Week 2. There are also positive coefficients in Week 3 and Week 4, with the higher number questions (Q5 and Q6 rather than Q1 and Q2 for example) tending to be positive. This may be because the questions tend to get more difficult, and also the students often would not see the last question in a lab until the day of the lab. All of the questions in Week 5 have positive coefficients, with Q4 and Q3 being the two highest coefficients so far.

	1	2	3	4	5
Q1	-0.08431	-0.04257	-0.04562	-0.12931	0.06685
Q2	-0.11132	-0.05622	-0.03485	-0.063	0.1101
Q3	-0.06388	0.00326	-0.00282	-0.06638	0.18698
Q4	-0.02778	-0.01139	0.05398	-0.05592	0.24215
Q5	N/A	0.01306	0.09361	-0.0061	0.16888
Q6	N/A	0.17289	0.10388	0.09624	N/A
Q7	N/A	N/A	0.14446	N/A	N/A

Table 7-9: COMPLEX File Size Linear Regression CA for Weeks 1 to 5

In the Linear Regression coefficient values shown in Table 7-10, we can see another high coefficient value in Week 6, Q6 of >0.27. Week 7 has three positive coefficient values out of four questions, and Week 8 has five positive coefficients out of six. Week 9 has only two positive coefficients out of five questions. Week 10 has four positive coefficients out of five questions, and the highest coefficient of the semester for CA file size in Q5.

	6	7	8	9	10
Q1	-0.04343	-0.05054	-0.07581	-0.05649	0.064731
Q2	-0.00693	0.01858	0.03763	-0.05	0.07166
Q3	0.01038	0.06305	0.0652	-0.00358	0.09888
Q4	-0.06164	0.24354	0.19592	0.06171	0.16748
Q5	-0.03446	N/A	0.03446	0.08994	0.33618
Q6	0.27555	N/A	0.18403	N/A	N/A

Table 7-10: COMPLEX File Size Linear Regression CA for Weeks 6 to 10

7.3.3.2 Written Exam

Similar to the results shown in Table 7-9 for the Linear Regression results for CA, the first positive coefficient for the Written Exam is in Week 2, but only in one question, Q6. Week 3 has four positive coefficients and Week 4 has one. Week 5 has four positive coefficients out of the five questions, including the two highest coefficients so far, Q5 and Q4.

In the Table 7-12, we see that the Written Exam Week 6 has the highest coefficient of the semester in Q6, and has four positive coefficients, compared to the CA results that has only two, as shown in Table 7-10. Week 7 has three positive

coefficients, Week 8 has only three positive coefficients, compared to the CA results of five of the six questions. Week 9 has two positive coefficients. Week 10 has five positive results out of five questions.

<i>Lab</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
<i>Q1</i>	-0.04822	-0.04372	-0.02446	-0.07147	-0.01261
<i>Q2</i>	-0.04642	-0.03803	-0.03242	-0.02198	0.09531
<i>Q3</i>	-0.04477	-0.00919	-0.0107	-0.03101	0.07761
<i>Q4</i>	-0.01851	-0.00996	0.05129	-0.03717	0.18156
<i>Q5</i>	N/A	-0.01606	0.04458	-0.00136	0.13225
<i>Q6</i>	N/A	0.06744	0.05156	0.03379	N/A
<i>Q7</i>	N/A	N/A	0.10393	N/A	N/A

Table 7-11: COMPLEX File Size Linear Regression Written Exam for Weeks 1 to 5

<i>Lab</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>Q1</i>	-0.03499	-0.02747	-0.03438	-0.05838	0.00045
<i>Q2</i>	0.0131	0.09852	-0.04604	-0.04103	0.00783
<i>Q3</i>	0.07007	0.06452	-0.00717	-0.02483	0.02207
<i>Q4</i>	-0.06461	0.12539	0.10896	0.00793	0.06998
<i>Q5</i>	0.06142	N/A	0.00283	0.02659	0.16175
<i>Q6</i>	0.19218	N/A	0.09417	N/A	N/A

Table 7-12: COMPLEX File Size Linear Regression Written Exam for Weeks 6 to 10

In the Linear Regression tests, we see that Week 5 has the highest and the most positive coefficients in the first half of the semester for both CA and the Written Exam, although there are also some high coefficient values in Week 2 and Week 3. The highest coefficients are Week 6 Q6 and Week 10 Q5, which are the two final questions in the formal in-lab examinations, suggesting that these are key assignments.

7.3.4 Linear Regression Test: Node Data

In this section, the Tables 7-13, 7-14, 7-15, and 7-16 present the results of the Linear Regression tests finding the correlation between the node data and the CA results, and the node data and Written Exam results.

7.3.4.1 Continuous Assessment

In the results of the Linear Regression tests for CA using the nodes data, shown in Table 7-13, we see that the first positive coefficient is in Week 2, with Q6. Week 3 has three positive coefficients, including a value of >0.17 for Q7,

higher than any result in Week 3 for the file size CA results shown in Table 7-9. Week 4 has one positive coefficient, and Week 5 has four positive coefficients out of five questions, including Q4, which has a value of >0.22 , the highest coefficient from the first half of the semester.

<i>Lab</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
<i>Q1</i>	-0.0558	-0.06862	-0.04202	-0.1649	0.06231
<i>Q2</i>	-0.0558	-0.04733	-0.06923	-0.07483	0.08168
<i>Q3</i>	-0.02669	-0.012	-0.01884	-0.092	0.17136
<i>Q4</i>	-0.06333	-0.05059	0.048169	-0.04654	0.22143
<i>Q5</i>	N/A	-0.06125	0.0657	-0.02449	-0.05299
<i>Q6</i>	N/A	0.0685	0.07059	0.04011	N/A
<i>Q7</i>	N/A	N/A	0.17182	N/A	N/A

Table 7-13: Nodes Linear Regression CA

In the Table 7-14, we can see the Week 6 results have two positive coefficients, including the highest result of the semester of >0.3 for Q5. Week 7 has one positive coefficient out of four questions, and Week 8 has four out of six questions, including one of the largest values, of >0.24 for Q6. Week 9 has one positive value, and Week 10 has three, similar to the file size results in Table 7-10.

<i>Lab</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>Q1</i>	-0.09061	-0.05482	-0.10363	-0.04057	-0.00531
<i>Q2</i>	-0.19423	-0.07044	0.11736	-0.04198	0.06782
<i>Q3</i>	-0.04442	-0.02721	0.0771	-0.00062	-0.02662
<i>Q4</i>	-0.12373	0.10363	0.15545	0.10106	0.03611
<i>Q5</i>	0.30048	N/A	-0.04618	-0.04136	0.18728
<i>Q6</i>	0.16537	N/A	0.2408	N/A	N/A

Table 7-14: Nodes Linear Regression CA

7.3.4.2 Written Exam

In the results of the Linear Regression tests for Written Exam using the nodes data, shown in Table 7-15, we see that the first positive coefficients are in Week 3, with Q4, Q5, Q6, and Q7. Week 4 has no positive coefficients, and Week 5 has three positive coefficients out of five questions, including Q4 which has a value of >0.17 , the highest coefficient from the first half of the semester.

In the Table 7-16, we can see the Week 6 results have three positive coefficients, including the highest result of the semester of >0.23 for Q5. Week 7

has two positive coefficients out of four questions, and Week 8 has four out of six questions, including one of the largest values, of >0.24 for Q6. Week 9 has one positive value, and Week 10 has three, similar to the file size results in Table 7-14.

<i>Lab</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
<i>Q1</i>	-0.03571	-0.04345	-0.00677	-0.10648	-0.01643
<i>Q2</i>	-0.03571	-0.03223	-0.03321	-0.06021	0.1092
<i>Q3</i>	-0.05851	-0.02703	-0.00905	-0.04325	0.0655
<i>Q4</i>	-0.04116	-0.0308	0.06666	-0.04836	0.1728
<i>Q5</i>	N/A	-0.04443	0.0505	-0.02672	-0.01136
<i>Q6</i>	N/A	-0.00093	0.05415	-0.00202	N/A
<i>Q7</i>	N/A	N/A	0.11135	N/A	N/A

Table 7-15: Nodes Linear Regression Written Exam

<i>Lab</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>Q1</i>	-0.03455	-0.03887	-0.03531	-0.05092	-0.04349
<i>Q2</i>	-0.13809	0.0115	0.03681	-0.02069	0.01016
<i>Q3</i>	0.00795	-0.0221	0.02175	-0.02328	-0.04914
<i>Q4</i>	-0.04601	0.00614	0.06634	0.02795	0.00452
<i>Q5</i>	0.23399	N/A	-0.04025	-0.02984	0.11079
<i>Q6</i>	0.04608	N/A	0.14652	N/A	N/A

Table 7-16: Nodes Linear Regression Written Exam

The results of the Linear Regression tests for the nodes data are similar to the file size data, in that again, Week 5 has the highest and the most positive coefficients in the first half of the semester for both CA and Written Exam, but unlike the file size results, and similar to the results of the node Wilcox Rank Sum test in Section 7.3.2.1, Q5 in Week 5 is not a positive coefficient. Overall, the Linear Regression tests with node data show less correlations than the file size tests.

7.4 Neural Network Classifiers

In this section, the weekly classifiers run using the node and the file size data will be discussed.

7.4.1 Comparison of CA Classifiers

In the Table 7-17, we can see that from Week 2 to Week 4, the threshold-0 classifiers have the highest accuracy when predicting student success according to

CA, and from Table 7-18, threshold-0 classifiers have the lowest number of False Passes, and so are the most successful. Although the threshold-0 classifiers are the most successful for these two weeks, it is important to note that the number of False Passes is very high (just under 13 False Passes out of 25 fails in Week 2), and the classifiers cannot be used reliably at this point.

<i>Lab</i>	2	3	4	5	6	7	8	9	10
0	0.669	0.709	0.722	0.788	0.809	0.828	0.83	0.828	0.847
0.1	N/A	0.688	0.708	0.795	0.813	0.823	0.844	0.843	0.859
0.15	N/A	0.684	0.684	0.81	0.821	0.841	0.853	0.858	0.876

Table 7-17: COMPLEX CA Classifier Accuracy

From Week 5 onwards, it can be seen in Tables 7-17 and 7-18 that the threshold-0.15 classifiers have the highest accuracy, and the lowest number of False Passes (other than in Week 6 when the threshold-0.1 classifiers have a slightly lower number of False Passes), showing that the threshold-0.15 classifiers are the most successful CA classifiers from Week 5 onward.

<i>Lab</i>	2	3	4	5	6	7	8	9	10
0	12.75	9.15	8.6	6.75	6.15	5.15	5.5	5.05	4.1
0.1	N/A	9.5	9.05	6.65	5.85	6	4.4	4.5	3.4
0.15	N/A	11.3	11.35	6.05	5.95	4.8	4.2	4.3	3.1

Table 7-18: COMPLEX CA Classifier False Passes

In Week 10, the threshold-0.15 classifier has an accuracy of 87.6%, with a False Pass rate of just 3.1 of 25 students. This is the most successful classifier of the three experiments so far, comparing MM, CRE, and COMPLEX.

	<i>Average Accuracy</i>	<i>Average False Pass</i>
0	0.7396667	8.1666667
0.1	0.7303333	8.4
0.15	0.726	9.5666667

Table 7-19: Compare Early Semester CA Classifiers

	<i>Average Accuracy</i>	<i>Average False Pass</i>
0	0.8284	5.19
0.1	0.8364	4.83
0.15	0.8498	4.47

Table 7-20: Compare Late Semester CA Classifiers

Comparing the first and the second half of the semester separately and looking at the averages of the accuracy and the False Passes, the best classifiers for the first half of the semester is the threshold-0.1, and the best classifiers for the second half are the threshold-0.15 classifiers.

7.4.2 Comparison of Written Exam Classifiers

In this section, the results of the Written Exam classifiers will be compared and discussed.

Similar to the CA classifiers shown in Tables 7-17 and 7-18, the Tables 7-19 and 7-20, showing the accuracy and the False Pass numbers for the Written Exam classifiers, show that from Week 2 to 5, the threshold-0 classifiers have the highest accuracy (except for Week 5 when 0.1-threshold has a very slightly higher accuracy) and the lowest False Pass number for the three weeks. The threshold-0 classifiers are therefore the most successful from Week 2 to Week 5.

<i>Lab</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>0</i>	0.706	0.7	0.704	0.756	0.749	0.752	0.738	0.751
<i>0.1</i>	N/A	N/A	0.708	0.776	0.776	0.774	0.782	0.767
<i>0.15</i>	N/A	N/A	N/A	0.782	0.778	0.779	0.779	0.783

Table 7-21: COMPLEX Written Exam Classifier Accuracy

In Table 7-19, we can see that the classifiers for threshold-0.1 and threshold-0.15 are similar, and higher, than threshold-0. The accuracy is slightly higher for threshold-0.15 overall, but the data in Table 7-20 shows that the threshold-0.1 classifiers are more reliable as they have lower False Pass numbers than the threshold-0.15 classifiers. Therefore, the most successful classifiers from Week 6 to Week 10 are the threshold-0.1 classifiers.

<i>Lab</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>0</i>	9.35	9.5	9.15	7.1	7.2	7.45	7.65	6.35
<i>0.1</i>	N/A	N/A	9.45	6.8	6.65	6.55	6.45	6.4
<i>0.15</i>	N/A	N/A	N/A	7.2	7.85	7.4	7.45	7.15

Table 7-22: COMPLEX Written Exam Classifier False Passes

	<i>Average Accuracy</i>	<i>Average False Pass</i>
<i>0</i>	0.7033333	9.3333333
<i>0.1</i>	0.708	9.45
<i>0.15</i>	N/A	N/A

Table 7-23: Comparing Early Semester Written Exam Classifiers

Comparing the first and second half of the semester separately and looking at the averages of the accuracy and the False Passes, the best classifier for the first half of the semester is the threshold-0, and the best classifier for the second half is the threshold-0.15.

	<i>Average Accuracy</i>	<i>Average False Pass</i>
0	0.7492	7.15
0.1	0.775	6.57
0.15	0.7802	7.41

Table 7-24: Compare Late Semester Written Exam Classifiers

7.5 Code Complexity Week-by-Week

Week 1

This week, the Wilcoxon Rank Sum test found significant differences in the file size and nodes data of higher achieving and lower achieving groups in CA for Q4. There are already positive Linear Regression results from the file size data in both CA and Written Exam tests from Q4, but not in the nodes data. The Q4 assignment asks students to declare variables, perform, and store the results of mathematical operations and print the stored results.

Week 2

The Wilcoxon Rank Sum test shows significant results for file size data CA tests for questions Q1, Q3, Q4, Q5, and Q6, and for Q3, Q4, Q5, and Q6 of the Written Exam file size tests. The nodes tests show significant differences in Q3 and Q6 in the CA tests and the Written Exam tests. Questions Q3, Q5, and Q6 all have positive coefficients from the file size Linear Regression tests in CA, but only Q6 is positive for the Written Exam. In the node data Linear Regression tests, only Q6 is positive in the CA tests, and no questions have positive coefficients in relation to the Written Exams. Only the CA classifier has enough features to run this week. The threshold-0 classifier already has an accuracy of 66.9% but has a high False Pass number of 12.75.

Week 3

Every question this week from the CA file size Wilcoxon Rank Sum Tests has significant results, except Q1. In the Written Exam test, every question has significant results except the first two. In the nodes Wilcoxon Rank Sum tests, again

in the CA results, every question has significant results, except Q1. In the Written Exam results, Q1, Q4, Q5, Q6, and Q7 all have significant results. In the Linear Regression tests for file size, the last four questions have positive coefficients in relation to CA and to the Written Exams. In the Linear Regression tests for the node data, the last three questions have a positive coefficient for CA, and the last four questions have a positive coefficient in relation to Written Exam data. All three classifier thresholds are able to run this week with CA data, the most successful being threshold-0, which already has an accuracy of 70.9%, and the False Pass number has dropped from 12.75 to 9.15. This week has the first Written Exam classifier, which has an accuracy of 70%, and a False Pass number of 9.35.

Week 4

The Wilcoxon Rank Sum test for file size shows significant differences in the CA groups for Q3, Q5, and Q6 and in the Written Exam groups for Q5 and Q6. In the nodes Wilcoxon Rank Sum test, Q1 and Q6 have significant results in CA, and Q6 only in the Written Exam. In the Linear Regression results for file size, Q6 has a positive coefficient in relation to CA and Written Exams. In the results for the nodes data Linear Regression tests, the only positive coefficient is for Q6 in relation to CA. The threshold-0 and threshold-0.1 increase in success this week for CA, with threshold-0 going from 70.9% accuracy to 72.2%, and the False Pass dropping from 9.15 to 8.6. The threshold-0.1 classifier goes from 68.8% accuracy to 70.8%, and the False Pass drops slightly from 9.5 to 9.05. The threshold-0.15 classifier remains the same, as it does not gain any additional features this week. The Written Exam threshold-0 classifier drops slightly in success this week.

Week 5

The Wilcoxon Rank Sum test for file size data returns a significant result for every question for both CA and the Written Exam. In the nodes Wilcoxon Rank Sum test, four of the five questions have significant results in relation to both CA and Written Exam classifications. In the Linear Regression tests for file size data, all five of the questions this week have positive coefficients for CA, with Q4 being the highest coefficient so far in this semester. The results for the Written Exam have four positive coefficients, with Q4 again being the highest of the semester for Written Exam. For the nodes Linear Regression tests, four of the five questions have positive coefficients in relation to CA, and three of the five have positive coefficients in relation to the Written Exam. There is a large leap in effectiveness of the CA classifiers at this point. For the CA classifiers, the threshold-0 classifier

increases in accuracy from 72% to almost 79%, and the False Pass drops from 8.6 to 6.75. The threshold-0.1 classifier increases from 70.8% to 79.5%, and the False Pass numbers drop from 9.05 to 6.65. The biggest change is in the threshold-0.15 classifier. Its accuracy increases from 68.4% to 81%, and the False Passes drop from 11.35 to 6.05, making it the most successful classifier this week. However, the threshold-0 classifier for the Written Exam is mostly the same as the previous week. The threshold-0.1 classifier is run for the first time and has similar results to the threshold-0 classifier, with an accuracy of 70.8% and 9.45 False Passes.

Week 6

In the Wilcoxon Rank Sum test for file size and for nodes, the questions Q2, Q3, Q5, and Q6 return significant results in relation to both CA and the Written Exam. In the Linear Regression tests for file size, Q3 and Q6 both have positive coefficients in relation to CA. In the nodes test, Q2, Q3, Q5, and Q6 have positive coefficients. In the Linear Regression nodes test, Q5 and Q6 have positive coefficients for CA, and Q3, Q5, and Q6 have positive coefficients for Written Exams. Again, each of the classifiers for CA improves this week. The threshold-0 classifier accuracy increases from 78.8% to 80.9%, and the False Passes drop from 6.75 to 6.15. The threshold-0.1 classifier increases from 79.5% to 81.3%, and the False Pass rate drops from 6.65 to 5.85. The threshold-0.15 classifier increases from 81% to 82.1%, and False Pass rate drops very slightly from 6.05 to 5.95, making it the most successful classifier again this week. In the Written Exam classifiers, there is a leap in success this week. The threshold-0 classifier goes from 70% to 75.6%, and the False Passes drop from 9.15 to 7.1. In the threshold-0.1 classifier, the accuracy goes from 70.8% to 77.6%, and the False Pass drops from 9.45 to 6.8. The threshold-0.15 classifier runs with Written Exam data for the first time and the accuracy is 78%, while the False Pass number is 7.2. It is arguable that threshold-0.1 or threshold-0.15 are the most successful this week.

Week 7

In the Wilcoxon Rank Sum tests for file size and for nodes, the Q2, Q3, Q4, are all significant for both CA and Written Exam groups. In the Linear Regression tests for file size, Q2, Q3, and Q4 has positive coefficients for CA and for the Written Exam. In the node Linear Regression tests, Q4 is positive for CA, and Q2 and Q4 are positive for the Exam. The threshold-0 CA classifier increases in accuracy from 80.9% to 83%, and the False Pass drops from 6.15 to 5.15. The threshold-0.1 CA classifier stays roughly the same. The threshold-0.15 CA

classifier accuracy goes from 82.1% to 84.1%, and the False Pass drops from 5.95 to 4.8. Threshold-0.15 is again the most successful CA classifier. For the Written Exam, the classifiers stay around the same as the previous week, with some slight drops in accuracy and increases in False Pass numbers.

Week 8

The Wilcoxon Rank Sum Test for both file size and nodes data found significant differences between the groups in Q2, Q3, Q4, Q5, and Q6 in CA and Written Exam. The Linear Regression tests for file size have positive coefficients for Q2, Q3, Q4, Q5, and Q6 in relation to CA, and in Q4, Q5, and Q6 for Written Exam. In the Linear Regression tests for nodes, the coefficients for Q2, Q3, Q4, and Q6 are all positive for CA and for Written Exams. The threshold-0 classifier for CA does not improve, the accuracy is slightly higher, and the False Pass rate increases slightly. The threshold-0.1 classifier increases in accuracy from 82.3% to 84.4%, and the number of False Passes drops from 6 to 4.4. The threshold-0.15 classifier accuracy rises from 84.1% to 85.3%, and the number of False Passes drops from 4.8 to 4.2. The Written Exam classifiers are again mostly the same as the previous week.

Week 9

The Wilcoxon Rank Sum test for file size finds significant differences between the CA groups for questions Q3, Q4, and Q5, and for the Written Exam groups, questions Q2, Q4, and Q5. For nodes it finds differences for questions Q3, Q4, and Q5 for CA, and for Q4 only for the Written Exam. For the Linear Regression tests for file size, the questions Q4 and Q5 have positive coefficients in relation to both CA and Written Exams. For nodes, the question Q4 only has positive coefficients for both CA and Written Exams. The classifiers for this week do not greatly change from the week before for CA. For the Written Exam, there is an increase in accuracy for threshold-0.1, from 77.4% to 78.2%, with a slight decrease in False Passes from 6.55 to 6.45.

Week 10

In this week, the students have their second lab exam. The Wilcoxon Rank Sum test for file size finds significant differences in the higher and lower achieving groups for every question for both CA and Written Exam. The nodes test found differences in four of the five questions, for CA and for Written Exam. The Linear Regression tests for file size show the last four of the five questions have positive

coefficients relating to CA, and all of the questions have positive coefficients relating to Written Exams. The Linear Regression tests for node data show positive coefficients for Q2, Q4, and Q5 in relation to CA and Written Exams. The CA classifiers have another leap in success at this point, with the threshold-0 classifier increasing in accuracy from 82.8% to 84.7%, and the False Passes dropping from 5.05 to 4.1. The threshold-0.1 classifier accuracy increases from 84.3% to 85.9%, and the False Passes drop from 4.5 to 3.4. Finally, the 0.15-threshold CA classifier increases to 87.6% accuracy, and the False Passes drop to just 3.1 out of 25 Fails (12.4%), making it the most successful classifier in the COMPLEX experiment. The threshold-0 classifier for Written Exams accuracy increases to 75.1%, and the number of False Passes drops to 6.35. The threshold-0.1 classifier drops to just under 77% and the False Passes stay the same. The threshold-0.15 classifier increases in accuracy slightly to 78.3%, and the number of False Passes drops slightly to 7.15.

7.6 Discussion of Results

In COMPLEX, many of the weeks have significant results from the Wilcoxon Rank Sum test, particularly for file size, for the last assignment of the week. These assignments are often not visible until the students are in their assigned lab time, and so only the students with a good grasp of the material, who are both able to formulate solutions based on their programming “tool-kit” and are able to work quickly, do well on these assignments. These assignments also tend to be the most challenging, as the questions for each lab increase in difficulty.

The threshold-0.15 classifiers are the most successful in classifying success in CA and the Written Exam, as they use the most relevant data. However, for early classification, the threshold-0 classifier that uses whatever data is available can also be useful. Week 5 data is the most important for CA according to the Linear Regression results, but Week 6 is more important for the Written Exam.

The code complexity classifiers were successful, with the highest accuracy of 87% in Week 10 for the CA classifier. The Wilcoxon Rank Sum tests were also successful, in that they showed a significant difference between the highest and lowest performing students throughout the semester, even in Week 1 in the case of the file size tests, as shown in Table 7-1 and Table 7-3. These tests worked so well that it was difficult to pinpoint key dates in the semester when the divergence in the two groups appears. From the file size tests, it seems that, as in the previous experiments, Week 5 is the key point of divergence, as all the assignments in this

week have significant differences between the two groups. It could also be argued that Week 2 and Week 3 are key points, as most of the assignments in these labs show significant differences.

The classifiers with threshold-0.15, using features with a Linear Regression relationship of more than 0.15, has the lowest percentage of False Passes for the last six weeks, (except Week 6, when the threshold-0.1 classifier is slightly lower). However, for Week 3 and Week 4, the classifier has a much higher rate of False Passes, possibly due to the fact that there are not many input features at this point.

There is a spike in CA classifier success in Week 5, with the threshold-0 classifiers jumping from 72% accuracy in Week 4 to 79% in Week 5, the threshold-0.1 classifiers jumping from 71% to 80% and the threshold-0.15 classifiers jumping from 68% to 81%. These additional assignments are Q2, Q3, Q4, and Q5 in the above Wilcoxon Rank Sum test results in Tables 7-1 and 7-5, and Linear Regression Tables 7-9 and 7-13. In both the Wilcoxon and Linear Regression tests, these assignments are among the most significant both in terms of difference between the two groups, and in association with outcome. Q4 is the most important of these, as it has one of the highest correlations with CA outcome according to the Linear Regression tests, but comes earlier in the semester than other assignments with similar correlations. Q4 asks the students to write code that takes a string as user input and to print out that string with the first and last characters swapped. On average, students who did well in their CA wrote more complex code for the Week 5 Q4 assignment, implying that this particular assignment is a key indicator as to whether or not a student is likely to do well in the module or not.

8. Experiment 4: The HOG Classifier

8.1 Introduction to HOG

This section discusses the final classifier that uses all three of the previously explored data types: MM, CRE, and COMPLEX. From the experiments on these three data types, a final classifier was built using the same method as in Chapters 5, 6, and 7 to use all three data types in order to classify students as being in the top or bottom 50% of the class grades under two categories:

- 1) Continuous Assessment.
- 2) Written Exam.

8.2 Features

The features for the HOG Neural Network Classifier are those described in the previous experiments features sections:

- MM
 - This is the Mouse Movement (MM) data type as described in Section 5.3.
- CRE
 - This is the Compile, Run, and Evaluate (CRE) data type as described in Section 6.4.
- COMPLEX
 - This is the Code Complexity (COMPLEX) data type, as described in Section 7.2.

Note that although all three data sets are used for this classifier, the Linear Regression results are never higher than 0.1 for MM data, and so the threshold-0.1 and threshold-0.15 classifiers only use CRE and COMPLEX inputs.

8.3 Neural Network Classifiers

In this section, the results of the HOG classifiers, run with the three data types that were explored in Chapters 5, 6 and 7, will be examined.

8.3.1 Comparison of Continuous Assessment Classifiers

The accuracy of the threshold-0, threshold-0.1, and threshold-0.15 HOG classifiers for CA can be seen in Table 8-1, and the number of False Passes (out of a possible total of 25) can be seen in Table 8-2.

<i>Lab</i>	2	3	4	5	6	7	8	9	10
0	0.654	0.723	0.74	0.789	0.816	0.838	0.836	0.793	0.823
0.1	N/A	0.719	0.711	0.79	0.813	0.842	0.853	0.846	0.858
0.15	N/A	0.694	0.642	0.812	0.791	0.846	0.851	0.856	0.876

Table 8-1: HOG CA Classifier Accuracy

In Weeks 2 to 4, the 0-threshold classifiers have the highest accuracy, as seen in Table 8-1, and the lowest number of False Passes as seen in Table 8-2 (although threshold-0.1 has the same False Pass number for Week 3). The most successful classifier threshold in Week 5 is threshold-0.15, as it has the highest accuracy and lowest False Pass number.

<i>Lab</i>	2	3	4	5	6	7	8	9	10
0	13	8.45	8.05	6.9	5.8	4.6	4.7	5.45	4.5
0.1	N/A	8.45	8.45	6.55	5.65	5.3	4.35	3.1	3.3
0.15	N/A	9.45	13	6.2	5.9	4.75	4.45	4.35	3.3

Table 8-2: HOG CA Classifier False Passes

However, because the threshold-0.15 False Pass numbers are so high in Week 3 and Week 4 and the accuracy is so low compared to threshold-0, the threshold-0 classifiers are the most successful overall in the first half of the semester, from looking at the averages of the Weeks 3,4, and 5, as can be seen in Table 8-3. Week 2 was not included in this, as only the threshold-0 classifier had any results for this week.

	<i>Average Accuracy</i>	<i>Average False Pass</i>
0	0.750667	7.8
0.1	0.74	7.816667
0.15	0.716	9.55

Table 8-3: Averages of Early Semester Classifiers for CA

Looking at the second half of the semester, from Week 6 to Week 10, the most successful classifiers, according to the averages of the accuracy and the False Passes, is the threshold-0.1 classifiers, as can be seen in Table 8-4. Although the

threshold-0.15 accuracy is higher than the threshold-0.1 accuracy, it also has a slightly higher average False Pass rate, and the threshold-0.1 has a lower average False Pass rate.

	<i>Average Accuracy</i>	<i>Average False Pass</i>
<i>0</i>	0.8212	5.01
<i>0.1</i>	0.8424	4.34
<i>0.15</i>	0.844	4.55

Table 8-4: Averages of Late Semester Classifiers for CA

The most successful classifiers for early semester classifications are the threshold-0 classifiers, and the most useful classifiers for later semester classifications are the threshold-0.1 classifiers. The most successful individual classifier overall is the threshold-0.15 classifier in Week 10, with an accuracy of over 87%, and only 3.3 False Passes of 25 Fails. It is interesting to note that for all thresholds, there is a large leap in classifier accuracy in Week 5, ranging from an increase of 0.04 to 0.17. There is also a reduction in False Pass numbers, ranging from 1 to almost 7. This suggests that Week 5 may be a key week in predicting student outcome in CA.

8.3.2 Comparison of Written Exam Classifiers

The accuracy of the threshold-0, threshold-0.1 and threshold-0.15 HOG classifiers for Written Exams can be seen in the Table 8-5, and the number of False Passes (out of a possible total of 25) can be seen in Table 8-6.

<i>Labs</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>0</i>	0.552	0.656	0.632	0.697	0.702	0.704	0.711	0.713	0.72
<i>0.1</i>	N/A	0.67	0.687	0.736	0.783	0.789	0.791	0.793	0.775
<i>0.15</i>	N/A	N/A	N/A	0.719	0.784	0.775	0.786	0.78	0.781

Table 8-5: HOG Written Exam Classifier Accuracy

Between Weeks 2 to 5, the most successful classifier is the threshold-0.1 classifier, according to the average accuracy and False Pass number as shown in Table 8-7.

<i>Labs</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>0</i>	15.1	11.75	12.95	9.2	8.95	9.4	9.15	9.05	8.95
<i>0.1</i>	N/A	9.05	8.2	7.95	6.45	6.35	6.15	6.1	6.05
<i>0.15</i>	N/A	N/A	N/A	10.45	6.5	6.9	6.45	6.5	6.5

Table 8-6: HOG Written Exam Classifier False Passes

The averages do not include Week 2, as only threshold-0 has a Week 2 result, and although threshold-0.15 has the highest accuracy, it also has a high False Pass number. Therefore, the most successful early classifier threshold is 0.1.

	<i>Average Accuracy</i>	<i>Average False Pass</i>
0	0.661667	11.3
0.1	0.697667	8.4
0.15	0.719	10.45

Table 8-7: Averages of Early Semester Classifiers for Written Exams

The most successful classifier threshold for late semester is the threshold-0.1, which can be seen in the averages in Table 8-8. The threshold-0.1 classifiers have the highest average accuracy and the lowest average False Pass number.

	<i>Average Accuracy</i>	<i>Average False Pass</i>
0	0.71	11.375
0.1	0.7862	7.775
0.15	0.7812	8.2125

Table 8-8: Averages of Late Semester Classifiers for Written Exams

The most successful classifier throughout the semester is the threshold-0.1 classifier, when considering both the accuracy and the False Passes, even when looking at the first and second half of the semester separately. The most successful individual classifier is the threshold-0.1 classifier for Week 9, with an accuracy of 79.3% and a False Pass number of 6.1. It is interesting to note that, like in the CA results, there is a leap in classifier accuracy for threshold-0 and threshold-0.1 in Week 5, compared to Week 4. There is also a reduction in False Pass numbers in threshold-0 of >3, going from 12.95 to 9.2. Although not as dramatic as the results in the CA section, this still suggests that Week 5 may be a key week in predicting student outcome in Written Exams.

8.5 Conclusions

Throughout this thesis, the data from Week 5 has been key to the differences in the higher and lower achieving groups. In this chapter, it is shown that Week 5 is when the successful and unsuccessful students begin to significantly diverge in observed behaviour. This is shown through the classifier results, the most successful classifier in CA making a leap from 71% to 79% in Week 4 to Week 5, and the False Passes dropping from 8.45 to 6.55. Similar to what was shown in the classifiers and Wilcox Rank Sum tests from Chapters 5, 6, and 7,

Week 5 and Week 6 is when the classifiers make a significant leap in accuracy, and False Passes make a drop.

In conclusion, Week 5 again seems to be a key week in determining student outcome. The most successful classifier threshold for CA is threshold-0 for early semester, and threshold-0.1 for late semester. The most successful classifier threshold for the Written Exam is threshold-0.1 for early and late semester.

9. Conclusions

In this chapter, the research instruments created for this study, MULE and HOG, and their success will be discussed. The success of the HOG classifiers using the three different data types, and the classifier using all of the data types will be compared. The student behaviour and its relation to student outcome is discussed on a week-to-week basis using results from all four experiments. The research questions are examined in the context of the results of the four experiments, and future research is discussed. Finally, the conclusions of the thesis will be outlined.

9.1 The Research Instruments

9.1.1 MULE

The MULE system has become an integral part of the Computer Science course in Maynooth University, and has also been used in both Beijing University of Technology and Fuzhou University. It has now been used in teaching Introduction to Programming to over 1000 students. The system is extremely modular and has huge potential for further expansion for pedagogical and research purposes. It has been used to teach novice programmers Java, C++, and Prolog, and can be used for many other programming languages. As a research tool, it has been used to non-intrusively collect large-scale behavioural data from novice programmers and subsequently provide valuable insight into the behaviours of novice programmers.

9.1.2 HOG

The HOG classifier is a classifier specifically built to work with the behavioural data from MULE to identify students who may need intervention. In this thesis, variants of this classifier are tested, with different subsections of the data, to varying degrees of success. In the following section, the results of the four different classifiers are compared and discussed for CA and the Written Exam in terms of which had the most success, in early semester and in late semester.

9.1.2.1 Early Semester CA classifiers

In Table 9-1 and Table 9-2, the most successful classifiers from each experiment, MM from Chapter 5, CRE from Chapter 6, COMPLEX from Chapter 7, and HOG from Chapter 8, are compared in terms of accuracy and number of

False Passes out of total Fails (always 25, as explained in Section 4.4) for early semester, Week 2 to Week 5.

	<i>Threshold</i>	<i>Lab 2</i>	<i>Lab 3</i>	<i>Lab 4</i>	<i>Lab 5</i>
<i>MM</i>	-0.1	0.508	0.561	0.556	0.545
<i>CRE</i>	0	N/A	0.677	0.706	0.73
<i>COMPLEX</i>	0	0.669	0.709	0.722	0.788
<i>HOG</i>	0	0.654	0.723	0.74	0.789

Table 9-1: Comparing Early Semester CA Classifier Accuracy

	<i>Threshold</i>	<i>Lab 2</i>	<i>Lab 3</i>	<i>Lab 4</i>	<i>Lab 5</i>
<i>MM</i>	-0.1	16	12.4	15.3	13.55
<i>CRE</i>	0	N/A	8.7	8.55	7.7
<i>COMPLEX</i>	0	12.75	9.15	8.6	6.75
<i>HOG</i>	0	13	8.45	8.05	6.9

Table 9-2: Comparing Early Semester CA Classifier False Passes

In Table 9-3, the averages of the accuracy and the False Passes for the four classifier types are compared. Note that Week 2 is not included in this average, as not all classifier types had enough data for a classifier at this point.

	<i>Threshold</i>	<i>Accuracy</i>	<i>False Passes</i>
<i>MM</i>	-0.1	0.554	13.75
<i>CRE</i>	0	0.704333	8.316667
<i>COMPLEX</i>	0	0.739667	8.166667
<i>HOG</i>	0	0.750667	7.8

Table 9-3: Comparing Early Semester CA Classifier Average Accuracy and False Passes

In Table 9-1, we can see that COMPLEX has the highest accuracy for Week 2, but the False Passes are high (>50% of the Fails classified as Passes), so the classifier is not reliable.

For Weeks 3, 4, and 5 the HOG classifier has the highest accuracy for every week, and the lowest False Pass number for Weeks 3 and 4. COMPLEX has a slightly lower False Pass number for Week 5 than HOG, and a slightly lower accuracy rate. Looking at the averages, as shown in Table 9-3, HOG threshold-0 is the most successful early semester classifier, but it is important to note the success of the COMPLEX classifier, especially in Week 5. The MM classifiers are unsuccessful and seem to perform as well as random classifiers (in that the

accuracy is around 50%) and CRE classifiers are successful as early semester classifiers, but not as successful as HOG or COMPLEX.

9.1.2.2 Late Semester CA classifiers

In Tables 9-4 and 9-5, the most successful classifiers from each experiment are compared in terms of accuracy and number of False Passes out of total Fails (always 25) for late semester, from Weeks 6 to 10. In Table 9-6, the averages of the accuracy and the False Passes for the four classifier types is compared.

	<i>Threshold</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>MM</i>	-0.1	0.556	0.568	0.512	0.495	0.493
<i>CRE</i>	0	0.758	0.763	0.766	0.767	0.767
<i>COMPLEX</i>	0.15	0.821	0.841	0.853	0.858	0.876
<i>HOG</i>	0.1	0.813	0.842	0.853	0.846	0.858

Table 9-4: Comparing Late Semester CA Classifier Accuracy

CRE results are mostly the same throughout late semester, with its accuracy staying around 76%, and the number of False Passes staying around 6.8, implying that CRE movements are more useful in classifying students in early semester, but later CRE data does not add anything useful to the classifier outcome. The HOG and COMPLEX classifiers are similar in accuracy and False Pass number to each other throughout Weeks 6 to 10, with COMPLEX having the highest accuracy of any classifier in this thesis at 87.6%, as seen in Table 9-4, and the lowest False Pass number of 3.1 as seen in Table 9-5.

	<i>Threshold</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>MM</i>	-0.1	11.1	13.2	21.8	25	25
<i>CRE</i>	0	6.9	6.7	6.8	6.75	6.7
<i>COMPLEX</i>	0.15	5.95	4.8	4.2	4.3	3.1
<i>HOG</i>	0.1	5.9	4.75	4.45	4.35	3.3

Table 9-5: Comparing Late Semester CA Classifier False Passes

	<i>Threshold</i>	<i>Accuracy</i>	<i>False Passes</i>
<i>MM</i>	-0.1	0.5248	19.22
<i>CRE</i>	0	0.7642	6.77
<i>COMPLEX</i>	0.15	0.8498	4.47
<i>HOG</i>	0.1	0.8424	4.55

Table 9-6: Comparing Late Semester CA Classifier Average Accuracy and False Passes

CRE results are successful, but do not improve with the addition of late semester data, and the MM classifiers are not successful, and continue to perform as well as random classifiers.

While the results for HOG and COMPLEX are similar, COMPLEX threshold-0.15 is the most successful late semester classifier, in average accuracy and False Passes, as can be seen in Table 9-6, and also in highest achieved accuracy rate and lowest achieved False Pass number.

9.1.2.3 Early Semester Written Exam HOG classifiers

In this section, the results of the four different Written Exam classifiers are compared, and the early semester success is discussed.

	<i>Threshold</i>	2	3	4	5
<i>MM</i>	-0.1	0.506	0.499	0.464	0.479
<i>CRE</i>	0	N/A	0.731	0.733	0.762
<i>COMPLEX</i>	0	N/A	0.706	0.7	0.704
<i>HOG</i>	0.1	N/A	0.67	0.687	0.736

Table 9-7: Comparing Early Semester Written Exam Classifier Accuracy

In Tables 9-7 and 9-8, the most successful classifiers from each experiment are compared in terms of accuracy and number of False Passes out of total Fails (always 25) for early semester, from Week 2 to Week 5. In Table 9-9, the averages of the accuracy and the False Passes for the four classifier types is compared, but Week 2 is not included in this average, as not all classifier types had enough data for a classifier at this point.

	<i>Threshold</i>	2	3	4	5
<i>MM</i>	-0.1	15.15	11.6	14.5	13.75
<i>CRE</i>	0	N/A	11	10.2	11.25
<i>COMPLEX</i>	0	N/A	9.35	9.5	9.15
<i>HOG</i>	0.1	N/A	9.05	8.2	7.95

Table 9-8: Comparing Early Semester Written Exam Classifier False Passes

	<i>Threshold</i>	<i>Accuracy</i>	<i>False Passes</i>
<i>MM</i>	-0.1	0.480667	13.28333
<i>CRE</i>	0	0.742	10.81667
<i>COMPLEX</i>	0	0.703333	9.333333
<i>HOG</i>	0.1	0.697666	8.4

Table 9-9: Comparing Early Semester Written Exam Classifier Average Accuracy and False Passes

For Weeks 3 and 4, the CRE classifier has the most accurate classifiers, but has very high False Pass numbers. HOG does not perform as well as CRE on average in terms of accuracy and has much lower False Pass numbers.

From the averages presented in Table 9-9, it could be argued that either of the two classifiers CRE and HOG are the most successful, CRE in terms of accuracy and HOG in terms of False Passes. If we consider the low number of False Passes as more important than accuracy, HOG is the most successful early semester classifier for Written Exams. The MM classifiers are not successful, and the COMPLEX classifiers are successful, but not as accurate as CRE, and have higher False Passes than HOG, as seen in Table 9-9.

9.1.2.4 Late Semester Written Exam HOG classifiers

In Tables 9-10 and 9-11, the Classifier Accuracy and the False Pass numbers are compared for late semester Written Exam classifiers. The classifiers for the CRE data peak in Week 6 and do not improve over the course of the last four weeks of the semester, implying that, like in the CA results, the CRE data is useful for early semester classification of Written Exam results only. There is little change in the other classifiers throughout this time, although the COMPLEX and HOG classifiers peak in Week 9.

	<i>Threshold</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>MM</i>	-0.1	0.464	0.499	0.468	0.486	0.485
<i>CRE</i>	0	0.781	0.77	0.777	0.78	0.776
<i>COMPLEX</i>	0.1	0.776	0.776	0.774	0.782	0.767
<i>HOG</i>	0.1	0.783	0.789	0.791	0.793	0.775

Table 9-10: Comparing Late Semester Written Exam Classifier Accuracy

	<i>Threshold</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>MM</i>	-0.1	14.8	13.45	16.75	20.85	20.3
<i>CRE</i>	0	7.5	7.25	7.5	6.8	7.15
<i>COMPLEX</i>	0.1	6.8	6.65	6.55	6.45	6.4
<i>HOG</i>	0.1	6.5	6.9	6.45	6.5	6.5

Table 9-11: Comparing Late Semester Written Exam Classifier False Passes

The averages of the classifier accuracies and False Passes from Week 6 to 10, is show in Table 9-12. The most successful classifier in late semester is the HOG classifier. Both CRE and COMPLEX also have high average accuracy (>77%), but COMPLEX has a lower False Pass number. The MM classifiers are again unsuccessful.

	<i>Threshold</i>	<i>Accuracy</i>	<i>False Passes</i>
<i>MM</i>	-0.1	0.4804	17.23
<i>CRE</i>	0	0.7768	7.24
<i>COMPLEX</i>	0.1	0.775	6.57
<i>HOG</i>	0.1	0.7862	6.57

Table 9-12: Comparing Late Semester Written Exam Classifier Average Accuracy and False Passes

While the classifiers, other than MM, have similar average success, the HOG threshold-0.1 is the most successful late semester classifier for Written Exam.

9.2 Discussion of Student Behaviour

In this thesis, key times and assignments in the Introduction to Programming course have been identified. Each of the three experiments found that Week 5 was a key week in the semester, and that the assignments could easily be a stumbling block for students. While the MM experiments were less successful when applied to the week-by-week analysis, we still found that there were connections between the student's MM data and their eventual outcome, though this may only be true for the second semester. This requires more research.

More successfully, we found that the CRE data and the COMPLEX data from Week 5 of the labs were significantly different in the higher and lower performing groups, and that this trend continued into Week 6 and Week 7.

The most successful features were those from the COMPLEX experiment. This fits in to the Neo-Piagetian framework, as students who are able to write more complex code and are able to write code that contains various concepts from an early stage, are more likely to do well in the module. This may suggest that as Computer Science educators, we should spend more time on teaching students how to combine different concepts, instead of teaching the concepts individually, and hoping the students will work the rest out for themselves. In the COMPLEX experiment, the analysis and classifiers showed a clear shift in Week 5, when the two groups seem to most clearly diverge from each other. In terms of assignments, the earliest indicator that a student will do well in a module is the Q4 assignment in Week 5.

The description of the weekly assignments in Section 4.1.2.1 suggests that Week 5 has a leap in complexity of assignment requirements, and that may be the first time in the semester that students have a large enough range of problem-solving techniques to allow for this kind of complexity. The result may be a sink-or-swim week, where students who are not able to quickly adapt to combining techniques will not be able to catch up for the rest of the semester.

In the following sections, there will be a week-by-week discussion of classifier success and when the higher and lower achieving students diverge in behaviour according to the different data types.

9.2.1 Week 1

In Week 1, we already see divergences in the behaviour of the higher and lower achieving students in CA and Written Exam in the Wilcox Rank Sum test for MMs, for CRE, and for COMPLEX. Specifically, there are differences in the file size tests for Q4, which asks the students to combine the first programming concepts they have learned, namely: mathematical operations; storing values in variables; and printing variables. This may be an early warning sign of students who are in need of assistance and would benefit from extra help. There are no positive Linear Regression results at this stage, and so there is no data for the classifiers.

9.2.2 Week 2

Week 2 has similar Wilcox Rank Sum test results for CA and the Written Exam for MM and CRE, but there is already a leap in the number of features with significant results for the COMPLEX tests, implying that differences in code complexity between the two groups are apparent very early on.

This week we see the first positive Linear Regression results, though the values are still very small (<0.03) for CRE tests. The results for COMPLEX data are also mostly very small, but the coefficient for Q6 in relation to CA is already 0.17, much higher than anything in CRE at this point.

These Linear Regression results mean the first classifier can be run for COMPLEX and HOG. The COMPLEX CA classifier is already at 66.9% accuracy, although it has a high False Pass number of 12.75 ($>50\%$). The HOG classifier is similarly successful, with an accuracy of 65% and a False Pass number of 13. The HOG Written Exam classifier is much less successful at 55% accuracy and a False

Pass number of 15.1. At this stage, the classifiers are not useful, as the False Pass number is over 50%, but the results do show that student behaviour is already diverging.

The assignments this week ask students to combine techniques, and the divisions in code complexity may show that successful students are able to write more complex code at this point.

9.2.3 Week 3

Week 3 has no significant Wilcoxon Rank Sum Test results for MM, but has slightly more for CRE tests. COMPLEX continues to have a majority of the features return significant results.

CRE Linear Regression results show two features with positive results for CA and for the Written Exam, all >0.1 . COMPLEX has an increase in the number of positive coefficients in relation to CA and the Written Exam, some of which are >0.1 .

These Linear Regression results mean the CRE classifiers run for the first time, and are successful for early in the semester, with the CA classifier having an accuracy of 67% and a False Pass number of 8.7 ($<35\%$). The Written Exam classifier is less successful, with a high False Pass number of 11, although it has an accuracy of 73%. The COMPLEX classifiers can now run with thresholds-0, thresholds-0.1 and thresholds-0.15 for CA, with the most successful being threshold-0 with an accuracy of 70.9% and a False Pass of 9.15, a significant improvement from the previous week. The first COMPLEX Written Exam classifier runs this week, with an accuracy of 70.6% and a False Pass rate of 9.35 ($<38\%$). The most successful classifier this week is the threshold-0 HOG classifier, with an accuracy of 72% and a False Pass number of 8.45.

Week 3 has clearer signs of divergence and is the first week with useful classifiers. This may be because, as well as the assignments becoming more difficult, this is the first week when the students are expected to apply the techniques they have learned to solve problems themselves, as mentioned in Section 4.1.2.1.

9.2.4 Week 4

For this week, there are just two features with significant results from the Wilcoxon Rank Sum test for CA and MM, and none for the Written Exam. The CRE

results are more successful with an increase in the number of significant features, but the COMPLEX results have a reduction in the number of significant features in proportion to total features. There is only one positive CRE feature in relation to CA, and none in relation to Written Exam results. There is again a reduction in the proportional number of positive results in the Linear Regressions tests for COMPLEX.

There are some slight changes in the classifier results, the CRE threshold-0 classifier for CA increases in accuracy, increasing from 67.7% to 70.6% and the False Pass number drops very slightly. The Written Exam threshold-0 classifier increases from 73.1% to 73.3% and the False Pass number drops from 11 to 10.2. The COMPLEX classifiers improve very slightly, the threshold-0 CA classifier accuracy increasing from 70.9% to 72.2% and False Pass dropping from 9.15 to 8.6. Written Exam classifiers do not change significantly.

There is also a slight increase in HOG classifier success, the threshold-0 CA accuracy increasing from 72.3% to 74%, and the False Pass number dropping from 8.45 to 8.05. The Written Exam threshold-0.1 classifier increases in accuracy from 67% to 68.7% and the False Pass number drops from 9.05 to 8.2. Similar to Week 3, the classifiers are useful at this stage, but do not improve enough to imply that Week 4 is a key week.

9.2.5 Week 5

Week 5 has a huge leap in significant results from the Wilcox Rank Sum test, with five significant results from the MM tests, and all eight features returning significant results for CRE tests. In COMPLEX, all five assignments return significant results with file size data for CA and the Written Exam, and four of the five for node data for both CA and the Written Exam.

The Linear Regression results for CRE have four positive coefficients in relation to CA, one of which is >0.1 , meaning it can be used in the threshold-0.1 classifier. There are two positive Written Exam results, both <0.1 . All of the assignments have positive coefficients in relation to CA for the COMPLEX Linear Regression file size results, with one result being >0.2 . Four of the five assignments have positive results for the CA results in relation to nodes, with one result being >0.15 . For the coefficients in relation to the Written Exam, four of the five are positive in relation to file size, with one result being >0.15 and four of the five are positive in relation to nodes, with two features being >0.15 .

The CRE classifiers in Week 5 continue to improve, with an increase in threshold-0 for CA increasing in accuracy from 70.6% to 73% and the False Pass dropping from 8.55 to 7.7. The Written Exam classifiers increase in accuracy from 73.3% to 76.2%, but False Passes also increase from 10.2 to 11.25, higher than it was in Week 2.

The COMPLEX classifiers have a significant leap in success. The threshold-0 classifier for CA increases from 72.2% to 78.8% and the False Pass drops from 8.6 to 6.75. The threshold-0.1 CA classifier also improves, with accuracy increasing from 70.8% to 79.5% and False Passes dropping from 9.05 to 6.65. Finally, the 0.15 CA classifier makes the biggest improvement, increasing in accuracy from 68.4% to 81% and the False Pass number dropping from 11.35 to 6.05. Written Exam results do not have a similar increase in success at this point. The HOG classifiers also have a significant increase in success at this point, the most pronounced being the CA threshold-0.15 classifier increasing in accuracy from 64.2% to 81.2%, and the False Pass number dropping from 13 to 6.2. The threshold-0.1 Written Exam classifier increases in accuracy from 68.7% to 73.6% and the False Pass number drops slightly from 8.2 to 7.95.

These changes in success in COMPLEX and HOG CA classifiers point to Week 5 being a key week in the semester, particularly in relation to students' ability to write complex code. As noted in Section 4.1.2.1, this week's assignments are an important step in learning to program: they test the students' ability to not just use the techniques, but their ability to understand when to apply the techniques.

9.2.6 Week 6

Week 6 is an exam week, so we would expect to see differences in behaviour between the two groups. In Week 6, the MM Wilcoxon Rank Sum test returns four significant results for CA and three for the Written Exam. The CRE results show six significant features for both CA and Written Exams. Four of the six questions in COMPLEX are significant across file size, nodes, the Written Exam and CA tests.

In the Linear Regression tests, three of the CRE coefficients relating to CA are positive, with two being >0.3 , the highest results from the CRE experiment. Five of the Written Exam CRE tests return positive results, with two being >0.2 . In

the COMPLEX Linear Regression file size CA tests, two assignments return positive results, Q6 being >0.25 . Four of the coefficients in relation to the Written Exam are positive, with one being >0.15 . In the tests for nodes, two coefficients in relation to CA are positive, with Q6 being >0.3 , the highest Linear Regression coefficient from the COMPLEX experiment. Three of the features for the Written Exam have positive results, with Q5 being >0.2 .

The most successful CRE classifier threshold for CA is threshold-0, which has a slight increase in accuracy from 73% to 75.8%, and a drop in False Passes from 7.7 to 6.9. The Written Exam classifier is more successful, with an increase from 76.2% to 78.1% for threshold-0 and the False Pass dropping from 11.25 to 7.5. The COMPLEX classifiers for CA all increase in accuracy and decrease in False Passes slightly. The HOG classifiers for CA also increase in accuracy by around and decrease in False Passes by up to 1. The HOG Written Exam classifiers improve, with the threshold-0.15 classifier increasing in accuracy from 71.9% to 78.4% and the False Pass number decreasing by almost four, from 10.45 to 6.5.

The Linear Regression results this week, which are some of the highest, and the leaps in classifier accuracy, suggest that Week 6 is a key week in predicting student outcome.

9.2.7 Week 7

In Week 7, the MM Wilcox Rank Sum tests have a total of 10 significant results for CA and eight for the Written Exam the highest amount for any week. The CRE tests have six significant features for both CA and the Written Exam, the same as Week 6. For COMPLEX, three of the four assignments, Q2, Q3, and Q4 all have significant results for both file size and nodes, and for CA and the Written Exam.

For Linear Regression, CRE has three positive coefficients for CA and two for Written Exam. For COMPLEX, file size has three positive results for CA, Q4 being >0.24 , and three for the Written Exam, Q4 being >0.1 . The nodes Linear Regression tests show one positive coefficient for CA, Q4 which is >0.1 and two for the Written Exam, Q2 and Q4.

The CRE classifiers for CA increase in accuracy slightly, and the False Passes decrease by around 0.5 for each threshold. The Written Exam classifiers do not change much and are slightly less successful in some respects. The

COMPLEX classifiers for CA all increase in accuracy and decrease in False Passes slightly. The HOG classifiers for CA each decrease in False Pass number by around 1, and increase in accuracy, the largest increase being threshold-0.15, which increases from 79.1% to 84.6%. The Written Exam classifiers for COMPLEX are almost identical to the previous week and perform less successfully than the week before in some cases. The HOG classifiers for CA continue to improve, with the threshold-0.15 increasing in accuracy from 79.1% to 84.6% and the False Pass rate dropping to 4.75 (<20%). However, the Written Exam classifiers do not improve in the same way and perform at around the same level as the previous week.

Week 7 is a key week according to the COMPLEX and HOG results, but the CRE classifiers have already peaked in usefulness, implying that CRE behaviours are less important indicators of student outcome at this point.

9.2.8 Week 8

The Wilcoxon Rank Sum test for MM has eight features with significant results for CA, and seven for the Written Exam. The CRE tests only have two for CA and the Written Exam, C2C, and C2R. However, in the COMPLEX tests, five of the six assignments for this week show significant differences in the file size and nodes data for CA and the Written Exam.

There are no positive Linear Regression results for CRE. For COMPLEX, there are five positive coefficients from the file size data in relation to CA, with Q4 being >0.15, and three in relation to the Written Exams. In the tests with nodes data, four have positive coefficients in relation to CA, with Q4 being >0.15, and four in relation to the Written Exams, with Q6 being >0.1.

There is no new data for CRE classifiers, and so the results are similar to the previous week. The COMPLEX classifiers for CA continue to increase in accuracy slightly and the False Passes continue to decrease. However, the Written Exam classifiers do not increase in success. The HOG classifiers for CA have a slight drop in False Passes but are mostly the same as last week. The Written Exam classifiers are also mostly the same as Week 7.

Despite a number of high coefficients from the Linear Regression results, the classifiers show Week 8 is not a key week.

9.2.9 Week 9

The MM Wilcox Rank Sum test for CA shows just four significant results, and the Written Exam has just three. The CRE tests have four for CA and for the Written Exam. In the COMPLEX tests, file size has three assignments with significant results for CA, and just two for Written Exam. In the nodes data test, the CA has three significant assignments, and the Written Exam has one.

In the Linear Regression results, six of CREs features for CA are positive, with C2C being >0.2 . The Written Exam results only have two positive correlations, with C2C being >0.1 . For COMPLEX file size, Q4 and Q5 both have positive coefficients in relation to CA and to the Written Exam, though all are <0.1 . For the nodes data, Q4 has a positive coefficient for both CA and the Written Exam.

The CRE, COMPLEX, and HOG classifiers do not significantly change this week, other than a drop in the accuracy of the HOG CA threshold-0 classifier.

Again, although there are some high values for the Linear Regression tests, due to a lack of improvement in the classifiers, Week 9 is not a key week.

9.2.10 Week 10

The Wilcox Rank Sum test for MM has just two significant results for CA and none for the Written Exam. The CRE tests have four significant results for CA and three for the Written Exam. These tests have shown a difference in C2C and C2R behaviour for the higher and lower achieving groups throughout the semester. The COMPLEX tests show a significant difference in all five assignments in terms of file size, and in four of the five assignments for nodes.

In the Linear Regression tests, C2R and R2C both have positive coefficients with CA, and C2R has a positive coefficient with the Written Exam results. In COMPLEX file size, four of the five assignments have positive coefficients with CA, with Q5 being >0.3 . All five of the assignments have positive coefficients in relation to the Written Exam, although Q1 is very close to zero. In the Linear Regressions tests using the nodes data, three of the five assignments have positive results in relation to CA, with Q5 having a coefficient of >0.15 . Three of the five have positive results in relation to the Written Exam, with Q5 being >0.1 .

In the classifiers for this week, the CRE again do not improve. The COMPLEX CA classifiers improve slightly this week for each threshold, and the False Passes each dropping by over 1. The COMPLEX Written Exam classifiers mostly do not improve, but the threshold-0 classifier increases in accuracy from 73.8% to 75.1% and the False Pass number drops by <1.

The HOG CA classifiers improve slightly again this week, and False Passes dropping by around 1, the most successful being threshold-0.15, which increases from 85.6% to 87%, and the False Pass dropping to just 3.3 (13.2%). The Written Exam classifiers, however, do not improve.

Although the data from this week results in the most successful classifier for CA, this is not a key week as the differences in the classifiers compared to the previous week are so low.

9.3 The Research Questions

In this section, the research questions originally presented in Section 1.2 will be discussed.

9.3.1 RQ1

How can we observe student behaviour as they learn to code in a non-intrusive way?

The MULE system was built to observe student behaviour as they learn to code in a close-to-authentic, online, desktop-like environment. Within this environment, the students can view their assignments, as well as write, compile, run and evaluate their code in a windowed coding system. Here they can view their assignments, multiple coding editor instances, and terminal instances. The system collects various behavioural data, including Mouse Movements (MM data), patterns of compilation, run and evaluation events (CRE data), and logs code written by the participants (COMPLEX data). The students are informed of the data collection at the beginning of the semester, and can choose to opt in or out, but after this they do not need to do anything to participate in the study other than complete the module tasks they would normally. This results in more “authentic” data on how the students learn to program than studies that may have used, for example, talk-aloud methods [43] or the collection of biometric data, such as pulse, sweat detection, Facial Action Coding System (FACS), or eye tracking to detect student behaviour. While these methods yield very interesting and important

results, the equipment used to gather this data is expensive, not widely available, and can be distracting for the student. By distracting the student, the data may be less valid, as it may make the student more stressed, for example.

The data collected by MULE is collected without interrupting the students' learning and takes place in their regular weekly labs. This data has been shown to be valuable both in highlighting key points and topics in the semester, and in building classifiers to detect students in danger of failing.

9.3.2 RQ2

Are there divergences in the observed student behaviour between the highest and lowest achieving students?

Throughout the experiments in the thesis, and the analysis into the three different types of data, divergences in behaviour have been observed in the higher and lower achieving groups of students.

As explained in Section 9.2, the key week in these divergences is Week 5 with significant results in Wilcox Rank Sum test observed in CRE data, in COMPLEX data, and even in the MM data, although this was the least successful of the experiments. Week 5 is also when the classifiers consistently make a jump in accuracy, and the False Pass rate drops. Weeks 6 and 7 are also key, with more leaps in classifier success. Despite large coefficient results from Linear Regression after this, the classifiers mostly stay at the same success, implying that students at this point have already diverged into the behaviour of higher or lower achieving students. There are many possible reasons that Week 5 is the key time, but the most likely is that the divergence in student behaviour is due to:

- 1) Increased complexity of assignments, as students are asked to use multiple programming concepts in conjunction.
- 2) Students are struggling to apply the concepts they have learned to solve problems, as the assignments no longer specifically tell the students which techniques to use.

These are key points in learning to program and aren't tied to Week 5 – this is just when it happens in our course, in the semester examined in this thesis. The changes in classifiers when this is happening implies that students would benefit from guidance and on how to apply the programming concepts they learn as problem solving tools.

9.3.3 RQ3

How early can students be classified as higher or lower achieving early on in the semester, to allow for interventions?

There are signs of divergence between the two groups as early as Week 1, and the first classifiers showing results are from around Week 3 onwards. Divergences in student behaviour between the higher and lower achieving groups can be seen as early as Week 1, when there are already significant differences in the C2C and C2R movements, as seen in Section 6.4.1. There are also already significant differences in COMPLEX data in Q4 of Week 1.

The most successful early semester classifier for CA and the Written Exam is the HOG classifier. The threshold-0 HOG classifier for CA has success as early as Week 3, with 72% accuracy, and 8.45 False Passes of 25 fails, meaning it can provide meaningful early warnings to students in danger. By Week 5, the accuracy rises to just under 79%, with 6.9 False Passes out of 25.

The most successful early semester classifier for the Written Exam is the HOG classifier with threshold-0.1. This classifier has success as early as Week 3, with just under 72% accuracy and 8.45 False Passes for 25 Fails. By Week 5 this rises to almost 80%, with 6.55 False Passes out of 25.

9.4 Future Work

The work in this thesis shows the potential for learning environments with passive, large-scale behavioural data collection, both as pedagogical tools and as research tools. The classifier system would have the most impact for students on an individual level if it were embedded in the learning environment, so that course co-ordinators could use it to predict which students need intervention.

Another promising avenue for research is peer learning. Peer learning [61] has been shown to be an effective pedagogical strategy, as students are required to articulate their thought processes, which could be of particular benefit to students who struggle to apply the concepts they have learned as problem solving strategies. Observation and analysis behaviour of students as they engage in remote pair programming in an authentic pedagogical environment could offer valuable insight into how to best teach novice programmers how to work collaboratively. The study described in the paper “*Gaps Between Industry Expectations and the Abilities of Graduates*” [60] found that students are lacking in personal skills, such as written communication and teamwork, and the study “*Struggles of New College Graduates*

in their First Software Development Job” [61] found that new programmers could not appropriately describe issues in written communication. I believe this suggests that students would benefit from collaborative work from the very beginning of Computer Science degree programs. Building collaborative tools within MULE, such as shared code files and chat functions, would allow for this to be implemented in the first-year labs.

This thesis focused on data from students who completed the course, and not on students who dropped out, but there may be value in investigating students who do not complete the course and the points at which these students diverge from the students who stay in the course. It may be that these students struggle to write compilable code more than the students investigated in this work. There is evidence to suggest that clearer error messages for novice programmers may improve student success. In the paper “*An Exploration Of The Effects Of Enhanced Compiler Error Messages For Computer Programming Novices*” [62] the use of enhanced compiler error messages was tested, and the results showed that the use of the Decaf editor resulted in fewer signs of struggling students in comparison to a control group, who saw standard error messages. Examining behavioural data from students using Enhanced Compiler Error Messages would provide an opportunity to further study how differently students behave when given clearer error messages.

The CRE and COMPLEX behavioural data has been shown to be valuable in studying the way that novice programmers learn how to code (as has MM data, although it was less successful with the HOG classifiers), but Computer Science education research tends to focus on the first year or semester of study. Using the existing data types, student behaviour can continue to be studied throughout their Computer Science course, with the aim of assisting students in need of intervention and improving the curriculum to address common problems in learning to write code. While the MM experiment was not successful in the weekly student outcome classifiers, there was some success in using MM as a stress classifier. It has been shown that Computer Science students experience stress and anxiety related to their programming ability [63], so this is an area that deserves further research.

9.5 Conclusion

In conclusion, it has been shown that authentic pedagogical coding environments with non-intrusive data collection features can be used to assist students both on an individual level, by alerting educators to struggling students,

and in the bigger picture, by highlighting stumbling blocks in the curriculum that cause the students to diverge into higher and lower achieving groups. It is difficult to teach programming, but by continuing to forensically investigate and identify the problems our students face, we can better guide our students on the road to mastering Computer Science.

10. Appendix

10.1 Consent Form

Consent Form	
Research Project:	MULE – Maynooth University Learning Environment
Researcher:	Dr Kevin Casey, Department of Computer Science, Maynooth University, Maynooth, Co. Kildare Natalie Culligan Department of Computer Science, Maynooth University, Maynooth, Co. Kildare
Contact details:	Email: natalie.culligan@mu.ie

The data gathered will be used by the researcher to improve the MULE system and the findings may be published in suitable conferences and journals. Some data will be accessible by the course co-ordinator to evaluate progress. I can access my data at my discretion.

I have received assurance from the researcher that the information that I will share will remain strictly confidential and that no information that discloses my identity will be released or published. However, I recognize that, in some circumstances, confidentiality of research data and records may be overridden by courts in the event of litigation or during investigation by lawful authority. In such circumstances the University will take all reasonable steps within law to ensure that confidentiality is maintained to the greatest possible extent

I am free to withdraw from the study up until the end of the academic year. I can refuse to answer any of the questions asked or to participate in any of the exercises.

All research-related data gathered will be stored in a secure manner and only the above-named researchers will have access to it.

If I have any questions about the research project, I may contact Natalie Culligan at the contact details provided.

I also understand that if I choose not to participate in the study, data will still be gathered by the system to provide reports for the course co-ordinator, but the data will not be used for research.

I, the participant, agree to participate in the research project being carried out by Dr Kevin Casey and Natalie Culligan to gather data on how students use the MULE system.

I consent to the automatic gathering, by the MULE system of the following:

- ✓ user interaction data
- ✓ performance data
- ✓ feedback
- ✓ code saved, compiled, run, evaluated, and submitted.

If during your participation in this study you feel the information and guidelines that you were given have been neglected or disregarded in any way, or if you are unhappy about the process, please contact the Secretary of the Maynooth University Ethics Committee at research.ethics@nuim.ie or +353 (0)1 708 6019.

Please be assured that your concerns will be dealt with in a sensitive manner.

10.2 Information Sheet

Information Sheet	
Research Project:	MULE – Maynooth University Learning Environment, funded by the Irish Research Council.
Researchers:	Dr Kevin Casey, Department of Computer Science, Maynooth University, Maynooth, Co. Kildare Natalie Culligan Department of Computer Science, Maynooth University, Maynooth, Co. Kildare
Contact details:	Natalie.culligan@mu.ie
<u>The Purpose:</u>	
<p>The purpose of this study is to gather data on how students are using the MULE system. MULE (Maynooth University Learning Environment) is a web-browser based system where students can edit, compile, and run their program. They can also use the chat function to get help from demonstrators and discuss problems with other students. There are two categories of data gathered by the system: research data and academic data. Research data will be used in improving future versions of the tool and in research (and subsequent publication) as to how useful the tool is and the usage patterns that students with different abilities have, and if it</p>	

is possible to detect stress from behavioural data. Academic data is used to provide reports on student performance and is necessary for the system to function in its role as a module content delivery platform.

The Participant:

The participant mentioned throughout this information sheet, refers to a student who is participating in a programming module.

The Data Gathering:

Participants will complete their module as normal using the MULE software system. The system will gather research data and academic data. The system gathers the users' interaction data, chat function data, user profile data, feedback and all code written by the user that is saved, compiled, executed, or evaluated. Periodically, a fully anonymised copy of the existing dataset is made to allow early-stage research.

Non-participation and exit from the study:

In accordance with the new GDPR guidelines, there is no requirement to participate in this study and the participant may still use the system. If a user chooses to opt out, academic data will still be gathered on the student, to provide reports to the course co-ordinator, but their data will not be used in the study. A participant may choose to exit the study at any time up until the end of the academic year. The participant must send their request in writing to the researcher, and all research-related data collected that has not yet been fully anonymised will be destroyed.

Anonymity and security of data:

As soon as the data is collected, it will be encoded with a unique identity key, to allow for separate sessions of use to be associated with a user. At the end of the academic year, the data will be fully anonymised. No records of the participant's identity will be stored for the purpose of our study, but the course co-ordinator will be able to review some aspects of the data. It must be recognized that, in some circumstances, confidentiality of research data and records may be overridden by courts in the event of litigation or in the course of investigation by lawful authority. In such circumstances the University will take all reasonable steps within law to ensure that confidentiality is maintained to the greatest possible extent.

The data will be stored only on a secure server located in Ireland. Only the above-named researchers will have access to it. The data will be kept for 10 years and will be destroyed thereafter.

Access to your data:

The subject is able to request a copy of their data that is stored for research by contacting the researcher, up until the data is fully anonymised.

Questions:

If you have any further questions, please contact the researcher using the above contact details.

10.3 Wilcoxon Rank Sum Test Results MM

10.3.1 CA

<i>Week</i>	<i>1</i>	<i>2</i>	<i>3</i>
<i>EFFICIENCY</i>	0.987437	0.323026	0.258799
<i>OVERSHOOTX</i>	0.703555	0.590466	0.484413
<i>DIRECTIONANGLE</i>	0.215464	0.948509	0.716484
<i>SEQUENCE_SPEED</i>	0.289042	0.702755	0.159286
<i>OVERSHOOTDIRECTIONANGLE</i>	0.111759	0.755683	0.713787
<i>OVERSHOOTY</i>	0.77283	0.500827	0.118085
<i>OVERSHOOT</i>	0.841912	0.532453	0.109021
<i>SEQUENCE_DURATION</i>	0.290238	0.878681	0.57172
<i>ANGLEDIFFERENCE</i>	0.289042	0.837198	0.740917
<i>DIRECTION</i>	0.029588	0.212553	0.196411
<i>HOVER_TIME</i>	0.157984	0.291519	0.085752
<i>OPTIMAL_DISTANCE</i>	0.784908	0.942278	0.929454
<i>DISTANCE_TRAVELLED</i>	0.147443	0.507071	0.09977
<i>VARIANCE1</i>	0.756812	0.347559	0.328339
<i>VARIANCE2</i>	0.914315	0.268867	0.294671
<i>VARIANCE3</i>	0.732982	0.904979	0.589041
<i>VARIANCEDIST1</i>	0.731007	0.268021	0.19392
<i>VARIANCEDIST2</i>	0.914315	0.268867	0.294671
<i>VARIANCEDIST3</i>	0.84602	0.939164	0.672452
<i>CLICKTIME</i>	0.280768	0.028114	0.560714
<i>HESITATE</i>	0.114143	0.023804	0.073371
<i>CLICKRATIO</i>	0.215464	0.003706	0.078136

Table 10-1: MM Wilcoxon Rank Sum Test CA for Weeks 1 to 3

<i>Week</i>	4	5	6	7
<i>EFFICIENCY</i>	0.420798	0.015108	0.740481	0.009588
<i>OVERSHOOTX</i>	0.34956	0.877951	0.274791	0.002774
<i>DIRECTIONANGLE</i>	0.093369	0.687177	0.03941	0.972124
<i>SEQUENCE_SPEED</i>	0.588681	0.918456	0.214753	0.195418
<i>OVERSHOOT</i>				
<i>DIRECTIONANGLE</i>	0.165815	0.961894	0.000429	0.000973
<i>OVERSHOOTY</i>	0.81847	0.509035	0.292012	0.018673
<i>OVERSHOOT</i>	0.817119	0.51013	0.285189	0.016298
<i>SEQUENCE_DURATION</i>	0.919672	0.357729	0.023814	0.769529
<i>ANGLEDIFFERENCE</i>	0.660008	0.971416	0.067558	0.356538
<i>DIRECTION</i>	0.017461	0.652375	0.265013	0.019592
<i>HOVER_TIME</i>	0.973646	0.234995	0.292012	0.656095
<i>OPTIMAL_DISTANCE</i>	0.012592	0.012978	0.271141	0.192956
<i>DISTANCE_TRAVELLED</i>	0.809024	0.970055	0.652937	0.206139
<i>VARIANCE1</i>	0.3451	0.019584	0.555292	0.033115
<i>VARIANCE2</i>	0.539364	0.354173	0.267524	0.06174
<i>VARIANCE3</i>	0.449438	0.362206	0.94745	0.939292
<i>VARIANCEDIST1</i>	0.137574	0.025512	0.795275	0.019036
<i>VARIANCEDIST2</i>	0.539364	0.354173	0.267524	0.06174
<i>VARIANCEDIST3</i>	0.053391	0.048165	0.168897	0.972124
<i>CLICKTIME</i>	0.626366	0.594471	0.13252	0.01131
<i>HESITATE</i>	0.823879	0.087633	0.052614	0.365013
<i>CLICKRATIO</i>	0.9681	0.164852	0.383295	0.016139

Table 10-2: MM Wilcox Rank Sum Test CA for Weeks 4 to 7

<i>Week</i>	8	9	10
<i>EFFICIENCY</i>	0.01516	0.031693	0.141307
<i>OVERSHOOTX</i>	0.004262	0.113071	0.221358
<i>DIRECTIONANGLE</i>	0.485603	0.869403	0.551862
<i>SEQUENCE_SPEED</i>	0.013254	0.109021	0.10326
<i>OVERSHOOTDIRECTIONANGLE</i>	0.003856	0.746383	0.004045
<i>OVERSHOOTY</i>	0.151661	0.029468	0.821522
<i>OVERSHOOT</i>	0.146422	0.028147	0.844168
<i>SEQUENCE_DURATION</i>	0.07856	0.124155	0.078657
<i>ANGLEDIFFERENCE</i>	0.020005	0.045101	0.160813
<i>DIRECTION</i>	0.067285	0.743648	0.043685
<i>HOVER_TIME</i>	0.82632	0.137978	0.202063
<i>OPTIMAL_DISTANCE</i>	0.903795	0.735463	0.241938
<i>DISTANCE_TRAVELLED</i>	0.062953	0.255764	0.197437
<i>VARIANCE1</i>	0.023864	0.991351	0.234068
<i>VARIANCE2</i>	0.718289	0.885072	0.703012
<i>VARIANCE3</i>	0.350573	0.624391	0.780738
<i>VARIANCEDIST1</i>	0.239165	0.926582	0.113104
<i>VARIANCEDIST2</i>	0.718289	0.885072	0.703012
<i>VARIANCEDIST3</i>	0.615618	0.344688	0.615227
<i>CLICKTIME</i>	0.870016	0.863718	0.326994
<i>HESITATE</i>	0.158683	0.031124	0.111909
<i>CLICKRATIO</i>	0.151661	0.402853	0.30784

Table 10-3: MM Wilcox Rank Sum Test CA for Weeks 8 to 10

10.3.2 Written Exam

<i>Week</i>	<i>1</i>	<i>2</i>	<i>3</i>
<i>EFFICIENCY</i>	0.456086	0.133361	0.824132
<i>OVERSHOOTX</i>	0.721161	0.338594	0.978379
<i>DIRECTIONANGLE</i>	0.06169	0.737898	0.549812
<i>SEQUENCE_SPEED</i>	0.970692	0.75866	0.735463
<i>OVERSHOOTDIRECTIONANGLE</i>	0.067771	0.631617	0.310773
<i>OVERSHOOTY</i>	0.478594	0.287066	0.465429
<i>OVERSHOOT</i>	0.516851	0.313537	0.516576
<i>SEQUENCE_DURATION</i>	0.723127	0.391366	0.762857
<i>ANGLEDIFFERENCE</i>	0.331549	0.244269	0.749121
<i>DIRECTION</i>	0.03914	0.533739	0.310773
<i>HOVER_TIME</i>	0.235546	0.932938	0.380875
<i>OPTIMAL_DISTANCE</i>	0.321201	0.818898	0.992792
<i>DISTANCE_TRAVELLED</i>	0.768816	0.529886	0.375011
<i>VARIANCE1</i>	0.295052	0.814338	0.926582
<i>VARIANCE2</i>	0.449767	0.422351	0.528325
<i>VARIANCE3</i>	0.729034	0.396796	0.204027
<i>VARIANCEDIST1</i>	0.472102	0.711485	0.961092
<i>VARIANCEDIST2</i>	0.449767	0.422351	0.528325
<i>VARIANCEDIST3</i>	0.803121	0.34256	0.517745
<i>CLICKTIME</i>	0.143811	0.074304	0.992792
<i>HESITATE</i>	0.508402	0.194474	0.169706
<i>CLICKRATIO</i>	0.475342	0.024542	0.150893

Table 10-4: MM Wilcox Rank Sum Test Written Exam for Weeks 1 to 3

<i>Week</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>
<i>EFFICIENCY</i>	0.457819	0.594471	0.79914	0.053057
<i>OVERSHOOTX</i>	0.112013	0.645011	0.362707	0.031453
<i>DIRECTIONANGLE</i>	0.054693	0.89682	0.433389	0.444697
<i>SEQUENCE_SPEED</i>	0.439085	0.116461	0.187719	0.252545
<i>OVERSHOOTDIRECTION ANGLE</i>	0.610436	0.937438	0.01481	0.005802
<i>OVERSHOOTY</i>	0.432939	0.581539	0.183286	0.042232
<i>OVERSHOOT</i>	0.42887	0.618313	0.184939	0.038224
<i>SEQUENCE_DURATION</i>	0.628833	0.263724	0.027804	0.983558
<i>ANGLEDIFFERENCE</i>	0.538215	0.621925	0.515745	0.391185
<i>DIRECTION</i>	0.219615	0.186607	0.524406	0.010417
<i>HOVER_TIME</i>	0.857867	0.319844	0.239785	0.603912
<i>OPTIMAL_DISTANCE</i>	0.087417	0.006531	0.769646	0.317776
<i>DISTANCE_TRAVELLED</i>	0.639984	0.064366	0.317962	0.329192
<i>VARIANCE1</i>	0.40886	0.060761	0.267524	0.011661
<i>VARIANCE2</i>	0.223559	0.033498	0.603226	0.096857
<i>VARIANCE3</i>	0.315738	0.454844	0.509299	0.822061
<i>VARIANCEDIST1</i>	0.186354	0.013678	0.938154	0.019036
<i>VARIANCEDIST2</i>	0.223559	0.033498	0.603226	0.096857
<i>VARIANCEDIST3</i>	0.449438	0.488459	0.421727	0.885299
<i>CLICKTIME</i>	0.515505	0.172238	0.374257	0.031453
<i>HESITATE</i>	0.81172	0.07292	0.007423	0.289976
<i>CLICKRATIO</i>	0.54628	0.025851	0.039092	0.002758

Table 10-5: MM Wilcox Rank Sum Test Written Exam for Weeks 1 to 3

<i>Week</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>EFFICIENCY</i>	0.015793	0.243869	0.193793
<i>OVERSHOOTX</i>	0.0259	0.466534	0.337519
<i>DIRECTIONANGLE</i>	0.45027	0.671134	0.650912
<i>SEQUENCE_SPEED</i>	0.049355	0.319477	0.256017
<i>OVERSHOOTDIRECTIONANGLE</i>	0.009798	0.963972	0.257123
<i>OVERSHOOTY</i>	0.094216	0.020249	0.726642
<i>OVERSHOOT</i>	0.094216	0.01911	0.704971
<i>SEQUENCE_DURATION</i>	0.392341	0.09977	0.215436
<i>ANGLEDIFFERENCE</i>	0.029862	0.205317	0.254914
<i>DIRECTION</i>	0.015633	0.554644	0.315411
<i>HOVER_TIME</i>	0.709964	0.0557	0.505276
<i>OPTIMAL_DISTANCE</i>	0.127307	0.897927	0.213487
<i>DISTANCE_TRAVELLED</i>	0.127772	0.586552	0.219371
<i>VARIANCE1</i>	0.008744	0.919408	0.773659
<i>VARIANCE2</i>	0.698919	0.885072	0.994737
<i>VARIANCE3</i>	0.174638	0.695006	0.772649
<i>VARIANCEDIST1</i>	0.030427	0.955334	0.206768
<i>VARIANCEDIST2</i>	0.698919	0.885072	0.994737
<i>VARIANCEDIST3</i>	0.333569	0.33013	0.470529
<i>CLICKTIME</i>	0.387216	0.574179	0.875254
<i>HESITATE</i>	0.382132	0.022279	0.138475
<i>CLICKRATIO</i>	0.067007	0.189006	0.099411

Table 10-6: MM Wilcox Rank Sum Test Written Exam for Weeks 1 to 3

10.4 Linear Regression Results MM

10.4.1 CA

<i>Week</i>	<i>1</i>	<i>2</i>	<i>3</i>
<i>EFFICIENCY</i>	-0.25709	-0.16574	-0.1278
<i>OVERSHOOTX</i>	-0.22264	-0.0867	-0.13951
<i>DIRECTIONANGLE</i>	-0.21731	-0.07566	-0.20588
<i>SEQUENCE_SPEED</i>	-0.26109	-0.08319	-0.15239
<i>OVERSHOOTDIRECTIONANGLE</i>	-0.23525	-0.08521	-0.13277
<i>OVERSHOOTY</i>	-0.2782	-0.10763	-0.1497
<i>OVERSHOOT</i>	-0.27826	-0.11099	-0.15171
<i>SEQUENCE_DURATION</i>	-0.40063	-0.07926	-0.11477
<i>ANGLEDIFFERENCE</i>	-0.26223	-0.17666	-0.1338
<i>DIRECTION</i>	-0.16599	-0.08044	-0.12805
<i>HOVER_TIME</i>	-0.43495	-0.07837	-0.08085
<i>OPTIMAL_DISTANCE</i>	-0.4696	-0.09271	-0.12142
<i>DISTANCE_TRAVELLED</i>	-0.25482	-0.07852	-0.14955
<i>VARIANCE1</i>	-0.30922	-0.16111	-0.08793
<i>VARIANCE2</i>	-0.24089	-0.08409	-0.13081
<i>VARIANCE3</i>	-0.2464	-0.26185	-0.10398
<i>VARIANCEDIST1</i>	-0.39094	-0.10177	-0.0859
<i>VARIANCEDIST2</i>	-0.24089	-0.08409	-0.13081
<i>VARIANCEDIST3</i>	-0.46522	-0.1849	-0.11233
<i>CLICKTIME</i>	-0.55715	-0.05803	-0.11254
<i>HESITATE</i>	-0.46923	-0.05589	-0.08896
<i>CLICKRATIO</i>	-0.2428	-0.18474	-0.11609

Table 10-7: MM Linear Regression CA for Weeks 1 to 3

<i>Week</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>
<i>EFFICIENCY</i>	-0.08626	-0.13532	-0.10022	-0.01352
<i>OVERSHOOTX</i>	-0.09082	-0.17932	-0.1529	-0.01719
<i>DIRECTIONANGLE</i>	-0.08934	-0.17469	-0.04705	-0.04552
<i>SEQUENCE_SPEED</i>	-0.09684	-0.17078	-0.16147	-0.04381
<i>OVERSHOOTDIRECTION ANGLE</i>	-0.09524	-0.18974	-0.03437	-0.00592
<i>OVERSHOOTY</i>	-0.07615	-0.2784	-0.04989	-0.03165
<i>OVERSHOOT</i>	-0.0762	-0.27943	-0.05037	-0.03108
<i>SEQUENCE_DURATION</i>	-0.15067	-0.1913	-0.13683	-0.11098
<i>ANGLEDIFFERENCE</i>	-0.14738	-0.17039	-0.05635	-0.04607
<i>DIRECTION</i>	-0.06674	-0.21568	-0.04394	-0.02926
<i>HOVER_TIME</i>	-0.11401	-0.15126	-0.11244	-0.04777
<i>OPTIMAL_DISTANCE</i>	-0.06364	-0.17189	-0.05935	-0.03213
<i>DISTANCE_TRAVELLED</i>	-0.09835	-0.17546	-0.09339	-0.03734
<i>VARIANCE1</i>	-0.09299	-0.17579	-0.05859	-0.03064
<i>VARIANCE2</i>	-0.09259	-0.18671	-0.07503	-0.03806
<i>VARIANCE3</i>	-0.17842	-0.1724	-0.06207	-0.04752
<i>VARIANCEDIST1</i>	-0.06645	-0.16763	-0.05243	-0.01726
<i>VARIANCEDIST2</i>	-0.09259	-0.18671	-0.07503	-0.03806
<i>VARIANCEDIST3</i>	-0.115	-0.15419	-0.08369	-0.05413
<i>CLICKTIME</i>	-0.07995	-0.18896	-0.11671	-0.05564
<i>HESITATE</i>	-0.1041	-0.14492	-0.10831	-0.05215
<i>CLICKRATIO</i>	-0.07903	-0.19393	-0.05405	-0.02726

Table 10-8: MM Linear Regression CA for Weeks 4 to 7

<i>Week</i>	8	9	10
<i>EFFICIENCY</i>	-0.0172	-0.54612	-0.17016
<i>OVERSHOOTX</i>	-0.02787	-0.17626	-0.16987
<i>DIRECTIONANGLE</i>	-0.06808	-0.03311	-0.19379
<i>SEQUENCE_SPEED</i>	-0.01472	-0.08308	-0.1532
<i>OVERSHOOTDIRECTIONANGLE</i>	-0.01594	-0.07473	-0.13424
<i>OVERSHOOTY</i>	-0.06226	-0.22505	-0.22404
<i>OVERSHOOT</i>	-0.06204	-0.22506	-0.22456
<i>SEQUENCE_DURATION</i>	-0.16748	-0.28076	-0.22677
<i>ANGLEDIFFERENCE</i>	-0.01924	-0.78433	-0.26156
<i>DIRECTION</i>	-0.04324	-0.06391	-0.17159
<i>HOVER_TIME</i>	-0.07058	-0.00058	-0.23295
<i>OPTIMAL_DISTANCE</i>	-0.06317	-0.03734	-0.18766
<i>DISTANCE_TRAVELLED</i>	-0.03657	-0.0633	-0.16672
<i>VARIANCE1</i>	-0.06751	-0.03065	-0.19214
<i>VARIANCE2</i>	-0.08563	-0.45219	-0.20431
<i>VARIANCE3</i>	-0.08167	-0.18943	-0.24616
<i>VARIANCEDIST1</i>	-0.0767	-0.03213	-0.18361
<i>VARIANCEDIST2</i>	-0.08563	-0.45219	-0.20431
<i>VARIANCEDIST3</i>	-0.09158	-0.43434	-0.21892
<i>CLICKTIME</i>	-0.23481	-0.28366	-0.21095
<i>HESITATE</i>	-0.08476	-0.01626	-0.25116
<i>CLICKRATIO</i>	-0.08948	-0.15675	-0.20161

Table 10-9: MM Linear Regression CA for Weeks 8 to 10

10.4.2 Written Exam

<i>Week</i>	<i>1</i>	<i>2</i>	<i>3</i>
<i>EFFICIENCY</i>	-0.1159	-0.17695	-0.03788
<i>OVERSHOOTX</i>	-0.16755	-0.02632	-0.02655
<i>DIRECTIONANGLE</i>	-0.07893	-0.00732	-0.06128
<i>SEQUENCE_SPEED</i>	-0.09399	-0.01722	-0.02733
<i>OVERSHOOTDIRECTIONANGLE</i>	-0.09097	-0.04294	-0.02636
<i>OVERSHOOTY</i>	-0.14654	-0.01241	-0.02321
<i>OVERSHOOT</i>	-0.14727	-0.01681	-0.02291
<i>SEQUENCE_DURATION</i>	-0.15821	-0.03073	-0.0249
<i>ANGLEDIFFERENCE</i>	-0.13819	-0.25058	-0.03576
<i>DIRECTION</i>	-0.06745	-0.01047	-0.05384
<i>HOVER_TIME</i>	-0.22261	-0.00152	-0.02759
<i>OPTIMAL_DISTANCE</i>	-0.17661	-0.0213	-0.0253
<i>DISTANCE_TRAVELLED</i>	-0.13645	-0.01073	-0.02478
<i>VARIANCE1</i>	-0.11943	-0.0053	-0.03719
<i>VARIANCE2</i>	-0.10449	-0.03462	-0.0262
<i>VARIANCE3</i>	-0.11589	-0.12696	-0.00751
<i>VARIANCEDIST1</i>	-0.12791	-0.00511	-0.06313
<i>VARIANCEDIST2</i>	-0.10449	-0.03462	-0.0262
<i>VARIANCEDIST3</i>	-0.12369	-0.16402	-0.02117
<i>CLICKTIME</i>	-0.3232	0.016025	-0.04628
<i>HESITATE</i>	-0.23711	0.02448	-0.0235
<i>CLICKRATIO</i>	-0.14094	0.052769	-0.01537

Table 10-10: MM Linear Regression Written Exam for Weeks 1 to 3

<i>Week</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>
<i>EFFICIENCY</i>	-0.01821	-0.05306	-0.13438	-0.03094
<i>OVERSHOOTX</i>	-0.00409	-0.05306	-0.20009	-0.02727
<i>DIRECTIONANGLE</i>	-0.14475	-0.05322	-0.03433	-0.03569
<i>SEQUENCE_SPEED</i>	-0.02244	-0.04987	-0.20262	-0.04817
<i>OVERSHOOTDIRECTION ANGLE</i>	-0.02216	-0.06516	0.005485	-0.00386
<i>OVERSHOOTY</i>	-0.03371	-0.13441	-0.05361	-0.03621
<i>OVERSHOOT</i>	-0.03334	-0.13639	-0.05536	-0.03587
<i>SEQUENCE_DURATION</i>	-0.08359	-0.05903	-0.10235	-0.10306
<i>ANGLEDIFFERENCE</i>	-0.0343	-0.05551	-0.04992	-0.03926
<i>DIRECTION</i>	-0.02098	-0.10575	-0.0407	-0.00886
<i>HOVER_TIME</i>	-0.02189	-0.01328	-0.08054	-0.03965
<i>OPTIMAL_DISTANCE</i>	-0.00813	-0.00219	-0.04597	-0.03865
<i>DISTANCE_TRAVELLED</i>	-0.02516	-0.04776	-0.11629	-0.03376
<i>VARIANCE1</i>	-0.02744	-0.03839	-0.06439	-0.09213
<i>VARIANCE2</i>	-0.00287	-0.07371	-0.03888	-0.04432
<i>VARIANCE3</i>	-0.03952	-0.05817	-0.03294	-0.04844
<i>VARIANCEDIST1</i>	-0.02236	-0.02156	-0.0325	-0.05884
<i>VARIANCEDIST2</i>	-0.00287	-0.07371	-0.03888	-0.04432
<i>VARIANCEDIST3</i>	-0.02755	-0.04954	-0.05406	-0.03896
<i>CLICKTIME</i>	-0.02312	-0.06235	-0.08616	-0.0189
<i>HESITATE</i>	-0.02633	-0.00033	-0.05929	-0.03441
<i>CLICKRATIO</i>	-0.09417	-0.07879	-0.02939	0.015701

Table 10-11: MM Linear Regression Written Exam for Weeks 4 to 7

<i>Week</i>	8	9	10
<i>EFFICIENCY</i>	-0.03839	-0.21587	-0.16056
<i>OVERSHOOTX</i>	-0.05121	-0.13292	-0.14631
<i>DIRECTIONANGLE</i>	-0.04849	-0.105	-0.15128
<i>SEQUENCE_SPEED</i>	-0.08235	-0.09776	-0.1404
<i>OVERSHOOTDIRECTIONANGLE</i>	-0.00176	-0.11307	-0.1331
<i>OVERSHOOTY</i>	-0.10391	-0.31639	-0.17737
<i>OVERSHOOT</i>	-0.10444	-0.31525	-0.17732
<i>SEQUENCE_DURATION</i>	-0.10635	-0.50468	-0.15981
<i>ANGLEDIFFERENCE</i>	-0.04301	-0.55688	-0.31605
<i>DIRECTION</i>	-0.01602	-0.11967	-0.14748
<i>HOVER_TIME</i>	-0.04638	-0.03787	-0.15084
<i>OPTIMAL_DISTANCE</i>	-0.04994	-0.14479	-0.1579
<i>DISTANCE_TRAVELLED</i>	-0.05696	-0.0832	-0.1498
<i>VARIANCE1</i>	-0.07963	-0.11496	-0.15224
<i>VARIANCE2</i>	-0.05714	-0.08182	-0.15137
<i>VARIANCE3</i>	-0.05607	-0.10103	-0.16047
<i>VARIANCEDIST1</i>	-0.071	-0.07974	-0.1503
<i>VARIANCEDIST2</i>	-0.05714	-0.08182	-0.15137
<i>VARIANCEDIST3</i>	-0.0518	-0.33642	-0.16674
<i>CLICKTIME</i>	-0.07656	-0.38137	-0.1634
<i>HESITATE</i>	-0.04232	-0.04071	-0.14071
<i>CLICKRATIO</i>	-0.04026	-0.08513	-0.18265

Table 10-12: MM Linear Regression Written Exam for Weeks 8 to 10

10.5 Full Classifier Results MM

10.5.1 CA

10.5.1.1 Threshold-0.1

<i>Week</i>	2	3	4
<i>Accuracy</i>	0.508	0.561	0.556
<i>AUC</i>	0.52808	0.5928	0.58296
<i>Precision_Recall</i>	0.543369	0.607421	0.690716
<i>Average_Loss</i>	1.35628	1.150318	5.951665
<i>Loss</i>	1.356276	1.150311	5.951643
<i>Precision</i>	0.50445	0.579935	0.566871
<i>Prediction Mean</i>	0.593196	0.545311	0.663546
<i>Recall</i>	0.656	0.618	0.724
<i>TrueFails</i>	9	12.6	9.7
<i>TruePasses</i>	16.4	15.45	18.1
<i>FalseFails</i>	8.6	9.55	6.9
<i>FalsePasses</i>	16	12.4	15.3

Table 10-13: MM CA Threshold -0.1 Classifier

<i>Week</i>	5	6	7
<i>Accuracy</i>	0.545	0.556	0.568
<i>AUC</i>	0.56224	0.59024	0.5888
<i>Precision_Recall</i>	0.656482	0.660806	0.653572
<i>Average_Loss</i>	7.050821	6.232789	6.061473
<i>loss</i>	7.050879	6.232834	6.061579
<i>precision</i>	0.547011	0.591255	0.550789
<i>predictionMean</i>	0.563162	0.49397	0.589728
<i>recall</i>	0.632	0.556	0.664
<i>TrueFails</i>	11.45	13.9	11.8
<i>TruePasses</i>	15.8	13.9	16.6
<i>FalseFails</i>	9.2	11.1	8.4
<i>FalsePasses</i>	13.55	11.1	13.2

Table 10-14: MM CA Threshold -0.1 Classifier

<i>Week</i>	8	9	10
<i>Accuracy</i>	0.512	0.495	0.493
<i>AUC</i>	0.58432	0.495	0.493
<i>Precision_Recall</i>	0.636419	0.746214	0.744682
<i>Average_Loss</i>	1.102824	1436.686	1673.947
<i>loss</i>	1.102815	1436.671	1673.947
<i>precision</i>	0.510205	0.497428	0.496365
<i>predictionMean</i>	0.722598	0.99526	0.993
<i>recall</i>	0.896	0.99	0.986
<i>TrueFails</i>	3.2	0	0
<i>TruePasses</i>	22.4	24.75	24.65
<i>FalseFails</i>	2.6	0.25	0.35
<i>FalsePasses</i>	21.8	25	25

Table 10-15: MM CA Threshold-0.1 Classifier

10.5.2 Written Exam

10.5.2.1 Threshold-0.1

<i>Week</i>	2	3	4
<i>Accuracy</i>	0.506	0.499	0.464
<i>AUC</i>	0.52456	0.473	0.46188
<i>Precision_Recall</i>	0.528641	0.548533	0.595284
<i>Average_Loss</i>	1.360249	6.425502	29.91753
<i>loss</i>	1.360246	6.42542	29.91732
<i>precision</i>	0.497013	0.463606	0.484249
<i>predictionMean</i>	0.558393	0.459704	0.542114
<i>recall</i>	0.618	0.462	0.508
<i>TrueFails</i>	9.85	13.4	10.5
<i>TruePasses</i>	15.45	11.55	12.7
<i>FalseFails</i>	9.55	13.45	12.3
<i>FalsePasses</i>	15.15	11.6	14.5

Table 10-16: MM Written Exam Threshold-0.1 Classifier

<i>Week</i>	5	6	7
<i>Accuracy</i>	0.479	0.464	0.499
<i>AUC</i>	0.47836	0.46856	0.49568
<i>Precision_Recall</i>	0.607675	0.613071	0.619542
<i>Average_Loss</i>	187.8078	66.00167	49.48129
<i>loss</i>	187.8066	66.00092	49.48214
<i>precision</i>	0.479081	0.480473	0.491156
<i>predictionMean</i>	0.531345	0.556057	0.536616
<i>recall</i>	0.508	0.52	0.536
<i>TrueFails</i>	11.25	10.2	11.55
<i>TruePasses</i>	12.7	13	13.4
<i>FalseFails</i>	12.3	12	11.6
<i>FalsePasses</i>	13.75	14.8	13.45

Table 10-17: MM Written Exam Threshold-0.1 Classifier

<i>Week</i>	8	9	10
<i>Accuracy</i>	0.468	0.486	0.485
<i>Accuracy_baseline</i>	0.5	0.5	0.5
<i>AUC</i>	0.4586	0.52284	0.51708
<i>Precision_Recall</i>	0.62386	0.636335	0.628207
<i>Average_Loss</i>	82.42663	3.868954	3.311844
<i>loss</i>	82.42532	3.868945	3.311835
<i>precision</i>	0.473558	0.490822	0.489373
<i>predictionMean</i>	0.642354	0.794328	0.784295
<i>recall</i>	0.606	0.806	0.782
<i>TrueFails</i>	8.25	4.15	4.7
<i>TruePasses</i>	15.15	20.15	19.55
<i>FalseFails</i>	9.85	4.85	5.45
<i>FalsePasses</i>	16.75	20.85	20.3

Table 10-18: MM Written Exam Threshold-0.1 Classifier

10.6 Full Classifier Results CRE Movements

10.6.1 CA

10.6.1.1 Threshold-0

<i>Week</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>
<i>Accuracy</i>	0.677	0.706	0.73	0.758
<i>AUC</i>	0.75868	0.79852	0.78832	0.8546
<i>Precision_Recall</i>	0.735137	0.789635	0.782734	0.851557
<i>Average_Loss</i>	0.626307	0.60079	0.570623	0.485475
<i>loss</i>	0.626308	0.600791	0.570624	0.485475
<i>precision</i>	0.672876	0.696093	0.718547	0.744403
<i>predictionMean</i>	0.502935	0.505485	0.51097	0.504778
<i>recall</i>	0.702	0.754	0.768	0.792
<i>TrueFails</i>	16.3	16.45	17.3	18.1
<i>TruePasses</i>	17.55	18.85	19.2	19.8
<i>FalseFails</i>	7.45	6.15	5.8	5.2
<i>FalsePasses</i>	8.7	8.55	7.7	6.9

Table 10-19: CRE CA Threshold-0 Classifier

<i>Week</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
<i>Accuracy</i>	0.763	0.766	0.767	0.767
<i>AUC</i>	0.86896	0.86732	0.86736	0.8778
<i>Precision_Recall</i>	0.875489	0.873641	0.872452	0.887014
<i>Average_Loss</i>	0.466736	0.46481	0.464857	0.448307
<i>loss</i>	0.466734	0.46481	0.46486	0.448309
<i>precision</i>	0.751874	0.751359	0.752479	0.753399
<i>predictionMean</i>	0.506499	0.515255	0.51384	0.515364
<i>recall</i>	0.794	0.804	0.804	0.802
<i>TrueFails</i>	18.3	18.2	18.25	18.3
<i>TruePasses</i>	19.85	20.1	20.1	20.05
<i>FalseFails</i>	5.15	4.9	4.9	4.95
<i>FalsePasses</i>	6.7	6.8	6.75	6.7

Table 10-20: CRE CA Threshold-0 Classifier

10.6.1.2 Threshold 0.1

<i>Week</i>	5	6	7
<i>Accuracy</i>	0.684	0.764	0.763
<i>AUC</i>	0.73712	0.852	0.85656
<i>Precision_Recall</i>	0.74689	0.849903	0.85382
<i>Average_Loss</i>	0.622947	0.495642	0.490722
<i>loss</i>	0.622946	0.49564	0.490723
<i>precision</i>	0.704249	0.746291	0.75313
<i>predictionMean</i>	0.483918	0.502364	0.489841
<i>recall</i>	0.614	0.81	0.788
<i>TrueFails</i>	18.85	17.95	18.45
<i>TruePasses</i>	15.35	20.25	19.7
<i>FalseFails</i>	9.65	4.75	5.3
<i>FalsePasses</i>	6.15	7.05	6.55

Table 10-21: CRE CA Threshold-0.1 Classifier

<i>Week</i>	8	9	10
<i>Accuracy</i>	0.762	0.761	0.756
<i>AUC</i>	0.8514	0.85176	0.84924
<i>Precision_Recall</i>	0.846903	0.848156	0.844744
<i>Average_Loss</i>	0.498402	0.503549	0.50171
<i>loss</i>	0.498401	0.503549	0.501712
<i>precision</i>	0.744467	0.746418	0.741885
<i>predictionMean</i>	0.496969	0.493509	0.491968
<i>recall</i>	0.808	0.8	0.796
<i>TrueFails</i>	17.9	18.05	17.9
<i>TruePasses</i>	20.2	20	19.9
<i>FalseFails</i>	4.8	5	5.1
<i>FalsePasses</i>	7.1	6.95	7.1

Table 10-22: CRE CA Threshold-0.1 Classifier

10.6.1.2 Threshold-0.15

<i>Week</i>	6	7	8	9	10
<i>Accuracy</i>	0.737	0.728	0.737	0.719	0.728
<i>AUC</i>	0.82892	0.80856	0.82852	0.80816	0.8284
<i>Precision_Recall</i>	0.817895	0.805647	0.814444	0.779761	0.820573
<i>Average_Loss</i>	0.530116	0.538838	0.526798	0.542029	0.527341
<i>loss</i>	0.530115	0.53885	0.526797	0.542036	0.527343
<i>precision</i>	0.720563	0.687334	0.718053	0.673736	0.714956
<i>predictionMean</i>	0.50768	0.502483	0.500114	0.510934	0.505458
<i>recall</i>	0.784	0.748	0.792	0.752	0.768
<i>TrueFails</i>	17.25	17.7	17.05	17.15	17.2
<i>TruePasses</i>	19.6	18.7	19.8	18.8	19.2
<i>FalseFails</i>	5.4	6.3	5.2	6.2	5.8
<i>FalsePasses</i>	7.75	7.3	7.95	7.85	7.8

Table 10-23: CRE CA Threshold-0.15 Classifier

10.6.2 Written Exam

10.6.2.1 Threshold-0

<i>Week</i>	3	4	5	6
<i>Accuracy</i>	0.651	0.664	0.636	0.725
<i>AUC</i>	0.71712	0.72772	0.7108	0.77776
<i>Precision_Recall</i>	0.731359	0.73331	0.76162	0.780754
<i>Average_Loss</i>	0.637468	0.630169	0.628805	0.571812
<i>loss</i>	0.637468	0.630179	0.628806	0.571813
<i>precision</i>	0.647881	0.655688	0.632102	0.717437
<i>predictionMean</i>	0.505255	0.50053	0.503575	0.495088
<i>recall</i>	0.742	0.736	0.722	0.75
<i>TrueFails</i>	14	14.8	13.75	17.5
<i>TruePasses</i>	18.55	18.4	18.05	18.75
<i>FalseFails</i>	6.45	6.6	6.95	6.25
<i>FalsePasses</i>	11	10.2	11.25	7.5

Table 10-24: CRE Written Exam Threshold-0 Classifier

<i>Week</i>	7	8	9	10
<i>Accuracy</i>	0.729	0.728	0.737	0.729
<i>AUC</i>	0.78152	0.78192	0.78496	0.78236
<i>Precision_Recall</i>	0.769857	0.776696	0.779933	0.775815
<i>Average_Loss</i>	0.567948	0.565877	0.56389	0.567747
<i>loss</i>	0.56795	0.565875	0.563888	0.567748
<i>precision</i>	0.724866	0.719843	0.737146	0.725651
<i>predictionMean</i>	0.496224	0.50459	0.493639	0.500003
<i>recall</i>	0.748	0.756	0.746	0.744
<i>TrueFails</i>	17.75	17.5	18.2	17.85
<i>TruePasses</i>	18.7	18.9	18.65	18.6
<i>FalseFails</i>	6.3	6.1	6.35	6.4
<i>FalsePasses</i>	7.25	7.5	6.8	7.15

Table 10-25: CRE Written Exam Threshold-0 Classifier

10.6.2.2 Threshold-0.1

<i>Week</i>	6	7	8	9	10
<i>Accuracy</i>	0.718	0.707	0.706	0.709	0.704
<i>AUC</i>	0.76972	0.772	0.75604	0.76768	0.77012
<i>Precision_Recall</i>	0.746164	0.740542	0.752207	0.731756	0.748504
<i>Average_Loss</i>	0.589908	0.589417	0.595132	0.591657	0.581265
<i>loss</i>	0.589907	0.589418	0.595131	0.591659	0.581265
<i>precision</i>	0.70974	0.696996	0.695669	0.696748	0.695487
<i>predictionMean</i>	0.491233	0.494364	0.490278	0.500048	0.49126
<i>recall</i>	0.752	0.748	0.762	0.752	0.734
<i>TrueFails</i>	17.1	16.65	16.25	16.65	16.85
<i>TruePasses</i>	18.8	18.7	19.05	18.8	18.35
<i>FalseFails</i>	6.2	6.3	5.95	6.2	6.65
<i>FalsePasses</i>	7.9	8.35	8.75	8.35	8.15

Table 10-26: CRE Written Exam Threshold-0.1 Classifier

10.6.2.3 Threshold-0.15

<i>Week</i>	6	7	8	9	10
<i>Accuracy</i>	0.71	0.71	0.713	0.713	0.712
<i>AUC</i>	0.7704	0.77016	0.77204	0.77148	0.77432
<i>Precision_Recall</i>	0.742669	0.747982	0.743751	0.745633	0.750403
<i>Average_Loss</i>	0.581102	0.589004	0.584548	0.583122	0.580263
<i>loss</i>	0.581106	0.589003	0.584547	0.583123	0.58026
<i>precision</i>	0.700753	0.696496	0.702882	0.702519	0.700598
<i>predictionMean</i>	0.493171	0.494459	0.490625	0.50209	0.487521
<i>recall</i>	0.746	0.754	0.75	0.748	0.746
<i>TrueFails</i>	16.85	16.65	16.9	16.95	16.95
<i>TruePasses</i>	18.65	18.85	18.75	18.7	18.65
<i>FalseFails</i>	6.35	6.15	6.25	6.3	6.35
<i>FalsePasses</i>	8.15	8.35	8.1	8.05	8.05

Table 10-27: CRE Written Exam Threshold-0.15 Classifier

10.7 Full Classifier Results COMPLEX

10.7.1 CA

10.7.1.1 Threshold-0

<i>Week</i>	2	3	4
<i>Accuracy</i>	0.669	0.709	0.722
<i>AUC</i>	0.72376	0.78424	0.79848
<i>Precision_Recall</i>	0.681588	0.768883	0.779994
<i>Average_Loss</i>	0.623944	0.566244	0.551358
<i>loss</i>	0.623944	0.566243	0.551354
<i>precision</i>	0.625121	0.686323	0.700056
<i>predictionMean</i>	0.520025	0.515713	0.516456
<i>recall</i>	0.848	0.784	0.788
<i>TrueFails</i>	12.25	15.85	16.4
<i>TruePasses</i>	21.2	19.6	19.7
<i>FalseFails</i>	3.8	5.4	5.3
<i>FalsePasses</i>	12.75	9.15	8.6

Table 10-28: COMPLEX CA Threshold-0 Classifier

<i>Week</i>	5	6	7
<i>Accuracy</i>	0.788	0.809	0.828
<i>AUC</i>	0.86264	0.89556	0.91384
<i>Precision_Recall</i>	0.835724	0.897094	0.912952
<i>Average_Loss</i>	0.467685	0.4274	0.39184
<i>loss</i>	0.467685	0.427404	0.391835
<i>precision</i>	0.76096	0.781625	0.810237
<i>predictionMean</i>	0.515317	0.520971	0.521544
<i>recall</i>	0.846	0.864	0.862
<i>TrueFails</i>	18.25	18.85	19.85
<i>TruePasses</i>	21.15	21.6	21.55
<i>FalseFails</i>	3.85	3.4	3.45
<i>FalsePasses</i>	6.75	6.15	5.15

Table 10-29: COMPLEX CA Threshold-0 Classifier

<i>Week</i>	8	9	10
<i>Accuracy</i>	0.83	0.828	0.847
<i>AUC</i>	0.91304	0.91208	0.933
<i>Precision_Recall</i>	0.915042	0.914544	0.939861
<i>Average_Loss</i>	0.399514	0.403811	0.359684
<i>loss</i>	0.399511	0.403809	0.35969
<i>precision</i>	0.804179	0.81446	0.845736
<i>predictionMean</i>	0.531913	0.519535	0.515071
<i>recall</i>	0.88	0.858	0.858
<i>TrueFails</i>	19.5	19.95	20.9
<i>TruePasses</i>	22	21.45	21.45
<i>FalseFails</i>	3	3.55	3.55
<i>FalsePasses</i>	5.5	5.05	4.1

Table 10-30: COMPLEX CA Threshold-0 Classifier

10.7.1.2 Threshold 0.1

labno	3	4	5
Accuracy	0.688	0.708	0.795
AUC	0.77572	0.78116	0.85984
Precision_Recall	0.76585	0.767767	0.820888
Average_Loss	0.576037	0.572627	0.460667
loss	0.576035	0.572627	0.460669
precision	0.668416	0.6842	0.764495
predictionMean	0.515744	0.512716	0.51763
recall	0.756	0.778	0.856
TrueFails	15.5	15.95	18.35
TruePasses	18.9	19.45	21.4
FalseFails	6.1	5.55	3.6
FalsePasses	9.5	9.05	6.65

Table 10-31: COMPLEX CA Threshold-0.1 Classifier

Week	6	7	8
Accuracy	0.813	0.823	0.844
AUC	0.89796	0.91536	0.91816
Precision_Recall	0.897729	0.911041	0.918603
Average_Loss	0.413128	0.391146	0.383919
loss	0.413128	0.391145	0.383922
precision	0.790228	0.792762	0.835013
predictionMean	0.516462	0.531437	0.51088
recall	0.86	0.886	0.864
TrueFails	19.15	19	20.6
TruePasses	21.5	22.15	21.6
FalseFails	3.5	2.85	3.4
FalsePasses	5.85	6	4.4

Table 10-32: COMPLEX CA Threshold-0.1 Classifier

<i>Week</i>	<i>9</i>	<i>10</i>
<i>Accuracy</i>	0.843	0.859
<i>AUC</i>	0.91736	0.93724
<i>Precision_Recall</i>	0.914396	0.944415
<i>Average_Loss</i>	0.382117	0.334469
<i>loss</i>	0.382114	0.334465
<i>precision</i>	0.832818	0.868474
<i>predictionMean</i>	0.518233	0.516113
<i>recall</i>	0.866	0.854
<i>TrueFails</i>	20.5	21.6
<i>TruePasses</i>	21.65	21.35
<i>FalseFails</i>	3.35	3.65
<i>FalsePasses</i>	4.5	3.4

Table 10-33: COMPLEX CA Threshold-0.1 Classifier

10.7.1.3 Threshold 0.15

<i>Week</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>
<i>Accuracy</i>	0.684	0.684	0.81	0.821
<i>AUC</i>	0.7478	0.7426	0.85368	0.89976
<i>Precision_Recall</i>	0.729168	0.749041	0.800013	0.894099
<i>Average_Loss</i>	0.590881	0.599657	0.470172	0.413643
<i>loss</i>	0.590887	0.599659	0.470174	0.413645
<i>precision</i>	0.655327	0.655919	0.783417	0.792379
<i>predictionMean</i>	0.523466	0.513708	0.517948	0.517893
<i>recall</i>	0.82	0.822	0.862	0.88
<i>TrueFails</i>	13.7	13.65	18.95	19.05
<i>TruePasses</i>	20.5	20.55	21.55	22
<i>FalseFails</i>	4.5	4.45	3.45	3
<i>FalsePasses</i>	11.3	11.35	6.05	5.95

Table 10-34: COMPLEX CA Threshold-0.15 Classifier

<i>Week</i>	7	8	9	10
<i>Accuracy</i>	0.841	0.853	0.858	0.876
<i>AUC</i>	0.91952	0.92292	0.92384	0.94096
<i>Precision_Recall</i>	0.91156	0.920781	0.92482	0.948239
<i>Average_Loss</i>	0.380421	0.374718	0.375989	0.3275
<i>loss</i>	0.380423	0.374716	0.375988	0.327499
<i>precision</i>	0.825157	0.84398	0.842022	0.881976
<i>predictionMean</i>	0.517085	0.514287	0.52357	0.518879
<i>recall</i>	0.874	0.874	0.888	0.876
<i>TrueFails</i>	20.2	20.8	20.7	21.9
<i>TruePasses</i>	21.85	21.85	22.2	21.9
<i>FalseFails</i>	3.15	3.15	2.8	3.1
<i>FalsePasses</i>	4.8	4.2	4.3	3.1

Table 10-35: COMPLEX CA Threshold-0.15 Classifier

10.7.2 Written Exam

10.7.2.1 Threshold 0

<i>Week</i>	3	4	5	6
<i>Accuracy</i>	0.706	0.7	0.704	0.756
<i>AUC</i>	0.74748	0.75508	0.78124	0.82808
<i>Precision_Recall</i>	0.729717	0.734802	0.773271	0.815417
<i>Average_Loss</i>	0.599087	0.597826	0.585641	0.530091
<i>loss</i>	0.599087	0.597824	0.585643	0.53009
<i>precision</i>	0.681052	0.676059	0.682344	0.7473
<i>predictionMean</i>	0.518862	0.517005	0.514705	0.520404
<i>recall</i>	0.786	0.78	0.774	0.796
<i>TrueFails</i>	15.65	15.5	15.85	17.9
<i>TruePasses</i>	19.65	19.5	19.35	19.9
<i>FalseFails</i>	5.35	5.5	5.65	5.1
<i>FalsePasses</i>	9.35	9.5	9.15	7.1

Table 10-36: COMPLEX Written Exam Threshold-0 Classifier

<i>Week</i>	7	8	9	10
<i>Accuracy</i>	0.749	0.752	0.738	0.751
<i>AUC</i>	0.8354	0.8256	0.82396	0.81632
<i>Precision_Recall</i>	0.823775	0.816269	0.80966	0.815072
<i>Average_Loss</i>	0.517992	0.535543	0.539123	0.552851
<i>loss</i>	0.517988	0.535544	0.539119	0.55285
<i>precision</i>	0.742617	0.735021	0.730168	0.759215
<i>predictionMean</i>	0.51575	0.527733	0.516182	0.496454
<i>recall</i>	0.786	0.802	0.782	0.756
<i>TrueFails</i>	17.8	17.55	17.35	18.65
<i>TruePasses</i>	19.65	20.05	19.55	18.9
<i>FalseFails</i>	5.35	4.95	5.45	6.1
<i>FalsePasses</i>	7.2	7.45	7.65	6.35

Table 10-37: COMPLEX Written Exam Threshold-0 Classifier

10.7.2.2 Threshold-0.1

<i>Week</i>	5	6	7
<i>Accuracy</i>	0.708	0.776	0.776
<i>AUC</i>	0.7708	0.84496	0.85552
<i>Precision_Recall</i>	0.770311	0.823149	0.846512
<i>Average_Loss</i>	0.587342	0.482156	0.475681
<i>loss</i>	0.587342	0.482154	0.47568
<i>precision</i>	0.679497	0.757443	0.763803
<i>predictionMean</i>	0.522417	0.508226	0.510422
<i>recall</i>	0.794	0.824	0.818
<i>TrueFails</i>	15.55	18.2	18.35
<i>TruePasses</i>	19.85	20.6	20.45
<i>FalseFails</i>	5.15	4.4	4.55
<i>FalsePasses</i>	9.45	6.8	6.65

Table 10-38: COMPLEX Written Exam Threshold-0.1 Classifier

<i>Week</i>	8	9	10
<i>Accuracy</i>	0.774	0.782	0.767
<i>AUC</i>	0.85272	0.85244	0.85076
<i>Precision_Recall</i>	0.847837	0.84706	0.846402
<i>Average_Loss</i>	0.484081	0.479841	0.488752
<i>loss</i>	0.484077	0.479841	0.48875
<i>precision</i>	0.763949	0.76815	0.763241
<i>predictionMean</i>	0.509686	0.512389	0.50823
<i>recall</i>	0.81	0.822	0.79
<i>TrueFails</i>	18.45	18.55	18.6
<i>TruePasses</i>	20.25	20.55	19.75
<i>FalseFails</i>	4.75	4.45	5.25
<i>FalsePasses</i>	6.55	6.45	6.4

Table 10-39: COMPLEX Written Exam Threshold-0.1 Classifier

10.7.2.3 Threshold 0.15

<i>Week</i>	6	7	8	9	10
<i>Accuracy</i>	0.782	0.778	0.779	0.779	0.783
<i>AUC</i>	0.84644	0.84084	0.839	0.84304	0.85692
<i>Precision_Recall</i>	0.825687	0.805755	0.81865	0.819155	0.851114
<i>Average_Loss</i>	0.494019	0.49595	0.503474	0.501981	0.481465
<i>loss</i>	0.494021	0.49595	0.50347	0.501979	0.481469
<i>precision</i>	0.751705	0.738913	0.748277	0.746889	0.752848
<i>predictionMean</i>	0.514158	0.533649	0.516537	0.516678	0.516898
<i>recall</i>	0.852	0.87	0.854	0.856	0.852
<i>TrueFails</i>	17.8	17.15	17.6	17.55	17.85
<i>TruePasses</i>	21.3	21.75	21.35	21.4	21.3
<i>FalseFails</i>	3.7	3.25	3.65	3.6	3.7
<i>FalsePasses</i>	7.2	7.85	7.4	7.45	7.15

Table 10-40: COMPLEX Written Exam Threshold-0.15 Classifier

10.8 Full Classifier Results HOG

10.8.1 CA

10.8.1.1 Threshold-0

<i>Week</i>	2	3	4	5
<i>Accuracy</i>	0.654	0.723	0.74	0.789
<i>AUC</i>	0.70724	0.80272	0.81164	0.86024
<i>Precision_Recall</i>	0.672865	0.77958	0.784943	0.824922
<i>Average_Loss</i>	0.624949	0.550188	0.535896	0.473129
<i>loss</i>	0.624949	0.550183	0.535899	0.47313
<i>precision</i>	0.61582	0.703158	0.716657	0.758027
<i>predictionMean</i>	0.516041	0.509815	0.503578	0.510214
<i>recall</i>	0.828	0.784	0.802	0.854
<i>TrueFails</i>	12	16.55	16.95	18.1
<i>TruePasses</i>	20.7	19.6	20.05	21.35
<i>FalseFails</i>	4.3	5.4	4.95	3.65
<i>FalsePasses</i>	13	8.45	8.05	6.9

Table 10-41: HOG CA Threshold-0 Classifier

<i>Week</i>	6	7	8	9	10
<i>Accuracy</i>	0.816	0.838	0.836	0.793	0.823
<i>AUC</i>	0.91196	0.91456	0.92204	0.89608	0.93324
<i>Precision_Recall</i>	0.917827	0.916947	0.926377	0.905773	0.94217
<i>Average_Loss</i>	0.401126	0.385291	0.375386	0.470676	0.394716
<i>loss</i>	0.401124	0.385295	0.375387	0.470674	0.394719
<i>precision</i>	0.790631	0.828401	0.823564	0.790046	0.826379
<i>predictionMean</i>	0.52014	0.520376	0.523299	0.519715	0.510721
<i>recall</i>	0.864	0.86	0.86	0.804	0.826
<i>TrueFails</i>	19.2	20.4	20.3	19.55	20.5
<i>TruePasses</i>	21.6	21.5	21.5	20.1	20.65
<i>FalseFails</i>	3.4	3.5	3.5	4.9	4.35
<i>FalsePasses</i>	5.8	4.6	4.7	5.45	4.5

Table 10-42: HOG CA Threshold-0 Classifier

10.8.1.2 Threshold 0.1

<i>Week</i>	3	4	5
<i>Accuracy</i>	0.719	0.711	0.79
<i>AUC</i>	0.79404	0.79348	0.85812
<i>Precision_Recall</i>	0.778512	0.776772	0.819388
<i>Average_Loss</i>	0.552793	0.554057	0.468464
<i>loss</i>	0.55279	0.55406	0.468463
<i>precision</i>	0.699248	0.694329	0.765839
<i>predictionMean</i>	0.505998	0.499822	0.516983
<i>recall</i>	0.776	0.76	0.842
<i>TrueFails</i>	16.55	16.55	18.45
<i>TruePasses</i>	19.4	19	21.05
<i>FalseFails</i>	5.6	6	3.95
<i>FalsePasses</i>	8.45	8.45	6.55

Table 10-43: HOG CA Threshold-0.1 Classifier

<i>Week</i>	6	7	8
<i>Accuracy</i>	0.815	0.827	0.847
<i>AUC</i>	0.9084	0.9178	0.92192
<i>Precision_Recall</i>	0.914411	0.921446	0.926431
<i>Average_Loss</i>	0.403165	0.390456	0.382811
<i>loss</i>	0.40316	0.390456	0.382812
<i>precision</i>	0.793439	0.807317	0.83733
<i>predictionMean</i>	0.517343	0.512925	0.509701
<i>recall</i>	0.856	0.866	0.868
<i>TrueFails</i>	19.35	19.7	20.65
<i>TruePasses</i>	21.4	21.65	21.7
<i>FalseFails</i>	3.6	3.35	3.3
<i>FalsePasses</i>	5.65	5.3	4.35

Table 10-44: HOG CA Threshold-0.1 Classifier

<i>Week</i>	<i>9</i>	<i>10</i>
<i>Accuracy</i>	0.846	0.858
<i>AUC</i>	0.9242	0.94196
<i>Precision_Recall</i>	0.929208	0.951258
<i>Average_Loss</i>	0.376153	0.326052
<i>loss</i>	0.376161	0.326052
<i>precision</i>	0.829256	0.85377
<i>predictionMean</i>	0.513526	0.507484
<i>recall</i>	0.876	0.868
<i>TrueFails</i>	20.4	21.2
<i>TruePasses</i>	21.9	21.7
<i>FalseFails</i>	3.1	3.3
<i>FalsePasses</i>	4.6	3.8

Table 10-45: HOG CA Threshold-0.1 Classifier

10.8.1.3 Threshold 0.15

<i>Week</i>	<i>3</i>	<i>4</i>	<i>5</i>
<i>Accuracy</i>	0.694	0.642	0.812
<i>AUC</i>	0.76112	0.70112	0.852
<i>Precision_Recall</i>	0.735319	0.672912	0.79263
<i>Average_Loss</i>	0.583244	0.610482	0.466722
<i>loss</i>	0.583243	0.610474	0.466724
<i>precision</i>	0.675557	0.62487	0.782215
<i>predictionMean</i>	0.515215	0.517748	0.51278
<i>recall</i>	0.766	0.804	0.872
<i>TrueFails</i>	15.55	12	18.8
<i>TruePasses</i>	19.15	20.1	21.8
<i>FalseFails</i>	5.85	4.9	3.2
<i>FalsePasses</i>	9.45	13	6.2

Table 10-46: HOG CA Threshold-0.15 Classifier

<i>Week</i>	6	7	8
<i>Accuracy</i>	0.813	0.842	0.853
<i>AUC</i>	0.91044	0.92216	0.9278
<i>Precision_Recall</i>	0.915991	0.923661	0.930221
<i>Average_Loss</i>	0.395246	0.377103	0.361779
<i>loss</i>	0.395244	0.377103	0.361775
<i>precision</i>	0.788346	0.825216	0.835917
<i>predictionMean</i>	0.516776	0.514567	0.517859
<i>recall</i>	0.862	0.874	0.884
<i>TrueFails</i>	19.1	20.25	20.55
<i>TruePasses</i>	21.55	21.85	22.1
<i>FalseFails</i>	3.45	3.15	2.9
<i>FalsePasses</i>	5.9	4.75	4.45

Table 10-47: HOG CA Threshold-0.15 Classifier

<i>Week</i>	9	10
<i>Accuracy</i>	0.856	0.876
<i>AUC</i>	0.92704	0.94424
<i>Precision_Recall</i>	0.930653	0.953114
<i>Average_Loss</i>	0.364385	0.311724
<i>loss</i>	0.364393	0.311722
<i>precision</i>	0.838975	0.874932
<i>predictionMean</i>	0.519358	0.517988
<i>recall</i>	0.886	0.884
<i>TrueFails</i>	20.65	21.7
<i>TruePasses</i>	22.15	22.1
<i>FalseFails</i>	2.85	2.9
<i>FalsePasses</i>	4.35	3.3

Table 10-48: HOG CA Threshold-0.15 Classifier

10.8.2 Written Exam

10.8.2.1 Threshold-0

<i>Week</i>	2	3	4	5
<i>Accuracy</i>	0.552	0.656	0.632	0.697
<i>AUC</i>	0.63144	0.71588	0.70228	0.77724
<i>Precision_Recall</i>	0.618634	0.692666	0.687021	0.761632
<i>Average_Loss</i>	0.772289	0.637888	0.654496	0.589386
<i>loss</i>	0.772288	0.637887	0.654496	0.589386
<i>precision</i>	0.564355	0.638821	0.636448	0.678242
<i>predictionMean</i>	0.577493	0.540688	0.542723	0.51104
<i>recall</i>	0.708	0.782	0.782	0.762
<i>TrueFails</i>	9.9	13.25	12.05	15.8
<i>TruePasses</i>	17.7	19.55	19.55	19.05
<i>FalseFails</i>	7.3	5.45	5.45	5.95
<i>FalsePasses</i>	15.1	11.75	12.95	9.2

Table 10-49: HOG Written Exam Threshold-0 Classifier

<i>Week</i>	6	7	8	9	10
<i>Accuracy</i>	0.702	0.704	0.711	0.713	0.72
<i>AUC</i>	0.77532	0.79056	0.79012	0.79292	0.80484
<i>Precision_Recall</i>	0.76625	0.782982	0.78133	0.784411	0.799473
<i>Average_Loss</i>	0.61816	0.602191	0.609482	0.604987	0.621336
<i>loss</i>	0.618163	0.602186	0.609482	0.604986	0.62134
<i>precision</i>	0.684606	0.682321	0.689194	0.692157	0.694934
<i>predictionMean</i>	0.532196	0.554068	0.549235	0.555158	0.556399
<i>recall</i>	0.762	0.784	0.788	0.788	0.798
<i>TrueFails</i>	16.05	15.6	15.85	15.95	16.05
<i>TruePasses</i>	19.05	19.6	19.7	19.7	19.95
<i>FalseFails</i>	5.95	5.4	5.3	5.3	5.05
<i>FalsePasses</i>	8.95	9.4	9.15	9.05	8.95

Table 10-50: HOG Written Exam Threshold-0 Classifier

10.8.2.2 Threshold-0.1

<i>Week</i>	3	4	5
<i>Accuracy</i>	0.67	0.687	0.736
<i>AUC</i>	0.75556	0.75896	0.81068
<i>Precision_Recall</i>	0.742749	0.735701	0.806698
<i>Average_Loss</i>	0.607336	0.616132	0.545871
<i>loss</i>	0.607333	0.616132	0.545874
<i>precision</i>	0.670086	0.684081	0.717977
<i>predictionMean</i>	0.514041	0.520497	0.515712
<i>recall</i>	0.702	0.702	0.79
<i>TrueFails</i>	15.95	16.8	17.05
<i>TruePasses</i>	17.55	17.55	19.75
<i>FalseFails</i>	7.45	7.45	5.25
<i>FalsePasses</i>	9.05	8.2	7.95

Table 10-51: HOG Written Exam Threshold-0.1 Classifier

<i>Week</i>	6	7	8
<i>Accuracy</i>	0.783	0.789	0.791
<i>AUC</i>	0.87676	0.87196	0.87552
<i>Precision_Recall</i>	0.880063	0.876539	0.88251
<i>Average_Loss</i>	0.460019	0.466028	0.460885
<i>loss</i>	0.460016	0.466029	0.460885
<i>precision</i>	0.769455	0.772684	0.778258
<i>predictionMean</i>	0.510909	0.517068	0.514362
<i>recall</i>	0.824	0.832	0.828
<i>TrueFails</i>	18.55	18.65	18.85
<i>TruePasses</i>	20.6	20.8	20.7
<i>FalseFails</i>	4.4	4.2	4.3
<i>FalsePasses</i>	6.45	6.35	6.15

Table 10-52: HOG Written Exam Threshold-0.1 Classifier

<i>Week</i>	<i>9</i>	<i>10</i>
<i>Accuracy</i>	0.793	0.775
<i>AUC</i>	0.87496	0.87008
<i>Precision_Recall</i>	0.881794	0.870965
<i>Average_Loss</i>	0.459104	0.459218
<i>loss</i>	0.459099	0.459217
<i>precision</i>	0.779337	0.772077
<i>predictionMean</i>	0.515584	0.501164
<i>recall</i>	0.83	0.792
<i>TrueFails</i>	18.9	18.95
<i>TruePasses</i>	20.75	19.8
<i>FalseFails</i>	4.25	5.2
<i>FalsePasses</i>	6.1	6.05

Table 10-53: HOG Written Exam Threshold-0.1 Classifier

10.8.2.3 Threshold-0.15

<i>Week</i>	<i>5</i>	<i>6</i>	<i>7</i>
<i>Accuracy</i>	0.719	0.784	0.775
<i>AUC</i>	0.78284	0.87764	0.8756
<i>Precision_Recall</i>	0.761441	0.88246	0.878185
<i>Average_Loss</i>	0.582213	0.454391	0.459704
<i>loss</i>	0.58222	0.454391	0.459702
<i>precision</i>	0.681394	0.767087	0.754203
<i>predictionMean</i>	0.514179	0.510067	0.516727
<i>recall</i>	0.856	0.828	0.826
<i>TrueFails</i>	14.55	18.5	18.1
<i>TruePasses</i>	21.4	20.7	20.65
<i>FalseFails</i>	3.6	4.3	4.35
<i>FalsePasses</i>	10.45	6.5	6.9

Table 10-54: HOG Written Exam Threshold-0.15 Classifier

<i>Week</i>	8	9	10
<i>Accuracy</i>	0.786	0.78	0.781
<i>AUC</i>	0.87844	0.87688	0.8704
<i>Precision_Recall</i>	0.881142	0.880199	0.879025
<i>Average_Loss</i>	0.455415	0.461232	0.461408
<i>loss</i>	0.455418	0.461231	0.461411
<i>precision</i>	0.768744	0.764232	0.766455
<i>predictionMean</i>	0.517419	0.509222	0.516676
<i>recall</i>	0.83	0.82	0.822
<i>TrueFails</i>	18.55	18.5	18.5
<i>TruePasses</i>	20.75	20.5	20.55
<i>FalseFails</i>	4.25	4.5	4.45
<i>FalsePasses</i>	6.45	6.5	6.5

Table 10-55: HOG Written Exam Threshold-0.15 Classifier

10.9 Sample VPL and MULE Scripts

10.9.1 vpl_run.sh

```

#!/bin/bash
cat > vpl_execution <<EEOOFF
#!/bin/bash
progl=HelloWorld
javac \${progl}.java &> grepLines.out
if ((\ $? > 0)); then
echo "Error compiling your program"
cat grepLines.out
exit
fi
java \${progl}
EEOOFF
chmod +x vpl_execution

```

Figure 10-1: Sample vpl_run.sh

10.9.2 vpl_evaluate.sh

```

#!/bin/bash

cat > vpl_execution <<EEOOFF
#!/bin/bash

# ----- PROGRAMS TESTED (WITHOUT EXTENSION) -----
-----
progl=Printing
compiled=true

# ----- STARTING GRADE -----
-----
grade=0

```

```

# ----- COMPILE STUDENT PROG -----
-----
javac \${prog1}.java &> grepLines.out

#--- if error, assign a minimal grade ---
if ((\ $? > 0)); then
    echo "Comment :=>> Your program has compiler
Errors. Use the Run command to help solve the
errors."
    cat grepLines.out
    echo "Comment :=>> -----"
    compiled=false
fi

if [ \${compiled} = true ] ; then
    grade=\$((grade+10))
fi

# ----- Remove comments from the code -----
-----

cat \${prog1}.java | sed 's://.*$:g' | sed
'/\/*\*/,\/*\*/ {s/*\*/.*//p; d}' > _\${prog1}.java

# ----- TEST THE CODE FOR PARTICULAR PATTERNS -
-----
# ----- TEST Code -----

if grep 'public *static *void *main' \${prog1}.java
then
    grade=\$((grade+20))
else
    echo "Comment :=>> you have no main method
created in Printing.java"
    echo "Comment :=>> -----"
fi

grep 'class *Printing' _\${prog1}.java | grep -v main
&> grepLines.out

if [ ! -s grepLines.out ] ;
then
    echo "Comment :=>> you have not created your
class called Printing in Printing.java"
    echo "Comment :=>> -----"
else
    grade=\$((grade+20))
fi

grep 'System.out.print' _\${prog1}.java | grep -v
main &> grepLines.out

if [ ! -s grepLines.out ] ;
then
    echo "Comment :=>> you have not created your
print statement in Printing.java"
    echo "Comment :=>> -----"
else

```



```

        if ((\ $? > 0)); then
            echo "Comment :=>> Your output is
incorrect."

            #--- display test file ---
            #echo "Comment :=>> Your program tested
with:"
            #echo "<|--"
            #cat data\${i}.txt
            #echo "--|>"

            echo "Comment :=>> -----"
            echo "Comment :=>> Your output:"
            echo "Comment :=>> -----"
            echo "<|--"
            cat user.out.org
            echo "--|>"
            echo ""
            echo "Comment :=>> -----"
            echo "Comment :=>> Expected output: "
            echo "Comment :=>> -----"
            echo "<|--"
            cat data\${i}.out
            echo "--|>"

            # ----- REWARD IF CORRECT
OUTPUT -----
        else
            #--- good output ---
            echo "Comment :=>> Congrats, your output is
correct."
            echo "Comment :=>> -----"
            echo "Comment :=>> Your output:"
            echo "Comment :=>> -----"
            echo "<|--"
            cat user.out.org
            echo "--|>"
            grade=\$((grade+30))
        fi
    done
fi

if (( grade > 100 )); then
    grade=100
fi

echo "Grade :=>> \$grade"

EEOFF

chmod +x vpl_execution

```

Figure 10-2: Sample vpl_evaluate.sh

10.9.3: vpl_compile.sh

```
#!/bin/bash

cat > vpl_execution <<EEOOFF
#!/bin/bash

prog1=Printing

javac \${prog1}.java &> grepLines.out

if ((\ $? > 0)); then
    echo "Error compiling your program"
    cat grepLines.out
    exit
else
    echo "Compilation succeeded"
    echo "compiled: ==> true"
fi

EEOOFF
```

Figure 10-3: Sample vpl_compile.sh

10.9.4: metadata.json

```
{
  "title": "Hello World",
  "Requested files": ["HelloWorld.java"],
  "qid": "CS1_Lab1_helloWorld"
}
```

Figure 10-4: Sample metadata.json

10.9.5: description.html

```
<H5><strong>Description</strong></H5>
<p>
  Write a java program which prints the message
  "Hello" on one line and
  "world!" on the next.
</p>
<H5><strong>Sample Output</strong></H5>
<pre>Hello<br>World!</pre>
```

Figure 10-5: Sample description.html

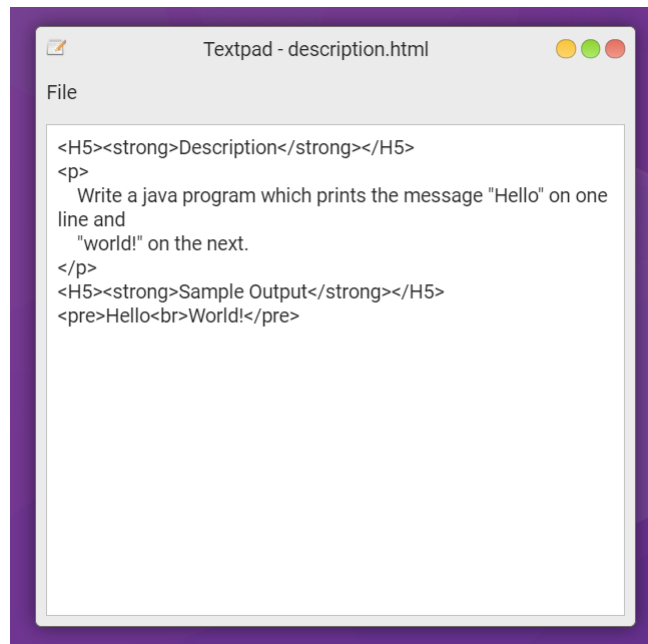


Figure 10-6: Sample description.html in MULE

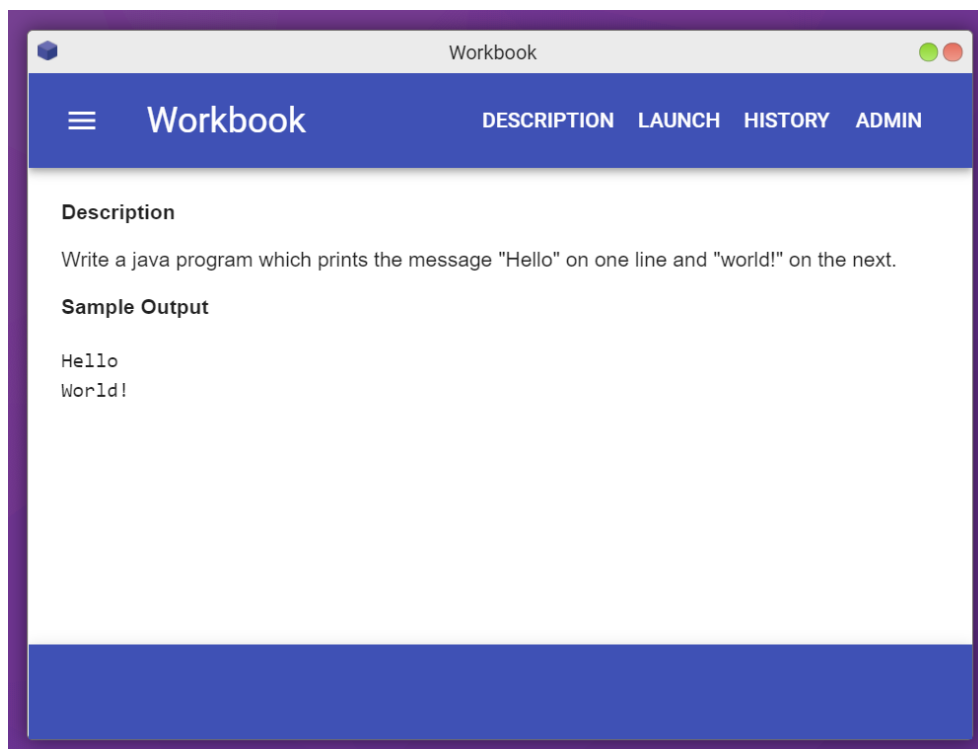


Figure 10-7: Workbook display of description.html

Bibliography

- [1] T. Beaubouef and J. Mason, "Why the High Attrition Rate for Computer Science Students: Some Thoughts and Observations," *ACM SIGCSE Bulletin*, vol. 37, no. 2, pp. 103-106, 2005.
- [2] M. Biggers, A. Brauer and T. Yilmaz, "Student Perceptions of Computer Science: A Retention Study Comparing Graduating Seniors with CS Leavers," *ACM SIGCSE Bulletin*, vol. 40, no. 1, pp. 402-406, 2008.
- [3] M. N. Giannakos, I. O. Pappas, L. Jaccheri and D. G. Sampson, "Understanding Student Retention in Computer Science Education: The Role of Environment, Gains, Barriers and Usefulness," *Education and Information Technologies*, vol. 22, no. 5, pp. 2365-2382, 2017.
- [4] P. Kinnunen and L. Malmi, "Why Students Drop Out CS1 Course," in *Proceedings of the second international workshop on Computing education research*, Canterbury, United Kingdom, 2006.
- [5] J. Bennedsen and M. E. Caspersen, "Failure Rates in Introductory Programming," *ACM SIGCSE Bulletin*, vol. 39, no. 2, pp. 32-36, 2007.
- [6] N. Culligan and K. Casey, "Building an Authentic Novice Programming Lab Environment," in *International Conference on Engaging Pedagogy*, Dublin, Ireland, 2018.
- [7] J. C. Rodríguez-del-Pino, E. Rubio-Royo and Z. J. Hernández-Figueroa, "A Virtual Programming Lab for Moodle with Automatic Assessment and Anti-Plagiarism Features.," in *Proceedings of The 2012 Internacional Conference on e-Learning, e-Business, Enterprise Information Systems, & e-Government.*, Las Vegas, Nevada, USA, 2012.
- [8] B. Kitchenham, "Procedures for Performing Systematic Reviews," *Keele, UK, Keele University*, vol. 33, pp. 1-26, 2004.
- [9] O. Bohl, J. Schellhase, R. Sengler and U. Winand, "The Sharable Content Object Reference Model (SCORM) - A Critical Review," in *International Conference on Computers in Education, 2002. Proceedings*, Auckland, New Zealand, 2002.
- [10] D. Thiébaud, "Automatic Evaluation of Computer Programs Using Moodle's Virtual Programming Lab (VPL) Plug-In," *Journal of Computing Sciences in Colleges*, vol. 30, no. 6, pp. 145-151, 2015.
- [11] N. Duncan, "Feed-Forward: Improving Students' Use of Tutors' Comments," in *7th international technology, education and development conference (INTED2013)*, Valencia, Spain, 2013.
- [12] B. Getzlaf, B. Perry, G. Toffner, K. Lamarche and M. Edwards, "Effective

- Instructor Feedback: Perceptions of Online Graduate students,” *Journal of Educators Online*, vol. 6, no. 2, pp. 1-22, 2009.
- [13] H. Kitaya and I. Ushio , “An Online Automated Scoring System for Java Programming Assignment,” *International Journal of Information and Education Technology*, vol. 6, no. 4, pp. 275-279, 2016.
- [14] S. L. Jackson, S. J. Stratford, J. S. Krajcik and E. Soloway, “Model-It: A Case Study of Learner-Centered Design Software for Supporting Model Building,” in *Working Conference on Applications of Technology in the Science Classroom*, 1995.
- [15] P. Sharmaa and M. J. Hannafinb, “Scaffolding in Technology-Enhanced Learning Environments,” *Interactive learning environments*, vol. 15, no. 1, pp. 27-46, 2007.
- [16] P. Blikstein, “Using Learning Analytics to Assess Students' Behavior in Open-Ended Programming Tasks,” in *Proceedings of the 1st international conference on learning analytics and knowledge*, Alberta, Canada, 2011.
- [17] M. Berland, T. Martin, T. Benton, C. Petrick Smith and D. Davis, “Using Learning Analytics to Understand the Learning Pathways of Novice Programmers,” *Journal of the Learning Sciences*, vol. 22, no. 4, pp. 564-599, 2013.
- [18] M. K. Bradshaw, “Ante Up: A Framework to Strengthen Student-Based Testing of Assignments,” in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, Kansas City, Missouri, 2015.
- [19] J. C. Caiza and J. M. Del Alamo, “Architecture to Support Automatic Grading Processes in Programming Teaching,” *Revista Politécnica*, vol. 36, no. 1, pp. 63-63, 2015.
- [20] M. Kölling, “Using BlueJ to introduce programming,” in *Reflections on the Teaching of Programming*, Berlin, Heidelberg, Springer, 2008, pp. 98-115.
- [21] M. Kölling and J. Rosenberg, “An Object-Oriented Program Development Environment for the First Programming Course,” in *SIGCSE '96: Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education*, Philadelphia, Pennsylvania, USA, 1996.
- [22] D. Hagan and S. Markham, “Teaching Java with the BlueJ Environment,” in *Proceedings of Australasian Society for Computers in Learning in Tertiary Education Conference ASCILITE*, Coffs Harbour, Australia, 2000.
- [23] R. da Silva Ribeiro, L. de Oliveira Brandão, T. V. Machado Faria and A. A. Franco Brandao, “Programming Web-Course Analysis: How to Introduce Computer Programming?,” in *IEEE Frontiers in Education Conference (FIE) Proceedings.*, Madrid, Spain, 2014.

- [24] M. C. Jadud, "A First Look at Novice Compilation," *Computer Science Education*, vol. 15, no. 1, pp. 25-40., 2005.
- [25] R. Romli, S. Shahida and Z. Z. Kamal , "Automatic Programming Assessment and Test Data Generation a Review on its Approaches," in *International Symposium on Information Technology*, Kuala Lumpur, Malaysia, 2010.
- [26] S. G. Hart and L. E. Staveland, "Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research.," *Advances in psychology.*, vol. 52, pp. 139-183, 1988.
- [27] X. Bai, A. Ola, S. Akkaladevi and Y. Cui, "Enhancing the Learning Process in Programming Courses Through an Automated Feedback and Assignment Management System," *Issues in Information Systems*, vol. 17, no. 3, pp. 165-175, 2016.
- [28] M. C. Jadud, "An Exploration of Novice Compilation Behaviour in BlueJ.," *PhD diss., University of Kent.*, 2006.
- [29] P. N. D., C. Hancock, R. Hobbs, F. Martin and R. Simmons, "Conditions of Learning in Novice Programmers," *Journal of Educational Computing Research*, vol. 2, no. 1, pp. 37-55, 1986.
- [30] K. Casey, "Using Keystroke Analytics to Improve Pass-Fail Classifiers," *Journal of Learning Analytics*, vol. 4, no. 2, pp. 189-211, 2017.
- [31] A. Vihavainen, M. Luukkainen and P. Ihanola, "Analysis of Source Code Snapshot Granularity Levels," *Proceedings of the 15th Annual Conference on Information Technology*, pp. 21-26, 2014.
- [32] K. Castro-Wunsch, A. Ahadi and A. Petersen, "Evaluating Neural Networks as a Method for Identifying Students in Need of Assistance," in *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education*, Seattle, Washington, USA, 2017.
- [33] A. L. R. Ahadi, H. Haapala and A. Vihavainen, "Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance," in *Proceedings of the eleventh annual international conference on international computing education research*, Omaha, Nebraska, USA, 2015.
- [34] S. Bergin and R. Reilly, "Programming: Factors that Influence Success," in *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, Louis, Missouri, USA, 2005.
- [35] C. D. Hundhausen and O. Adesope, "The Normalized Programming State Model: Predicting Student Performance in Computing Courses Based on Programming Behavior," in *Proceedings of the eleventh annual International Conference on International Computing Education Research*, Omaha, Nebraska, USA, 2015.

- [36] D. Sun, P. Paredes and J. Canny, "MouStress: Detecting Stress from Mouse Motion," in *Proceedings of the SIGCHI conference on Human factors in computing systems.*, Toronto, Ontario, Canada, 2014.
- [37] Y. Takashi, "Mouse Trajectories and State Anxiety: Feature Selection with Random Forest," in *Humaine Association Conference on Affective Computing and Intelligent Interaction*, Geneva, Switzerland, 2013.
- [38] S. L. Beilock and T. H. Carr, "When High-Powered People Fail: Working Memory and "Choking Under Pressure" in Math," *Psychological science*, vol. 16, no. 2, pp. 101-105, 2005.
- [39] S. L. Beilock, W. A. Jellison, R. J. Rydell, A. R. McConnell and T. H. Carr, "On the Causal Mechanisms of Stereotype Threat: Can Skills that Don't Rely Heavily on Working Memory Still be Threatened," *Personality and Social Psychology Bulletin*, vol. 32, no. 8, pp. 1059-1071, 2006.
- [40] A. Kapoor, W. Burlison and R. W. Picard, "Automatic Prediction of Frustration," *International journal of human-computer studies*, vol. 65, no. 8, pp. 724-736, 2007.
- [41] D. Teague, "Neo-Piagetian Theory and the Novice Programmer," *PhD diss., Queensland University of Technology*, 2015.
- [42] R. Lister, "Concrete and Other Neo-Piagetian Forms of Reasoning in the Novice Programmer," in *Thirteenth Australasian Computing Education Conference*, Perth, Australia, 2011.
- [43] D. Teague, R. Lister and A. Ahadi, "Mired in the Web: Vignettes from Charlotte and Other Novice Programmers," in *17th Australasian Computing Education Conference*, Sydney, Australia, 2015.
- [44] B. Adelson, "Problem Solving and the Development of Abstract Categories in Programming Languages," *Memory & cognition*, vol. 9, no. 4, pp. 422-433, 1981.
- [45] K. Casey and D. Azcona, "Utilizing Student Activity Patterns to Predict Performance," *International Journal of Educational Technology in Higher Education*, vol. 14, no. 1, pp. 1-15, 2017.
- [46] J. Vanvinkenroye, C. Grüniger, C.-J. Heine and T. Richter, "A Quantitative Analysis of a Virtual Programming Lab," in *IEEE International Symposium on Multimedia*, Anaheim, CA, USA, 2013.
- [47] A. Begel and B. Simon, "Novice Software Developers, All Over Again," in *Proceedings of the fourth international workshop on computing education research*, Sydney, Australia, 2008.
- [48] . A. Evenrud, "OS.js," [Online]. Available: <https://www.os-js.org/>. [Accessed 27th February 2021].

- [49] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser and J. Bright, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261-272, 2020.
- [50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer and R. Weiss, "Scikit-Learn: Machine Learning in Python.," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.
- [51] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu and X. Zheng, "Tensorflow: A System for Large-Scale Machine Learning.," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI})*, Savannah, GA, USA, 2016.
- [52] J. Wahlström, M. Hagberg, P. Johnson, J. Svensson and D. Rempel, "Influence of Time Pressure and Verbal Provocation on Physiological and Psychological Reactions During Work with a Computer Mouse," *European Journal of Applied Physiology*, vol. 87, no. 3, pp. 257-263, 2002.
- [53] S. Bergin and R. Reilly, "The Influence of Motivation and Comfort-Level on Learning to Program," in *17th Workshop of the Psychology of Programming Interest Group*, Brighton, UK, 2005.
- [54] B. C. Wilson and S. Shrock, "Contributing to Success in an Introductory Computer Science Course: a Study of Twelve Factors," *ACM SIGSCE Bulletin*, vol. 33, no. 1, p. 133.1, 2001.
- [55] N. Culligan and K. Casey, "What the Mouse Said: How Mouse Movements Can Relate to Student Stress and Success," in *Psychology of Programming Interest Group*, Toronto, Canada, 2020.
- [56] N. Culligan and K. Casey, "Exploring the Coding Behaviour of Successful Students in Programming by Employing Neo-Piagetian Theory," in *Psychology of Programming Interest Group*, Toronto, Canada, 2020.
- [57] A. Hagberg, P. J. Swart and S. A. Daniel, "Exploring Network Structure, Dynamics, and Function Using NetworkX," in *Proceedings of the 7th Python in Science Conference*, Pasadena, CA, 2008.
- [58] T. J. McCabe and C. W. Butler, "Design Complexity Measurement and Testing," *Communications of the ACM*, vol. 32, no. 12, p. 1989, 1989.
- [59] J. Sitthiworachart and M. Joy, "Effective Peer Assessment for Learning Computer," *ACM SIGCSE Bulletin*, vol. 36, no. 3, pp. 122-126, 2004.
- [60] A. Radermacher and G. Walia, "Gaps Between Industry Expectations and the Abilities of Graduates," in *Proceeding of the 44th ACM technical symposium*

on Computer science education, Denver, Colorado, USA, 2013.

- [61] A. Begel and B. Simon, "Struggles of New College Graduates in Their First Software Development Job," in *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, Portland, OR, USA, 2008.
- [62] B. A. Becker, "An Exploration of the Effects of Enhanced Compiler Error Messages for Computer Programming Novices," in *Masters Thesis*, DIT, Dublin, 2015.
- [63] C. Connolly, E. Murphy and S. Moore, "Programming Anxiety Amongst Computing Students—A Key in the Retention Debate?," *IEEE Transactions on Education*, vol. 52, no. 1, pp. 52 - 56, 2009.
- [64] R. da Silva Ribeiro, T. V. M. Faria, L. de Oliveira Brandão and A. A. F. Brandão, "Programming web-course analysis: how to introduce computer programming?," in *IEEE Frontiers in Education Conference (FIE) Proceedings*, Madrid, Spain, 2014 .
- [65] M. Bernard and T. Martin, "Clusters and Patterns of Novice Programmers," in *The meeting of the American Educational Research Association*, New Orleans, USA, 2011.
- [66] J. C. Caiza and J. M. Del Alamo, "Programming Assignments Automatic Grading: Review of Tools and Implementations," in *7th international technology, education and development conference (INTED2013)*, Valencia, Spain, 2013.
- [67] A. Katrutsa and V. Strijov, "Comprehensive Study of Feature Selection Methods to Solve Multicollinearity Problem According to Evaluation Criteria.," *Expert Systems with Applications*, vol. 76, pp. 1-11., 2017.
- [68] T. Martin, T. Benton, C. Petrick Smith and D. Davis, "Using Learning Analytics to Understand the Learning Pathways of Novice Programmers," *Journal of the Learning Sciences*, vol. 22, no. 4, pp. 564-599, 2013.
- [69] H.-T. Pao, "A Comparison of Neural Network and Multiple Regression Analysis in Modeling Capital Structure," *Expert Systems with Applications*, vol. 35, p. 720–727, 2008.