The Connection Between Coding Habits and Coding Success

by

Scott Dougan

A Thesis
presented to
The University of Guelph

In partial fulfilment of requirements
for the degree of
Master of Science
in
Computer Science

Guelph, Ontario, Canada

© Scott Dougan, April, 2015
ABSTRACT

THE CONNECTION BETWEEN CODING HABITS AND CODING SUCCESS

Scott Dougan
University of Guelph, 2015

Advisor: Dr. Judi McCuaig

Many college and university students struggle to grasp programming concepts during their introductory courses. This could be partially caused by the transition towards e-learning software in education which reduces hands-on learning and makes it difficult for instructors to know which students are struggling. This thesis proposes an e-learning system plug-in which allows students to write programs in a web-based development environment, while at the same time logging students’ actions. The logs of coding actions, visually observed events, and participants self-reports were recorded as participants’ completed programming exercises. This data was later analyzed to examine the connections between coding habits and success on programming tasks. By providing more opportunity for hands-on programming practice and helping to inform instructors of the areas in which their students are lacking understanding, the learning experience for students in computer science can be greatly improved.
Acknowledgements

First of all, I would like to thank my advisor, Dr. Judi McCuaig. She provided thought-provoking discussions and gave critical feedback on many occasions. Dr. McCuaig’s door was always open to answer questions, no matter how busy she was. She has been extremely helpful and is a great advisor.

I would also like to thank my partner, Alex Vermeulen, for her positive attitude and willingness to discuss ideas regarding my thesis. She showed interest and was patient and encouraging throughout the thesis process. I could not have done it without her. Alex’s parents, Fred Vermeulen and Mary Jo Hind, were instrumental in their support. They treated me as a member of their family and generously opened their home to me during the completion of my degree.

Also, I would like to extend my gratitude to Marcia Franz who was kind enough to contribute her incredible talent as an editor. Her advice and constructive criticism taught me many things and made an undeniable improvement to this work.

Lastly, and most importantly, I would like to thank my parents, Richard and Jackie Dougan. They were the core of my inspiration and motivation while pursuing my education. I would not be where I am today without their continuous support.
Contents

List of Figures vii
List of Tables ix

1 Introduction 1
  1.1 Thesis Statement 4
  1.2 Overview of Thesis 4

2 Literature Review 6
  2.1 E-learning 7
  2.2 Educational Data Mining 13
    2.2.1 Methods 16
  2.3 Web-based Programming for Hands-on Learning 20
  2.4 Data Collection for Analysis of Coding Actions 23
  2.5 Identifying At-Risk Students 27
    2.5.1 Emotion Detection 28
    2.5.2 Success prediction 31

3 Methodology and Implementation or Experimental Design 34
  3.1 Plug-in Development 35
    3.1.1 Requirements 36
3.1.2 Design .................................................. 38
  3.1.2.1 Study Requirements .......................... 38
  3.1.2.2 Student Requirements ...................... 39
3.1.3 Implementation ................................. 42
  3.1.3.1 Communication and Event Handling ...... 42
  3.1.3.2 Security Precautions ......................... 45
  3.1.3.3 Implementation of Editor Features ........ 46
3.2 Experimental Design ................................. 46
3.3 Materials ............................................... 51
  3.3.1 Participants ..................................... 51
  3.3.2 Experimental Environment .................. 52
  3.3.3 Programming Tasks ............................ 52
  3.3.4 Data Collection and Analysis ............... 53

4 Results, Analysis and Discussion 55
  4.1 Results ............................................... 56
    4.1.1 Summary Statistics ......................... 56
    4.1.2 Correlations .................................. 59
  4.2 Analysis ............................................ 63
    4.2.1 Bayesian Networks .......................... 63
    4.2.2 Decision Trees ................................ 68
  4.3 Discussion .......................................... 71

5 Summary and Future Work 76
  5.1 Future Work ........................................ 77
    5.1.1 Improvements and Validation for the Study .. 78
    5.1.2 Refinement of the Experimental Procedure ... 79
    5.1.3 Improvement and Expansion of the Moodle IDE .. 80
5.2 Conclusion ......................................................... 84

A Ethical Considerations and Participant Package 85
  A.1 Programming Task #1 ........................................ 91
  A.2 Programming Task #2 ........................................ 91
  A.3 Programming Task #3 ........................................ 92
  A.4 Programming Task #4 ........................................ 92

B Data tables ...................................................... 94

C Additional Results .............................................. 98
  References ......................................................... 107
List of Figures

3.1 Screen shot of the Moodle IDE with a Hello World program. . . . 40
3.2 Screen shot of the Moodle IDE with a Hello World program and
run results. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
3.3 Diagram of communication between servers and plug-in. . . . . . 43
3.4 Screen shot of the four different tasks in Moodle. . . . . . . . . . . 50

4.1 Spearman correlation matrix for recorded variables over all four tasks. 60
4.2 Accuracy (as percent correct) for Naïve Bayes and Bayes Net results. 64
4.3 Root mean square error for Naïve Bayes and Bayes Net results. . . 65
4.4 Kappa measure of reliability for Naïve Bayes and Bayes Net results. 66
4.5 Trial 1 - Decision tree predicting completion level formed from all
collected attributes. . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
4.6 Trial 2 - Decision tree predicting completion level formed without
visual observation attributes. . . . . . . . . . . . . . . . . . . . . . . 70
4.7 Trial 3 - Decision tree predicting completion level formed without
visual observations or TLX attributes. . . . . . . . . . . . . . . . . 70
4.8 Trial 4 - Decision tree predicting completion level formed with only
computer observation attributes and time. . . . . . . . . . . . . . . 71

A.1 Page one of the consent form signed by all participants. . . . . . . 87
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.2</td>
<td>Page two of the consent form signed by all participants.</td>
<td>88</td>
</tr>
<tr>
<td>A.3</td>
<td>Pre and post surveys</td>
<td>89</td>
</tr>
<tr>
<td>A.4</td>
<td>TLX survey completed after each of the task attempts. This survey was identical for each of the 4 tasks.</td>
<td>90</td>
</tr>
<tr>
<td>B.1</td>
<td>Density graphs of completion percentage and time to determine normality</td>
<td>97</td>
</tr>
<tr>
<td>C.1</td>
<td>Spearman rank-order correlations for task 1 data.</td>
<td>102</td>
</tr>
<tr>
<td>C.2</td>
<td>Spearman rank-order correlations for task 2 data.</td>
<td>103</td>
</tr>
<tr>
<td>C.3</td>
<td>Spearman rank-order correlations for task 3 data.</td>
<td>104</td>
</tr>
<tr>
<td>C.4</td>
<td>Spearman rank-order correlations for task 4 data.</td>
<td>105</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Summary statistics for all tasks combined. ............................... 57
4.2 Variables showing significant Spearman correlations to completion level across all four tasks. ............................................. 61
4.3 Naïve Bayes Trials. ............................................................. 64
4.4 Naïve Bayes results. ............................................................. 67
A.1 Data Protection Plan containing the considerations taken in order to protect the participants’ privacy. ................................. 86
B.1 Description of attributes used for analysis ......................... 94
B.2 Bins used for Bayes net analysis ......................................... 96
C.1 Task 1 Summary statistics .................................................. 98
C.2 Task 2 Summary statistics .................................................. 99
C.3 Task 3 Summary statistics .................................................. 100
C.4 Task 4 Summary statistics .................................................. 101
C.5 Bayes Net results for determining success. ......................... 106
Chapter 1

Introduction

The use of e-learning in education has greatly increased in recent years. E-learning systems are used as teaching aids in many traditional classroom environments and also as stand-alone learning environments for offering online courses. E-learning offers many benefits some of which include allowing students to access their courses at anytime from anywhere, letting students learn at their own pace, helping instructors manage large class sizes, and increasing the cost effectiveness of hosting courses for institutions (Coates, James, & Baldwin, 2005). However, the shift to an electronic learning environment impacts teachers and students in both positive and negative ways. As the use of e-learning systems become more common, it is important to understand these effects and how they change the learning process.

Although the use of e-learning systems as teaching aids help with the learning process, they also have drawbacks. E-learning systems lack hands-on learning, they cannot identify at-risk students, and they are limited in the quality of feedback they can provide. This is problematic, especially in computer science
programs where a number of recent studies have reported that students’ understanding of programming concepts remain poor even after one or two years of education (Rodrigo et al., 2014; Tabanao, Rodrigo, & Jadud, 2011).

Computer programming is a skill that students learn by doing (Mow, 2008). Unfortunately, e-learning does not offer much opportunity for hands-on learning. At present, e-learning systems rely primarily on forms of assessment that can be automatically marked (such as multiple choice and short answer tests) (Coates et al., 2005), greatly limiting the scope of exercises instructors can assign to their students.

A lack of hands-on experience is one of the main reasons why students have a hard time learning to program (Babb, Longenecker, Baugh, & Feinstein, 2014). To effectively teach programming in an online-only course, the instructor needs to be able to create and administer exercises where the students are actively problem solving through writing code. This ability could be added to e-learning by integrating a web-based development environment directly into the system. Instructors would also be able to incorporate small programming exercises directly into their quizzes and could determine how their students were approaching the programming problems.

With the programming environment accessible through the e-learning software, students would be able to practice coding without first needing to install, learn, and set up all of the prerequisites for writing and executing their programs. Some students do poorly on programming assignments simply because their programs do not run successfully on the environment used for marking. However, their program may have worked perfectly on the student’s own computer. Since all of the students would be working in the same programming environment, they would not have to worry about testing their work in the environment used for assessment.
Another major upside to an integrated programming environment is that data can be collected through the e-learning system as students program. This data could be analyzed to learn valuable information about how students learn to program and where they commonly encounter problems. It could also be added to other data logged by the e-learning system to provide better or more specific results for the classification or clustering of computer science students.

In traditional classroom learning where the classroom size is relatively small, the instructor can identify students who are frustrated, struggling, or at risk of failing. However, as class sizes increase, teaching and learning become less effective (Schanzenbach, 2014). In large classes instructors interact with individual students less frequently, making it more and more difficult to identify those who are struggling (Bascia, 2010). It is also hard for instructors to know the level of understanding their students have about concepts being taught.

In programming classes it is particularly challenging to identify which students are struggling without being able to look over their shoulder as they code. To solve this problem, recent studies have attempted to automatically identify at-risk students (those struggling, frustrated, or likely to fail) from data collected while the students program. To distinguish these students, most studies have taken either an emotion detection approach (Dragon et al., 2008; McQuig-gan, Lee, & Lester, 2007; Picard & Klein, 2002; Rodrigo & Baker, 2009) or a success prediction approach (Baldwin, 2013; Romero, López, Luna, & Ventura, 2013; Kovacic, 2011). The emotion detection method aims to identify frustration, confusion, or anxiety in real time whereas success prediction method attempts to identify students who are likely to fail in the future. For computer science students specifically, researchers have used data logged from programming actions to predict success overall and on the task at hand (Heinonen,
Many researchers have done analysis on at-risk students, however very few studies have included emotional attributes along with the attributes that have shown reliability in identifying successful students. It would be interesting to analyze both groups of data together to determine whether emotion attributes add to the prediction accuracy of student’s success in programming.

Being able to identify when students are struggling is also a big step towards e-learning systems providing better feedback. If the system knew what the student was having trouble with, and the time at which help was needed, then it could generate useful and specific feedback for the task at hand. Alternatively, the system could inform the instructor which students are having difficulties and the instructor could provide assistance.

1.1 Thesis Statement

The thesis statement of this research is that students’ behaviours while coding can be used to predict their success on programming tasks. This thesis creates a web-based development environment integrated with an e-learning system. Students’ automatically logged programming actions were collected and analyzed along with self-reports and visual observations of their behaviours. This thesis has shown that students’ success on programming tasks can be identified by applying machine learning techniques to this data.

1.2 Overview of Thesis

The remainder of this thesis consists of four chapters. Chapter 2 is a literature review exploring e-learning, data mining, and similar experiments that examine
student coding habits. Chapter 3 explains how the experiment was conducted as well as describing the design and construction of the experimental coding environment. Chapter 4 contains the analysis of the experiment data and description of the algorithms used. Lastly, Chapter 5 is a summary of the thesis and ideas for future work.
Chapter 2

Literature Review

Every day the face of learning continues to evolve into a more technological form. This movement towards electronic learning (e-learning) has major effects on both students and teachers. Without a physical classroom and teacher, many students lack supervised hands-on learning in addition to specific and constructive feedback. Likewise, teachers are hindered because it is more difficult for them to identify when their students are frustrated, struggling, or possibly at risk of failing the course. This chapter explores these downsides to e-learning and examines the literature on possible solutions.

This chapter presents a review of the literature which was the motivation behind the experiment carried out in this thesis. Section 2.1 further discusses the benefits and downsides of e-learning. Section 2.2 provides information about the ways in which data collected from students as they learn can be used to improve e-learning systems. The section also presents the data mining and machine learning techniques often used in the analysis of educational data. Section 2.3 investigates the emergence of independent online programming courses and their inclusion of hands-on programming experience within
the lessons they offer. The incorporation of similar web-programming environments and the benefits they could offer within e-learning systems is considered. Finally, Section 2.4 describes the approaches to collecting and analyzing data collected while students program. The section also discusses how this information could be used to improve the learning experience of computer science students. Section 2.5 reviews existing literature on identifying at-risk students using computer automated methods. Research on diagnosing frustration in real time and the research on the prediction of success or failure is explored.

2.1 E-learning

As with many things in today’s world, the advancement of technology has opened new opportunities in education. The use of educational software is quickly increasing in post-secondary institutions and is, and will continue to, radically change learning and teaching (Coates et al., 2005). In 2005 over 70% of institutions in Australia, the UK, and Canada held licenses for either WebCT or Blackboard (both e-learning software) (Coates et al., 2005).

Electronic learning (E-learning) or Learning Management Systems (LMS) are tools that use computer technology to deliver learning materials and create a learning environment that students can access remotely (Coates et al., 2005; Welsh, Wanberg, Brown, & Simmering, 2003; Zhang, Zhao, Zhou, & Nunnemaker, 2004). They assist with the administration, delivery, and management of courses and course material. E-learning systems are primarily web-based software offered through the internet (Welsh et al., 2003). Two examples of e-learning software hosted as websites are Blackboard and Moodle, the two most popular choices for LMS solutions (Machado & Tao, 2007). Building e-learning systems that are accessible through any modern web-browser, makes them convenient for both instructors and students.
The use of e-learning in institutions is only going to increase (Welsh et al., 2003). E-learning systems offer many benefits over traditional classroom learning, but as with anything, they also come with downsides. With education shifting towards a computerized and much more automated path, it is important to understand the ways in which e-learning differs from classroom learning and how these differences affect teachers and students.

Traditional classroom learning (in a physical room with students and a teacher) provides certain advantages to teachers and students. A classroom learning environment is often more familiar and comfortable for both. (Zhang et al., 2004). In a face-to-face setting teachers can motivate or engage their students simply through more interaction and good instruction (Zhang et al., 2004). If the class is small, teachers can also identify which students are struggling and act accordingly. Teachers also have more control over class content and the learning process when instructing in an actual classroom (Zhang et al., 2004).

For students, the structure and routine of traditional learning often helps to keep them on track. Students also benefit from the social community of their classmates. They can easily talk to one another to externalize their ideas and identify possible knowledge gaps they may have (Krause, Stark, & Mandl, 2009). In a classroom setting, students are also able to ask questions and receive immediate feedback (Zhang et al., 2004).

Another major upside to classroom learning is the option for students to learn through supervised hands-on exercises. “Learning by doing” takes advantage of students’ natural learning mechanisms and builds functional skills and knowledge that are relevant to the learner (Schank, 1995). Hands on learning is argued to be the best educational approach, yet is not often put into practice (Schank, 1995). When teaching through an e-learning system, there is a
lack of mechanisms to provide students with hands-on learning. Because of this most e-learning courses lean towards drilling and testing students on content rather than providing relatable real-world experience.

Learning in traditional classrooms also has its disadvantages. Because of the need for a physical classroom and instructor there are time and location constraints (Zhang et al., 2004). Students may not be able to take a course because the time at which it is offered causes scheduling conflicts. The students also need to live close enough to attend the classes, making the course unavailable to students who live too far away to commute. Courses taught using traditional classroom learning also tend to be more expensive to deliver than those offered through e-learning (Zhang et al., 2004). For traditional learning there needs to be an institution with an available classroom, which is expensive to maintain.

Classroom learning is also much less scalable. Again, there would need to be a classroom large enough to hold all of the students, or classes need to be split into multiple sections, requiring more time from the instructor. It is also difficult for instructors to manage large classes. They have less time to dedicate to each student, therefore drastically reducing the amount of individualized attention and feedback. (Zhang et al., 2004).

E-learning systems make managing classes, especially large ones, easier for teachers. They provide the ability to quickly and easily distribute course material such as lectures, assignments, and tutorials. Teachers can post quizzes with automatic start and end times, interact with students through the online forums at any time, and have complete control over when and how students can view course content. For instance, students may be required to complete a quiz on the topics learned in one section before gaining access to content in the next section.
E-learning systems help to keep instructors organized by recording grades, automatically scoring online components, and presenting students with their marks. In addition, teachers can use the information recorded and presented through the system to track their students’ activities and progress on course materials (Welsh et al., 2003). All of this helps streamline the repetitive preparation aspects of teaching so that instructors can focus on providing students with the best learning experience possible.

Students benefit from e-learning systems just as much, if not more, than teachers. Students can download class notes and assignments at anytime and from anywhere (Zhang et al., 2004). E-learning software allows for entire courses to be taught online, removing any time and location constraints and it allows students to earn credits from remote locations (Zhang et al., 2004). In courses without a classroom component, e-learning is centred on the students and they can learn at their own pace (Zhang et al., 2004).

Online forums provide students with access to additional information and discussions about course material. The forums help to give students awareness about what is going on in the course and what fellow students are struggling with (Lambropoulos, Faulkner, & Culwin, 2012). The students’ grades, quiz results, class summary statistics, and any instructor-submitted feedback are available online to keep students aware of their progress in the course. All in all, the use of e-learning provides many conveniences that are not available in traditional classroom learning.

Unfortunately e-learning systems also introduce some drawbacks. Teachers are required to develop or transform their learning material into a digital format (Zhang et al., 2004). This requires both expertise and a significant amount of time from the instructor. Additionally, instructors may be required to tailor
their material to the e-learning software. E-learning systems are fairly limited in the types of exercises that can be created and administered.

In a classroom, teachers can have their students problem solve as a group or role play a scenario, but in e-learning, students are limited to basic exercises such as multiple choice or short answer questions (Coates et al., 2005). Structured assessment is a prominent aspect of E-learning and causes teaching to be shifted towards a more mechanical form, which lacks the type of feedback needed for quality student reflection (Coates et al., 2005). The forms of automatic grading currently available are not a replacement for an instructor looking over a student’s shoulder (Prichard & Vasiga, 2013).

In courses offered solely through the e-learning system instructors are unable to observe their students as they work. This makes it difficult for instructors to be aware of which students are struggling and which course concepts their students do not understand. Instructors need to be aware of what areas students are having difficulty with and why, something that is not apparent from simply looking at the students’ grades or generic test results.

Students experience the limitations of e-learning as well. Because e-learning is self-paced, students need to be self disciplined and motivate themselves (Zhang et al., 2004). Many students thrive with the structure and regularity of a classroom environment, but do not do well when they are fully responsible for their own learning. These requirements for higher maturity levels in students help to explain why drop rates for e-learning are higher than traditional classroom learning (Zhang et al., 2004).

Some students are also uncomfortable in a completely digital environment and can feel isolated from the class community (Zhang et al., 2004). It is more difficult for students to identify their knowledge gaps and misconceptions without the ability to communicate with others (Krause et al., 2009). Because
of this students need to be independent, have good self-assessment skills, and be willing to put effort in to finding answers to their questions.

Another downside to learning in an online environment is that students cannot get immediate feedback from the teacher when they have a question (Zhang et al., 2004). Instead they must go through the class notes or write a message and wait for a response. Weaker or less motivated students may need specific feedback referring to their individual errors or knowledge gaps (Krause et al., 2009). Example solutions to problems are often provided, however students require prior knowledge to be able to problem-solve on their own from the example (Zhang et al., 2004). Similarly, auto-marked exercises often provide pre-defined feedback about each question, however more feedback does not necessarily equal more learning (Price, Handley, Millar, & O’Donovan, 2010).

The lack of hands-on learning and proper assessment and feedback are becoming increasingly noticeable as e-learning becomes increasingly prevalent. One of the most obvious limitations of e-learning is the reliance on forms of assessment which can be automatically marked (Coates et al., 2005). Students are being asked to complete generic exercises such as multiple choice and short response tests rather than hands-on problem solving with real-world scenarios. With the importance of hands-on learning, especially in fields like computer programming, this is something that needs to be remedied.

Students also lack constructive feedback specific to their attempts at solving problems. Price et al. (2010) argues that quality feedback is the most important part of the assessment process; therefore it is important that students are given good feedback even as education turns more automated. If the creation of feedback is going to be automated, then the system needs to be able to identify when and why a student is having trouble, and provide assistance that helps them move forward.
Another major problem encountered by instructors using e-learning systems is the lack of awareness about how well their students understand class topics. Without face-to-face contact, it is difficult for instructors to identify the students who are struggling or may be at-risk of failing. Because the teachers are unable to observe their students, it is necessary for the e-learning system to do that for them. Unfortunately, e-learning systems are not yet capable of reliably identifying states of frustration, although many researchers are working towards resolving this (Kapoor, Burleson, & Picard, 2007; McQuiggan et al., 2007; Picard & Klein, 2002; Rodrigo & Baker, 2009). They can only show the instructor the numerical grade assessments of the students for evaluation.

To improve the learning and teaching experience these three issues need to be addressed. First, there is a need for more mechanisms within e-learning systems to allow for hands-on exercises. Hands-on learning is possibly the best educational approach (Schank, 1995) and cannot be sacrificed in the conversion from classroom learning to e-learning environments. Second, the quality of feedback for students needs to be improved. Students should receive specific and constructive criticism that improves their understanding and promotes active learning. Third, e-learning systems need to be able to automatically identify at-risk students so that instructors can be made aware of the students who are struggling. Depending on the situation, struggling or frustrated students could be identified in real-time, or predictions could be made about the students that are at-risk of failing the course.

2.2 Educational Data Mining

Educational Data Mining (EDM) uses a variety of techniques from statistics, machine learning, and data mining to explore and analyze data collected in an educational context to extract useful information (R. Baker, 2010; Bienkowski,
Data mining (DM) is very similar to EDM in that it follows the same general steps (Romero & Ventura, 2010) and can be defined as the analysis of large data sets to find relationships and summarize the data in a way that is understandable and useable to the data owner (Hand, Mannila, & Smyth, 2001; Leventhal, 2010). Some of the key differences between the two research fields is that EDM uses only data collected in an educational context and some EDM techniques such as regression, correlation and visualization are not considered to be data mining techniques in the broader field (Romero & Ventura, 2010; R. S. Baker & Yacef, 2009).

EDM’s main purpose is to analyze educational data in order to solve educational research problems and to better understand how students learn and the environments in which they learn (R. Baker, 2010; R. S. Baker & Yacef, 2009; Romero & Ventura, 2010). Baker (2009) further defines this objective into four goals that EDM attempts to tackle:

1. The improvement of student models. A student model contains a variety of information about a student such as gender, semester level, current knowledge, and so on. Developing individual models allows the software to act differently depending on the user or situation, which can improve student learning (Corbett, 2001).

2. Discovering or improving domain models. Some researchers have looked at psychometric modeling and space-searching algorithms to automatically create domain structure models from data (R. S. Baker & Yacef, 2009).

3. Research into the pedagogical support that e-learning systems provide. Different types of pedagogy are studied to figure out which strategies are more effective for specific students or overall.
4. Research to better understand learning and learners. Research into well-known educational theories can provide a better understanding of how to target the mindset of students in order to improve e-learning systems.

The use of software in education has introduced the availability of large amounts of data generated by students as they learn. By examining this data using data mining techniques, knowledge can be extracted about the ways in which students learn (Koedinger, Cunningham, Skogsholm, & Leber, 2008). In addition, the use of e-learning software continuously generates new data about how students interact with the system while learning and doing exercises (Castro, Vellido, Nebot, & Mugica, 2007). Together this information has produced an absolute gold mine for educational data (Mostow & Beck, 2006). In fact, the sheer amount of data and how to deal with it is one of the problems institutions are facing (Goyal & Vohra, 2012). EDM takes all of this raw data from various e-learning systems and transforms it into useful information that can be used to improve educational environments (Romero & Ventura, 2010). The use of this data can impact all aspects of education, from teachers and learners all the way to course developers and network administrators (Romero & Ventura, 2010).

There are many methods and algorithms that are used within EDM. Each class of algorithms has scenarios and data types for which they are best suited. The following subsection discusses the data mining techniques and algorithms which are most prevalent in the mining of data collected through e-learning systems.
2.2.1 Methods

The different EDM methods used to solve problems include data mining, statistics, visualization, computational modeling and machine learning and psychometrics (R. S. Baker & Yacef, 2009). Baker (2010; 2009) has suggested the following categories for data mining tasks. This categorization is nearly identical to those forming the chapters of the Handbook of Educational Data Mining (Romero, Ventura, Pechenizkiy, & Baker, 2011):

1. Prediction
2. Clustering
3. Relationship mining
4. Distillation of data for human judgment
5. Discovery with models

Data mining is often done within education to make predictions about students. For instance, prediction mining techniques are often used to identify students that are at-risk for passing courses early enough that they can be helped. One of the most useful tasks in data mining is classification (Romero, Ventura, & García, 2008). The idea behind classification is to place each object (or student) into one class or category based on its characteristics (Romero et al., 2011, ch. 5, p. 57). Classifiers are supervised algorithms, meaning that they are trained on a set of data with known labels, and the predicted variable they produce is binary or categorical (R. S. Baker, 2014). Some examples of classification methods are decision trees, bayesian networks, neural networks, and nearest neighbour classifiers (Romero et al., 2011, ch. 5, p. 57). Regression analysis is similar except that the predicted variable is a continuous variable (R. S. Baker, 2014). Popular regression methods in EDM are linear regression, neural networks, and support vector machines (R. S. Baker, 2014).
There are many options available for prediction in data mining, and choosing the most appropriate method for a given task is a problem that has no general answer (Romero et al., 2011, ch. 5, p. 70). However, some are more prevalent than others in the field of EDM. Decision trees and Naïve Bayes (a Bayesian network classifier) have shown to be effective machine learning techniques for generating preliminary predictive models with educational data (McQuiggan et al., 2007). Linear regression, although not actually a classification method, also works well for prediction when all attributes are numeric ones (Romero et al., 2011, ch. 5, p. 69).

Decision trees are likely the best-known group of classification algorithms (Romero et al., 2011, ch. 5, p. 65). They have the advantage of being simple and easy to understand, are capable of handling mixed variable types, can quickly classify new additions once created, and are flexible (Romero et al., 2011, ch. 5, p. 65). Once a decision tree is created it provides interpretable rules that support runtime decision making for the system (McQuiggan et al., 2007). The most commonly used decision tree algorithms are ID3 (J. R. Quinlan, 1986) and C4.5 (or C5) (J. Quinlan, 1993).

Bayesian Network classifiers are also effective machine learning techniques for predicting students’ affective states (McQuiggan et al., 2007). Bayes classification produces probability tables that can also be implemented in systems to provide probabilities for predicting these states in real-time. The two most commonly seen Bayesian Network algorithms in EDM are Naïve Bayes and Bayes Net (Frank, Trigg, Holmes, & Witten, 1999; McQuiggan et al., 2007). Naïve Bayes relies on the assumption that the attributes used for the prediction are independent from one another (Frank et al., 1999; Kumar & Sahoo, 2012) whereas Bayes Net simply assumes that the attributes are nominal and that there are no missing values (Kumar & Sahoo, 2012). In practical learning
problems the assumption that the attributes are independent from one another is rarely true. However, it has been demonstrated that in classification problems where the classifying labels are categorical this independence assumption is not too limiting (Frank et al., 1999). Because of this, both algorithms have been used in a wide range of data mining tasks with success (Kumar & Sahoo, 2012).

Clustering involves grouping data into categories based on some measure of similarity (Alpaydin, 2010). It is specifically characterized by the fact that labeling information is not considered in the formation of the groups (Romero et al., 2011, ch. 6, p. 76). Unlike classification, clustering is an unsupervised method, since it aims to find meaning in unlabelled data. This makes it good for discovery of information but not for prediction. In EDM clustering is often used to find groups of students who share similar behavioural patterns (Bovo, Sanchez, Héguy, & Duthen, 2013). There are many clustering algorithms available; one of the best-known and most widely used is K-means (Romero et al., 2011, ch. 6, p. 78). Several authors have suggested the use of cluster analysis techniques as a tool for the improvement of e-learning (Chan, 2007; Fu & Foghlu, 2008; Romero et al., 2011, ch. 6).

Relationship mining is a well-studied data mining task. Its goal is to discover relationships between variables within a data set containing many different variables (R. S. Baker, 2014). Baker (2010; 2009) further divides the task into the four subcategories: of association rule mining, correlation mining, sequential pattern mining, and causal data mining.

Association rule mining is used to discover relationships between attributes in a data set and then create if-then statements which concern the values of the attributes (Agrawal, Imielinski, & Swami, 1993). The if-then statements are of the form that if some variable values are known, then another variable
will have a specific value. Association rule mining has been used in web-based e-learning systems in two ways (Romero et al., 2011, ch. 7, p. 94):

1. To help professors obtain detailed feedback of the e-learning process (how students learn, their navigation patterns, and to classify groups of students)
2. To help improve students’ interaction with the e-learning system (e.g. adaptation of the course or recommendations for personalized learning paths)

Correlation mining attempts to find correlations (either positive or negative) between variables (R. S. Baker, 2014). Sequential pattern mining looks for temporal associations between events (such as the path of student actions that lead to a specific learning event). Lastly, Causal data mining has the goal of finding whether a singular event was the cause of another event (R. S. Baker, 2014).

Distillation of data for human judgment allows humans to make inferences about data that may be beyond the capabilities of automated data mining (R. S. Baker, 2014). Visualizations of the data are created so that humans can more easily see the data as a whole. Most commonly, it is done to allow for identification (to distinguish patterns in the ways students learn) or classification (to be used for prediction as discussed above) (R. S. Baker, 2014). In order to do this the data must first be refined to the point where humans can identify well-known patterns. The goal of this method is to summarize and present information in a way that is useful, interactive, and visually appealing so that large amounts of data can be understood and used for decision making (Romero & Ventura, 2013).

The discovery with models approach, as the name implies, requires a model of a phenomenon which is developed through prediction, clustering, or in some
cases, human reasoning. The model is then used as a component in a secondary analysis, usually prediction or relationship mining (R. S. Baker, 2014). When used with prediction methods, the model’s predictions are used to predict a new variable, such as assessing the probability that a student is “gaming the system” (Walonoski & Heffernan, n.d.). When used with relationship mining, the relationships between the created model’s predictions and additional variables are studied.

2.3 Web-based Programming for Hands-on Learning

In recent years several new independent online courses have been created to allow anyone to learn how to code. They range from free tutorials teaching the very basics of programming to membership-based sites offering multiple courses in a range of programming languages and topics. Many of these “learn to code” websites use gamification, themed lectures, interactive coding exercises, and optional hints to provide fun and stress-free learning experiences. While each of the sites vary from one another in some ways, they are all similar in that they provide web-based online coding environments for hands-on programming exercises.

Many people are interested in learning to program, however it is unreasonable to expect novices to download a compiler or to re-load webpages and re-upload files so that they can debug their first program (Prichard & Vasiga, 2013). A web-based programming environment provides learners with a common platform that handles these details (Helminen, Ilvantola, & Karavirta, 2013). This allows learners to code on any device with a modern web browser and from anywhere with an internet connection. They do not need to have a computer, specific operating system, or any additional applications in order to
use the same development environment as everyone else (Barnett, Fähndrich, de Halleux, Logozzo, & Tillmann, 2009).

On the research side, a web platform provides many opportunities for data collection. In a web-browser it is possible for each keystroke to be sent right when the key is pressed, allowing for real-time logging of events (Barnett et al., 2009). The websites developed for teaching the general public to program have realized the usefulness of web-based programming environments and have opened up the world of programming to everyone.

With the available selection of public websites that teach coding, there is an option for almost anyone. The mainstream sites all offer beginner level coding courses, in varying programming languages. The course offerings in more advanced levels differ much more, with end goals for development defining which site is most suitable. Some, such as Codecademy (Codecademy, 2014), are focused on languages and skills used for web-development. Others, like Khan Academy (Khan Academy, 2014), aim to teach only programming topics and theories and do so in only select languages. Code School (Code School LLC, 2014), one of the membership-based sites, offers courses for several different development paths. They cover languages and frameworks used for both web and app development, as well as several “elective” courses that cover tools useful for any type of development.

While the sites are all tailored for learning (Prichard & Vasiga, 2013), the “virtual classroom” environment varies from one site’s courses to the next. Code School (Code School LLC, 2014) aims to imitate the structure of traditional classroom learning by first presenting learners with a video lecture at the start of each lesson. Following the video, the learner is asked to complete a series of hands-on programming exercises along with written guidance reiterating the topics presented in the video. Codecademy (Codecademy, 2014) omits
the video lectures and instead provides written lessons which are frequently interspersed with programming exercises. Both Code School and Codecademy show exercises on one side of the browser and a coding window with console output on the other. The learner’s code is retained in the coding window between steps of a multi-part exercise, which helps to provide continuity while progressing through the lesson (Prichard & Vasiga, 2013). These two sites, and several others, use gamification to motivate their learners. Code School rewards their learners with badges and points indicating their completion status on each development path.

Khan Academy takes a different approach, imitating a more traditional e-learning system, but with added interactive functionality. During lecture videos the students can see the lecturer’s code window which can be interrupted at any time and edited so the students can learn how changes effect the outcome. During programming exercises the students’ code is dynamically re-parsed and re-executed (Prichard & Vasiga, 2013). This allows Khan Academy to provide automatically-generated suggestions on how to fix syntax and run-time errors, thereby teaching the students what the errors mean and reducing their frustration in trying to remedy them (Prichard & Vasiga, 2013).

Other sites omit the traditional teaching approach all together and have instead created games that help to improve learners programming skills. Pex4Fun (Microsoft Research, 2014) teaches programming concepts and skills through a puzzle game where users must create a program that produces output matching that of a “mystery program”. The users are provided with a partially revealed set of outputs from the unknown problem and must figure out the pattern to produce the correct input/output combinations (Tillmann, Halleux, Xie, & Bishop, 2012). Pex4Fun also allows users to create their own “mystery programs” for their friends to solve, adding a competition aspect.
The inclusion of similar web-based Integrated Development Environments (IDEs) within e-learning systems would be very beneficial to students in computer science. Students would be able to experience more hands-on learning and to solve problems that are closer to those in the real world. Numerous international studies have shown that programming students still struggle to be proficient coders even after two years of formal education on the topic (Babb et al., 2014; Rodrigo et al., 2014). A lack of hands-on experience is one of the reasons why programming is difficult for students to learn (Babb et al., 2014).

In many cases, the lack of hands-on experience happens outside of the classroom because students do not have access to programming software (Mow, 2008). Another reason why students do not practice their programming skills is that they lack the auxiliary skills necessary for basic development (Mow, 2008). These skills include proficiency in dealing with the development environment and knowing how to compile, run, and debug their programs. While these skills are absolutely necessary, it is hard for students to master them while also trying to learn the basics of programming. The web-based IDEs discussed in this section solve this problem by taking care of the auxiliary details and letting the user focus their energy on first learning to write basic programs. Adding similar IDEs to e-learning systems would allow more opportunity for hands-on learning and help students to gain a more solid understanding of the skills that are needed.

2.4 Data Collection for Analysis of Coding Actions

The availability of programming environments that log users’ actions has allowed researchers to analyze the ways in which students program. Analysis of the data could be done to predict success on the programming task at hand,
to diagnose frustration in real-time, and to identify areas where students commonly struggle. Rodrigo et al. (2013) claims that a better understanding of student difficulties, misconceptions, behaviours, and affective states can lead to the development of proactive tools for education. To gain more knowledge about how students program and to create tools that can aid students in this area, data on the coding actions of students must first be collected.

To collect data, other studies have developed plug-ins for specific Integrated Development Environments (IDEs) to add the ability to track and log users’ actions. One popular choice is BlueJ (University of Kent, 2014), a free java IDE which is commonly extended for logging of actions (Heinonen et al., 2014). Other examples include CodeBrowser, a browser-side code snapshot analysis tool (Heinonen et al., 2014), Test My Code, an automated assessment system for NetBeans (Vihavainen, Vikberg, Luukkainen, & Pärtel, 2013), and Marmoset, also an assessment system developed for Eclipse (Spacco, Strecker, Hovemeyer, & Pugh, 2005).

Unfortunately, there are several limitations to plug-ins designed for a specific IDE. First of all, the IDE may be dependent on a certain operating system or hardware, so not all users have the option of using it. Secondly, installing, setting up, and learning to use an IDE can be difficult for beginners just starting to learn programming. Users likely have their own preference for which IDE they use, so providing them with a choice would mean creating multiple plug-ins for each IDE. Finally, the data logging plug-ins are made for the IDEs which are popular at the time. However there is no guarantee they will stay the most popular and switching to a new IDE would mean creating an entirely new module.

To solve some of these problems, web-based programming environments (as discussed in Section 2.3) can be used instead. They are available on any
operating system and hardware through a modern web-browser. They do not require any set-up from the user, and they can be extended and improved upon to keep with the times. In addition, the logging capabilities are integrated directly with the programming environment, so there is more flexibility for what can be recorded.

Systems that collect data from their users’ programming activities do so by monitoring actions such as key strokes, mouse movement, and compiler interactions. Most also keep track of changes in the code by recording regular snapshots (records of the contents in each file at particular times). Although the amount and types of information collected vary slightly from system to system, there are many similarities. The variables commonly collected include the following (Blikstein, 2011; Helminen et al., 2013; Watson et al., 2013; Tabanao, Rodrigo, & Jadud, 2008; Worsley & Blikstein, 2013):

- Code edits (can include lines or characters added/removed/modified)
- Run and compile attempts (both successful and unsuccessful)
- Error and warning messages
- Timestamps and line numbers related to the above actions
- Start time and end time of coding sessions
- Copies and pastes

Useful information has been attained from these attributes by applying machine learning methods (discussed in Section 2.2.1. The approaches mentioned in the studies discussed include correlations (Tabanao et al., 2008), linear regression (Watson et al., 2013), clustering (Piech, Sahami, Koller, Cooper, & Blikstein, 2012), and classification (Jadud, 2006). However, neural networks, bayesian networks, and classification methods are also commonly used in educational data mining and could likely be applied here as well.
One of the challenges of collecting data from remote users is managing the storage and transmission of that data. There are two approaches for data storage in this situation. The data can be stored locally, then sent to a central server manually or automatically at a specific time (such as at the end of each day). Alternatively, the data can be sent in real time, meaning that every event is sent to the server as it occurs and no data is stored locally.

Both techniques have advantages and disadvantages. For instance, storing user information locally means that the user can work on coding tasks offline and have their logged data sent to the central server when they reconnect. However, depending on what is being logged and the device that is being used, the amount of data could exceed disk space. The opposite is true for constant communication with the server. The user would have to be online in order for the software to work.

The decision for which storage approach works best depends on the user-end requirements and how the data are being used. For example, if the data were being analyzed to determine when intervention (such as automatic feedback) is required, then the data would need to be made accessible to the server where the analysis occurs in real-time. In this scenario, either continuous communication would need to be established with the remote server where the feedback system is located, or the feedback system would need to be installed locally. In a case where the purpose is to study trends on how students interact with an e-learning system then all of the data recoded would need to be gathered at a central server so that it could be combined for analysis.

Data collected from students as they program can be beneficial to the advancement of e-learning in several ways. Analysis of the data can help educators better understand how students learn programming, and in turn help to improve computer science programs. Examination of this data can be used
to identify at-risk students, to determine when students are struggling and on what aspects of the exercise, and to aid in the development of feedback which is specific to the learning exercise. Therefore, the implementation of e-learning integrated programming environments that log students’ actions is a valuable investment.

2.5 Identifying At-Risk Students

When courses are taught solely through e-learning software, instructors are unable to directly observe their students. A major side-effect is that instructors are unaware of which students are frustrated, struggling, or at risk of failing or dropping the course (Zhang et al., 2004). In traditional classroom instruction with a small class, teachers can tell, at least to some extent, when their students are having difficulty. However within e-learning this is more difficult. Since instructors must rely on the e-learning system for monitoring their students, it would be extremely helpful if the system were able to detect at-risk students automatically.

Past studies have attempted to tackle this problem in two main ways. First, there is research on the automatic identification of human emotions (Dragon et al., 2008; Rodrigo & Baker, 2009; Picard & Klein, 2002). This type of research works towards systems being able to recognize students who are experiencing frustration or anxiety in real time. If e-learning systems could identify these students, then either the instructor or the system itself could offer assistance. Second, there are those who attempt to identify students who are at risk of failing (Heinonen et al., 2014; Romero et al., 2013; Tabanao et al., 2011; Watson et al., 2013). This approach is a predictive one, and aims to recognize these students so that the instructor can step in and try to prevent the undesirable
outcome. The studies focus on students’ actions and results within the e-learning system rather than on their emotions.

2.5.1 Emotion Detection

The problem of automatically identifying emotions is a difficult one. For a human it is usually easy to recognize and respond to one another’s behavioural cues and body language; however computers are not so fortunate. Since the ability to perceive human emotions is natural and instinctive, it is challenging to implement within a machine (Picard & Klein, 2002). Nonetheless, research is being done to facilitate the creation of models that allow a computer to diagnose emotion through data collected in various ways.

There has been some success at measuring physiological signals, such as skin conductance, heart rate, and brain activities in order to detect emotions (McQuiggan et al., 2007; Rodrigo & Baker, 2009; Picard & Klein, 2002). However, measuring these signals requires intrusive physiological sensors which can be limited in their ability to distinguish positive states from negative ones (e.g., excitement versus anger) (Yannakakis, Hallam, & Lund, 2008). It can also be difficult to generalize between individuals due to their vast differences in physiology.

Other studies (Dragon et al., 2008; McQuiggan et al., 2007) have aimed to detect frustration or confusion through body language, behavioural cues, and physiological signals, all of which can be observed without sensors. Most do so by having trained observers code the different behaviours as events. The Facial Action Coding System (FACS), devised by Ekman and Friesen (1978), has been used to help measure facial expressions of participants (Montalvo, Baker, Sao Pedro, Nakama, & Gobert, 2010). Researchers have also used the FACS system along with conversational cues to detect students’ emotional
states (Craig, D’Mello, Witherspoon, & Graesser, 2008). In general, those who have recorded behavioural, emotional, and affective states of students have done so by observing the following (Craig et al., 2008; Dragon et al., 2008; Montalvo et al., 2010; Rodrigo & Baker, 2009):

- Facial expressions
- Body language
- Utterances
- Interactions with fellow students/teachers/observers
- Conversational cues

Dragon et al. (2008) looked specifically at physical behaviours (such as chair and head position, movement, and gestures), facial expressions (including smiles, frowns, and nods), and verbal behaviours (such as loud comments and talking with others). These behaviours were coded by researchers and analyzed to identify how emotional states are linked to student learning. Similarly, DMello et al. (2008) used trained judges to identify frustration and confusion. Frustration was defined as dissatisfaction or annoyance and confusion was defined as a noticeable lack of understanding. The study was able to identify emotions that are shown through animated facial expressions (delight and confusion for instance) but was unable to reliably identify frustration. By collecting the data through an observer, these approaches are not as intrusive as requiring the participant to wear physical sensors. However, getting outside observers to rate feelings and emotions is an enormous task and is not practical as a permanent solution (Kapoor et al., 2007).

In an attempt to identify frustration and confusion while remaining practical and unobtrusive, researchers have been working on linking emotional states to actions that are observable by a computer. This way, models can be created so that the machine can automatically recognize emotions through the
data it collects. In many cases, creating these models initially requires self-reports (Helminen et al., 2013; Kapoor et al., 2007; Rodrigo et al., 2014) from the students in order to accurately identify their emotions, but once a reliable model is made, the process could be fully automated.

Rodrigo and Baker (2009) recorded students’ compilation patterns while they programmed and used the data to create linear regression models of students’ frustration before and after their lab activities. They were able to detect frustration across all five lab activities together, but were unsuccessful at finding frustration on a per-lab basis. Using a self-report approach, Kapoor et al. (2007) had students indicate when they were frustrated or struggling by clicking two different buttons while they worked. This information was used to identify situations which were likely to cause the students’ trouble and therefore help the tutoring system know when to intervene. Likewise, Rodrigo et al (2014) used an “ease of learning” survey to determine how difficult students thought the problems were. They also attempted to identify the differences between low-achieving, average, and high-achieving students.

If e-learning systems had the capability to detect emotion (specifically frustration) then the system could intervene at the right time to provide assistance. Additionally, providing students with encouragement and specific feedback related to both the task and behaviour may help students stay motivated (McQuiggan et al., 2007). It could also help students decrease their anxiety and frustration and increase their ability to cope with the situation through guidance and building on self-assessment ability (McQuiggan et al., 2007). The studies mentioned, and others similar to them, help to create a model of frustration from characteristics that are observable to a computer. Therefore getting one step closer to truly non-intrusive emotion detection, and a huge improvement in automated assistance and tutoring for e-learning.
2.5.2 Success prediction

Rather than trying to detect the emotions of students, other researchers have instead aimed to predict students’ success or failure. This could be their outcome on a single task, or the end result for an entire course. For an instructor, knowing which of their students is at risk of failing is helpful because it allows them to step in and provide guidance.

The prediction of student success is a widely researched area and has been done using many different student characteristics. Some existing studies have focused primarily on past academic success, using the grades from past evaluations to anticipate grades in the future (Baldwin, 2013). Others have looked at in-class response data from clicker usage (Porter, Zingaro, & Lister, 2014) or participation in online forums (Romero et al., 2013) to predict success. Rather than focusing on events in the classroom, studies have also examined the predictive abilities of students’ psychosocial factors (such as stress, time management, and involvement in activities) (Krumrei-Mancuso, Newton, Kim, & Wilcox, 2013) or the role of students’ self-reported confidence levels on success (DeTure, 2004). Many researchers have even attempted to predict the risk of students dropping their programs through demographic enrolment data alone (Kovacic, 2011).

While these approaches have been somewhat successful at predicting outcomes for an entire course, they lack the ability to predict success or failure for individual tasks. Additionally, studies such as these are, for the most part, generalized solutions designed for all students regardless of their specific programs. Just because a student does well in one class does not mean that they will do well in another one. For instance, learning to program has proved particularly difficult for many students, regardless of their success in other disciplines (Mow, 2008).
To target programming students specifically, researchers have also investigated the use of programming errors and actions to identify at-risk students early enough that they can be helped (Heinonen et al., 2014; Tabanao et al., 2011, 2008; Watson et al., 2013). Tabanao et al. (2011, 2008) has attempted to quantify the indicators of novice programmer progress by identifying which frequently encountered errors and compilation behaviours were characteristic of less successful students. Their goal was to determine whether at-risk novice programmers could be accurately identified from compilation events. Although they were not successful in attaining their goal, they were able to identify some compilation behaviours common among at-risk students. Similarly, Rodrigo et al. (2014) looked at the differences in programming behaviours, affect, perceptions, and syntax errors between low, average, and high achieving novice programmers. They also incorporated an “ease of learning” survey to determine how difficult students thought the problems were.

Outside of the identification of at-risk students, researchers have also classified or clustered students based on programming logs in order to learn more about the students coding habits. Jadud (2006) looked at the differences between the students he classified as “stoppers” (those who get stuck on a single error) versus those he called “movers” (students who ignore an error, move on, and later return to fix it). Blikstein (2011) grouped students into three groups “copy and pasters”, “mixed-mode” and “self-sufficients” using students’ programming logs. He used automatically-generated logs of students programming to infer patterns about how they go about the exercises, with a future goal of being able to detect critical points in the programming task where assistance would be needed. Rodrigo et al. (2014) later carried out a very similar study, classifying into an additional category of “extreme movers”. Seo et al (2014) were more curious about which errors most frequently cause failed program
builds. A study by Brown aimed to evaluate the accuracy of instructors’ perceptions of which errors were the most common among novice programmers (Brown & Altadmri, 2014).

One would assume that combining data collected during programming tasks with data used to identify behavioural and affective states of students would improve the quality of predictions made. However, no studies were encountered which attempted to analyze both behavioural attributes as well as those of logged programming actions. It would be interesting to explore both types of data to determine whether that would aid in the identification of at-risk students.
Chapter 3

Methodology and Implementation or Experimental Design

As expressed in Chapter 1 the thesis statement of this research is that *students’ behaviours while coding, can be used to predict their success on programming tasks*. Previous researchers have examined the behaviours exhibited by novice programmers. Most of the research was conducted by collecting and examining computer logged data collected while students programmed (Helminen et al., 2013; Tabanao et al., 2008; Watson et al., 2013). Existing studies have looked at compile errors and warnings, changes in code, user keystrokes, and lengths of coding sessions. Some researchers have used the data to find the behaviours that correlate with success (Blikstein, 2011; Brown & Altadmri, 2014; Heinonen et al., 2014). Others have used it to classify or cluster students into different groups in order to better understand their actions while coding (Piech et al., 2012; Jadud, 2006). However, studies were not found that have compared recorded behavioural attributes to logged data of actions while coding.
This study used logged programming actions, human observations, and self-report data to create a predictive model of students’ success on programming tasks. It was found that success can be predicted with better-than-chance reliability using only actions logged by a computer. The predictions were improved with the additional inclusion of the students’ self-reported confidence and semester level.

These results were found by analyzing data which were collected in several ways. Body language, utterances, and physical actions were recorded through human observation. Measures of participants’ confidence, semester level, and perceived task load were collected through surveys. Participants’ interactions with the computer while programming were logged automatically through a plug-in created for the Moodle (Modular Object-Oriented Dynamic Learning Environment) e-learning system. All of these data were aggregated together for the analysis used to create the model.

This chapter starts with a discussion of the development and implementation of the Moodle plug-in which was used both to log programming actions and to create an environment where students could write, compile, and execute code from within the e-learning system. Section 3.2 describes the data which were collected and the procedure taken to gather that data from the participants. Finally, Section 3.3 describes the materials needed to carry out the experiment. This included the recruitment of participants, the creation of an experimental environment, the writing of programming tasks, and the methods used for data collection and analysis.

3.1 Plug-in Development

Given the popularity of e-learning systems, a programming environment that integrates directly into a learning management system has several advantages.
Students are becoming more accustomed to using e-learning systems alongside traditional classroom learning. They already know where and how to access the e-learning environment when needed, so integrating other learning tools into that same environment helps reduce the frustration of remembering additional URLs, user names and passwords. E-learning systems are web based, and are therefore accessible on any device with a modern web browser. When used to provide a development environment, e-learning systems would also offer a consistent testing platform for the students.

In order to log students’ computer interactions while programming, an environment was needed where students could write, compile, and execute code. Given the benefits of an integrated programming environment, a plug-in was created for Moodle which fulfilled these needs. This plug-in, which will be referred to as the “Moodle IDE” also logged students’ actions as they programmed for use in data analysis. This section describes the requirements of the plug-in and how those requirements were designed and implemented for this research.

3.1.1 Requirements

There were several requirements for the functionality of the Moodle IDE plug-in. The requirements were created from two perspectives: the features needed to carry out this study, and the features needed to create a basic, but functional, coding environment from the student’s standpoint.

The requirements of the plug-in for the study were as follows:

1. The plug-in needed to be able to log students’ actions while they were coding.
2. The plug-in needed to keep students’ information secure.
3. The plug-in needed to protect against malicious code.
The requirements for the students included the following:

1. Students needed to be able to write basic, single-file programs.
2. Students needed to be able to compile and run their code.
3. Students needed to be able to see the results of their compile and run attempts.
4. Students needed to have access to basic code editor features.

Additional functionalities were considered but were not implemented as they were not necessary to complete this study. For example, security was partially implemented by sending the users’ code to a secondary server to be compiled and executed, thereby protecting the web server hosting Moodle from malicious attacks. Security on this secondary server could be improved by also compiling and executing the users’ code in a Virtual Machine or chroot environment. Both Virtual Machines and chroot create isolated environments to protect the host machine from malicious activity, giving the Moodle IDE more security.

In addition, from the student’s perspective, the Moodle IDE could have been improved further as some of the features that one would expect from a modern IDE are missing. In the current state of the plug-in, students do not have a way to save their work. Obviously, this functionality would be needed if students were to use the plug-in for learning in a real course. Another limitation is that the user is restricted to working on a single file and has no access to debugging features. While users could see the output of the terminal, they could not interact with it in any way. This means that they cannot execute commands nor manually enter input for their programs.

Most modern IDEs also provide tools to aid in the productivity when programming in a range of languages. These tools include abilities such as syntax highlighting, auto-complete for variables and function names, and searching
through or inserting available functions. The Moodle IDE as it was created for this study only provides syntax highlighting for the C programming language and does not have any auto-complete functionality. Additional productivity features would be useful, but were omitted at this time because they were not necessary for the completion of this study.

3.1.2 Design

Based on the requirements listed in Section 3.1.1, the Moodle IDE was developed and used for the completion of this study. This subsection discusses the design of the plug-in and addresses how each of the requirements were incorporated.

3.1.2.1 Study Requirements

For the study, the first requirement was the ability for the plug-in to log student’s actions while coding. In order to achieve this, each action that needed to be logged was turned into an event trigger which caused the recording of the students’ action. This resulted in logs of all trigger actions that occurred during each of the student’s programming sessions. The logging of these actions happened behind the scenes, and thus did not cause any sort of disruption to the students as they worked.

The next two requirements for the study were security related. The plug-in needed to keep students’ information secure and protect against malicious code. These requirements were fulfilled through architecture and security considerations.

In the design for the back-end communication of the plug-in, the Moodle e-learning system was kept on a completely separate physical machine from the one where the students’ code was compiled and executed (see Figure 3.3).
Therefore, even if the machine were compromised, the Moodle e-learning environment and the students’ personal information would be kept secure.

Another precaution was put into place to prevent programs from running indefinitely. Both the compile and run actions were allowed 15 seconds to complete before being terminated. This prevented students’ programs from taking up 100% of the server’s resources. It also prevented the Moodle IDE from hanging while waiting for a response from the server, thus stalling the progress of the experiment.

As mentioned before, security was not fully required for the completion of this study. The students were under the direct supervision of the researcher at all times during the experiment. This made it very unlikely that a participant would perform a malicious act that would go un-noticed. Even if the participant were to somehow break the Moodle IDE, the only individual affected would be the current user. In this case the researcher could easily restart the Moodle IDE and continue on with the study.

In addition, the experiment had a Moodle installation solely for the purpose of hosting the plug-in. This installation of Moodle did not have any real student information such as grades, feedback, forum posts, assignment submissions, user names or passwords. Therefore, the security of the Moodle IDE for the study was not as important as it would be when used in a real classroom scenario. Regardless, preventative measures were designed into the plug-in for use in real-world classroom situations or future research.

### 3.1.2.2 Student Requirements

In terms of the needs from a student’s perspective, the first requirement was the ability to write basic, single-file programs. The Moodle IDE provided this functionality through a basic one-window text editor in which students could
write their code (see Figure 3.1). Students could not save their work, nor could they create more than one file at a time. These limitations were not a problem for the purposes of the study, but for use in a classroom these functionalities would need to be added.

In order to make this plug-in an IDE and not just a text editor, students needed the ability to compile and execute the code they had written. Two buttons above the text editor, labeled “compile” and “run”, allowed students to test out their code. By clicking these buttons the students could either compile their program into machine language or compile and then run their program if the compile was successful.

In order for students to be aware of the outcome of their testing, they needed to be able to see the results of their compile and run attempts. To present this information, both the compile and run results were displayed in an expandable text box underneath the text editor (see the black area at the base of Figure 3.2). To be less distracting and obtrusive when not in use, this box
In order to create an editor that students would be comfortable using, a basic set of code editing features were required. These features included keyboard shortcuts, coloured syntax highlighting, bracket pair identification, line numbering, automatic indentation, and line wrapping. Many students use modern IDEs when they program and have become accustomed to such tools to aid in their productivity. To avoid creating any frustration, the plug-in was designed with these basic code editing tools as well.

Figure 3.1 shows a screenshot of the Moodle IDE with a basic “hello world” program written in C. Line numbering can be seen along the left side of the
editor window with the current line highlighted in the document. The plug-in also provides syntax colouring to aid in identifying the various keywords and content classes within the C programming language. The editor also helps identify bracket pairs within the document. If the cursor is placed on one bracket, the program automatically highlights the corresponding opening or closing bracket. Finally, the Moodle IDE was designed so that students could utilize keyboard shortcuts such as cut, copy, and paste within the editor.

### 3.1.3 Implementation

The Moodle IDE was implemented with future expansion and flexibility in mind. The plug-in was created using a Model View Controller (MVC) architecture for the backend which allowed it to be modular and easily expandable. This section describes the technical details about the implementation of the plug-in.

#### 3.1.3.1 Communication and Event Handling

The implementation of the plug-in required three separate resources. First, a web browser was needed to display the Moodle IDE to the student as well as record their actions within the interface. Although the plug-in code is located on the server hosting Moodle, the editor interface runs locally in the web browser on the students’ machines. Second, a server was used to host the Moodle e-learning environment. This server was also responsible for relaying information from the student’s web browser to a second server. Third, this secondary server was used to do the work of compiling and running the code created by the students. This server also contained the database to which all of the logged actions were recorded. Figure 3.3 shows a visual outline of the
communication between these three resources used for the implementation of the plug-in.

In order for the three resources of the plug-in to exchange data between one another, a form of communication was necessary. It was decided that the communication would be over Hypertext Transfer Protocol (HTTP) using an Application Programming Interface (API). An API provides access to external programs so that they can execute tasks and share information among them. This API consisted of three HTTP “post” method calls which were referred to as action, compile and run.

Each time one of the three event methods were called, an entry was added to the log database. The events were sent to the database in real-time, meaning that each time an event happened, the information was sent right away. Nothing was stored locally, so if the student closed the Moodle IDE by mistake, all of their work would disappear and they would have to start from the beginning (there was also no save feature). Luckily, this only happened once during the experiment. It was at the very beginning of a task and therefore

Figure 3.3: Diagram of communication between servers and plug-in.

...
caused little frustration and inconvenience. However, if this kind of event had occurred during the experiment then the data for the task would have to be discarded. Even if the participant were to repeat the task, the data would be biased since they had already spent time solving the problem.

The *action* method was called when the Moodle IDE needed to record a computer interaction but did not require a response (this included copies, cuts or pastes). At the time of the event, method call data was sent to the secondary server via the Moodle server to be stored in the database. The information recorded included experiment ID, question number, a copy of the user’s current code within the editor, and the triggering event action. The actions which triggered the event method did not feedback from the plug-in to the user. Therefore this method only sent information to the database.

The *compile* method was called when the user clicked on the “compile” button located at the top of the Moodle IDE text editor (see Figure 3.1). The student’s code was sent to the secondary server where it was compiled using the GNU C Compiler (GCC). The GCC output was then sent back to the user’s web browser via Moodle and displayed in the black window at the bottom of the text editor. If the compiler did not output any errors or warnings, then a message saying “Compile was successful!” was displayed (see Figure 3.1).

The *run* method was called when the user clicked on the “run” button which was located next to the “compile” button. The student’s code was first compiled as described above. If the compile was successful and an executable file was created, the program was also executed. As with the compile results, the output from running the program was sent back to the user. Run results were displayed to the user in the same window following the compile results (see Figure 3.2).
As with the *action* method, both the *compile* and *run* methods sent data to the database to be logged. They recorded the same information as the *action* method with the addition of any compiler or program output.

### 3.1.3.2 Security Precautions

For the security reasons mentioned in Section 3.1.2, the machine hosting the Moodle environment did not record student information or execute any of the students’ code. Instead, Moodle acted as a relay between the students’ web browsers and the secondary server that performed these actions (See Figure 3.3). For security reasons, the secondary server did not have access to the Internet and could not be accessed from a browser directly. Since the secondary server lacked any sort of authentication to use the API, restricting the communication prevented others from sending in foreign code to be executed. Additional reasons for this implementation approach are mentioned in Section 5.1.

As an additional precaution, the amount of time the compile or execute processes were limited to 10 seconds before being terminated. This prevented processes from running continuously in an infinite loop and hanging the Moodle IDE. However, killing a process before it has finished means that the output that is buffered and waiting to be displayed is lost. To prevent this, the process needed to be told ahead of time that it should not buffer any of the output. As a workaround, students were required to include the line "`setbuf(stdout, NULL);`" in their programs in order to see output from their programs if they ran for longer than the 10 second time limit.
3.1.3.3 Implementation of Editor Features

The final requirement for students was the need for access to basic code editor features (such as syntax highlighting and keyboard shortcuts). In order to implement this requirement, a program called Ace (Ajax.org B.V., 2014) was used. Ace is a high performance text editor that is written in Javascript and runs directly in the user’s web browser.

Ace includes many pre-built features that cover basic code editor capabilities. For this plug-in, Ace was used to provide coloured syntax highlighting, bracket pair identification, line numbering, keyboard shortcuts, automatic indentation, and line wrapping. There are many more features available through Ace that were not implemented in the plug-in at this time. This allows for further expansion of the editor’s capabilities should there be a need. (See Section 5.1 for a discussion on ideas for this future work expansion.)

3.2 Experimental Design

In order to find the relationships between student actions and programming success, a data set was needed for experimentation. To gather this data participants were observed while completing a series of programming exercises. The data set consisted of attributes collected through computer logs, human observation, and survey responses from the participants.

Attributes which were logged by the computer included the following actions or events:

- compile
- run
- copy
- cut
Each action was recorded along with a timestamp of when it occurred and a code snapshot, documenting the state of the participant's code at that time. This was to allow any comparison of changes in the code between events. However, analysis of the code snapshots was not done for this study.

If the participant did not trigger any of the computer interaction events (listed above) during the programming exercise, then the researcher asked the participant to compile at the end of the time limit. The code that the participant produced was recorded along with the compile event. However, as this compilation was requested by the researcher rather than performed naturally by the participant it needed to be discarded from the data set. A note was made in the CSV file when this situation occurred so that it could be removed later.

Human observations were recorded through visual monitoring of the participants as they programmed and included the following:

- comments (negative or positive statements directed at the tasks)
- fidgeting (repetitive movements over a short period of time)
- use of man pages (software documentation accessible via the command prompt)
- noise (any utterances or nonsensical noises)
- staring (looking at the same spot without moving for approximately 5 seconds or more)
- questions (anything asked by the participant directed at the researcher)
- start and end time of each task
- approximate completion percentage of each task

These visual observations were recorded manually by the researcher who sat beside the participant during the study. Each occurrence was recorded along
with the time at which it occurred. Reoccurrences of the same event within five seconds of the one previous were not recorded. The time the researcher used to log these interactions was synced with the server’s time (the machine which logged the computer actions) so that all the times of the events would be synchronized. In addition to these observations, the researcher also recorded the time taken for each task and an estimate at the completion level achieved by the participant.

Visual observations were kept very general so as to be consistent from participant to participant. For example, instead of trying to distinguish negative comments from positive ones, all comments were recorded regardless of their content. Likewise, all nonsensical sounds were recorded as noise. Noises such as groans or sighs were not distinguished.

To avoid influencing the participant, the researcher aimed to maintain neutral body language and only answered direct questions regarding the task. The answers were kept as simple and straightforward as possible and only addressed the question asked.

The NASA Task Load Index (TLX) (Hart & Staveland, 1988) was used to determine the self-perceived difficulty of each programming exercise. An example of the TLX survey administered can be seen in Figure A.4 in AppendixA. The attributes measured using the TLX were:

- mental demand
- physical demand
- temporal demand
- performance
- effort
- frustration
Additional information about each of the attributes collected and how they were measured is available in Table B.1 in Appendix B.

Participants were asked to answer a few questions before and after completing the programming exercises. This was done to collect some basic information on the participant’s programming background (such as semester level and programming confidence level). The questions asked in these surveys can be seen in Figures A.3a and A.3b in Appendix A.

Each participant was asked to attempt four different programming tasks. Each programming task touched on different computer science topics such as string manipulation, reading from disk, working with arrays, and recursion. The task descriptions as provided to the participants can be seen in Sections A.1 to A.4 in Appendix A. The topics in the programming exercises were selected to match those covered in the courses that the participants had taken.

Before starting the tasks, the participants were asked to complete a pre-programming survey (see Figure A.3a). For each task the participants were given a time limit of 15 minutes. This amount of time was approximately double the time needed for an experienced programmer. After each task the participant was asked to complete the TLX survey (see Figure A.4 in Appendix A). Finally, following the completion of all four tasks, each participant was asked to complete a post-programming survey (see Figure A.3b in Appendix A). The maximum time required of any one participant did not exceed an hour and a half (a maximum of one hour for programming and 30 minutes to explain what was required and to complete the surveys).

In order to collect the data through both computer logs and human observation, two types of environments were needed. First, a virtual environment was needed which could log the programming interactions of the participants.
Second, a physical environment was needed where a human observer (the researcher) could record visual observations of the participants’ actions and behaviours.

The Moodle IDE (as described in Section 3.1) was used to fulfill the requirement for a virtual environment that logged programming actions. Within the plug-in separate modules were created for each task so as to keep track of which data was from which task. Each task module was accessible from the main Moodle page for the study under the same title as its corresponding task (as seen in the screen shot in Figure 3.4).

![Figure 3.4: Screen shot of the four different tasks in Moodle.](image)

The Interactive Cognitive Computing (ICC) lab located at the University of Guelph in the Reynolds building was used as the physical environment for the study. The lab was kept quiet with as few distractions as possible while the study was taking place. The ICC lab has eight different work stations, all of which are nearly identical to one another. Each work station has the same
computer hardware, software, chairs, desk, mice, and keyboard. This helps to eliminate any variations in the physical environment that might affect the research data. For even more consistency the same workstation was used for every participant during the study and the researcher always sat on the left side of the participant.

At the workstation used for the experiment, a computer was ready with a web browser opened to the Moodle IDE and an open terminal window for “Man page” access. “Man pages”, short for manual pages, are a set of software documentation accessible via the command prompt which covers computer programs, libraries, and system calls. Participants were given access to the man pages in order to look up information if they wished.

3.3 Materials

3.3.1 Participants

Participants were recruited from a first year intermediate programming course (CIS*2500) and a second year operating systems (CIS*3110) course. Recruitment was done through an announcement made by the researcher requesting the voluntary participation of the students and explaining what would be required of them. The students were informed the study would take no longer than a hour and a half to complete and would be a great opportunity to practice important C programming skills. As a result of the recruitment, 15 participants completed the study.

To protect the privacy of the participants no identifying information was recorded. The steps taken to ensure the participants’ identities were protected can be seen in Figure A.1 in Appendix A. This study was approved by the University of Guelph Research Ethics Board.
3.3.2 Experimental Environment

As discussed in Section 3.2, both a virtual environment for logging programming actions and a physical environment for observing the participants were needed. The Moodle IDE described in Section 3.1 was designed and implemented to fulfill the requirements of the virtual environment. The server used to host Moodle was provided by Sharcnet at the University of Guelph for the duration of this study. The ICC lab (as described in Section 3.2) was used as the experiment’s physical environment. To ensure a quiet setting, the lab was reserved during the times used for the study.

3.3.3 Programming Tasks

During the study participants were required to attempt four different programming tasks. The tasks were designed to cover a range of programming concepts that students are expected to understand by the completion of their first year in the Computer Science program. Each task was short enough that it could be completed in under 15 minutes if the participant knew how to approach it. This was designed so that the total time required from each participant would be under an hour and 15 minutes (to allow for short pre-programming, post-programming, and TLX surveys in between each task). The requirements of the tasks were as follows:

1. The first task involved reading numbers from a file and storing them into a dynamically allocated 2-dimensional array. From this array the participant was expected to print each of the numbers separated by commas to the screen (see full task description in Section A.1).

2. The second task involved string manipulation. Participants were expected to create a string containing a sentence, capitalize the first letter
of each word in the sentence, then print the words on individual lines (see full task description in Section A.2).

3. The third task also involved strings but this time the participant had to read in an unknown string from a file, count how many numerical digits were present in the string, then output their result to the screen (see full task description in Section A.3).

4. The fourth task required the participants to calculate the factorial of a number using recursion (see full task description in Section A.4).

While the participants attempted each programming task the researcher recorded specific visually observed behaviours. During this time the participants’ interactions with the plug-in were also logged automatically. For more information about the attributes that were recorded see Section B.1 in Appendix A.

3.3.4 Data Collection and Analysis

Data for the study was collected through computer logs, human observation, and surveys (as described in Section 3.2). The data from each of these sources was aggregated into two comma-separated value (CSV) files. The first CSV file consisted of one row per participant, resulting in 15 rows containing separate columns for each task’s collection of measured attributes. This file was used to extract data about each individual task and the participants’ survey responses. The second CSV file contained one for per participant per task, resulting in a total of 60 rows (15 participants and 4 tasks each). This file was used for examining the trends across all four tasks together.

Much of the data was entered manually and most of the computer-generated data was already in the proper format, so the data required little cleaning. During the experiment, the researcher had asked some students to compile so that
their data could be logged (Section 3.2 includes a more detailed explanation). These compile events were removed during the data cleaning since they were done by request.

Three calculated variables were added after the collection of the data. The total time taken for each task (in seconds) was calculated by taking the difference between the start and end times of the task. The percentage of successful compile and run attempts were also calculated and added to the data set.

Spearman correlations between each pairwise set of attributes in the data set were calculated using R (R Core Team, 2014). This was done to learn about the relationships between attributes which can be automatically recorded and those which require human observation or self-reports. For instance, frustration measures from the TLX were compared to automatically logged attributes to identify those that may indicate a frustrated state.

Several trials were performed by submitting data to the Naive Bayes and J48 decision tree algorithms to determine if success on the programming tasks could be determined from the collected data. These machine learning methods were applied using Weka (Hall et al., 2009). In each trial, one source of data was omitted to examine the change in accuracy and reliability due to that group of data (for instance, the first trial included all of the data and the second trial eliminated the visually observed behavioural attributes).
Chapter 4

Results, Analysis and Discussion

Data was collected through several methods from the 15 participants who completed this study. This data was combined and analyzed to learn about how the participants’ coding behaviours are connected to their programming success. The data included visual observations from the researcher, TLX measurements, survey responses, participant feedback and actions logged by the Moodle IDE.

The objectives for the analysis of the data included:

- Identifying which coding habit observations (both manual and automatic) can be used to determine success.
- Identifying the relationships between manually observed actions and actions that can be automatically logged by the e-learning system.

This chapter discusses the results found through summary statistics and correlations. It presents the analysis done using Bayesian network and decision tree machine learning techniques and provides discussion on the significance of the results.
4.1 Results

For the initial examination of the data summary, statistics and correlation matrices were created using R (R Core Team, 2014). The summary statistics provides a description of the data collected. Significant relationships between attributes in the data set were identified through the Spearman rank order correlation.

4.1.1 Summary Statistics

The summary statistics (minimum, median, maximum, mean, and standard deviation) for data collected across all programming tasks combined is shown in Table 4.1. These summary statistics were calculated from all of the numerical and non-binary data collected. The summary statistics for each individual task are available in Tables C.1 to C.4 in Appendix C.

The combined summary statistics (in Table 4.1) describes the data collected for all participants on all four tasks. The mean completion time across all the tasks for all participants was 11:51 minutes (691 seconds) with a median of 14:50 minutes (870 seconds). Since the average estimated task completion was only 49.17%, it made sense that participants would use most of the available time in their attempt to complete the programming tasks.

The most frequently observed actions were fidgeting and noise, with averages of 4 and 4.18 occurrences respectively per participant per task. Copies, cuts, and pastes were rarely used, resulting in a median of 0 and mean values between 0.35 and 1.08 across the four tasks as a whole. This indicates that less than half of the participants chose to use the copy, cut, and paste actions.

Statistics for the TLX survey attributes revealed that the highest recorded mental demand measurement was 19. For each of the other five measurements,
a value of 21 was recorded (the maximum score) at some point across the four tasks. Two participants scored their physical demand as high even though the task did not involve any physical demand. This could indicate that these participants misinterpreted which side of the scale was high and which was low.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Min</th>
<th>Median</th>
<th>Max</th>
<th>Mean</th>
<th>Stdev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completion</td>
<td>0%</td>
<td>50%</td>
<td>100%</td>
<td>49.17%</td>
<td>34.7%</td>
</tr>
<tr>
<td>Time</td>
<td>60s</td>
<td>870s</td>
<td>900s</td>
<td>691s</td>
<td>264s</td>
</tr>
<tr>
<td>Semester level</td>
<td>2</td>
<td>4</td>
<td>10</td>
<td>5.33</td>
<td>2.75</td>
</tr>
<tr>
<td>Confidence</td>
<td>1</td>
<td>7</td>
<td>10</td>
<td>7.07</td>
<td>2.16</td>
</tr>
</tbody>
</table>

**Visual Observations**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Min</th>
<th>Median</th>
<th>Max</th>
<th>Mean</th>
<th>Stdev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments (C)</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0.18</td>
<td>0.43</td>
</tr>
<tr>
<td>Fidgets (F)</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>4.00</td>
<td>3.11</td>
</tr>
<tr>
<td>Man pages (M)</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>1.78</td>
<td>1.92</td>
</tr>
<tr>
<td>Noise (N)</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>4.18</td>
<td>2.83</td>
</tr>
<tr>
<td>Question (Q)</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0.97</td>
<td>1.16</td>
</tr>
<tr>
<td>Staring (S)</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>2.05</td>
<td>1.98</td>
</tr>
</tbody>
</table>

**Computer Observations**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Min</th>
<th>Median</th>
<th>Max</th>
<th>Mean</th>
<th>Stdev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiles</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4.53</td>
<td>3.98</td>
</tr>
<tr>
<td>Successful compiles</td>
<td>0%</td>
<td>50%</td>
<td>100%</td>
<td>48.47%</td>
<td>37.27%</td>
</tr>
<tr>
<td>Runs</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4.12</td>
<td>4.35</td>
</tr>
<tr>
<td>Successful runs</td>
<td>0%</td>
<td>100%</td>
<td>100%</td>
<td>73.09%</td>
<td>42.08%</td>
</tr>
<tr>
<td>Cuts</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0.35</td>
<td>0.84</td>
</tr>
<tr>
<td>Copies</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0.55</td>
<td>1.13</td>
</tr>
<tr>
<td>Pastes</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td>1.08</td>
<td>2.20</td>
</tr>
</tbody>
</table>

**TLX**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Min</th>
<th>Median</th>
<th>Max</th>
<th>Mean</th>
<th>Stdev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mental</td>
<td>1</td>
<td>10.5</td>
<td>19</td>
<td>9.88</td>
<td>5.68</td>
</tr>
<tr>
<td>Physical</td>
<td>0</td>
<td>1</td>
<td>21</td>
<td>3.25</td>
<td>4.61</td>
</tr>
<tr>
<td>Temporal</td>
<td>1</td>
<td>8.5</td>
<td>21</td>
<td>9.57</td>
<td>5.97</td>
</tr>
<tr>
<td>Performance</td>
<td>1</td>
<td>11</td>
<td>21</td>
<td>11.10</td>
<td>7.28</td>
</tr>
<tr>
<td>Effort</td>
<td>1</td>
<td>10</td>
<td>21</td>
<td>9.60</td>
<td>5.59</td>
</tr>
<tr>
<td>Frustration</td>
<td>1</td>
<td>8</td>
<td>21</td>
<td>9.05</td>
<td>6.32</td>
</tr>
</tbody>
</table>

Table 4.1: Summary statistics for all tasks combined.

Summary statistics were also calculated for each task individually. These statistics can be seen in Tables C.1 through C.4 in Appendix C. The results from these tables show additional trends which are discussed below.

Participants took the longest time on task 1. The mean time was 13:21 minutes (801 seconds) and the median was 15 minutes (900 seconds), indicating that at least half of the participants required the full amount of time
allowed. All the tasks had at least one participant take the full amount of time. In addition, task 1 was the only task where no participant was able to fully complete the task. However, this could be partially due to the fact that the task order was not randomized. The highest completion level for task 1 was estimated by the observer to be 80%. This also helps to explain why participants spent more time on this task. It is also worth noting that since the time was truncated at 900 seconds, the standard deviation does not accurately represent the dispersion at the higher end of the scale.

Task 2 had the highest mean completion percentage (62% compared to 45% for task 1, 47% for task 3, and 41% for task 4). Task 2 also had the highest median completion rate (80% compared to 50% for tasks 1, 3, and 4). Overall, the participants were the most successful on task 2. Task 4 had the highest standard deviation in completion percentage indicating that it had the highest degree of variation in participant success.

In terms of the visual observations, every participant was caught fidgeting at least once in tasks 1 and 3. Task 4 was the only task where no participants accessed the man pages. Observations of comments made by the participants occurred most frequently during task 1 (mean of 0.33 comments per participant) with a maximum of 2 comments from a single participant. Fidgeting was observed as occurring the most during task 2 (median of 6 and mean of 5.4). The other three tasks had only 3 (for tasks 1 and 3) and 2 (for task 4) for their median number of fidgets.

With regard to computer observations, task 2 had the highest mean and median number of attempted compiles (5.80 and 6 respectively) and the highest percentage of successful compiles (mean of 68% and a median of 67%). Task 2 also had the most attempted runs (mean of 5.07 and median of 4) however the percentage of successful runs was slightly higher in task 1 (mean of 98.75% in
task 1 and 96.37% in task 2). Task 1 also had the most failed compiles (with a mean of 42.75% success and median of 33% success). However, all of the tasks had a fairly high run success rate (lowest mean was 85.42% on task 3).

Regarding the TLX survey responses, the temporal demand measure was the lowest on task 4. Participants’ performance measures were high for task 2, matching the high completion rate for that task. The effort measurement was lowest on task 4 (median of 6, mean of 8.07) and highest on task 3 (median of 13, mean of 10.53). Tasks 1 and 3 had the highest frustration measures (both with means and medians around 10). This high frustration was paired with low completion rates for these tasks (means of 46% for task 1 and 47.33% for task 3).

4.1.2 Correlations

Relationships in the data were examined using Spearman’s rank-order correlation. Much of the data was discrete and therefore not normally distributed (such as counts, semester levels, and rankings on a set scale). Only the two attributes of “time taken” and “completion” were relatively continuous. Density plots of these variables showed non-normal distributions as well (refer to figures B.1). Because of the lack of normality and continuity in the data, the Spearman correlation was chosen for analysis as it does not assume the data are normal.

All of the non-binary attributes in the data were run through a Spearman correlation matrix to identify the relationships present. Figure 4.1 contains the results of this correlation matrix with the data from all four tasks combined. Separate matrices for each individual task were also created and can be seen in Figures C.1 to C.4 in Appendix A.
One of the goals of this thesis was to identify which of the attributes in the data are connected to student success. Success levels were identified from the estimated completion level achieved by the participants on the programming tasks. Several of the attributes displayed significant correlations to the completion levels. Table 4.2 contains the correlation coefficient and p-value for each attribute that showed significant correlations to completion level.

Participants’ self-reported confidence levels in the pre survey (Figure A.3a) were positively correlated with their success on the programming tasks ($r_S(60) =$
The number of man page look-ups while coding showed a positive correlation with completion level \( r_S(60) = 0.26, p = .0457 \). The number of times the participant compiled their programs and how often the compile was successful were both positively correlated with success \( r_S(60) = 0.31, p = .0146 \) and \( r_S(60) = 0.43, p = .0006 \) respectively. The number of times the participants ran their programs and the proportion of those runs which were successful, showed positive correlations with success \( r_S(60) = 0.49, p < .0001 \) and \( r_S(60) = 0.53, p < .0001 \) respectively. From the TLX survey (shown in Figure A.4 in Appendix A) both performance and frustration measures were negatively correlated with success \( r_S(60) = -0.36, p = .0043 \) and \( r_S(60) = -0.45, p = .0003 \) respectively. It is worth noting that a low performance score on the TLX indicated that the participant thought their performance was high. Therefore, the negative correlation between success and performance is, in reality, a positive one.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Completion %</th>
<th>( R ) – value</th>
<th>( P ) – value</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidence</td>
<td>0.42</td>
<td>.0009</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Man pages</td>
<td>0.26</td>
<td>.0457</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Compiles</td>
<td>0.31</td>
<td>.0146</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Compile Success</td>
<td>0.43</td>
<td>.0006</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Runs</td>
<td>0.49</td>
<td>.0000</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Run Success</td>
<td>0.53</td>
<td>.0000</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>-0.36</td>
<td>.0043</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Frustration</td>
<td>-0.45</td>
<td>.0003</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Variables showing significant Spearman correlations to completion level across all four tasks.

All visual observations were positively correlated with time taken \( 0.35 \geq r_S(60) \geq 0.64, p < .05 \) except for the number of comments made by the participant. Time taken was also positively correlated with the number of compiles and runs made by the participant \( r_S(60) = 0.40, p = .0017 \) and \( r_S(60) = 0.28, p = .0326 \) respectively.
From TLX survey, mental demand, temporal demand and effort were positively correlated with time ($r_S(60) = 0.38, p < .0029$, $r_S(60) = 0.27, p < .0387$ and $r_S(60) = 0.29, p < .0226$ respectively). Higher semester levels were correlated with higher frustration ($r_S(60) = 0.38, p = .0027$). Semester level was also negatively correlated with occurrences of noise and TLX scores of physical demand ($r_S(60) = -0.34, p = .0072$ and $r_S(60) = -0.27, p = .0378$ respectively). Higher confidence indicated higher performance ($r_S(60) = -0.38, p = .0025$). Confidence was positively correlated with runs and run success ($r_S(60) = 0.38, p = .0025$, $r_S(60) = 0.32, p = .135$) and was negatively correlated with mental demand, physical demand, effort, and frustration ($r_S(60) = -0.37, p = .0035$, $r_S(60) = -0.28, p = .0304$, $r_S(60) = -0.30, p = .0214$, $r_S(60) = -0.34, p = .0078$ respectively). Higher performance was correlated with higher run success ($r_S(60) = -0.44, p = .0004$) and confidence ($r_S(60) = -0.38, p = .0025$).

Another objective of this analysis was to identify the relationships between the attributes which were observed manually or self-reported by the participants to the attributes which were automatically logged by the e-learning system. From the results of the correlation matrix, only frustration and performance measures showed significant correlations to attributes which were recorded by the computer. Frustration was negatively correlated with run success ($r_S(60) = -0.35, p = .0059$). Performance showed a negative correlation with both run success ($r_S(60) = -0.44, p = .0004$) and runs ($r_S(60) = -0.33, p = .0110$). Although not logged by the computer in this study, confidence showed a negative correlation with both performance and frustration ($r_S(60) = -0.38, p = .0025$ and $r_S(60) = -0.34, p = .0078$ respectively) and semester level was positively correlated to frustration ($r_S(60) = 0.38, p = .0027$).
4.2 Analysis

Analysis was done using both Bayesian networks and decision trees through Weka (Hall et al., 2009). Bayesian networks were used to create models for predicting success based on the attributes recorded during the programming tasks. Several trials were done to examine the level of improvement gained by visual observations, self-reported results and TLX data. A second set of models were created using decision trees. These were used to identify specific variables in the data set which were indicative of success. The following subsections describe the results of these machine learning techniques.

The Naïve Bayes and Bayes Net algorithms were experimented with for the creation of Bayesian networks. The decision trees were created using Weka’s J48 algorithm (a re-implementation of C4.5). All three of these machine learning algorithms require the data set to be categorical. Therefore, the values in the data set were converted into equally spaced categorical bins. The range of values for each attribute’s bins can be seen in Table B.2 in Appendix B.

4.2.1 Bayesian Networks

The data was used to create a Bayesian network to examine how well student success could be predicted. Both the Naïve Bayes and Bayes Net algorithms were experimented with. These algorithms are used frequently in the analysis of educational data and in some cases have been used to predict students’ performance (Baradwaj & Pal, 2011; McQuiggan et al., 2007).

The Bayesian networks were used to predict both success (with two possible classifications, yes and no) and completion level (with three possible classifications, 0-33%, 34-67%, and 68-100%). For each of these two output scenarios four trials of the algorithm were run. Each subsequent trial eliminated one
group of attributes, with trial 1 containing all attributes and trial 4 containing only time and computer observations (The four trails can be seen in Table 4.3). The separate trails were done to examine how much improvement, if any, each group of data added to the accuracy of the prediction.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Trial 1</th>
<th>Trial 2</th>
<th>Trial 3</th>
<th>Trial 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Computer observations</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Confidence &amp; Sem. Lvl.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TLX</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Visual observations</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 4.3: Naïve Bayes Trials.

The Naïve Bayes algorithm was selected for discussion as it provided slightly better results. Figure 4.2 shows the accuracy (as percentage correct) for both algorithms in predicting both completion level and success. As the figure shows, Naïve Bayes was more accurate in all cases except for predicting success with visual observations and TLX measures included.

Figure 4.2: Accuracy (as percent correct) for Naïve Bayes and Bayes Net results.
The root mean square error (RMSE) indicates the differences between the values predicted by the network and those actually observed. Figure 4.3 displays the RMSE for the two algorithms. Here it can be seen that Naïve Bayes had slightly lower error than Bayes Net in all cases.

![Figure 4.3: Root mean square error for Naïve Bayes and Bayes Net results.](image)

The Kappa coefficient of agreement is a measure of reliability (Carletta, 1996). It indicates the proportion of agreement beyond that expected by chance (Sim & Wright, 2005). The range of possible values for kappa is from -1 to 1, where a value of 0 indicates that the reliability of the predictions made is equal to chance (Sim & Wright, 2005). A value of 1 indicates perfect agreement and a negative value indicates agreement worse than that expected by chance. Figure 4.4 displays this reliability measurement for each algorithm. Naïve Bayes had slightly higher reliability for predicting completion levels in all cases and for predicting success with the minimal number of attributes available.
Figure 4.4 shows that the prediction of completion levels was more reliable than the prediction of yes/no success. In several cases the predictions for success were worse than the results expected by chance, whereas predictions for completion levels were better than chance in all cases.

The results of the analysis using the Naïve Bayes algorithm can be seen in Table 4.4. This table shows the percentage and number of correct classifications, the Kappa coefficient, the mean absolute error (MAE), and the root mean square error (RMSE). The same table of results for Bayes Net can be seen in Table C.5 in Appendix C for comparison.

Table 4.4 shows that the percentage of correct classifications was lower when predicting completion level, however one must consider that the probability of classifying correctly simply by chance is higher for two bins than it is for three. For the two bin success approach only 10/60 students were successful. If the algorithm predicted failure for all students its reported accuracy
<table>
<thead>
<tr>
<th>Attributes</th>
<th>Trial 1</th>
<th>Trial 2</th>
<th>Trial 3</th>
<th>Trial 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attributes included</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Computer observations</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Confidence &amp; Sem. Lvl.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TLX</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Visual observations</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

### Results - Completion (3 bins)

<table>
<thead>
<tr>
<th></th>
<th>Percent correct</th>
<th>Number correct</th>
<th>Kappa</th>
<th>MAE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>61.7%</td>
<td>37/60</td>
<td>0.4113</td>
<td>0.2881</td>
<td>0.4565</td>
</tr>
<tr>
<td></td>
<td>60.0%</td>
<td>36/60</td>
<td>0.3854</td>
<td>0.2892</td>
<td>0.4514</td>
</tr>
<tr>
<td></td>
<td>58.3%</td>
<td>35/60</td>
<td>0.3682</td>
<td>0.3305</td>
<td>0.4538</td>
</tr>
<tr>
<td></td>
<td>48.3%</td>
<td>29/60</td>
<td>0.2145</td>
<td>0.354</td>
<td></td>
</tr>
</tbody>
</table>

### Results - Success (yes/no)

<table>
<thead>
<tr>
<th></th>
<th>Percent correct</th>
<th>Number correct</th>
<th>Kappa</th>
<th>MAE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>73.3%</td>
<td>44/60</td>
<td>0.04</td>
<td>0.2497</td>
<td>0.4053</td>
</tr>
<tr>
<td></td>
<td>75.0%</td>
<td>45/60</td>
<td>-0.0227</td>
<td>0.2747</td>
<td>0.4217</td>
</tr>
<tr>
<td></td>
<td>75.0%</td>
<td>45/60</td>
<td>-0.125</td>
<td>0.29</td>
<td>0.4227</td>
</tr>
<tr>
<td></td>
<td>76.7%</td>
<td>46/60</td>
<td>-0.1053</td>
<td>0.2824</td>
<td>0.4131</td>
</tr>
</tbody>
</table>

Table 4.4: Naïve Bayes results.

would be 83.3%. Thus the maximum success rate (76.7%) of the two bin approach is rather misleading and does not accurately show the correctness of the algorithm. For these reasons, the prediction of completion levels was chosen as the better approach.

As one might expect, Naïve Bayes produced the best results when all of the attributes were provided and the worst results when only time and computer observations were provided (see “Percent correct” under the Completion results in Table 4.4). Interestingly enough, the presence of visual observations (trial 1) and TLX data (trial 2) only improved the results by 1.7% each. However, adding only the students’ semester levels and self-reported confidence (trial 3) to the base case (trial 4) caused a 10% improvement in the results.
4.2.2 Decision Trees

A decision tree is a graph resembling a tree-like structure that models decisions and their most likely consequences (see Figures 4.5 to 4.8). Each “node” in the tree (shown in grey) represents the decision, each “branch” represents the outcome of that decision, and each “leaf node” (shown in green, yellow, or orange) represents the outcome. Trees such as those displayed in this thesis are read top-to-bottom, with the path between representing the classification rules that lead to the final classification. The labels in the centre of each branch show the possible choices for the decision above them (or in this case they show the bins available for the given attribute). The leaf nodes show the final completion level classification category. Green leaf nodes indicate completion between 68% and 100%, yellow leaf nodes indicate completion between 34% and 67%, and orange leaf nodes indicate completion levels between 0% and 33%.

Figure 4.5: Trial 1 - Decision tree predicting completion level formed from all collected attributes.

48.3% correct
Kappa = 0.2256
MAE = 0.3464
RMSE = 0.5192
Consider Figure 4.5 as an example. Here the first classifying decision is the percentage of successful runs. If the participant had less than 68% success, the completion level was immediately classified. If their run success was between 68% and 100%, they were next classified based on effort, then subsequently on semester level or frustration depending on their effort level. For instance, according to Figure 4.5 a participant with 75% run success, a moderate effort level of 10, and a low frustration level of 5 would most likely end up with a completion level between 68% and 100%.

The data collected was used to create a J48 decision tree to identify which attributes were most predictive of the participants’ completion levels. As with the Bayesian networks, this was done in four trials using 3 bins, resulting in four trees. Trial 1 used all available attributes (Figure 4.5). Trial 2 eliminated the use of any visual observations (Figure 4.6). Trial 3 eliminated visual observations and TLX data (Figure 4.7). Trial 4 used only computer logged observations and the time taken for task completion (Figure 4.8).

The prediction accuracies of the J48 trees was lower than the accuracies using the Naïve Bayes algorithm. The trees ranged from 46.6% to 53.3% correct classifications. Unlike the Bayesian networks, the best results were produced when using the least amount of data (only computer observations and time).

The trees formed in trials 1 and 2 were identical (Figures 4.5 and 4.6 respectively). Visual observations did not appear at all in the tree when they were available, indicating that they were of little use in making better predictions. However, the trial 1 tree did have slightly better accuracy (higher by 1.6%) and a better Kappa measure of reliability (higher by 0.05) than the trial 2 tree. Both of these trees used run success, effort, semester level, and frustration in their prediction of completion level.
Figure 4.6: Trial 2 - Decision tree predicting completion level formed without visual observation attributes.

Figure 4.7: Trial 3 - Decision tree predicting completion level formed without visual observations or TLX attributes.
In trial 3, with visual observations and TLX data removed, the tree maintained the same prediction correctness, 46.7%, as when effort and frustration were available. The Kappa statistic was lower, but only by 0.0018. This tree used run success and semester level like the first two, but replaced effort and frustration with the students’ self-reported confidence level.

![Decision tree predicting completion level formed with only computer observation attributes and time.](image)

Figure 4.8: Trial 4 - Decision tree predicting completion level formed with only computer observation attributes and time.

The trial 4 tree produced the highest prediction correctness of all (53.5%), the highest Kappa measure (0.2638), and the lowest error (MAE of 0.3667 and RMSE of 0.4572). This tree used only run success and the number of cuts made by the participant in its prediction of completion level.

4.3 Discussion

Correlations of the data revealed several attributes related to the participants’ completion levels on the programming tasks. Confidence, use of man pages, number of compiles, the proportion of successful compiles, number of runs, and the proportion of successful runs were all positively correlated with completion level. Additionally, the participants’ self-reported performance and frustration scores were negatively correlated with completion level. While these results do
not provide a concrete list of attributes for determining success while program-
ning, they do give an idea of some areas that would be useful to observe in
the future.

The use of man pages was the only visual observation that showed a signif-
icant correlation to success, and fortunately it would be the easiest to observe
automatically through a computer. Confidence measures, which also showed
significant correlations to success, were collected via self-reports from the par-
ticipants. All other attributes that were found to be helpful for determining
success were automatically logged by the computer.

Most visual observations, and the number of compiles and runs were posi-
tively correlated with time, which made sense. If the participants took more
time then there was more opportunity for observations, compiles, and run at-
ttempts. Curiously enough, there was no significant correlation between time
and completion level. This suggests that simply spending more time on the
problems does not mean the students will do better. Alternatively it could
have meant that the time limit was too short, causing more participants to use
the full allocation of time without being able to complete the problem.

Results from the Spearman correlation also showed that both the partici-
pants with higher self-reported frustration and performance scores had poorer
run success. (Recall that a low performance score meant the participant
thought their performance was good.) This means that students who had more
failed run attempts reported feeling more frustrated and/or lacking in perfor-
ance. However, it is unclear which direction this relationship goes, whether
failed run attempts cause frustration or if frustration causes more mistakes and
thus more failed run attempts. Maybe participants felt frustrated because their
run attempts had failed so often, or maybe their run attempts failed because
they were frustrated and therefore ran their programs without first fixing their
errors. At this point, more experimentation would need to be done to determine the extent to which run success can be used to determine performance and level of frustration.

It was found that self-reported confidence was related to both frustration and performance as well. Those that felt more confident experienced less frustration and better performance. In addition, participants in higher semester levels of their program tended to experience more frustration. Exactly why semester levels are correlated with frustration in this way is unknown but a few reasons can be speculated. First, students in higher semester levels might put more pressure on themselves and believe that they should be able to easily complete first year tasks. Second, students in higher semesters have had more coding experience and might be used to working within a particular IDE for development. Transitioning from the IDE they are comfortable with to the Moodle IDE might have been difficult since certain features and functions that they are accustomed to were unavailable. Third, students in higher semesters are likely familiar with certain errors, and know that some are more difficult than others to repair. Therefore, if a particularly difficult error appeared during one of the tasks, the participant may have become frustrated more quickly than someone who was less familiar with that error message.

The analysis done using Naïve Bayes showed that the completion levels of the participants could be determined with a 48.3% accuracy using only computer observations and time. By simply adding the confidence and semester levels to the data, this accuracy was improved by 10%. Subsequently adding TLX data and visual observations only improved the accuracy another 1.7% each. From these results, it does not appear that visual observations of the students’ physical actions nor their self-reported feelings about task load, adds much improvement to the prediction of their success. It is useful to know
that it is not necessary to collect much self-report data from the students or
to do obtrusive physical observations to improve the predictions. Due to the
improvement caused by simply adding students’ confidence and semester levels
to the prediction data, these attributes are worth collecting.

The J48 decision tree was able to predict completion levels with an accuracy
of 53.3% using only computer observations and time (5% better than Naïve
Bayes). However, this accuracy was not improved by adding any additional
data. Although the results from the trees were worse overall than those from
the Bayesian networks, the trees showed which attributes were most important
in their prediction.

When all of the data was available the trees utilized run success, then effort,
then semester level and frustration. When the visual observations and TLX
data were removed, they used run success, then semester level, then confidence.
Finally, with only computer observations and time to choose from, the tree used
run success and the number of cuts made in its prediction. Of these attributes,
run success, frustration, and confidence had also shown significant correlations
to completion level.

The thesis statement of this research is that students’ behaviours while cod-
ing can be used to predict their success on programming tasks. The Naïve Bayes
model created in this study was able to predict the success of students on pro-
gramming tasks with an accuracy of 61.7% using all of the data collected. This
model performed better when more data was available, however the J48 de-
cision tree outperformed Naïve Bayes when only computer observations and
time were used (53.5% versus 48.3% respectively).

As both models performed with a better-than-chance reliability, it can be
concluded that students’ actions and behaviours while coding can indeed be
used to predict their success on programming tasks. However, the prediction
capabilities of the models presented in this study could still use improvement. The following chapter includes a discussion of ideas for how these prediction accuracies could be increased.
Chapter 5

Summary and Future Work

This study looked at computer observations, visual observations, and surveys in order to understand students’ behaviours while coding and predict their success on programming tasks. A Moodle IDE was developed to allow students to program while their actions were automatically logged by a computer. Data was collected and analyzed from fifteen participants each of whom completed four programming tasks.

The results were shown using summary statistics and Spearman correlation matrices (see Section 4.1). On average, participants used 11.5 minutes out of the available time limit of 15 minutes. Visual observations of fidgeting and noise were seen the most, with average occurrences of 4 and 4.18 respectively per participant per task. The copy, cut and paste actions were rarely used, with a median of 0 and a mean between 0.35 and 1.08. Also, the attributes of confidence, man pages, compiles, compile success, run, and run success showed positive correlations to success while performance and frustration were negatively correlated to success.
Analysis of the variables was done through the use of Bayesian networks and decision trees to see how well student success could be predicted. The Naïve Bayes algorithm was used to predict completion as one of three classification bins (0-33%, 34-67%, and 68-100%). The best results were seen when all of the data was available to the Bayesian network. Eliminating visually observed data and data collected from the TLX surveys only reduced the prediction accuracy by 3.4%. However, removing the students’ self-reported confidence and semester level worsened the prediction by an additional 10%. From these results it was concluded that asking the students for their confidence levels, and obtaining their semester levels from the e-learning system would be a worthwhile endeavour.

The same data sets were run though decision trees using the J48 (or C4.5) algorithm. The trees showed slightly better results than the Bayesian network with the minimal data set, however their results were worse for the other three trials. Although their prediction accuracies were worse overall, the trees helped to identify which attributes might be more useful in predicting student success. The attributes of run success, semester level, confidence, effort, frustration, and the number of cuts all made appearances in the final structure of the trees. This adds further evidence that these variables, especially the ones that were seen to improve the Bayesian network or were correlated with completion level, should be explored further.

5.1 Future Work

This study has demonstrated that students’ behaviours while coding can be used to create a predictive model of their success on programming tasks. While the capabilities of the Moodle IDE were sufficient for this study, there remains a lot of potential for improvements and expansions to the tool. There are
also several alterations to the study and the experiment itself which would be interesting to explore in future research. This section details ideas for how the research and plug-in could be expanded and improved upon in future research.

The section starts by proposing ideas for how the study itself could be improved. This discussion includes the need for validation in a real-world setting and suggestions for how prediction accuracy of the model might be improved. Next, ideas for improving the experimental procedure used for the study are presented. The remainder of the section focuses on how the Moodle IDE could be improved for future use in a real classroom setting. These ideas are separated into three groups: improvements for the students, improvements for the teachers, and improvements for the general function and security of the plug-in.

5.1.1 Improvements and Validation for the Study

The Naïve Bayes model created in this study was able to predict students’ success with a prediction accuracy of 61.7% when all of the data was used. There is a possibility that this accuracy could be improved by simply collecting and analyzing a larger sample of data. Since there was little incentive to participate in the study, the students who volunteered were likely those who had a higher level of interest in programming. Because the group was not representative of computer science students as a whole, the results could have been biased towards this particular group. By gathering a broader collection of data, this bias could possibly be reduced and the results may be more applicable to all students.

Another way prediction accuracy could potentially be improved is through the inclusion of additional attributes or measurements. One possibility would be to include the analysis of students’ compiler errors. Several other researchers
have had success using error analysis to identify student success (see Section 2.5.2) therefore adding them to the data used in this study could improve the results. If the Moodle IDE were to later include an automated feedback system then the analysis of compiler errors would be a good asset to provide specific and relevant feedback.

As mentioned, this study created models to predict student programming success using the data collected from a sample of 15 students. These models would need to be validated using a secondary data set to determine whether they can reliably be applied to students outside of the sample group. This could be done by deploying the Moodle IDE into a real classroom to be used for solving programming problems in quizzes or lab exercises. During this time, another set of data could be gathered through the IDE. Once this data is collected the models could then be applied and the results could be compared with the actual success of the students to determine the models reliability.

5.1.2 Refinement of the Experimental Procedure

There are several ways in which the experiment itself could be revised if it were to be repeated. It may be worthwhile to change the format of the programming tasks that were completed by the participants. The tasks could either be made shorter so the participant can complete more tasks in the 1.5 hour time-slots, or alternatively, could be expanded into a single large programming task that required several steps and more time to complete. Both of these would yield different data about programming behaviour and might shed more light on the programming trends linked to success.

Another idea for improvement is to have multiple observers recording and labeling the students’ behaviours and actions. This could help to achieve better consistency between the observations. It would also improve the reliability
and lessen any bias of the completion level estimates since they could be averaged across all of the observers. Understandably, participants may not feel comfortable being observed by a group of people while they work. To remedy this, the participant could instead be video recorded during the experiment, allowing the observers to categorize their behaviour and actions after the fact.

After each programming task, participants were asked to complete the TLX survey (Figure A.4 in Appendix A) using pen and paper. This shifted their focus away from the computer and created interruptions in the experimental procedure. The TLX surveys could be digitized and presented from within the Moodle IDE in order to help streamline the experiment sessions and avoid such a drastic shift in focus between tasks. Not only would this be an upside for the participants, it would greatly reduce the amount of manual data entry required on the part of the researcher.

Finally, the Moodle IDE could include more automation for the collection of student information. The results of this study indicated that students’ semester levels play a significant part in the prediction of their success on programming tasks. During this experiment, the semester level was gathered manually from the students in the pre-survey (Figure A.3a in Appendix A). However, the Moodle system itself could store this information in the students’ profiles. Since the Moodle IDE has access to the Moodle databases, the collection of semester levels could be automated, thus reducing inconvenience to the students.

### 5.1.3 Improvement and Expansion of the Moodle IDE

For the purpose of this study the Moodle IDE only required basic features (see Section 3.1.1). The participants simply needed a platform to be able to work on the programming tasks without becoming frustrated due to a lack of editor capabilities. The post-programming survey (see Figure A.3b in Appendix A)
indicated that four out of the fifteen participants reported that their experience with the Moodle IDE was negative. This means that 27% of participants were not satisfied with the editor, so effort should be put into improving the capabilities of the editor itself if it is to be used in future research or applications.

From the students’ point of view, the usability of the Moodle IDE could be improved by offering more of the features offered by other modern IDEs and code editors. These include features such as debugging tools, auto-complete for variable and function names, searching and inserting available functions and variables, displaying syntax errors prior to compiling and tools for performance testing. There are also features that could be added which have more focus on aesthetics rather than programming functionality. Most IDEs offer the ability to change custom preferences for how code is displayed, including choices for syntax colouring schemes, fonts, sizes, and options for the application’s appearance and behaviour (such as auto-converting tabs to spaces). Through adding some of these features, students would be able to create a more personalized programming environment.

The plug-in as it was designed for this study did not allow students to save their work. This functionality would certainly be required for the Moodle IDE to be useful to students. The students would need to be able to save their progress so that they could pick up where they left off each time they open the IDE to continue working. They would also need to be able to work on multiple projects or files and be able to switch between the files while programming.

In addition, the plug-in used for this study contained an expandable black window at the bottom which presented command line output and resembled a terminal in its appearance. At present, the window only displays compile and run results of the students’ programs. However, the functionality could be expanded so that it can actually be used as a real terminal with both input and
output. This could be done by mirroring the terminal of the computer where the students programs are actually being compiled and run. Additionally, this would provide students with access to operating system commands, such as Man pages, right in the e-learning system as well. For the study the participants had to open an actual terminal window on the local computer to access these commands which could have contributed to a negative experience.

In terms of accessibility, the Moodle IDE could be improved through adding responsive design (the user interface would dynamically adapt to various screen sizes). A great benefit of a web-based programming environment is that it can be run on many types of devices. However, the current Moodle IDE was designed with the assumption that users would have a reasonably large monitor (such as the average laptop) and therefore used static text sizes and buttons as well as windows which were only partially scalable. The plug-in’s front-end should be re-created using responsive design, making it suitable for use on devices with any screen size. In addition, the plug-in should be thoroughly tested for usability, both on computers and tablets, before being deployed for use by a classroom of students.

The Moodle IDE could be incorporated into e-learning environments to introduce more hands on learning for programming students. Instructors could use it to incorporate real programming questions into their quizzes or labs, allowing them to ask more complex and thought provoking questions. With the full potential of an integrated IDE, these programming problems could be auto-marked just like the existing multiple choice or short answer questions, therefore supporting use in large classes. It would be even more beneficial if the e-learning system could automatically analyze the logged data while the students attempt to solve the programming problems. Students could then be given automated feedback to help them understand more easily. At the same
time instructors could be informed of the students who are having difficulty or
the topics where their students are struggling with as a whole.

The rough design of the Moodle IDE had other potential EDM research
and applications in mind. One of the ideas was the ability to deploy the
Moodle IDE to an entire class with unlimited access. This means the Moodle
IDE needs to be able to scale well in order to serve an unspecified amount of
students from multiple courses simultaneously. Running and compiling code
is a resource-intensive process which could require multiple servers in order
to provide students with responses in a timely manner. The secondary server
which handles all the compile and run events, could only be upgraded so far
before it would make more sense to divide the load across multiple machines.
In the current design, the information passes through the Moodle server to the
secondary server (see Figure 3.3 in Section 3.1.2). With this structure, the
Moodle server could be converted into a load balancer in order to divide the
work between as many servers as needed. As an added benefit, this increases
the reliability of the Moodle IDE because it could still operate if one of the
multiple secondary servers were to go offline.

During this study security risks were considered, but only partially imple-
mented since the plug-in was only being used by a small group of users under
the researcher’s supervision. However, if the IDE were to be accessible to an
entire classroom of students additional precautions would need to be made.
The environment where the students’ code is compiled and executed should
be isolated from other students as well as the host machine. Either Virtual
Machines or environments within a chroot would need to be created for each
student. This would isolate any malicious code from reaching other students’
environments or the host operating system. Certain system functions might
also need to be removed depending on the reasons why the students will be using the Moodle IDE.

5.2 Conclusion

The results of this study have shown that it is possible to predict student success through Bayesian networks using the automatic logging of actions while programming. The study has also demonstrated that predictions of success can be improved considerably by simply including the students’ semester level and their self-reported confidence on the task at hand. Correlations of the data and the creation of decision trees have further identified the attributes which are most indicative of success. These results will need further validation with a larger data set collected from actual classroom scenarios. However, they provide useful direction for the types of data and machine learning methods that work well for the prediction of programming success.

Further development of the Moodle IDE presented in this thesis could provide students with more learning opportunities in the Computer Science field. With the availability of an integrated coding environment the students would have access to more hands-on experience and real-time feedback. Instructors would be able to create interactive programming problems to challenge their students and would be able to see how they were being approached and where the students struggled. These features could potentially help students overcome their frustration when learning to program, and thus start a new path for e-learning in Computer Science.
Appendix A

Ethical Considerations and Participant Package
Table A.1: Data Protection Plan containing the considerations taken in order to protect the participants’ privacy.

<table>
<thead>
<tr>
<th>Short Names</th>
<th>Description</th>
<th>Identity Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surveys</td>
<td>Pre and post surveys will be completed by the participants at the beginning and end of the experiment (see Figures A.3a and fig:PostSurvey).</td>
<td>Participants’ names will not be put on the survey papers. Only an experiment ID will be printed to match the users’ logged data with the survey.</td>
</tr>
<tr>
<td>NASA TLX</td>
<td>The NASA Task Load Index questionnaire is a subjective workload assessment tool (see Figure A.4).</td>
<td>Participants’ names will not be put on the survey papers. Only an experiment ID will be printed to match the users’ logged data with the survey.</td>
</tr>
<tr>
<td>Participant interactions</td>
<td>During the experiment the Moodle plug-in will log users’ interactions while they develop the given programming task. Things that will be logged include: Source code in between compile, timestamps, any copy, cut, or paste actions along with a source code snapshot, anything that the compiler displays when executed, and what the users’ code outputs.</td>
<td>Data will be stored under the participants’ experimental ID. They will log into Moodle with their experimental ID, so at no time will their own username be connected with the data.</td>
</tr>
<tr>
<td>Researcher observations</td>
<td>While the participant is running through the experiment the researcher will observe and score the participant actions using the observation form.</td>
<td>Participants’ names will not be put on the survey papers. Only an experiment ID will be printed to match the users’ logged data with the survey.</td>
</tr>
</tbody>
</table>
SAMPLE CONSENT FORM

PLEASE USE UNIVERSITY OF GUELPH LETTERHEAD HERE

CONSENT TO PARTICIPATE IN RESEARCH

Automatic identification of students’ behaviours while programming.

You are asked to participate in a research study conducted by Dr. Judi McCuaig and Scott Dougan, from the School of Computer Science at the University of Guelph. The results of this study will be used in Scott Dougan's graduate thesis.

If you have any questions or concerns about the research, please feel free to contact Dr. McCuaig. (extension 55534, judi@uoguelph.ca)

PURPOSE OF THE STUDY

This study will use a web-based tool that logs student’s interactions in real-time while they develop an application through the use of a web-based compiler. Information from this experiment will help us design a tutoring system that can identify which concepts students are having difficulty with as they code. The tutoring system could then alert the student’s instructor or provide the student with immediate help.

PROCEDURES

If you volunteer to participate in this study, we would ask you to do the following things:

1. Complete a short pre-survey that is multiple choice.
2. Write a series of small C programs using an prototype web-based editor as your compiler.
3. Indicate on a set of scale the level of effort each program required.
4. Complete two short surveys about your experiences with using the plugin to write the C programs, how difficult you found the questions and any suggestions you feel appropriate. Both surveys are multiple choice.

POTENTIAL RISKS AND DISCOMFORTS

There are no foreseeable risks associated with participating in this study. You may withdraw from the study at any time if you feel uncomfortable. It should be noted that Dr. McCuaig is the teacher for CIS*2500 and thus will not be participating in the recruitment and will not have any access to names of participants or consent forms until after the grades for courses have been submitted. Therefore there is absolutely no risk (or benefit) to your grades by participating in this study. Scott Dougan is not a TA for either CIS*2500 or CIS*3110 and thus has no conflict of interest.

POTENTIAL BENEFITS TO PARTICIPANTS AND/OR TO SOCIETY

The participant will receive no potential benefits from this study. Participants will not receive feedback after their participation in the study but may request a copy of the study results by contacting the researchers. The research from could help educators better understand novice programmers habits and to develop sophisticated environments to help students learn to program.

PAYMENT FOR PARTICIPATION

There is no payment offered for participation. However, four $25 dollar gift certificates to Future Shop will be drawn from the participant pool at the end of the semester and will be able to be claimed from Dr. Judi McCuaig. Chances of winning a certificate with a maximum number of participants is 1 in 10. Participants will be notified at by e-mail if their name as been drawn.

Figure A.1: Page one of the consent form signed by all participants.
CONFIDENTIALITY

Every effort will be made to ensure confidentiality of any identifying information that is obtained in connection with this study.

Any data about individual participant will be kept confidential at all times. On the same day of the experiment the research will enter in the survey data manually into the database along with any logged results during the experiment and replace your username with an experimental ID number.

PARTICIPATION AND WITHDRAWAL

You can choose whether to be in this study or not. If you volunteer to be in this study, you may withdraw at any time without consequences of any kind. You may exercise the option of removing your data from the study at any point before you leave the study itself. Removal of your data after this point is impossible since there is no links between the data and personal identifying information. You may exercise the option of removing your data from the study. You may also refuse to answer any questions you don’t want to answer and still remain in the study. The investigator may withdraw you from this research if circumstances arise that warrant doing so.

RIGHTS OF RESEARCH PARTICIPANTS

You may withdraw your consent at any time and discontinue participation without penalty. You are not waiving any legal claims, rights or remedies because of your participation in this research study. This study has been reviewed and received ethics clearance through the University of Guelph Research Ethics Board. If you have questions regarding your rights as a research participant, contact:

Director, Research Ethics
University of Guelph
437 University Centre
Guelph, ON N1G 2W1

Telephone: (519) 824-4120, ext. 56006
E-mail: sauld@uoguelph.ca
Fax: (519) 821-5236

SIGNATURE OF RESEARCH PARTICIPANT/LEGAL REPRESENTATIVE

I have read the information provided for the study "[Automatic identification of students’ behaviours while programming]" as described herein. My questions have been answered to my satisfaction, and I agree to participate in this study. I have been given a copy of this form.

______________________________________
Name of Participant (please print)

______________________________________
Signature of Participant

SIGNATURE OF WITNESS

______________________________________
Name of Witness (please print)

______________________________________   _______________
Signature of Witness      Date

Figure A.2: Page two of the consent form signed by all participants.
Pre-Programming Survey

1) What semester level are you currently in?
____________________________________________________________________

2) How would you rate your coding confidence level? (1 being not very confident, 10 being very confident)
____________________________________________________________________

3) Have you used man pages before?
   Yes [ ] No [ ]

4) When making an application do you mainly use the internet for information on C library functions?
   Yes [ ] No [ ]

5) Do you mainly use an Integrated Development Environment (IDE) when developing an application (Eclipse, Visual Studio, Xcode etc.)
   Yes [ ] No [ ]
   If yes which IDE do you use?
____________________________________________________________________

Post-Programming Survey

1) Did you find the programming questions difficult?
   Yes [ ] No [ ]

2) Did you find developing using the online Moodle editor different from your regular programming editor?
   Yes [ ] No [ ]
   If so, different in what way?
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________

3) Did you have a positive experience using the online Moodle editor?
   Yes [ ] No [ ]

4) What did you like about the editor?
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________

5) What did you not like about the editor?
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________

Participant #______

Figure A.3: Pre and post surveys
Participant #______

**NASA Task Load Index - Programming Task #1**

Hart and Staveland’s NASA Task Load Index (TLX) method assesses work load on five 7-point scales. Increments of high, medium and low estimates for each point result in 21 gradations on the scales.

**Mark each scale by circling the tick at the point you feel describes yourself during the completion of the task.**

![Scale Diagram](image.png)

Figure A.4: TLX survey completed after each of the task attempts. This survey was identical for each of the 4 tasks.
A.1 Programming Task #1

Include the following line as the first line in your main:

```c
setbuf(stdout, NULL);
```

Read in the file called "/tmp/task1.txt" and store the contents in a dynamically allocated 2D integer array.

One example of a possible file is as follows:

```
2 4
10 13 32 39
43 65 84 97
```

The first line of the file contains two numbers separated by a space. These two numbers are the rows and columns (respectively) of the array. Malloc a 2D integer array of this size, then read in the rest of the file, storing each number in its respective position in your array.

Print the contents of your array as a comma-separated list of numbers on a single line. Then print the numbers in backwards order on the next line.

The numbers from the example file printed forwards should look like:

```
10, 13, 32, 39, 43, 65, 84, 97
```

A.2 Programming Task #2

Include the following line as the first line in your main:

```c
setbuf(stdout, NULL);
```

Write a program that reads in a sentence, capitalizes each word, and prints each word on its own line.
Create a string containing the sentence:

"Zebra’s are awesome animals with stripes!"

Use `strtok` to separate each word at the spaces. Capitalize the first letter or each word using the appropriate function from the `ctype` library, then print the word on its own line.

The output should look like this:

Zebra’s
Are
Awesome
Animals
With
Stripes!

A.3 Programming Task #3

Include the following line as the first line in your main:

```c
setbuf(stdout, NULL);
```

Read in the file called "/tmp/task3.txt". Using only `fscanf` count how many numbers are in the file.

For example, if the file contained:

"The war of 1813 lasted for 32 months."

Your program should output "2".

A.4 Programming Task #4

Include the following line as the first line in your main:
setbuf(stdout, NULL);

Calculate the factorial of a number using recursion (For example, 3! would be 3\times2\times1). Do not ask for user input. Simply define and set a variable for the number.
Appendix B

Data tables

Table B.1: Description of attributes used for analysis

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completion</td>
<td>Approximation of how much of the task the participant completed</td>
<td>Percentage of completion</td>
</tr>
<tr>
<td>Time</td>
<td>Time taken to complete task</td>
<td>Seconds elapsed</td>
</tr>
</tbody>
</table>

Pre-Programming Survey Responses

<table>
<thead>
<tr>
<th>Semester level</th>
<th>What semester level are you in?</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidence</td>
<td>How would you rate your coding confidence level?</td>
<td>Not confident (1) to very confident (10)</td>
</tr>
<tr>
<td>Man page use</td>
<td>Have you used man pages before?</td>
<td>Yes or no</td>
</tr>
<tr>
<td>Internet search</td>
<td>Do you usually use the internet to find information on C library functions?</td>
<td>Yes or no</td>
</tr>
<tr>
<td>IDE use</td>
<td>Do you usually use an Integrated Development Environment (IDE) to code? Which one?</td>
<td>Yes or no + explanation</td>
</tr>
</tbody>
</table>

Post-Programming Survey Responses

<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Were the programming questions difficult?</th>
<th>Yes or no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editor difference</td>
<td>Did you find this editor different from your usual editor?</td>
<td>Yes or no + explanation</td>
</tr>
<tr>
<td>Satisfied</td>
<td>Did you have a positive experience with this editor?</td>
<td>Yes or no</td>
</tr>
<tr>
<td>Pros</td>
<td>What did you like about this editor?</td>
<td>Explanation</td>
</tr>
</tbody>
</table>

Continued on next page
Table B.1 – Continued from previous page

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cons</td>
<td>What did you dislike about this editor?</td>
<td>Explanation</td>
</tr>
</tbody>
</table>

**Visual Observations**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments (C)</td>
<td>Making general comments about the task</td>
<td>Count of occurrences</td>
</tr>
<tr>
<td>Fidgets (F)</td>
<td>Appears to be Fidgeting</td>
<td>Count of occurrences</td>
</tr>
<tr>
<td>Man Pages (M)</td>
<td>Accessed the man pages</td>
<td>Count of occurrences</td>
</tr>
<tr>
<td>Noise (N)</td>
<td>Made a noise, sound or incomplete sentence</td>
<td>Count of occurrences</td>
</tr>
<tr>
<td>Staring (S)</td>
<td>Staring into space or is blankly Staring at the screen</td>
<td>Count of occurrences</td>
</tr>
<tr>
<td>Question (Q)</td>
<td>Asked a question</td>
<td>Count of occurrences</td>
</tr>
</tbody>
</table>

**Computer Observations**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiles</td>
<td>Number of attempted compiles</td>
<td>Count of occurrences</td>
</tr>
<tr>
<td>Successful compiles</td>
<td>Percentage of successful compiles</td>
<td>Percentage</td>
</tr>
<tr>
<td>Runs</td>
<td>Number of attempted runs</td>
<td>Count of occurrences</td>
</tr>
<tr>
<td>Successful runs</td>
<td>Percentage of successful runs</td>
<td>Percentage</td>
</tr>
<tr>
<td>Cuts</td>
<td>Number of “cut” actions</td>
<td>Count of occurrences</td>
</tr>
<tr>
<td>Copies</td>
<td>Number of “copy” actions</td>
<td>Count of occurrences</td>
</tr>
<tr>
<td>Pastes</td>
<td>Number of “paste” actions</td>
<td>Count of occurrences</td>
</tr>
</tbody>
</table>

**Self-Reported Measures**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mental demand</td>
<td>How mentally demanding was the task?</td>
<td>Very low (0) to very high (21)</td>
</tr>
<tr>
<td>Physical demand</td>
<td>How physically demanding was the task?</td>
<td>Very low (0) to very high (21)</td>
</tr>
<tr>
<td>Temporal demand</td>
<td>How hurried or rushed was the pace of the task?</td>
<td>Very low (0) to very high (21)</td>
</tr>
<tr>
<td>Performance</td>
<td>How successful were you in accomplishing what you were asked to do?</td>
<td>Perfect (0) to failure (21)</td>
</tr>
<tr>
<td>Effort</td>
<td>How hard did you have to work to accomplish your level of performance?</td>
<td>Very low (0) to very high (21)</td>
</tr>
<tr>
<td>Frustration</td>
<td>How insecure, discouraged, irritated, stressed, and annoyed were you?</td>
<td>Very low (0) to very high (21)</td>
</tr>
</tbody>
</table>
Table B.2: Bins used for Bayes net analysis

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completion</td>
<td>0% - 33%</td>
<td>34% - 67%</td>
<td>68% - 100%</td>
</tr>
<tr>
<td>Time</td>
<td>60s - 340s</td>
<td>341s - 620s</td>
<td>621s - 900s</td>
</tr>
<tr>
<td>Semester level</td>
<td>2 - 5</td>
<td>6 - 7</td>
<td>8 - 10</td>
</tr>
<tr>
<td>Confidence</td>
<td>1 - 4</td>
<td>5 - 7</td>
<td>8 - 10</td>
</tr>
<tr>
<td><strong>Visual observations</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comments (C)</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Fidgets (F)</td>
<td>0 - 4</td>
<td>5 - 8</td>
<td>9 - 12</td>
</tr>
<tr>
<td>Man pages (M)</td>
<td>0 - 2</td>
<td>3 - 5</td>
<td>6 - 7</td>
</tr>
<tr>
<td>Noise (N)</td>
<td>0 - 3</td>
<td>4 - 7</td>
<td>8 - 10</td>
</tr>
<tr>
<td>Question (Q)</td>
<td>0 - 1</td>
<td>2 - 3</td>
<td>4</td>
</tr>
<tr>
<td>Staring (S)</td>
<td>0 - 2</td>
<td>3 - 5</td>
<td>6 - 7</td>
</tr>
<tr>
<td><strong>Computer observations</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiles</td>
<td>0 - 6</td>
<td>7 - 12</td>
<td>8 - 18</td>
</tr>
<tr>
<td>Successful compiles</td>
<td>0% - 33%</td>
<td>34% - 67%</td>
<td>68% - 100%</td>
</tr>
<tr>
<td>Runs</td>
<td>0 - 6</td>
<td>7 - 11</td>
<td>12 - 17</td>
</tr>
<tr>
<td>Successful runs</td>
<td>0% - 33%</td>
<td>34% - 67%</td>
<td>68% - 100%</td>
</tr>
<tr>
<td>Cuts</td>
<td>0 - 1</td>
<td>2 - 3</td>
<td>4</td>
</tr>
<tr>
<td>Copies</td>
<td>0 - 2</td>
<td>3 - 4</td>
<td>5 - 6</td>
</tr>
<tr>
<td>Pastes</td>
<td>0 - 5</td>
<td>6 - 9</td>
<td>10 - 14</td>
</tr>
<tr>
<td><strong>TLX</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mental</td>
<td>1 - 7</td>
<td>8 - 13</td>
<td>14 - 19</td>
</tr>
<tr>
<td>Physical</td>
<td>1 - 7</td>
<td>8 - 14</td>
<td>15 - 21</td>
</tr>
<tr>
<td>Temporal</td>
<td>1 - 7</td>
<td>8 - 14</td>
<td>15 - 21</td>
</tr>
<tr>
<td>Performance</td>
<td>1 - 7</td>
<td>8 - 14</td>
<td>15 - 21</td>
</tr>
<tr>
<td>Effort</td>
<td>1 - 7</td>
<td>8 - 14</td>
<td>15 - 21</td>
</tr>
<tr>
<td>Frustration</td>
<td>1 - 7</td>
<td>8 - 14</td>
<td>15 - 21</td>
</tr>
</tbody>
</table>
Figure B.1: Density graphs of completion percentage and time to determine normality
Appendix C

Additional Results

Table C.1: Task 1 Summary statistics

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Min</th>
<th>Median</th>
<th>Max</th>
<th>Mean</th>
<th>Stdev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>0s</td>
<td>900s</td>
<td>900s</td>
<td>801s</td>
<td>214s</td>
</tr>
<tr>
<td>Completion</td>
<td>0%</td>
<td>50%</td>
<td>80%</td>
<td>46%</td>
<td>27.46%</td>
</tr>
</tbody>
</table>

Task 1 Visual Observations

| Comments (C)          | 0   | 0      | 2   | 0.33 | 0.62   |
| Fidgets (F)           | 1   | 3      | 9   | 3.60 | 2.03   |
| Man pages (M)         | 0   | 2      | 7   | 2.53 | 2.10   |
| Noise (N)             | 0   | 5      | 10  | 4.47 | 3.11   |
| Question (Q)          | 0   | 1      | 4   | 1.47 | 1.30   |
| Staring (S)           | 0   | 1      | 7   | 1.73 | 1.94   |

Task 1 Computer Observations

| Compiles              | 1   | 4      | 11  | 4.80 | 3.51   |
| Successful compiles   | 0%  | 33%    | 100%| 42.75%| 36.08% |
| Runs                  | 0   | 1      | 17  | 2.73 | 4.56   |
| Successful runs       | 88% | 100%   | 100%| 98.75%| 3.95%  |
| Cuts                  | 0   | 0      | 2   | 0.27 | 0.59   |
| Copies                | 0   | 0      | 6   | 0.80 | 1.70   |
| Pastes                | 0   | 0      | 6   | 1.40 | 1.99   |

Task 1 TLX

| Mental                | 1   | 8      | 17  | 8    | 5.66   |
| Physical              | 1   | 1      | 14  | 1    | 3.61   |
| Temporal              | 1   | 9      | 21  | 9    | 5.74   |
| Performance           | 1   | 8      | 21  | 8    | 7.40   |
| Effort                | 1   | 10     | 21  | 10   | 6.04   |
| Frustration           | 1   | 10     | 21  | 10   | 7.04   |
Table C.2: Task 2 Summary statistics

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Min</th>
<th>Median</th>
<th>Max</th>
<th>Mean</th>
<th>Stdev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>270s</td>
<td>815s</td>
<td>900s</td>
<td>676s</td>
<td>245s</td>
</tr>
<tr>
<td>Completion</td>
<td>0%</td>
<td>80%</td>
<td>100%</td>
<td>62%</td>
<td>37.83%</td>
</tr>
</tbody>
</table>

**Task 2 Visual Observations**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0.07</th>
<th>0.26</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments (C)</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2.00</td>
<td>1.36</td>
</tr>
<tr>
<td>Fidgets (F)</td>
<td>0</td>
<td>6</td>
<td>12</td>
<td>5.40</td>
<td>3.78</td>
</tr>
<tr>
<td>Man pages (M)</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>2.00</td>
<td>1.36</td>
</tr>
<tr>
<td>Noise (N)</td>
<td>0</td>
<td>4</td>
<td>10</td>
<td>4.80</td>
<td>2.73</td>
</tr>
<tr>
<td>Question (Q)</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0.80</td>
<td>1.15</td>
</tr>
<tr>
<td>Staring (S)</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>1.80</td>
<td>1.90</td>
</tr>
</tbody>
</table>

**Task 2 Computer Observations**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>0</th>
<th>6</th>
<th>18</th>
<th>5.80</th>
<th>5.06</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiles</td>
<td>0</td>
<td>4</td>
<td>14</td>
<td>5.07</td>
<td>0.96</td>
</tr>
<tr>
<td>Successful compiles</td>
<td>0%</td>
<td>67%</td>
<td>100%</td>
<td>68.13%</td>
<td>31.07%</td>
</tr>
<tr>
<td>Runs</td>
<td>0</td>
<td>4</td>
<td>14</td>
<td>5.07</td>
<td>0.96</td>
</tr>
<tr>
<td>Successful runs</td>
<td>60%</td>
<td>100%</td>
<td>100%</td>
<td>96.37%</td>
<td>11.11%</td>
</tr>
<tr>
<td>Cuts</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0.40</td>
<td>1.06</td>
</tr>
<tr>
<td>Copies</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0.53</td>
<td>0.74</td>
</tr>
<tr>
<td>Pastes</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0.93</td>
<td>1.44</td>
</tr>
</tbody>
</table>

**Task 2 TLX**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>3</th>
<th>12</th>
<th>19</th>
<th>11.47</th>
<th>5.08</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mental</td>
<td>1</td>
<td>1</td>
<td>21</td>
<td>3.67</td>
<td>5.50</td>
</tr>
<tr>
<td>Physical</td>
<td>7</td>
<td>15</td>
<td>21</td>
<td>13.47</td>
<td>5.21</td>
</tr>
<tr>
<td>Temporal</td>
<td>4</td>
<td>17</td>
<td>21</td>
<td>15.67</td>
<td>5.69</td>
</tr>
<tr>
<td>Performance</td>
<td>4</td>
<td>11</td>
<td>17</td>
<td>9.87</td>
<td>5.17</td>
</tr>
<tr>
<td>Effort</td>
<td>1</td>
<td>6</td>
<td>21</td>
<td>7.67</td>
<td>6.33</td>
</tr>
<tr>
<td>Frustration</td>
<td>1</td>
<td>6</td>
<td>21</td>
<td>7.67</td>
<td>6.33</td>
</tr>
</tbody>
</table>
Table C.3: Task 3 Summary statistics

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Min</th>
<th>Median</th>
<th>Max</th>
<th>Mean</th>
<th>Stdev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>480s</td>
<td>840s</td>
<td>900s</td>
<td>771s</td>
<td>156s</td>
</tr>
<tr>
<td>Completion</td>
<td>0%</td>
<td>50%</td>
<td>100%</td>
<td>47.33%</td>
<td>29.63%</td>
</tr>
</tbody>
</table>

**Task 3 Visual Observations**

<table>
<thead>
<tr>
<th>Comments (C)</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0.13</th>
<th>0.35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fidgets (F)</td>
<td>1</td>
<td>3</td>
<td>11</td>
<td>4.27</td>
<td>3.31</td>
</tr>
<tr>
<td>Man pages (M)</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>2.60</td>
<td>2.13</td>
</tr>
<tr>
<td>Noise (N)</td>
<td>0</td>
<td>4</td>
<td>9</td>
<td>4.40</td>
<td>2.29</td>
</tr>
<tr>
<td>Question (Q)</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0.93</td>
<td>1.10</td>
</tr>
<tr>
<td>Staring (S)</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>2.73</td>
<td>1.58</td>
</tr>
</tbody>
</table>

**Task 3 Computer Observations**

<table>
<thead>
<tr>
<th>Compiles</th>
<th>0</th>
<th>4</th>
<th>13</th>
<th>5.00</th>
<th>3.89</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful compiles</td>
<td>0%</td>
<td>50%</td>
<td>100%</td>
<td>57.14%</td>
<td>37.10%</td>
</tr>
<tr>
<td>Runs</td>
<td>0</td>
<td>3</td>
<td>17</td>
<td>4.20</td>
<td>4.55</td>
</tr>
<tr>
<td>Successful runs</td>
<td>0%</td>
<td>100%</td>
<td>100%</td>
<td>85.42%</td>
<td>31.00%</td>
</tr>
<tr>
<td>Cuts</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0.40</td>
<td>1.12</td>
</tr>
<tr>
<td>Copies</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0.47</td>
<td>1.13</td>
</tr>
<tr>
<td>Pastes</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td>1.27</td>
<td>3.59</td>
</tr>
</tbody>
</table>

**Task 3 TLX**

<table>
<thead>
<tr>
<th>Mental</th>
<th>1</th>
<th>13</th>
<th>17</th>
<th>10.60</th>
<th>5.30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical</td>
<td>1</td>
<td>1</td>
<td>17</td>
<td>2.93</td>
<td>4.38</td>
</tr>
<tr>
<td>Temporal</td>
<td>2</td>
<td>9</td>
<td>21</td>
<td>10.27</td>
<td>5.38</td>
</tr>
<tr>
<td>Performance</td>
<td>1</td>
<td>14</td>
<td>21</td>
<td>12.33</td>
<td>6.43</td>
</tr>
<tr>
<td>Effort</td>
<td>1</td>
<td>13</td>
<td>16</td>
<td>10.53</td>
<td>10.07</td>
</tr>
<tr>
<td>Frustration</td>
<td>1</td>
<td>10</td>
<td>21</td>
<td>10.07</td>
<td>5.89</td>
</tr>
</tbody>
</table>
Table C.4: Task 4 Summary statistics

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Min</th>
<th>Median</th>
<th>Max</th>
<th>Mean</th>
<th>Stdev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>60s</td>
<td>450s</td>
<td>900s</td>
<td>515s</td>
<td>331s</td>
</tr>
<tr>
<td>Completion</td>
<td>0%</td>
<td>50%</td>
<td>100%</td>
<td>41.33%</td>
<td>41.90%</td>
</tr>
</tbody>
</table>

**Task 4 Visual Observations**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Min</th>
<th>Median</th>
<th>Max</th>
<th>Mean</th>
<th>Stdev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments (C)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.20</td>
<td>0.41</td>
</tr>
<tr>
<td>Fidgets (F)</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>2.73</td>
<td>2.74</td>
</tr>
<tr>
<td>Man pages (M)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Noise (N)</td>
<td>0</td>
<td>2</td>
<td>10</td>
<td>3.07</td>
<td>3.08</td>
</tr>
<tr>
<td>Question (Q)</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0.67</td>
<td>1.05</td>
</tr>
<tr>
<td>Staring (S)</td>
<td>0</td>
<td>1</td>
<td>7</td>
<td>1.93</td>
<td>2.43</td>
</tr>
</tbody>
</table>

**Task 4 Computer Observations**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Min</th>
<th>Median</th>
<th>Max</th>
<th>Mean</th>
<th>Stdev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiles</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>2.53</td>
<td>2.72</td>
</tr>
<tr>
<td>Successful compiles</td>
<td>0%</td>
<td>50%</td>
<td>100%</td>
<td>52.86%</td>
<td>33.20%</td>
</tr>
<tr>
<td>Runs</td>
<td>0</td>
<td>5</td>
<td>10</td>
<td>4.47</td>
<td>3.80</td>
</tr>
<tr>
<td>Successful runs</td>
<td>50%</td>
<td>100%</td>
<td>100%</td>
<td>93.33%</td>
<td>14.97%</td>
</tr>
<tr>
<td>Cuts</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.33</td>
<td>0.49</td>
</tr>
<tr>
<td>Copies</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0.40</td>
<td>0.74</td>
</tr>
<tr>
<td>Pastes</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0.73</td>
<td>1.10</td>
</tr>
</tbody>
</table>

**Task 4 TLX**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Min</th>
<th>Median</th>
<th>Max</th>
<th>Mean</th>
<th>Stdev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mental</td>
<td>1</td>
<td>7</td>
<td>18</td>
<td>8.53</td>
<td>6.61</td>
</tr>
<tr>
<td>Physical</td>
<td>0</td>
<td>1</td>
<td>17</td>
<td>3.60</td>
<td>5.14</td>
</tr>
<tr>
<td>Temporal</td>
<td>1</td>
<td>4</td>
<td>16</td>
<td>5.27</td>
<td>4.93</td>
</tr>
<tr>
<td>Performance</td>
<td>1</td>
<td>4</td>
<td>21</td>
<td>6.93</td>
<td>7.01</td>
</tr>
<tr>
<td>Effort</td>
<td>1</td>
<td>6</td>
<td>18</td>
<td>8.07</td>
<td>6.05</td>
</tr>
<tr>
<td>Frustration</td>
<td>1</td>
<td>7</td>
<td>18</td>
<td>7.13</td>
<td>5.62</td>
</tr>
</tbody>
</table>
Figure C.1: Spearman rank-order correlations for task 1 data.
Figure C.2: Spearman rank-order correlations for task 2 data.
Figure C.3: Spearman rank-order correlations for task 3 data.

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Completion</th>
<th>Comments</th>
<th>Fidgets</th>
<th>Man Pages</th>
<th>Noise</th>
<th>Questions</th>
<th>Staring</th>
<th>Compiles</th>
<th>Compile Success</th>
<th>Runs</th>
<th>Run Success</th>
<th>Cuts</th>
<th>Copies</th>
<th>Pastes</th>
<th>Mental</th>
<th>Physical</th>
<th>Temporal</th>
<th>Performance</th>
<th>Effort</th>
<th>Frustration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>0.4</td>
<td>0.25</td>
<td>0.59</td>
<td>0.67</td>
<td>0.02</td>
<td>0.3</td>
<td>0.47</td>
<td>0.37</td>
<td>0.34</td>
<td>0.48</td>
<td>0.48</td>
<td>0.32</td>
<td>0.25</td>
<td>0.35</td>
<td>0.35</td>
<td>0.43</td>
<td>0.39</td>
<td>-0.28</td>
<td>0.34</td>
<td>-0.19</td>
<td></td>
</tr>
<tr>
<td>Compilation</td>
<td>0.34</td>
<td>0.14</td>
<td>0.77</td>
<td>-0.05</td>
<td>0.14</td>
<td>0.33</td>
<td>0.36</td>
<td>-0.27</td>
<td>0.26</td>
<td>0.69</td>
<td>0.69</td>
<td>0.05</td>
<td>0.04</td>
<td>0.05</td>
<td>0.02</td>
<td>-0.27</td>
<td>0.28</td>
<td>-0.63</td>
<td>0.1</td>
<td>-0.32</td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>-0.19</td>
<td>0.41</td>
<td>0.46</td>
<td>-0.61</td>
<td>-0.15</td>
<td>-0.27</td>
<td>-0.37</td>
<td>-0.11</td>
<td>-0.37</td>
<td>0.85</td>
<td>0.85</td>
<td>0.04</td>
<td>0.05</td>
<td>0.02</td>
<td>0.02</td>
<td>-0.06</td>
<td>0.12</td>
<td>-0.11</td>
<td>0.2</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>Fidgets</td>
<td>0.32</td>
<td>-0.04</td>
<td>0.94</td>
<td>0.38</td>
<td>0.76</td>
<td>0.34</td>
<td>0.5</td>
<td>0.32</td>
<td>0.37</td>
<td>0.32</td>
<td>0.32</td>
<td>0.49</td>
<td>0.55</td>
<td>0.56</td>
<td>0.13</td>
<td>-0.04</td>
<td>0.29</td>
<td>-0.02</td>
<td>0.29</td>
<td>-0.52</td>
<td></td>
</tr>
<tr>
<td>Man Pages</td>
<td>-0.03</td>
<td>0.95</td>
<td>0.45</td>
<td>0.46</td>
<td>0.86</td>
<td>0.4</td>
<td>0.11</td>
<td>0.05</td>
<td>0.11</td>
<td>0.11</td>
<td>0.11</td>
<td>0.32</td>
<td>0.18</td>
<td>0.11</td>
<td>0.18</td>
<td>0.28</td>
<td>0.18</td>
<td>-0.28</td>
<td>0.18</td>
<td>-0.45</td>
<td></td>
</tr>
<tr>
<td>Noise</td>
<td>0.11</td>
<td>-0.5</td>
<td>-0.27</td>
<td>-0.97</td>
<td>0.03</td>
<td>0.12</td>
<td>-0.17</td>
<td>0.25</td>
<td>0.02</td>
<td>0.25</td>
<td>0.25</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>-0.09</td>
<td>0.18</td>
<td>-0.97</td>
<td>0.18</td>
<td>-0.54</td>
<td></td>
</tr>
<tr>
<td>Questions</td>
<td>0.3</td>
<td>-0.36</td>
<td>-0.21</td>
<td>-0.12</td>
<td>-0.29</td>
<td>0.39</td>
<td>0.01</td>
<td>0.13</td>
<td>0.13</td>
<td>0.19</td>
<td>0.19</td>
<td>0.25</td>
<td>0.14</td>
<td>0.14</td>
<td>0.26</td>
<td>0.14</td>
<td>0.19</td>
<td>0.26</td>
<td>0.14</td>
<td>0.14</td>
<td></td>
</tr>
<tr>
<td>Staring</td>
<td>0.34</td>
<td>0.18</td>
<td>0.16</td>
<td>0.26</td>
<td>-0.34</td>
<td>-0.13</td>
<td>0.15</td>
<td>-0.05</td>
<td>-0.07</td>
<td>-0.03</td>
<td>-0.03</td>
<td>0.83</td>
<td>0.35</td>
<td>0.35</td>
<td>0.35</td>
<td>0.83</td>
<td>0.35</td>
<td>-0.03</td>
<td>0.35</td>
<td>-0.03</td>
<td></td>
</tr>
<tr>
<td>Compile Success</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run Success</td>
<td>0.19</td>
<td>0.42</td>
<td>0.11</td>
<td>0.21</td>
<td>0.19</td>
<td>0.02</td>
<td>0.14</td>
<td>-0.13</td>
<td>0.33</td>
<td>-0.04</td>
<td>0.04</td>
<td>0.23</td>
<td>0.03</td>
<td>0.34</td>
<td>0.34</td>
<td>0.34</td>
<td>0.34</td>
<td>-0.04</td>
<td>0.34</td>
<td>-0.03</td>
<td></td>
</tr>
<tr>
<td>Cuts</td>
<td>0.55</td>
<td>0.72</td>
<td>0.28</td>
<td>0.37</td>
<td>0.19</td>
<td>0.05</td>
<td>0.25</td>
<td>0.46</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>Copies</td>
<td>0.87</td>
<td>0.36</td>
<td>0.53</td>
<td>-0.02</td>
<td>0.36</td>
<td>0.55</td>
<td>0.35</td>
<td>0.32</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>Pastes</td>
<td>0.4</td>
<td>0.59</td>
<td>0.52</td>
<td>-0.11</td>
<td>0.44</td>
<td>0.39</td>
<td>0.39</td>
<td>0.39</td>
<td>0.39</td>
<td>0.39</td>
<td>0.39</td>
<td>0.39</td>
<td>0.39</td>
<td>0.39</td>
<td>0.39</td>
<td>0.39</td>
<td>0.39</td>
<td>0.39</td>
<td>0.39</td>
<td>0.39</td>
<td></td>
</tr>
<tr>
<td>Mental</td>
<td>0.72</td>
<td>0.2</td>
<td>-0.52</td>
<td>0.76</td>
<td>-0.05</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
<td></td>
</tr>
<tr>
<td>Physical</td>
<td>0.41</td>
<td>0.16</td>
<td>0.46</td>
<td>0.1</td>
<td>0.41</td>
<td>0.16</td>
<td>0.41</td>
<td>0.41</td>
<td>0.41</td>
<td>0.41</td>
<td>0.41</td>
<td>0.41</td>
<td>0.41</td>
<td>0.41</td>
<td>0.41</td>
<td>0.41</td>
<td>0.41</td>
<td>0.41</td>
<td>0.41</td>
<td>0.41</td>
<td></td>
</tr>
<tr>
<td>Temporal</td>
<td>-0.24</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>-0.12</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
<td></td>
</tr>
<tr>
<td>Effort</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frustration</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure C.4: Spearman rank-order correlations for task 4 data.
Table C.5: Bayes Net results for determining success.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Trial 1</th>
<th>Trial 2</th>
<th>Trial 3</th>
<th>Trial 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes included</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Time</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Computer observations</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Confidence &amp; Sem. Lvl.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TLX</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Visual observations</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**Results - Completion (3 bins)**

<table>
<thead>
<tr>
<th>Percent correct</th>
<th>58.3%</th>
<th>60.0%</th>
<th>56.7%</th>
<th>46.7%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number correct</td>
<td>35/60</td>
<td>36/60</td>
<td>34/60</td>
<td>28/60</td>
</tr>
<tr>
<td>Kappa</td>
<td>0.3625</td>
<td>0.3857</td>
<td>0.3432</td>
<td>0.1926</td>
</tr>
<tr>
<td>MAE</td>
<td>0.2905</td>
<td>0.2894</td>
<td>0.3305</td>
<td>0.3558</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.4669</td>
<td>0.4606</td>
<td>0.4655</td>
<td>0.4775</td>
</tr>
</tbody>
</table>

**Results - Success (yes/no)**

<table>
<thead>
<tr>
<th>Percent correct</th>
<th>78.3%</th>
<th>78.3%</th>
<th>73.3%</th>
<th>75.0%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number correct</td>
<td>47/60</td>
<td>47/60</td>
<td>44/60</td>
<td>45/60</td>
</tr>
<tr>
<td>Kappa</td>
<td>0.3036</td>
<td>0.1875</td>
<td>-0.0435</td>
<td>-0.125</td>
</tr>
<tr>
<td>MAE</td>
<td>0.2409</td>
<td>0.2663</td>
<td>0.2986</td>
<td>0.2875</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.4127</td>
<td>0.4250</td>
<td>0.4310</td>
<td>0.4152</td>
</tr>
</tbody>
</table>
References


Barnett, M., Fähndrich, M., de Halleux, P., Logozzo, F., & Tillmann, N.


systems. In Proceedings of the 2nd international conference on innovative computing, information and control (pp. 147–150). Kumamoto, Japan.


compilation behaviour. In *Proceedings of the 2nd international workshop on computing education research* (pp. 73–84). Canterbury, UK.


Tabanao, E. S., Rodrigo, M. M. T., & Jadud, M. C. (2011). Predicting at-risk novice Java programmers through the analysis of online protocols. In *Proceedings of the 7th international workshop on computing education research* (pp. 1–8). Providence, RI, USA.


Watson, C., Li, F. W. B., & Godwin, J. L. (2013). Predicting Performance in


