Pair Programming and Unobtrusive Monitoring:

Toward an Automated Partner Matching System

By

L Hobbes LeGault

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

University of Wisconsin–Madison

2018

Date of final oral examination: 12/07/2018

The dissertation is approved by the following members of the Final Oral Committee:

Dr. Matthew Berland, Associate Professor, Curriculum & Instruction
Dr. Martina Rau, Assistant Professor, Educational Psychology
Dr. Andrea Arpaci-Dusseau, Professor, Computer Sciences
Dr. Ben Liblit, Associate Professor, Computer Sciences
Dr. Jim Williams, Associate Faculty Associate, Computer Sciences

Contents

Chapter 1

Introduction

The research presented in the following document is focused on the study of novice-level programmers, working in pairs. The goal of this research was to investigate unobtrusive methods for observing programming work *in situ*, as well as the utility of a non-compulsory policy for partnered work in an introductory course (colloquially, "CS1"). The goal of this research is to provide the groundwork for future introductory course practice and policy in the form of two research questions:

> **Research question 1:** Is self-selection of partners (or self-selection to individual work) an educationally-sound policy in an introductory programming course?

One of the primary issues with self-selection of partners is one of equity of access — already-disadvantaged students with little or no preexisting social network in computer science courses are further disadvantaged when it comes to partner selection, as many students tend to select people they already know. Therefore it is a social responsibility of instructors of introductory courses who allow self-selection to support these students in their search for partners.

As discussed in Section 1.1.1, many strategies exist, but they depend on randomness, honesty in student self-reporting, or time-intensive and intrusive observational methods. This leads to the second research question.

> **Research question 2:** What can machine learning methods, applied to unobtrusively-gathered observations, tell us about the skills and working style of (pair) programmers?

The combination of answers to these research questions will lead to advances in how instructors support their CS1 students, discussed in depth in Chapter 8.

1.1 Introduction to Pair Programming

Pair Programming is not uncommon in either the computer science classroom or the programming industry. In contrast to general applications of group work, Pair Programming (PP) has a formalized structure beyond simply "two people working together"; to fit the formal definition of "Pair Programming", two programmers collaborate on a single task using one computer, and differentiate into two predefined roles, the **driver** and the **navigator**[1], which they alternate between over the course of development [22].

Industry embraces this paradigm for reasons of productivity [60]: while only 50% of programmers are actually writing code at a given time, the code produced has been shown to have up to 90% fewer errors, require less debugging, and twice as many programmers are familiar with the contents of the code than if it was produced individually. This is the classic "two heads are better than one" justification.

In an educational setting, programmers' deliverables are of less import than the learning process, so industry's justifications for using PP may be less enticing to an

[1]Formal terminology definitions are located at the end of this chapter.

educator. However, PP has been shown to have additional benefits beyond code quality that make it a valuable classroom resource [7]: it decreases grading workload, as students produce half as many programs; it allows students to engage in peer teaching and learning; and it introduces a social aspect to programming. Additionally, it counteracts the misconception of programming as a solitary lifestyle and making CS as a field more inviting to traditionally underrepresented groups (e.g. [13, 57, 58]), who may be put off by the stereotype of a lone programmer working in a windowless room at 3am eating Cheetos [41, 16].

Computer Science departments across the country are experiencing booming enrollments, but this is not a new problem; departments experienced the "PC boom" in the mid-1980s and the "dot-com boom" in the late 1990s. However, the current enrollment increase is less of a boom and more of a paradigm shift to a computing-oriented society [11]. Enrollments are growing more quickly than associated instructional staff and faculty resources can keep up. Many courses have turned to automated testing, but as personalized feedback is still valuable to students, halving the number of programs to be reviewed by graders or instructors has its utility.

More importantly, increasing class sizes result in less individual attention for students (resulting in adverse affects on academic outcomes, unless consciously mitigated with collaborative learning strategies [4]). The primary resource which scales with class size, however, is number of peers available for students. While instructors might not be able to provide the individualized attention students require to progress in their studies, well-selected partners who are at a similar level in their programming careers can provide even more valuable feedback than an instructor whose mastery of a concept can potentially blind them to the misconceptions a student holds.

Vygotsky's theory of the Zone of Proximal Development ([53], Figure 1.1) includes a visual representation of the tasks which a learner can and cannot accomplish. The

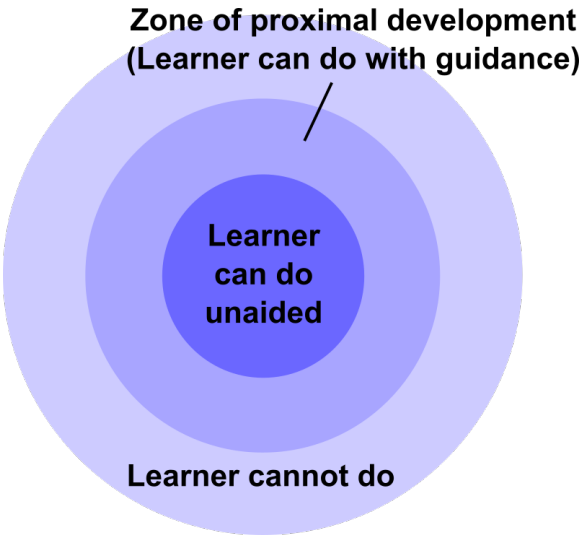


Figure 1.1: ZPD visualization; figure by Wikipedia user Dcoetzee - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=20903046

innermost zone contains the tasks a learner can accomplish independently – concepts they have mastered. The outermost zone contains the tasks a learner is unable to accomplish at their current level of understanding. Between these two lies the Zone of Proximal Development (ZPD): the realm of tasks which a learner can accomplish with aid, the work they are currently *primed* to learn to do. In an ideal peer learning situation, two students with complementary ZPDs are paired. They aid each other in progressing through the ZPD, and accomplish tasks together that neither learner could have accomplished alone.

Peer teaching and learning focus on mastery of course content, but in studies of underrepresented groups and their decisions to continue or not in computer science courses, students cite not their understanding of the material, but rather a feeling that they do not belong or are unwelcome in the field [12]. Women and underrepresented racial and ethnic groups are much less likely to encounter people in the industry or even in their classes who look like them, which leads to a sense of alienation [57, 58].

By introducing group work at an early stage in programming courses and encouraging discussion and social connections, not only can members of underrepresented groups forge connections and find commonalities that transcend gender identity and racial or ethnic background, but also members of well-represented groups can gain exposure to the insights that such individuals bring to the table. Diversity in a field benefits everyone [38].

1.1.1 Partner assignment strategies

Presently, most assignment of partners or group members relies on one of two approaches: either students select their own partner/group without interference (self-selection), or students are assigned a partner/group by some coordinator. There are a number of different assignment methods: a human (usually a supervisor or instructor) may assign pairs by hand, assignment may be done randomly by computer, or assignment may be done either by hand or automatically but in some informed way. For example, relying on a survey or other self-reported information from the programmers to make assignment decisions.

Each of these approaches has its positives and negatives and may be appropriate in different contexts. Assigning pairs by hand is imprecise and can be a significant time investment, particularly if assigning a large number of pairs. However, it allows the assigning party fine-grained control over the matching. Random assignment ensures that everyone has an equal likelihood of being assigned a good or bad partner, but carries the baggage of a compromise: rarely is anyone actually happy with the result. In contrast, self-selection gives an advantage to those with preexisting social networks, and functions little better than random assignment (and potentially even worse) for those without those connections. Informed matching (like that done

by the CATME tool [37]; see Chapter 2 for a more detailed discussion) can potentially improve upon random assignment, but still relies on programmers' self-reported responses to questions, which may not be an accurate representation of reality.

1.2 Overview of the research

The motivating observation for this research topic is that matching pair programmers is akin to matching dating couples. The working title for the tool, "OKCoder", is a play on a popular dating website's name, wherein individuals looking for a partner answer multiple choice questions and are presented with a selection of potential dates along with compatibility scores. Users set up their own dates based on the website's advice, and if a match does not work out, there are generally plenty of other options.

When selecting something as major as a life partner, there is *guidance* in selection available, but for the most part actors still enjoy a level of *agency*.[2] Why not provide similar guidance and allow similar agency when selecting a programming partner?

The general practice in introductory programming courses at the University of Wisconsin–Madison is to permit but not require group work. Students' motivations for electing to work individually rather than finding a partner were examined in the second study, detailed in Chapter 3. Students in the study detailed in Chapter 4 were required to work with a partner for the studied activity only.

Study participants were students in an introductory programming course for non-majors and were assumed to have little to no experience with programming. Students were generally mid-level undergraduates from a variety of backgrounds but a majority of students were from the College of Engineering. I did not examine the effects

[2]While arranged marriages exist, they are uncommon in the United States, which is the context of this research. In cultures where arranged marriages are more common, one may analogously encounter different attitudes vis-à-vis assigned partnerships.

of seniority or major on perceptions of or success in group work situations, and replicating these studies in a similar course geared at potential CS majors would provide at least interesting sociological insight.

1.3 Stakeholders

The present work has three distinct audiences: computer science educators, researchers in the area of machine learning applications with a specific nod to the use of ML in web application analytics, and programmers wishing to better understand and apply a pair programming paradigm in their own work.

Computer science educators who currently allow, encourage, or require pair programming in their classes have expressed an interest in the product of this research, a utility for suggesting partners to students from within a large class. Anecdotally, students express that the task of finding a "good" partner in their large lectures (whether "large" means 50 or 500 students) is daunting-to-impossible, and it is in both their interest and the interest of the instructor to have an automated solution for narrowing that field.

Computer science educators who require *or prohibit* partnered work in their course will find Chapter 3 of particular interest, where I compare student perceptions and outcomes in a course where partnering is voluntary and self-selected. While I do not wish to challenge the assertion that all students should have the experience of working both alone and with others, I found that there is no detriment to allowing students to work as they prefer in an introductory course.

Additionally, the review of educational literature in Chapter 2 is of particular relevance to those educators considering incorporating pair programming into their curricula. As with all choices in an educational setting it is beneficial to have an

understanding of the research (or lack thereof) backing up the choice to allow partnered work so that it can be properly understood by the instructor and adequately motivated to students in the course.

Researchers interested in the applications of machine learning techniques to other fields, particularly to the domain of web application analysis, will be interested in the techniques employed in working style logging and analysis presented in the first study. I explore a number of different machine learning techniques as applied to usage logs in a web application to classify, cluster, and characterize different categories of user, and remark upon the efficacy of each technique.

While the review of relevant machine learning literature likely holds nothing new to ML researchers, the survey of educational literature may serve as an introduction to the background of CS education research for ML researchers considering education as a field for applications of these algorithms.

From another angle, programmers interested in a deeper understanding of their own work will be interested in the findings regarding working style in the first study in Chapter 5. It may be illustrative for such programmers to self-analyze working style and compare results to their colleagues to determine working compatibility, or even to help quantify the idea of "working style" in the first place. Additionally, they may be interested in incorporating some of the findings of the third study (chapter 6) to their pair programming selection; though I again focused on introductory skill sets, the complementarity of programmer knowledge bases remains important at all levels.

1.4 Chapter overviews

The remainder of this document is laid out as follows: Chapter 2 contains a review of relevant literature, Chapter 3 describes the study of student perceptions of pair programming activities over their full semester course, Chapter 4 describes the study of pair programming behaviors through interaction with a web-based tool which are further investigated in Chapters 5 and 6, Chapter 7 summarizes the findings and Chapter 8 details impacts and future research directions.

The literature reviewed in Chapter 2 is drawn from a variety of disciplines. First I review the educational literature detailing the current practice of pair programming in classroom settings, including best practices for pair programmers and pairing strategies. Then I turn to more industry-focused literature for strategies in analyzing website transaction logs, and expand upon those strategies by incorporating machine learning literature.

Chapter 3 discusses a detailed survey study completed over a full semester. Unlike the previous study wherein pair programming was compulsory, students could choose to work alone or find their own partners for each assignment; both individual programmers and partnered programmers filled out the survey, which included situationally-tailored questions about their solo/paired choices as well as common questions regarding, for example, perceptions of the assignment's difficulty. These responses were matched with assignment scores for analysis of the real and perceived advantages and disadvantages of students' chosen working configuration.

Chapter 4 involves the observation of pair programming activities using only the pair's interactions with a web-based programming tool. The study also included a simple perception survey that inspired the full survey in the following chapter, and this data was combined with the log data as a basic analysis of quality of experience.

This chapter also includes an in-depth discussion of the unobtrusive observational data. I answer the following six questions and discuss the power and limitations implicit in the data: which data are analyzed, how is this data defined, what is the population from which the researcher has drawn the data, what is the context in which the researcher has analyzed the data, what are the boundaries of the analysis, and what is the target of the inferences. I describe in detail the transaction log data from the pair programming behavior study, the survey data from the perceptions study, and the multiple choice question data used for the final proof of concept.

Chapter 5 involves the observation of pair programming activities using only the pair's interactions with a web-based programming tool. These interaction logs were used to predict success or failure at the task, as well as exploration of certain "coding styles". Additionally, the study included a simple perception survey that inspired the full survey in the following chapter, and this data was combined with the log data as a basic analysis of quality of experience.

Chapter 6 further explores the web transaction logs obtained in this pair programming study using a graph model from game analytics. The goal of this study was to examine the paths taken by successful and unsuccessful programmers through the solution space of a difficult task, and explore trends in these paths.

Chapter 7 summarizes the results of the full course of research across all three studies and Chapter 8 lays the groundwork for future research directions. The studies in this dissertation form a progression of investigation of paired or partnered programming, but also of machine-supported coding and behavior analysis of novice programmers in a much lower-stress setting than a formal examination.

1.5 Definitions

For the purposes of this document, **Pair Programming (PP)** will refer to the formal paradigm wherein two students use a single computer to create a program, using the roles of **driver** and **navigator** as described here:

Definition. The *driver* manipulates the physical computer, controlling the input devices (keyboard/mouse), entering code into the editor, and actually running the programs created.

Definition. The *navigator* proofreads the code entered by the driver, offers suggestions, and may have a separate device for looking up answers to questions the pair has.

However, PP was not the only partnered paradigm that I observed in my research:

Definition. In *partnered programming*, two students work next to each other on separate computers to create a program. They may or may not adhere to the formal roles of driver and navigator, but each student controls their own device and has the ability to write and run code while also observing the other student's work directly.

Definition. In *divide-and-conquer*, two students create a program by completing sections of the work independently. Students are not co-located in this paradigm, and observe only their partner's finished work.

Students in the studied course were not required to work with a partner except in the first study; students completing their work individually without the help of a partner will be referred to as *solo programmers*. Solo programmers were not forbidden from using additional resources (e.g. tutors or teaching assistants), nor were they forbidden from discussing their work with other students at a high level, but did not work directly with another student to produce their programs.

Chapter 2

Literature Review

Pair Programming as an explicit paradigm gained its popularity with Extreme Programming in the late 90s and early 2000s [3]. Since then it has been adopted into the educational sphere, and the educational research literature has explored several questions which this chapter will review:

1. What is the educational benefit of **collaborative learning** in general, and Pair Programming specifically?

2. What **metrics** are used to evaluate the relative quality of collaborative learning groups or pairs?

3. What are effective methods for **creating** collaborative learning groups or pairs?

Following a review of this literature, this chapter will review the current state of applications of machine learning in education, specifically in computer-supported collaborative learning (CSCL), to explore a fourth question:

4. How is **machine learning** used to analyze programmers' performance in an educational setting?

2.1 Collaborative Learning

The concept of learning in community has regained traction in the past 30 years as educational researchers call attention to the shortcomings of the traditional lecture, "sage on the stage" style classroom in favor of a more participatory learning setting. Aronson and Bridgeman's 1978 work on 'Jigsaw' [2] inspired further work in the 1990s [8, 9] examining the utility of grouping students with complimentary skill sets to encourage peer learning. The idea of Reciprocal Teaching [8], wherein students ask each other questions and collaboratively create responses by synthesizing information, directly influences the PP paradigm in that students in an actively communicative partnership can not only create a better product (the justification used for PP in industry), but also can augment each others' learning.

Particularly given the rise of internet-based learning, computer-supported collaborative learning (CSCL) is of particular interest across disciplines. Dillenbourg and Schneider [17] revisited collaborative learning in the context of a movement to internet-based distance collaboration, underlining its mechanisms and situations in which it becomes optimal: when collaborating, students are faced with viewpoints other than their own and must either assimilate and learn from those views, defend their own, or otherwise combine their understanding with that of their peers.

2.1.1 What is Pair Programming?

The literature generally agrees (e.g. [19, 23]) that Pair Programming (PP) as a formal practice is the process where two programmers share a single computer to perform a task. The practice began in the 1950s [28] and gained popularity as a component of Extreme Programming (XP) in the 1990s, before finding its way into classrooms in

the US, UK, Germany, New Zealand, India and Thailand in the late 90s and early 2000s [23].

Wray [60] highlights four mechanisms by which PP has particular advantages over traditional individual programming:

1. **Pair Programming chat.** The problem-solving approach of talking through a problem aloud is so well-documented that many companies will include a rubber duck as a part of their onboarding package for new programmers [10], so that when stuck on a problem the programmer can explain the situation to the duck without taking up another programmer's time (industry programmers report that more difficult problems sometimes require more than one duck [31]). However in PP, verbal communication between programmers is frequent and so problem-solving occurs in a multi-modal framework from the onset, and programmers are less likely to become stuck.

2. **Pair Programmers notice more details.** Different people tend to pay attention to different aspects of their code. Regardless of the magnitude of that difference, having two people reading code while it is being produced means that a broader range of errors, problems, and solutions will be noticed.

3. **Fighting poor practices.** When programming individually it can be tempting to cut corners in terms of style and documentation; when another person is actively reading the code, it can encourage programmers to be more thorough in terms of best practices for coding. In particular, a pair can hold each other accountable.

4. **Sharing and judging expertise**. Each person in a partnership has their own unique skillset, and combining skillsets will allow for a more robust solution.

What PP can also do, though, is allow the partners to assess each other's skill level at very close range.

While the fourth mechanism may be useful in terms of peer review, educational applications of PP tend to focus on the first three mechanisms as desirable contributions to student learning, resulting in improved outcomes for students in courses which use PP (e.g. [6]).

2.2 Pair Programming evaluation metrics

As identified in Umapathy and Ritzhaupt [52], there are four themes to the metrics used in pair programming studies for identifying the effectiveness of pair programming (typically in contrast to individual programming) in an academic setting:

1. **Assignment performance** including both scores and measures of quality of code produced)
2. **Exam scores**
3. **Affective measure scores**, self-reported attitudes regarding the programming experience
4. **Passing rates** and other measures of student retention

Studies utilizing these metrics generally look at the comparative measures on each between pairs and individuals (see also [45]), but these metrics are also applicable for within-groups analysis to determine more- or less-effective pairs of programmers. This approach is frequently used in analyses of the gender composition of pairs (e.g. [34]).

It is important to note that these thematic metrics are non-standardized. As Umapathy and Ritzhaupt note, there is inconsistent reporting across the primary

Factor	Positive Effect	Negative Effect	No Effect
Personality	2	-	5
Actual Skill Level	9	2	1
Perceived Skill Level	4	-	-
Self-esteem	-	-	3
Gender	1	-	1
Ethnicity	1	-	-
Learning style	-	-	2
Work ethic	1	-	1
Time management	-	-	1
"Feelgood" factor	2	-	-
Confidence level	1	1	-
Type of role	-	-	1
Type of tasks	1	-	-
Communication skills	-	-	1

Table 2.1: Adapted from [45], a list of factors investigated in Pair Programming studies and the number of studies reporting a significant positive or negative effect (or no effect).

studies. Assignment specifications and rubrics are rarely reported, exams are generally custom-written for each studied course, and many of the instruments for course grading may include other forms of bias not detectable from published article contents.

2.2.1 Factors affecting programming effectiveness

A large number of factors have been identified as potentially contributing to the efficacy of different pairs of programmers. In [45], 14 different categories of factors were distinguished from the literature, 9 of which were found to have some kind of significant effect on the pair (see Table 2.1).

The most reproduced results in the table are those of the top three rows, showing that personality has no significant effect, and that actual and perceived skill level in the pairing both have a significant positive effect on the efficacy of the programmers.

2.3 Creating effective programming pairs

The literature reflects three general approaches to pair assignment in educational settings [21]: self-selection, where students choose their own partners from the class; random assignment, where an agent (usually a computer) randomly pairs students from the class; and informed assignment, where individuals complete a survey or pretest and the results are used by an agent (computer or human) to create pairs. In this section, I will examine each of these approaches, and their stated advantages and disadvantages, and ultimately why I opted to craft the approach of this research.

2.3.1 Self-selection

Self-selection of partners is the least instructor-guided approach to partnering – generally the understanding is that all students are required to have one and only one partner, so that as more students pair off, the field of selection becomes narrower. Initial partner selections might follow very stringent criteria, or reflect preexisting networks within the classroom, while subsequent selections must necessarily broaden their acceptable criteria until "they didn't have a partner yet" becomes the most important metric.

Students who are able to select their partner freely from their class view self-selection positively [26]; they know that they will get along with the person they have selected, they reinforce their sense of belonging in the classroom and the subject at large, and they feel more control over their educational situation than they do when assigned a partner externally.

However in Dillenbourg and Schneider [17], instructors are warned against allowing self-selection. Students tend to select partners who are similar to themselves and hold similar viewpoints and understandings, defeating the mechanism of collaborative

learning wherein heterogeneity leads to conflict, which must be resolved as a learning experience for group members. Instructors are encouraged to introduce"an optimal amount" of heterogeneity to spark this sort of conflict and prevent students from simply confirming their own biases.

Students working with friends can also be distracted and are therefore less likely to engage in deep learning.

Additionally, those students who do not have preexisting networks within the class or the field may find themselves essentially selecting a partner at random, losing out on the benefits that a more well-connected student experiences. As this tends to affect already-underrepresented students in our classrooms at a disproportionate rate, using an alternative to self-selection presents an opportunity to level the playing field to a certain extent.

2.3.2 Random assignment

Random assignment of partners is straightforward and ultimately "fair": each student is paired with a random student in the class, and there are no advantages to being well-connected or disadvantages to choosing later. However, what may be considered fair in terms of an algorithm — in that all students are treated equally — may not actually be an equitable solution; wild mismatches in skill levels or personality clashes can be further discouraging to the very students the "fair" algorithm had been intended to help.

2.3.3 Informed assignment

In response to concerns about partner compatibility from both students and instructors, a number of studies were performed examining the effects of various individual

characteristics on the efficacy of a partnership. In particular, much attention was paid to factors like personality, quantified in various ways including but not limited to Myers-Briggs Type Indicators [15, 24, 47]; self-confidence and self-esteem [59, 48]; and demographic factors like ethnicity and gender identity [16, 33]. Additionally, investigations into the relative skill levels of partners were performed [14, 29], many of which found significant effects.

Most frequently investigated [45] were personality and skill level. Only two of nine studies reported any significant effect of personality though all studies suggested that diverse personalities were advantageous; seven of ten studies found skill level to be a significant determining factor in pair success. Skill level was additionally broken down into actual skill (determined by academic background, programming experience, and performance) and perceived skill (determined by self-reports of students' relative skill compared to their partner's). Generally agreement was that similar skill levels were beneficial to students, while skill levels that differed too much were detrimental.

One very popular method for creating instructor-assigned groups in courses with computer assistance is the Comprehensive Assessment of Team Member Effectiveness, or CATME [37]. CATME offers a suite of tools for group creation based on student responses to survey questions (for example, "when are you unable to meet?"), and allows for significant customization of both questions and grouping metrics by the instructor. For example, the instructor might require that the gender distribution in a group include at least two non-male-identified students, as some studies (e.g. [16]) have detected significant differences in compatibility and communication in groups of different gender compositions.

The benefits of assignment over selection are primarily the uniformity of application — a student with preexisting connections within the course has the same partner selection methodology as the student who knows no one. However, it has been shown

that not only does self-selection vs assignment not impact performance, but also that self-selection "aids in the satisfaction of a student and therefore impacts their decision to retain computer science as their major" [58]. Therefore, rather than simply using CATME for all of our pairing needs, we will continue to investigate self-selection paradigms and instead work to support students without networks in finding their own partners in the future.

2.4 Applications of machine learning to CSCL

Of particular interest for this research is the work of Rosé et al [43], which concerns an application of machine learning techniques to a problem in computer-supported collaborative learning (CSCL). Their application involved developing an automated text annotation scheme for an analysis of transcripts generated by students in collaborative learning environments using techniques from Natural Language Processing (NLP). This work was an automation of existing manual processes, wherein annotation of transcripts or other text documents was completed by trained humans following a coding scheme as in Weinberger and Fischer [56]; Rosé et al looked to automation for efficiency, simplification of the process, and reallocation of researcher resources to more interesting problems.

2.4.1 Web transaction log analysis

The approach used for much of this research is the application of machine learning to automate web log and supplemental data analysis to create and group user profiles for optimal matching of students into pair programming partnerships. Having established the utility of pair programming, I now turn to the machine learning mechanisms by which I will accomplish this task.

Srivastava et al [50] established an early framework for mining web transaction logs for usage patterns, following three basic stages: preprocessing of data, pattern discovery, and pattern analysis.

In the preprocessing stage, transaction logs are converted from their raw form into abstractions that can be used in analysis. This involves identifying individual users so their logs can be grouped, dividing the usage logs of these users into sessions using a suggested 30-minute timeout for session breaks, and performing any content preprocessing necessary to the individual application. This step is necessarily somewhat heuristic and specific to the application and desired outcome.

The pattern discovery stage incorporates techniques from machine learning, statistics, data mining, and any variety of other fields, applied to the preprocessed logs. Once patterns have been discovered, uninteresting (or insignificant) patterns are filtered in the analysis stage.

Among the suggestions for transaction log pattern discovery in Srivastava et al are statistical analysis, clustering and classification, and sequential patterns.

Statistical analysis

Descriptive statistical analysis can function as an introductory exploration of usage logs. Basic statistics like frequency, mean, and median can highlight immediately apparent trends in usage.

Clustering and classification

Clustering and classification are two of the most basic machine learning algorithms. Clustering is an unsupervised machine learning approach with no "correct" answer, wherein data points are grouped using either "hard" or "soft" clusters (see e.g. [35]). With hard clusters a point belongs to exactly one cluster, based on the points it is

closest to in the relevant dimension; when using soft (or probabilistic) clusters, each point is assigned a likelihood that it belongs to the set of clusters. Frequently an algorithm like Expectation-Maximization (EM) is used to create and assign these clusters, whether hard or soft.

Classification of data points is a supervised machine learning approach that requires predefined classes that an agent is trained to recognize, and is subsequently tested by classifying previously unseen data points into the same predefined classes. As discussed in Kotsiantis [32], there are numerous techniques for classification, including everything from logic-based algorithms like decision trees, perceptron-based techniques like neural networks, statistical methods like Bayesian networks, Support Vector Machines (SVMs), and comparative techniques like k-nearest neighbors. Each of these approaches accomplishes the same goal; i.e., the labeling of data points based on predefined classes; the choice of one over another is generally application-dependent.

Sequential patterns

By examining the sequence of actions and events in an application, researchers can gain insight to the particular usage patterns of classes of users as well as a wide variety of other analytics, including uncovering potential bugs or even systematic defects in the system, or studying the effects of an update to the application [55].

Of particular interest in this application is the graph-based method of sequential pattern analysis detailed in [54]. This methodology is explored in more depth in Chapter 6.

Chapter 3

Perceptions of Pair Programming

The purpose of this investigation was to improve upon the post-activity survey originally included in the study in Chapter 4 by refining its self-reported measures of satisfaction. The survey in that study included only four sliders and a free-text entry box, and as it was provided to all students immediately upon completing the activity, all students filled it out in their partner's presence. This iteration of the partner survey (and its complementary solo programmer survey) includes a large number of questions and was completed at the students' leisure after every assignment during the semester.

Openness to partnering is a predictor of partner success [46], a claim we wish to investigate in the context of motivations and perceptions of pair programming and their differences between students who choose to work alone and those who prefer a partner.

Hypothesis 1: Students working with a partner perceive the course workload to be easier and have higher satisfaction ratings with the course material and better course outcomes.

Based on study results, this hypothesis may soften slightly:

Hypothesis 1.1: Students working *in a manner of their own choosing* perceive the course workload to be easier and have higher satisfaction ratings with the course material and better course outcomes.

As discussed in Section 2.1, the literature is highly supportive of pair programming for improved outcomes with novice programmers. However these studies are generally completed using compulsory pairing or individual coursework, and the motivations for students who would choose to work in one paradigm or the other are rarely if ever studied.

3.1 Method

3.1.1 Participants

Participants in this investigation were students in a CS1 course for non-majors at the University of Wisconsin–Madison across two semesters. The course was taught in Python three instructors over the course of the study (one of which was me).

In the first semester, the course had 389 students at the end of the semester; each week partner surveys received between 137–178 (mean=155.58, SD=11.83) responses and solo surveys received between 64–184 (mean=130.58, SD=37.67) responses; of those an average of 44.75 (29%, SD=7.11) partnered students reported working in a strict pair programming style, 82.08 (53%, SD=9.26) reported using partner programming, and 28.75 (19%, SD=11.58) reported using divide-and-conquer (see Figure 3.1). An average of 55% of respondents each week completed partner surveys, with a maximum value of 73.55% for the final program.

In the second semester, the course had 401 students at the end of the semester; surveys received an average of 151.67 (partner, SD=28.89) and 188.33 (solo, SD=39.95)

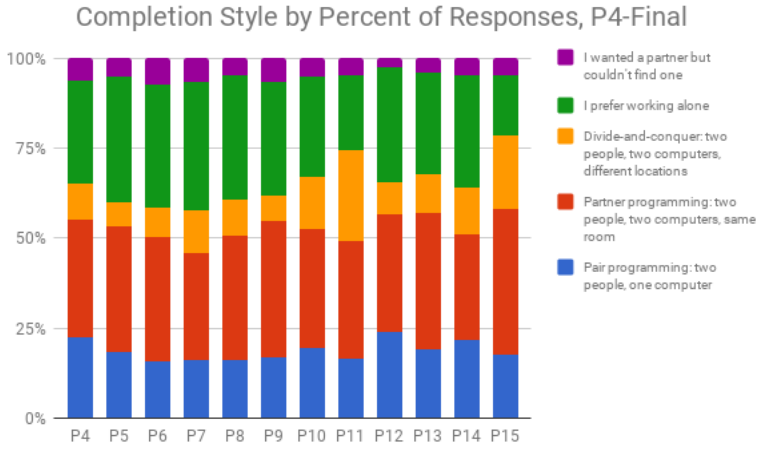


Figure 3.1: Percent of students reporting working in each style, fall 2017.

responses each week; of those on average approximately 61.8 (18.43% SD=22.51) partnered students reported working in a strict pair programming style, 49.33 (14.41%, SD=15.77) reported using partner programming, and 40.53 (12.06%, SD=16.35) reported using divide-and-conquer (see Figure 3.2). An average of 45% of respondents each week completed partner surveys, with a maximum value of 59.54% for the final program.

During both semesters students self-selected to either the partnered or solo programming condition and could switch partners or conditions for each assignment.

Survey completion was not required for class credit, nor were participants compensated in any other way for completing the surveys, though completion was strongly encouraged as a method of giving feedback to instructors. In the first semester, students completed an average of 10 surveys out of a possible 17 (SD = 3.89) and only 12 students completed surveys for every assignment during the semester. Consistent participation was higher in the second semester; students completed an average of 12.17 surveys out of a possible 16 (SD = 3.24) with 23 students completing every offered survey.

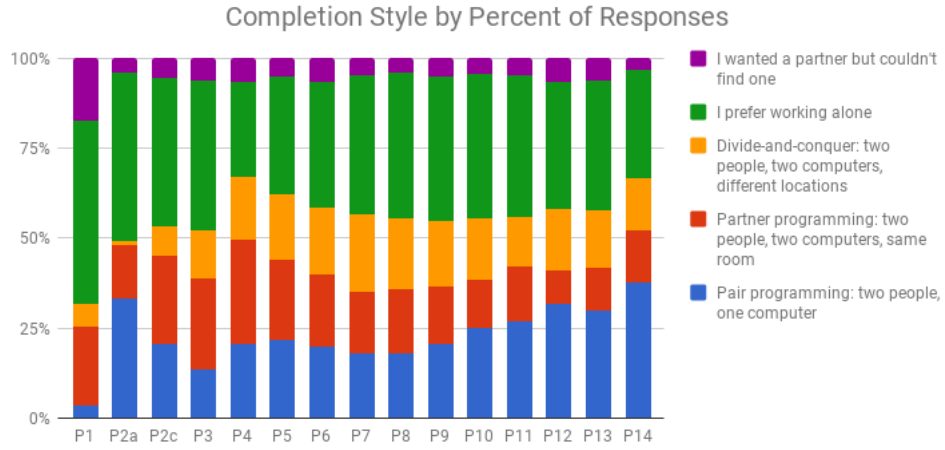


Figure 3.2: Percent of students reporting working in each style, spring 2018.

Data was not collected anonymously, but was access-restricted to only instructors and TAs. At the conclusion of the semester and after final grades were submitted, student responses were matched to corresponding grade information from Canvas using an ID number, and identifying information was then removed. All data included in the following analysis is described only in the aggregate.

3.1.2 Materials

The surveys were developed using Google Forms for simplicity and accessibility, as seen in Figure 3.3. Students were required to complete these surveys weekly for grading purposes, in part because the partner-tracking methods used by the university's learning management system (Canvas) were not suitable for the course[1]. Google Forms allows for immediate exporting of survey data into a spreadsheet, which can be incorporated easily into grading scripts.

[1]Students were allowed to work individually or in pairs, and could switch partners on each assignment. Canvas' grouping system requires a new group to be created and joined for each student and each assignment; for 300-450 students across 15 assignments this becomes unfeasible. We tried.

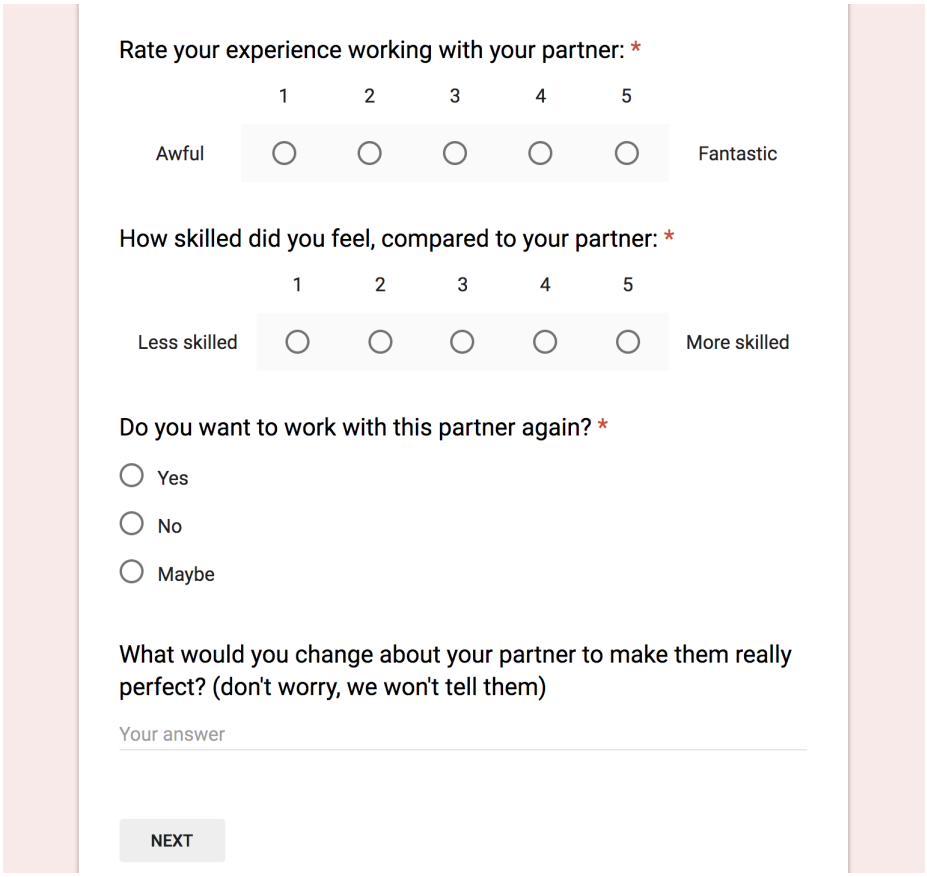


Figure 3.3: A partial view of the partner survey interface in Google Forms.

Question Text	Response	Which Survey
Which program did you complete?	Select	Both
What is your name?	Text	Both
How long did this assignment take to complete?	Select	Both
How difficult did you find this assignment?	1-5	Both
What kinds of work did you do on this assignment?	Multiple Select	Both
Was there anything in particular you thought was too easy/ hard about this assignment?	Text	Both
Did you learn anything from doing this assignment? If so, what?	Text	Both
Why did you choose to work alone?	Multiple Select	S only
Are you interested in a partner for future projects?	Select	S only
What is your partner's name?	Text	P only
Why did you choose to work with this partner?	Select	P only
Rate your experience working with your partner:	1-5	P only
How skilled did you feel, compared to your partner:	1-5	P only
Do you want to work with this partner again?	Select	P only
What would you change about your partner to make them really perfect?	Text	P only
How did you complete the majority of this assignment?	Select	P only
About how much of the work would you say you did?	1-5	P only

Figure 3.4: Program completion survey questions for both solo (S) and partnered (P) students.

In the associated spreadsheet, I created a dashboard of response analysis for several questions individually as well as dependencies between questions, so that as the semester progressed I could get a sense as the instructor for the perceived difficulty of various assignments and adjust the amount of time spent on various topics in class (or amend assignment descriptions accordingly). An example of this dashboard can be seen in Figure 3.5.

3.1.3 Procedure

All groups in this study were self-selected at each iteration, and students could move freely between the two groups. Students were reminded at the end of each assignment to not only submit their work but also the appropriate survey to their working conditions, and were provided a link to both surveys.

P7: Spa Day

Why did you choose to work with this partner?			Rate your experience working with your partner.			How skilled did you feel, compared to your partner.			Do you want to work with this partner again?			About how much of the work would you say you did?			How difficult did you find this assignment?			How long did this assignment take to complete?			How did you complete the majority of this assignment?			What kinds of work did YOU do on this assignment?	
We've worked together before	136	88.3%	1	0	0.0%	1	1	0.6%	Yes	145	94.2%	1	0	0.0%	1	2.33%	Pair programming: two people, one computer	Less than an hour	3	1.9%	Pair programming: two people, one computer	43	27.9%	Debugged code	124	80.5%
We haven't worked together before, but we're friends outside of class	15	9.7%	2	2	1.3%	2	10	6.5%	Maybe	8	5.2%	2	6	3.9%	2	20.00%	Partner programming: two people, two computers, same room	1-2 hours	35	22.7%	Partner programming: two people, two computers, same room	80	51.9%	Figured out the design/logic	129	83.8%
We haven't worked together before, but we sit next to each other in lecture	1	0.6%	3	6	3.9%	3	104	67.5%	No	1	0.6%	3	117	76.0%	3	46.25%	Divide-and-conquer: two people, two computers, different locations	2-3 hours	45	29.2%	Divide-and-conquer: two people, two computers, different locations	31	20.1%	Wrote code	140	90.9%
We haven't worked together before, but we sit next to each other in lab	1	0.6%	4	20	13.0%	4	27	17.5%				4	25	16.2%	4	25.00%		3-4 hours	40	26.0%	Other	0	0.0%	Wrote comments	107	69.5%
We haven't worked together before, but they live in my dorm/near my apartment	0	0.0%	5	126	81.8%	5	12	7.8%				5	6	3.9%	5	7.50%		4-5 hours	17	11.0%						
Other	0	0.0%														0.00%		5 hours or more	14	9.1%				n =	154	

Figure 3.5: An example of the instructor dashboard view for Program 7, displaying aggregated responses for partnered programmers.

While the chronology was not enforced, based on survey submission times versus assignment submission times most students adhered to the general suggested procedure of complete the assignment, submit it, and then complete the appropriate survey.

As this study was descriptive in nature, no experimental manipulations were performed.

3.2 Results

The purpose of this study was to explore the attitudes and beliefs of paired and individual students about their work in a course over a full semester's worth of pairing-optional assignments. In this study we asked a broader range of questions and solicited more fine-grained feedback than the previous study, though data was not collected during the program development process.

Reported time spent on assignments varied from mostly "less than an hour" on early assignments through "2-3 hours" on most later assignments. The most reports of "5 hours or more" in the first semester occurred on P9 and P10 as well as the final program (which was intentionally a longer, larger assignment); in the second semester, most reports of "5 hours or more" occurred on P4 (the final program was not intentionally longer or larger than previous assignments), though 57.14% of students working alone because they could not find a partner reported working "5 hours or more" on P13.

Perceived assignment difficulty varied across assignments (Figures 3.6, 3.7). In the first semester, the lowest mean perceived difficulty across all students occurred on P12 (mean = 2.514, SD = 0.863). The highest mean difficulty across all students was P10 (mean = 4.103, SD = 0.784). In the second semester, the lowest mean perceived

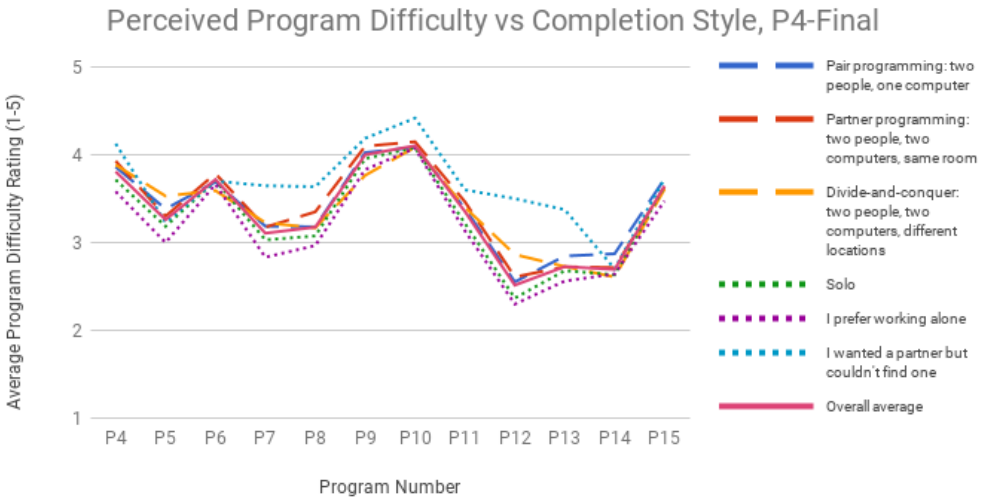


Figure 3.6: Average perceived difficulty of assignments in Fall 2017

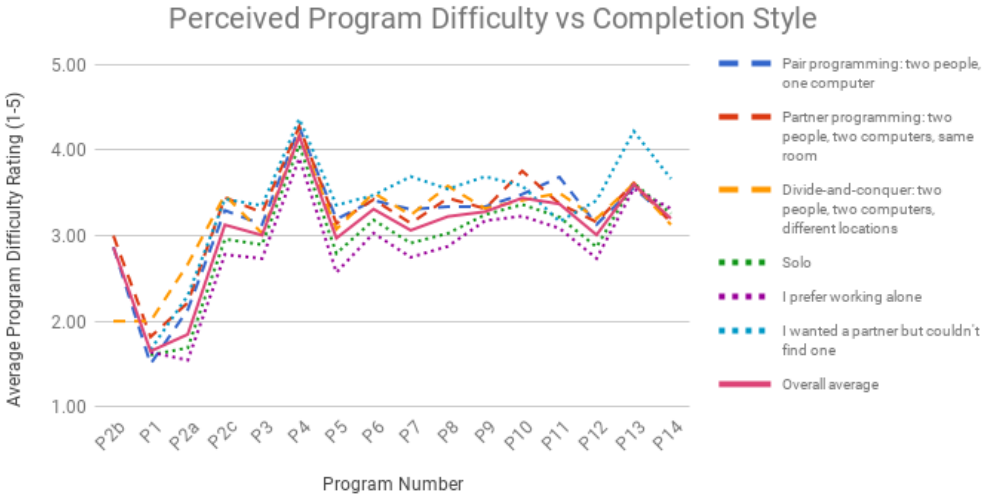


Figure 3.7: Average perceived difficulty of assignments in Spring 2018

difficulty across all students was P1 (mean = 1.649, SD = 0.757); the highest mean difficulty across all students was P4 (mean = 4.162, SD = 0.752).

After P1 well over 50% of partnered students (and usually over 75%) chose to work with their partner because they had worked together before. Average partner satisfaction levels were well above 4.5 on a scale of 5 for all projects except P10 (which also received the highest average perceived difficulty rating of the semester, at 4.103 out of 5 across both partner and solo surveys). Over 80% of partner survey respondents rated their partner as 4 or 5 across assignments, and on average only about 1.28% of respondents rated their partner as 1 or 2 (max = 3.18% on P6). Well over 90% of respondents reported wanting to work with their partner again.

On all programs, partnered students reported an average relative skill level of slightly over 3 on a scale of 1-5 when compared to their partner, with standard deviations of less than 1 on all programs. For division of labor, 70-80% of student reported doing half of the work, with most of the remaining students reporting that they did more than half of the work. At most 8.7% of students reported doing less than half of the work.

For solo programmers, an average of 55.95% of respondents reported working alone because they prefer it; 58.26% believed they learn better when they have to do the entire project themselves. An average of 9.92% reported wanting a partner but being unable to find one; and an average of 13.01% said that finding a partner was too much work.

Across sections

In both semesters where data was gathered, the course was taught in three sections by two instructors (one instructor taught two of the sections and the other instructor taught the third). The section in which a student was enrolled was not significantly

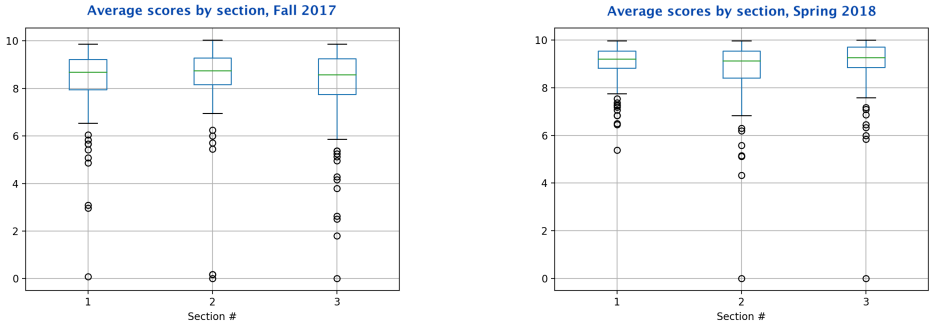


Figure 3.8: Average scores across assignments, grouped by enrollment section

related to their outcomes on the assignments. For fall 2017, $F(2, 386) = 2.31$ and $p = 0.10$; for spring 2018, $F(2, 398) = 2.66$ and $p = 0.07$. A visualization of the average scores divided out by section can be seen in Figure 3.8.

Partners vs solo programmers

Our analysis of exam scores versus partner status on programs revealed no correlation between number of times a student had worked with a partner and performance on exams ($R = -0.067$ for exam 1, $R = -0.050$ for exam 2, $R = 0.095$ for exam 3). Assignment scores showed little difference based on completion style; the only significant difference in performance occurred on the final program, where students working with a partner scored significantly higher than their individual counterparts ($p < 0.01$ for a two-tailed t-test).

A two-tailed t-test of the perceived difficulty of assignments between partnered and solo programmers revealed a significant difference ($p < 0.05$) on several assignments in fall 2017: P4, P5, P8, and P12. For all of these programs, the individual programmers perceived the assignments as being easier on average than the partnered programmers.

Students who couldn't find partners

An average of 12.5 students per first-semester assignment (full course N=389) and 15.8 students per second-semester assignment (full course N=401) reported wanting a partner but being unable to find one. While we observed no correlation ($R = -0.004$) between final exam scores and number of assignments on which a student wanted a partner but was unable to find one, a two-tailed t-test of the general student population's scores on assignments and these students' scores revealed a significant difference ($p < 0.05$) on two programs in fall 2017: P7 and P12. On programs P7, P9, and P12, students who wanted but couldn't find partners received significantly ($p < 0.01$) lower grades than students who worked alone by choice.

Students who could not find partners tended to perceive programs as more difficult. In the fall 2017, they rated P7 and P12 as significantly more difficult than the class in general ($p = 0.011$ and $p = 0.006$ respectively) and also significantly more difficult than the rest of students working individually ($p = 0.006$ and $p = 0.004$ respectively). In the spring, they rated P7 and P13 as significantly more difficult than the class in general ($p = 0.007$ and $p = 0.001$ respectively).

For programs P7 and P12 from the fall semester, we analyzed the rubric-level scores for students who couldn't find partners versus the rest of the students who responded to the surveys. Of the seven rubric elements for P7, four showed a significant difference ($p < 0.05$) for students who couldn't find partners; of the six rubric elements for P12, two showed a significant difference ($p < 0.05$).

Students who eventually found partners

Of particular interest in this study are the subset of students who began the semester working individually but later found a partner and worked with them on more than

one occasion. To define this subset of students, I require that they worked individually on at least the first four programs (with the exception of the compulsory pair programming exercise), and then worked with the same partner on at least two of the later programs. In the first semester, 174 of 389 students (44.7%) did not report working with a partner on the first four programs where solo programming was allowed; of these 174 students, 59 (about 15.2% of the class) reported working with a partner on at least two other occasions.

3.3 Discussion

This study was intended to test the following hypothesis:

Hypothesis 1: Students working with a partner perceive the course workload to be easier and have higher satisfaction ratings with the course material and better course outcomes.

The results of this study demonstrated across semesters that this was *not* the case in general for partnered students versus students who chose to work individually. In fact, for the assignments with a statistically significant difference in perceived difficulty between working styles, solo programmers working alone by their own preference perceived the assignments as easier. Somewhat surprisingly, in the second semester this reduction in difficulty perception was statistically significant in a two-tailed t-test with $p<0.01$ on fully two-thirds of the assignments in which working alone was an option.

Where we did see this hypothesis borne out, however, was when isolating the roughly 3% of students who wanted a partner but were unable to find one – these students worked alone not by choice, and both perceived the assignments as more difficult and achieved lower scores.

For this reason, we instead accept the softened version of Hypothesis 1:

Hypothesis 1.1: Students working *in a manner of their own choosing* perceive the course workload to be easier and have higher satisfaction ratings with the course material and better course outcomes.

3.3.1 Threats to validity

Programs for the course studied were assigned and due weekly, and the scope of the programs reflected this frequency. It is possible that the alignment of graphs between programmers working in each of the various styles is due more to the limited scope of assignments and that the benefit of a pair or partnered programming situation would become more pronounced even over those students who elected to work alone if the task at hand were longer, more complicated, or even more open-ended. Indeed we did see an increase of students choosing to work with a partner on the final program, which was more open-ended than the weekly homework assignments.

Given that observation coupled with the results of this study, however, I would anticipate that the negative effect of belonging to the category of "wanted but could not find a partner" would be even more pronounced in a situation with larger, more complicated, or more open-ended tasks.

3.3.2 Contributions

The literature makes many very important arguments for compulsory partnering – and compulsory solo programming. Compulsory partnering forces students to learn to work collaboratively, articulate their ideas to another person, and meet other people within their area of interest (e.g. [13]). Conversely, individual programming requires students to demonstrate their individual skill without relying on the expertise of their

peers (e.g. [42]). There are undeniably many situations where requiring students to work in a particular style are advantageous, even if the students themselves do not perceive it as such at the time.

However, the results of this study strongly suggest that allowing introductory programming students to work in their preferred partner/solo configuration has no adverse effects on their achievement in the course and in fact does not even contribute to how difficult they perceive the assignments. Additionally, the results of this study suggest that the small proportion of students who are unable to find partners when given the opportunity to work in a partnered situation would be well-served by the sort of partner matching program I aim to create in future work.

Chapter 4

Differential features of partnerships from usage log data

This component of the research explores transaction logs combined with basic survey data as a method for identifying and classifying skillsets. The popular group assignment tool CATME [37] relies on participant self-reports for their input rather than direct observation, which is subject to all of the potential issues of self-reporting. By exploring how usage logs of programming activities within a web-based tool can be used to supplement or even replace self-reported data, I establish a basis for future work in automated partner matching protocols.

Previous studies which have used observation to categorize and evaluate Pair Programming (PP) have primarily concentrated on direct observation of the individuals in the form of video and audio recordings or even in-person observations (e.g. [14]). Transaction log analysis is significantly less intrusive and may lead to more authentic behaviors [27], though what I am observing is merely the interaction of programmers with the tool and not the interaction of partners with each other.

To this end, I developed a web-based tool in which participants completed ten levels of a maze game, wherein they controlled a "pegman" by assembling a program in a block-based language called Blockly [39], similar to Scratch. Participants completed these levels in pairs using PP, and then individually filled out surveys about their experience. Before beginning, they consented or withheld consent for retention of their log data for analysis.

Research question: What are the differential features of partnerships for Pair Programming in CS courses?

The purpose of this study is to explore the features of programmers which are observable both through self-reporting and usage log data, and to determine the utility of those features in describing the quality of a PP partnership.

Hypothesis 1: "Successful" partnerships are predicted by specific features of the data.

For the purpose of this study, "successful" partnerships will refer to both partnerships wherein the partners self-reported ideal results in their post-activity surveys (positive experience, shared workload, high amount learned, and/or desire to work together again) as well as partnerships that created correct solutions to the final, challenge level in the activity.

In studies evaluating the effects of various factors on PP effectiveness, it is frequently implied that pairs considered to be more effective work differently from ineffective pairs (e.g. [16, 24]). If this hypothesis holds and this intuition is quantifiable, I will be able to learn a classifier from the log data vectors to predict (with some degree of reliability) the evaluation ratings from the partnership — a partnership's observable behaviors in the tool will predict their subjective experience with the task.

Additionally, I will also be able to create a classifier to predict (with reasonable reliability) whether the partnership successfully completes the final level of the task.

> **Hypothesis 2:** Reported positive experiences, completeness/correctness of work can be influenced by a small amount of education on pair programming procedures.

The University of Wisconsin–Madison's current practice is, informally, that students are expected to pair program with no introduction to the theory behind Pair Programming or even much of the best practices corpus. If Hypothesis 2 holds, dividing subjects by their randomly-assigned briefing level will result in significant differences in the self-reported evaluations and/or the partnership's success on the final challenge level.

> **Hypothesis 3:** Reported happiness is a meaningful predictor of complete, correct work.

While I define "successful" pairs as both reporting high satisfaction levels and producing correct and complete code, I hypothesize that the two are related. If this holds, I will be able to learn a classifier using only evaluation data to predict solutions within the tool with some degree of reliability.

4.1 Method

4.1.1 Participants

Participants in this investigation were students in CS 301, an introductory programming course for students not intending to major in CS and/or students with no prior programming experience. During the studied semester, the course was taught in

Python by two instructors. Class size ranged from 115 to 140 students per section with 3 sections taught during the studied semester.

The semester examined in this study included 193 partnerships in which at least one student consented to their data being retained. The plurality (∼50%) of students in the course were in the College of Engineering, with the remainder of students distributed across the College of Letters and Science (∼30%), the School of Business (∼8%), and other colleges. Students are primarily sophomores and juniors, with some graduate students (∼8%) and international exchange (study abroad) students.

All participants were required to complete the activity for course credit, which was granted for completion of the activity and did not take into account quality of work or time spent on the activity. Data was retained based on student consent, which was obtained prior to the activity beginning; all data in the tool was retained completely anonymously and no association was made between student identity and participant data. Any non-consenting student's survey data was deleted before analysis was performed, though if their partner consented to the study then partnership log data was retained. If both students withheld consent, log data was also deleted.

As the activity was performed completely anonymously, demographics for consenting participants were not recorded, and there may be underlying confounding factors which we are therefore unable to uncover.

Participation in the study was not compensated, though all students were required to complete the activity for course credit whether they consented to retention of their data for study purposes or not.

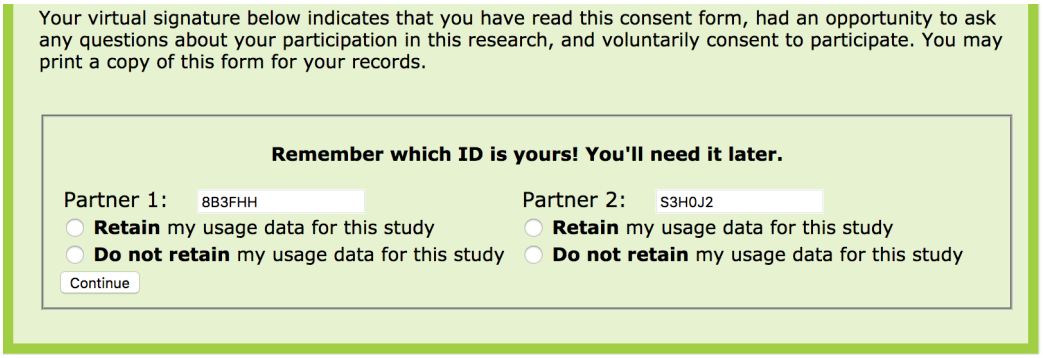


Figure 4.1: Each participant was assigned their own random 6-digit ID, and could consent or withhold consent individually.

4.1.2 Materials

The OKCoder tool created for this study was implemented in Python using the Django framework [51]. Blockly and the maze levels are an open-source project called Blockly Game [39] implemented in JavaScript and modified slightly for the purposes of the study, either cosmetically or for log data retention. For details on transaction log data, see the Data description section below.

The tool itself includes 5 stages:

1. **Study information and consent form.** The IRB consent form[1] was displayed on this page and each participant was assigned a random 6-character alphanumeric ID (Figure 4.1), under which they granted or withheld consent to retain their data for the purposes of the study. Each group of two participants, referred to as a "partnership", was also assigned a random 6-character alphanumeric ID at this time.

2. **Briefing.** Partnerships were randomly assigned one of three levels of briefing: none (Figure 4.2), minimal (Figure 4.3), or complete (Figure 4.4). Partners receiving no briefing were simply told to continue; partners receiving minimal

[1]IRB information is included in Appendix A.

Figure 4.2: The "no briefing" condition.

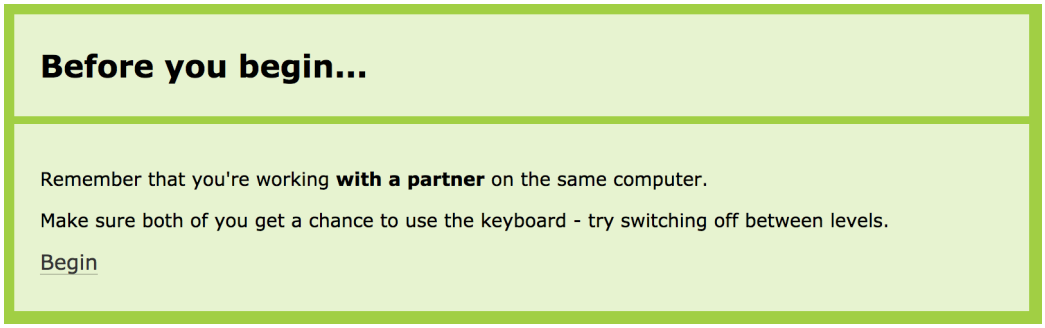


Figure 4.3: The "minimal briefing" condition.

briefing were reminded that they had a partner and should switch control of the keyboard occasionally; partners receiving complete briefing saw a page with definitions of driver and navigator, and some suggestions for best practices in Pair Programming.

3. **Activity.** The coding activity completed for this assignment was in the Blockly block-based programming language [39] and involved "programming" an icon to move through 10 maze stages (e.g. Figure 4.5). Each stage progressively increased in difficulty, introducing concepts like if statements and while loops, with some limitations on number of blocks allowed so coders were not allowed to simply brute force their solutions. The final stage was a challenge problem involving the "left hand rule" for maze solution, and after about 10 minutes offered an option to skip the problem and end early (so students were not required to complete level 10 to get credit for the activity).

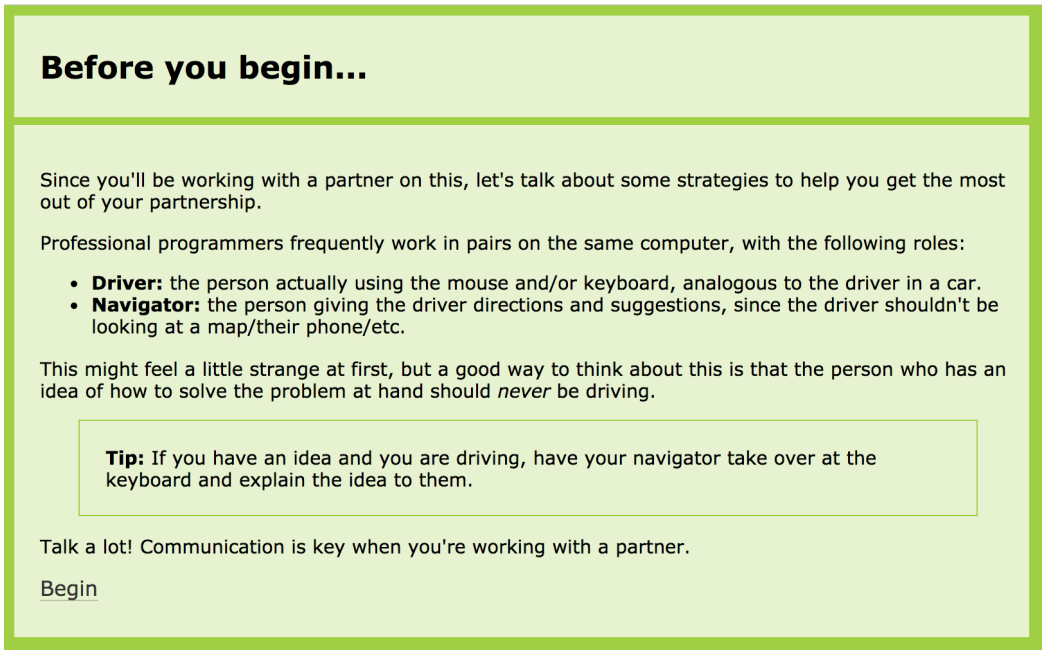


Figure 4.4: The "full briefing" condition.

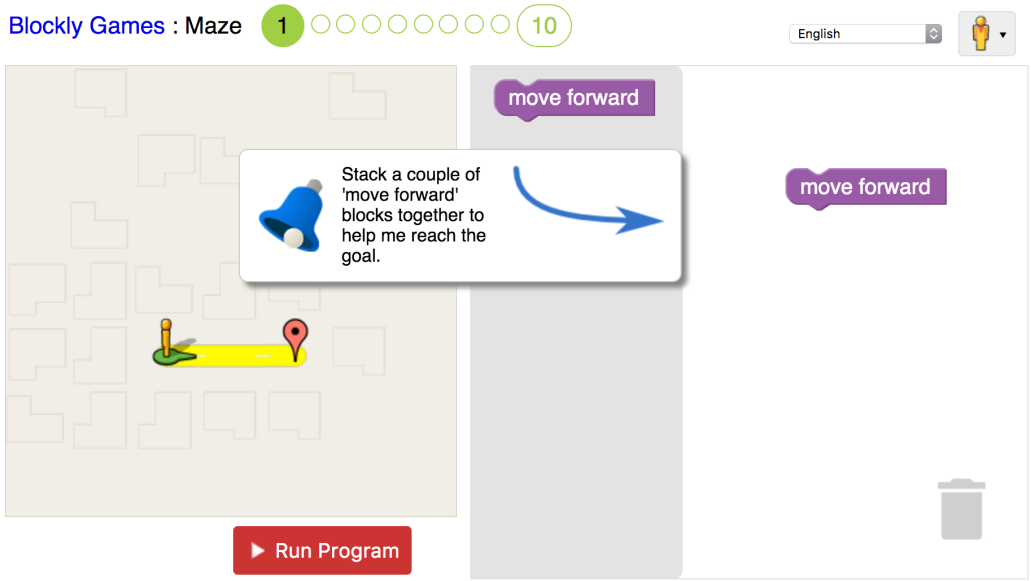


Figure 4.5: Level 1 of the Blockly maze activity.

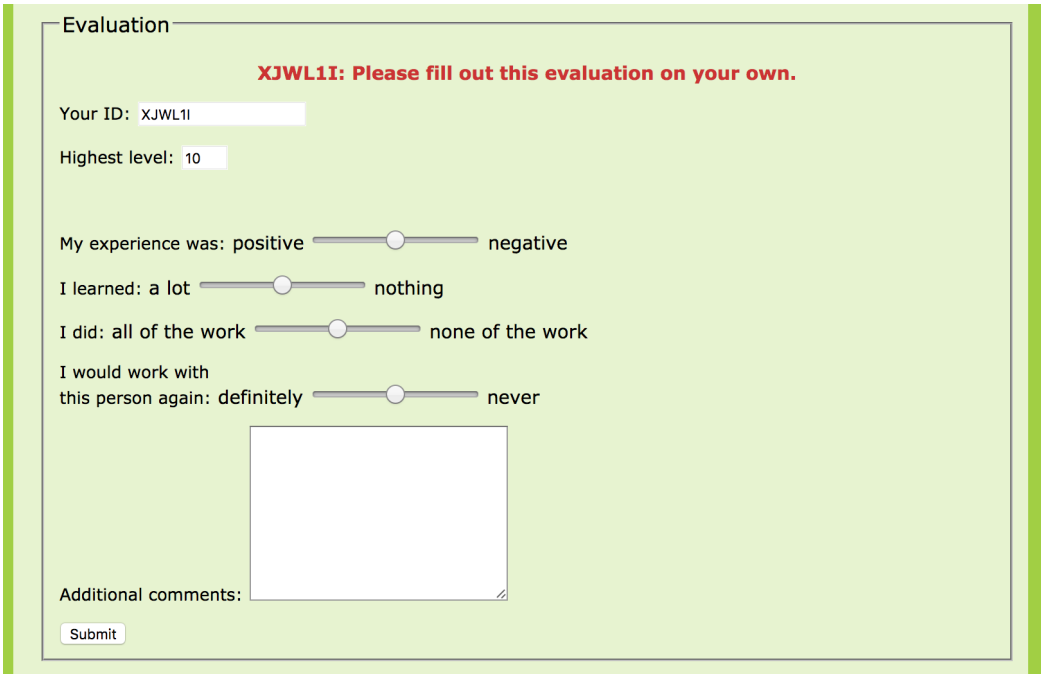


Figure 4.6: Participants each filled out a simple, short evaluation of their experience.

4. **Evaluation.** Each partner was asked to evaluate the activity and partnership using four sliders, reporting their overall enjoyment of the exercise, how much they thought they learned, how much of the work they did, and whether they would work with their partner again in the future[2]. Students were also given a text entry box for any additional comments (Figure 4.6).

5. **Completion code.** Once both evaluations were completed, students were presented with a completion code (Figure 4.7) which they could paste into the course learning management system to receive credit for completing the assignment.

Upon completion of the course, I performed analysis of the transaction log data in Python using the pandas framework [36] and Scikit-learn [40].

[2]While participants were instructed to give their partners privacy while filling out the survey, many students reported anecdotally filling out the surveys together, potentially threatening the validity of these responses. This directly inspired the study detailed in the next chapter.

Figure 4.7: Completion was verified anonymously for the class using a completion code.

Design and coding

The only experimental manipulation completed in the study was a random assignment of briefing levels for stage 2 of the tool. Participants were required to self-select their partners, as this is the mechanism used for partner selection in the course during the rest of the semester. However, for the rest of the semester students were not required to actually have a partner, and many students did elect to work individually (see the study in Chapter 3 for complete analysis of partnered vs individual work over the semester).

Data description

The OKCoder study was primarily designed around unobtrusive observation and data collection in the form of server-side website usage or transaction logs. As discussed at length in Jansen et al. [27], the very act of observation can have an influence on the factors being observed, particularly in the context of website usage. For example, people who are aware they are being recorded in a laboratory setting tend to search for rather less pornography than passively scraped log data suggest they do in a non-laboratory setting. While these studies were not concerned with any habits that were nearly so controversial, there may still be the tendency to want to "perform"

differently as a student if you are being actively video- or audio-recorded, so I wanted to select the least obtrusive method of data collection I could. Students were still informed that their actions on the site were being logged, but were given the option to give or withhold consent for me to retain those logs for analysis, and the logs themselves played no part in their course grade.

Per Jansen et al., building on Holsti [25], we answer the following questions about the log data collected and analyzed for these studies:

1. Which data are analyzed?

Analysis of log data included initially timing and density of interactions with the tool, but expanded to look at the quality and patterns of these interactions.

2. How is this data defined?

Logs for the OKCoder tool take three forms: level logs, event logs, and run logs.

A **level log** is generated any time a level in the maze activity loads, and records the timestamp, level number, and ID of the partnership loading the level. In general, this type of log is generated any time a user loads a programming problem.

An **event log** is generated any time an action is taken in the Blockly workspace. These logs contain the timestamp and partnership as with "level" logs, but also include a high level event type ("ui", "create", "move", "change", or "delete") and a string of characters representing the interaction: the workspace ID, old and/or new coordinates of the block, and the ID and type of block if a block was involved in the event. For example:

```
ws:.8u(Y-/|L,+OIS1,;f4z;
n(34.043025970458984,226.0625);
```

```
bl:lL=rNi7D@,jiyP|4mFnw,maze_if
```

is an information string for a move action in workspace `.8u(Y-/|L,+OISl,;f4z` moving the `if` statement block `lL=rNi7D@,jiyP|4mFnw` to new coordinates (34.04, 226.06).

For the Blockly implementation, "create" logs indicate a block was generated in the workspace (including upon initial load of the workspace). "Move" logs indicate a block was moved within the workspace. "Change" logs indicate a block was modified in some way (for example, the condition in an if statement might be changed). "Delete" logs indicate the user destroyed a block that had previously been in their workspace. "UI" logs indicate that the user interacted with the workspace in some other way, for example to click a button or interact with a menu.

In general, event logs are generated any time a user interacts with the programming problem in some way.

A **run log** is generated any time a user clicks the Run button in the workspace. These logs contain the timestamp, level number, and partnership as with "level" logs, but also include the status of the run represented numerically (1 = success, -1 = failure; 2 = timeout, -2 = error) and a JavaScript representation of the code that was run. In general, this type of log is generated any time a user runs their program within the tool.

3. What is the population from which the researcher has drawn the data?

Participants were students in an undergraduate introductory programming course for non-majors. Details are given above, in Section 4.1.1.

4. What is the context in which the researcher has analyzed the data?

Participants in the studies were allowed to access the tool from anywhere. Many chose to do so from a shared computer lab in the Computer Sciences building on the UW campus, where they were monitored and potentially aided by a course teaching assistant, but this was not a requirement of the task. The task was completed in the third week of the semester, before conditionals or loops had been introduced, but after variables and basic functions were covered and after students had written a few very basic mathematical programs in Python.

5. What are the boundaries of the analysis?

Using a combination of these log data, I am able to determine when levels were loaded, how much time was spent on a level before successful completion (if the level was successfully completed at all), how many actions and what kind were taken. I am not able to determine the level of interaction between partners or any other actions (for example, algorithm planning) that were taken outside of the context of the tool. A long pause followed by a flurry of actions and a successful run might indicate that the user reasoned through the problem on paper and then correctly implemented it; it might indicate that they searched for a solution online and then recreated it; it might indicate that they got up and went to the bathroom and would have been able to correctly implement the solution immediately had they not left the computer – these are not inferences I can draw.

6. What is the target of the inferences?

The target of the inferences in this study is descriptive – by analyzing the patterns in the log data, I create usage profiles and match them with both self-reported outcomes

like a numeric Likert scale rating of the experience and observable outcomes like time to ultimate success (if success is achieved at all).

Data preprocessing

Before beginning analysis of the generated data, all logs underwent preprocessing to ease interpretation. Raw format usage logs are described above; for ease of analysis, the logs were downloaded in CSV files and loaded directly into Python using the pandas package. Individual users were identified by session, using individual and partnership IDs; two individuals using one computer to interact with the site were given one partnership ID, but each individual could choose whether their data should be retained for the study. If at least one individual consented, usage data was retained; any non-consenting partners' evaluations were discarded. If both individuals withheld consent, usage data was also discarded.

Event logs, run logs, and level loading logs were ordered by time and grouped by partnership. To incorporate the idea of "session", the 30-minute timeout suggested in Srivastava et al. [50] was used; any periods of inactivity (i.e. no logs generated) lasting longer than 1800 seconds were considered session breaks and logs were grouped accordingly.

Two partnerships' logs were discarded for reporting less than one second elapsed on the level (time between level load and last recorded event on level); 12 other partnerships' logs were discarded for logging more than 30 minutes of downtime between events (ranging from 2905 seconds/48 minutes up to 168,659 seconds, or 46.8 *hours*).

As the usage logs and evaluations were given unique identifying ID numbers *in situ*, I additionally verified that no log data was replicated in the common situation where both partners indicated consent.

4.1.3 Procedure

Participants were required by their course assignment to complete the activity in pairs. These pairs were self-selected with very little intervention from instructors or TAs except when a student could not find a partner. Students were not allowed to complete the activity with multiple partners, though I am aware of roughly 1-2 cases each semester where this requirement was not followed. Additionally I am aware of approximately 5 cases each semester where students who did not find partners instead opted to complete the activity individually for reduced credit. As the data collected from this tool was completely anonymized and the completion codes were not connected to the partnership IDs in any way, I am unable to designate which "partnership" logs belong to only a single participant rather than a pair.

Participants were introduced to the activity via a brief write-up of the assignment on their course website, which included some brief explanations of the partnering requirements and steps for submitting their completion code for credit:

> **Summary**
>
> This week, you will be practicing your problem solving skills by moving a marker through a maze using programming-style commands. However, rather than writing a program in Python, you'll be using a drag-and-drop language where you won't have to worry about the syntax, just actions and the order in which you're taking them.
>
> **Program Requirements**
>
> To complete this week's assignment, you must:
>
> 1. Find a partner and sit down next to them.
> 2. Complete the 10 mazes on the site linked below.
> 3. Complete the partner evaluations.
> 4. Copy and paste the completion code into the text entry box for your P2 partner group.
>
> I'll explain each step a little more below:

1. Find a partner

You can work with anyone in the class, but everyone must work with a partner! If you don't know someone to work with in the class, I recommend attending lab and seeing who else needs a partner.

Not only must you work with a partner, you must work with them in person - so attending lab is a really good idea this week.

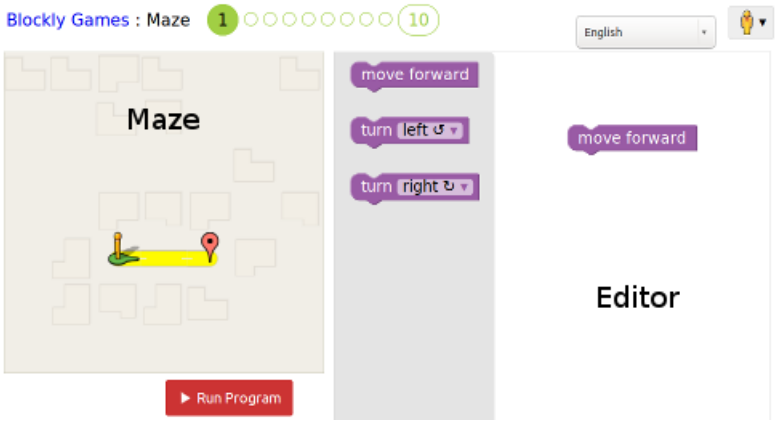
Both partners should submit a copy of the completion code.

Both partners should fill out the partner survey (takes ∼5 min)

2. Complete the mazes

When you go to the site, you'll first see a wordy explanation of research - MY research! If you let me use your (completely anonymized) data, you'll be helping not only to make this site better but also helping me learn what makes good partner programmers. Please let me use your data logs, it would make me happy (and will not affect your grade in any way).

Once you consent (or not), you'll be taken to a series of maze levels:



On the left is the maze (if you don't like the Google Maps aesthetic, the menu in the top right has a couple of other themes you can pick) - this is where you'll see the output of your program. This is a graphical version of our repl.it console.

On the right is your editor - unlike the repl.it editor, you won't type here. You'll just drag and drop those blocks from the middle section into a short program to tell the maze marker how to move.

Note: eventually you'll start seeing blocks with spaces for other blocks inside of them. You can fit as many blocks as you want in there - they'll keep expanding!

3. Complete the evaluations

When you finish level 10, there will be a couple of short evaluations for you to complete about your partner. Each of you will fill out ONE of these. Please be honest!

4. Copy the completion code

After you complete everything - all the mazes and both evaluations - you'll be given a randomly-generated completion code. This is unique to your partner and you, but isn't at all tied to your activity on the site. Each code may only be redeemed by one pair for a grade on this assignment!

All participants were allowed a one-week period to complete the activity on their own time. The environment in which they completed it was not controlled in any way, to provide logs that would more authentically mimic those observable in the final version of the tool.

4.2 Results

Data analysis for this study follows [50], a three-step process: data pre-processing, pattern discovery, and pattern analysis. Pre-processing is described in Section 4.1.2. Data analysis was performed in two stages: the first examined level 9 (the most difficult training level, though almost all students were still able to complete it), and the second focused on level 10, the challenge level.

Activity on level 10 can be categorized as successful or unsuccessful, based on whether the partnership ultimately found a solution to the level. Successfully completing level 10 was not required for credit, and was only accomplished by 56.8% of partnerships.

4.2.1 Pattern discovery

The only experimental intervention performed in the study was a randomly assigned pre-task briefing (see Section 4.1.2 for details). A one-way between-subjects ANOVA was conducted to compare the effect of this briefing on success rates in the no-briefing, minimal-briefing, and full-briefing conditions. This briefing level did not have a significant effect on success rates at the $p = 0.05$ level for the three conditions ($F(2, 191) = .915$, $p = 0.402$).

Additionally, a one-way between-subjects ANOVA was conducted to compare the effect of briefing on the post-study evaluation ratings across the briefing conditions, but the briefing levels did *not* have a significant effect on any of the evaluation ratings (overall experience rating, likelihood to work with partner again, or perceived division of workload) at the $p = 0.05$ level for the three conditions (three separate univariate ANOVAs had $F(2, 378) = 1.614$, $p = 0.201$; $F(2, 378) = 1.626$, $p = 0.198$; $F(2, 378) = .249$, $p = 0.780$ respectively).

The remaining analysis disregards briefing level as a distinguishing characteristic of partnerships.

On level 9, 14 partnerships did not attempt to run a solution on the level but instead proceeded directly to level 10 (possible through navigation buttons on the top bar). The 6 partners who skipped level 9 without generating any event logs at all were all successful on level 10, including 3 who solved the level in a single attempt, spending less than 7 seconds total on level 10.

Times for level 10, broken out by level 9 activity and level 10 success, are listed in Table 4.1 and shown in Figure 4.8. A two-tailed t-test analysis showed that skipping level 9 was not associated with time spent on level 10 for either successful ($p = 0.11$) or unsuccessful ($p = 0.67$) partnerships.

Attempted 9?	Successful 10?	N	Mean (sec)	SD
No	**Yes** (Group 1)	7	407.4	750.6
	No (Group 2)	7	957.4	659.1
Yes	**Yes** (Group 3)	98	2278.8	2208.5
	No (Group 4)	69	1109.2	903.0

Table 4.1: Mean and standard deviation of time spent on level 10, broken out by whether level 9 was attempted and whether level 10 was ultimately successful.

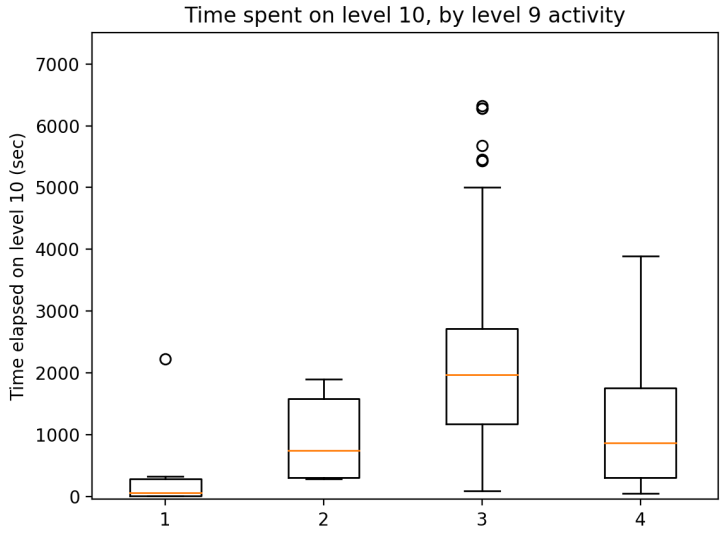


Figure 4.8: Time spent on level 10. Groups 1 and 2 did not attempt level 9; group 1 was successful at level 10 and group 2 was not. Groups 3 and 4 attempted level 9 at least once; group 3 was successful at level 10 and group 4 was not.

Of the 167 partnerships who attempted to run a solution on level 9, 159 were ultimately successful.

All but one partnership attempted to run a solution on level 10 at least once; of the 193 partnerships, 109 ultimately discovered solutions to the level.

It took successful partners an average of 2767.53 seconds (or 46.13 minutes, $SD =$ 3994.03 sec) and 22.4 runs ($SD = 28.0$) to find a solution to level 10; discarding the longest times as outliers (more than five hours), unsuccessful partners made their

Evaluation measure	Correlation coefficient (R)		
	Level 9 events	**Level 10 events**	**Level 10 success**
Overall experience	-0.006	0.057	-0.199
Amount learned	-0.113	-0.070	-0.168
Workload proportion	-0.157	-0.043	-0.088
Work together again	0.022	-0.070	-0.108

Table 4.2: Correlation coefficients between each of the each of the evaluation metrics and each of the number of events logged on level 9, level 10, and success status of level 10.

last run attempt after an average of 3127.22 seconds (52.12 minutes, $SD = 10644.77$ sec) and 19.6 runs ($SD = 23.4$). Successful partners logged an average of 42.7 events ($SD = 17.1$) for each run attempt; unsuccessful partners logged an average of 40.7 events ($SD = 15.1$).

Correlation and variance

Correlation between the post-activity evaluation results and several factors, including the number of events logged on level 9, level 10, and success/failure on level 10 was calculated using NumPy. Lower numbers on the evaluation measures were "better" — for example, a score of 0 on overall experience means "very positive" where a score of 10 means "very negative". Negative correlations indicate that "better" evaluations were associated with more events (or success, where success=1 and failure=0).

All four numeric evaluation ratings were effectively uncorrelated to any of these other measures (see Table 4.2 for details).

Classification

For classification analysis, partnerships were represented by one of two types of vectors containing summary information about their activity in level 10, either a magnitude vector or a scaled vector.

Magnitude vectors contained:

1. total number of runs
2. total number of events
3. maximum number of events between run attempts
4. number of events before first run attempt
5. the total number of each type of event
6. total time spent on level (seconds)
7. time before first run attempt (seconds)
8. maximum duration of inactivity between events (seconds)
9. success label (1=success, 0=failure)

Scaled vectors contained:

1. ratio of time elapsed on level 10 to number of events
2. percent of time spent on level before first run attempt
3. percent of events before first run attempt
4. percent of events of each type
5. success label (1=success, 0=failure)

I trained two SVM classifiers from Scikit-learn [40] with a Gaussian radial basis function kernel (a decaying function of the distance between the two vectors) and a simple linear kernel on the vectors to determine whether their success or failure could be predicted based on these descriptive components.

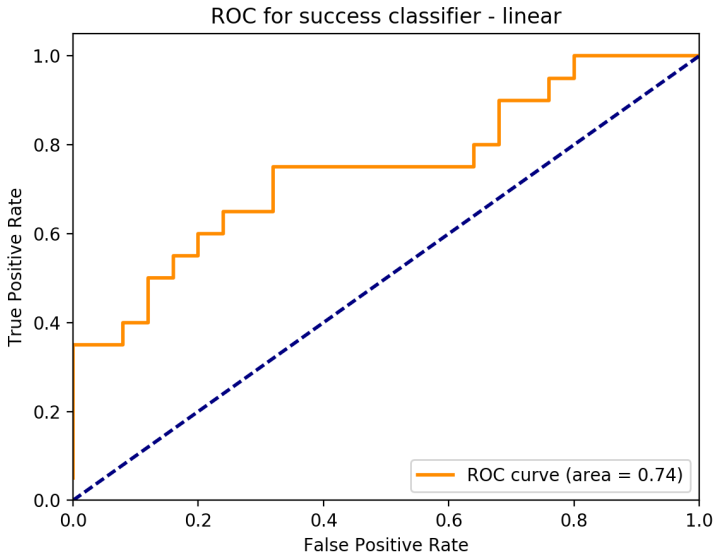


Figure 4.9: ROC for magnitude data using a linear kernel

For the linear classifier, I created receiver operating characteristic (ROC) curves using both the magnitude vectors (Figure 4.9) and the scaled, relative vectors (Figure 4.10). Both curves indicated a better-than-random success prediction. For the radial basis function, only the scaled vectors indicated a better-than-random success prediction (Figure 4.11).

4.3 Discussion

This study originally proposed three hypotheses:

1. "Successful" partnerships are predicted by specific features of the data.

2. Reported positive experiences, completeness/correctness of work can be influenced by a small amount of education on pair programming procedures.

3. Reported happiness is a meaningful predictor of complete, correct work.

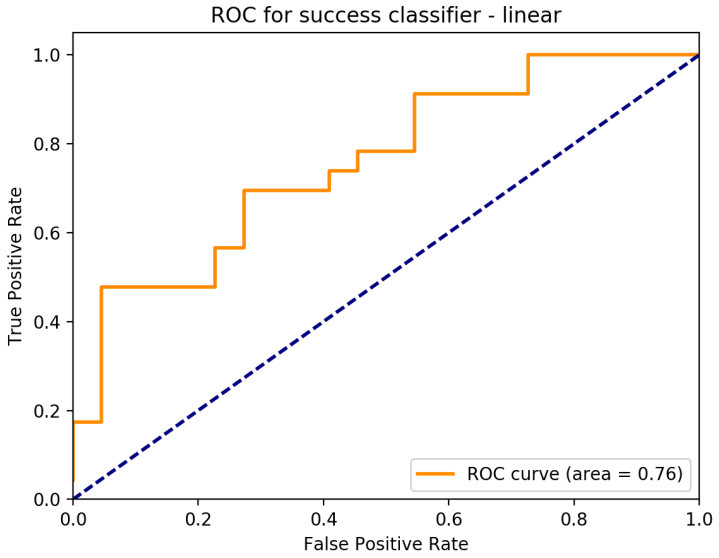


Figure 4.10: ROC for relative data using a linear kernel

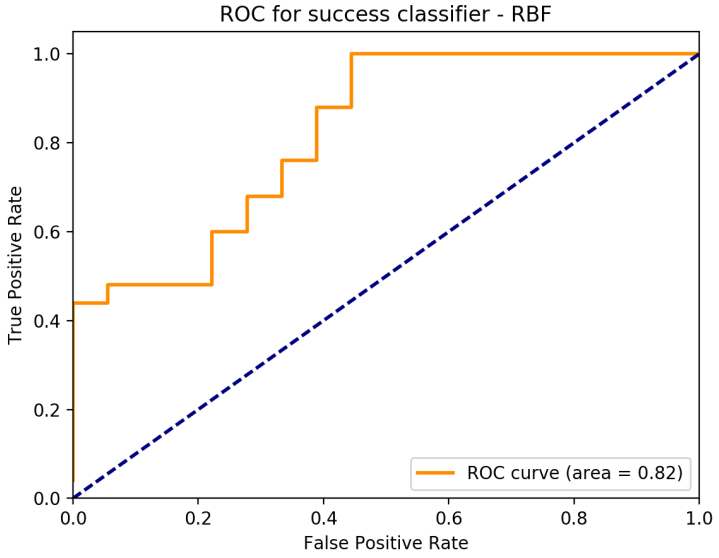


Figure 4.11: ROC for relative data using a radial basis function kernel

Of these three hypotheses, only the first was borne out by the data: the classifiers on level 10 success based on both magnitude and scaled data showed promising levels of predictive ability (see section 4.2.1).

Unfortunately, the other two hypotheses regarding education and self-reported measures were not supported by the data. There was no significant difference between briefing groups for any of the investigated measures, and the self-reported measures showed no correlation with more objective measures of success in the task (see section 4.3.1 below for a discussion of possible threats to the validity of these results).

This study begins to demonstrate the utility of the data provided by web application transaction logs, compared to the data gathered in traditional, more-intrusive observational methods for pair programming analysis — indeed the only positive results arose from the passively-gathered observational data rather than from the evaluation results. As described in Section 4, most pair programming studies rely on video and/or audio recordings as well as in-person interviews with the subjects; the study detailed in this chapter did inform subjects that they were participating in a study but recorded only their interactions with the tool (and to a much lesser extent, the responses provided in a short, four-question survey).

Pair dynamics may change when partners are aware that they are being observed [27], but it is unethical for researchers to pretend they are not observing. As the methods used here are no different from the low-level observation present in most web-based activities, I argue that the behaviors recorded by these methods are more authentic than those performed for researchers during more intrusive studies. Most programmers, even novice programmers, are used to having their web-based activities

observed[3] and so they are less likely to change their behavior even when told that their activity in the tool is recorded.

Log-based observation requires additional implementation time before deployment, where video/audio observations and interviews can be set up very quickly. Once deployed, however, a log-based observation requires no additional researcher time before data analysis, allowing large-scale studies to be performed efficiently by a single researcher.

When evaluating a pair of programmers for their efficacy, metrics are broadly categorized into technical productivity, program/design quality, academic performance, and satisfaction [45]. While the satisfaction metric is generally confined to directly interacting with a programmer (and academic performance is outside the scope of this study), technical productivity and program quality can be gleaned directly from usage log data. The majority of this study was focused on technical productivity as a metric for pair effectiveness, and explored many measures and methods for quantifying the measure from the data.

Per the direction of the literature on collaborative learning (e.g. Dillenbourg and Schneider [17]), interesting collaborations arise when the collaborative task is sufficiently complex that conflict may arise between the collaborators and learning can therefore occur. While I do not measure learning in this study, it is illustrative that by changing from analysis of the final training task (level 9) to the challenge task (level 10), patterns began to arise in the data, suggesting that the final training task was still too simplistic for real analysis. This in itself stresses the importance of nontrivial tasks – even frustrating ones – for differentiating student strategies in the learning process.

[3]There are running jokes on the internet about, for example, following up your less-savory online activities by looking at pictures of kittens purely for the mental health and well-being of the NSA agent monitoring your browser history.

4.3.1 Threats to validity

There are several potential threats to the validity of this study, some of which directly influenced the direction of the previous study (Chapter 3).

For the second hypothesis in this study, that a minimal amount of education provided immediately before the task could significantly influence the outcomes of the task itself, there exists a very real confounding factor in that all students in the course were given a lecture on pair programming best practices before attempting the task, and also participated in a pair programming activity during that same lecture.

It would be highly instructive to compare the data from this study with data from a course where students did not receive in-class pair programming instruction, or at least did not receive that instruction until after completing this activity. As it stands, we did observe a small difference in successful completion rates across the briefing levels that suggested that more education led to a higher completion rate, but the differences between groups were not large enough to be significant and could have been due to random chance.

The first and third hypotheses in this study included the self-reported measures from post-activity evaluations. These evaluations included sliding scale ratings from 0-10 for overall impression of the activity, amount learned, percent of the workload done, and likelihood to work with the partner again. I chose to incorporate these evaluations into the activity to increase participation, but unfortunately the reality of the situation meant that most participants filled out the evaluation with their partner watching. This could introduce the very real possibility that partners were rated more highly than they would have been if the evaluations had been truly private.

Additionally, the scope of the post-activity evaluation was somewhat limited, and the wording of questions allows for potential misinterpretation. For example, asking

participants to rate the overall experience as positive or negative may or may not have included information about their partner, and may have been more reflective of their feelings about the maze task itself.

Chapter 5

Uncovering working style using automated data collection

In Chapter 4, vectors describing both the magnitude and relative characteristics of user activity within the OKCoder tool were meaningful predictors of a partnership's correctness and completion of the level 10 challenge. This leads to the research question for this study:

> **Research question:** Can programmers' **working styles** be differentially quantified using unobtrusively collected data?

This post hoc study will use the same data detailed in Chapter 4, looking more deeply at individual features of the data and their chronology in an attempt to uncover exactly which components of these vectors are important, and at whether partnerships that will ultimately be successful and those that will not can be differentiated from this data.

> **Hypothesis 1:** Transaction log data will demonstrate some patterns indicating distinct working styles.

The concept of working style is not new to programming (e.g. [44]), though it is either left un-quantified, discussed in study-specific terms like "copy and paste" vs. "write as you go" [20], or defined by the degree of presence of expert strategies like "beacon" searching [44]. If the concept of working styles is valid in general, which its pervasiveness in the underlying intuition of study design would suggest, I anticipate that patterns relating to distinct working styles of some type will be discernible from the transaction log data using machine learning methods.

5.1 Method

The data used in this study was collected in the same investigation as the previous chapter; information on participants can be found in section 4.1.1, materials can be found in section 4.1.2, procedure can be found in section 4.1.3.

Incorporating the success indicator value into the partnership vectors, I created a set of "working style" clusters using both the magnitude and scaled data and the Birch clustering implementation in Scikit-learn [40]. Birch (Branched Iterative Reducing and Clustering using Hierarchies, [61]) is a deterministic hierarchical clustering algorithm for large datasets.

5.2 Results

As with most clustering algorithms, the number of clusters is provided initially to the Birch algorithm. The first step in clustering pattern discovery was to determine the most appropriate number of clusters. Comparing the within-clusters sum of squares across cluster numbers from 2 to 8 using the elbow method [30], informative numbers of clusters appear to be either 3 and either 6 or 7 for the magnitude data and 5 for

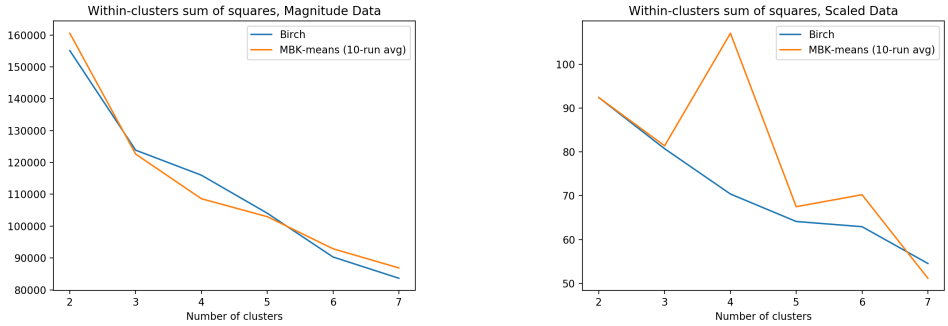


Figure 5.1: Within-clusters sum of squares for clustering with both types of vectors

the scaled data (Figure 5.1). This section will compare results between these sets of clusters.

Magnitudes: Three clusters

Cross-algorithm validation of the Birch-created clusters using the mini-batch k-means (MBK) clustering algorithm mostly preserved the three clusters; 84% of points remained in cluster 3.0, 95% of points remained in cluster 3.1, and 94% of points remained in cluster 3.2. Of the points which were not preserved as a part of their Birch cluster, four points from each of clusters 3.0 and 3.1 were clustered with cluster 3.2, and five points from cluster 3.2 were assigned to cluster 3.1. The within-cluster sum of squares for the Birch algorithm was 123,848.48, and for mini-batch k-means it was 122,044.76.

A one-way between-subjects ANOVA was conducted to compare the Birch and mini-batch k-means clusters' differences across the features in the magnitude vectors[1] in the three-cluster condition. The only feature in the vectors in which the clusters

[1]Magnitude vector features were: number of runs, number of events, events before first run, maximum number of events before a run, number of UI-type events, number of change-type events, number of create-type events, number of delete-type events, number of move-type events, total time elapsed on level, time elapsed before first run, maximum time between events, success/failure, time-to-events ratio.

did *not* differ significantly at the $p = 0.05$ level was the number of events before the first run attempt ($F(2, 177) = 1.478$, $p = 0.231$).

Decreasing the confidence threshold to $p = 0.005$, the elapsed time before the first run attempt $F(2, 177) = 4.633$, $p = 0.010$) and the maximum timeout $F(2, 177) = 5.172$, $p = 0.007$) lose their significance, but the variance in all other features remains significant. Full statistics across all clusters and features can be found in Appendix B.

For a graphical representation of the three-cluster characteristics, see Figure 5.2.

Cluster 3.0 (Birch $n = 25$) successfully solved the level 76.0% of the time. They ran the level the most frequently, and logged the most events on average.

Cluster 3.1 (Birch $n = 76$) successfully solved the level 43.4% of the time. They performed the fewest runs and logged the fewest events on average.

Cluster 3.2 (Birch $n = 78$) successfully solved the level 66.7% of the time. Their average runs, events, and time spent on level were between those of the other two clusters.

Magnitudes: Seven clusters

Cross-algorithm validation using MBK remained mostly faithful to the Birch clusters; cluster 1 retained 95% of its points, cluster 3 retained 91%, cluster 4 retained 81%, cluster 5 retained 100%. Cluster 0 was split into two clusters of 2 and 3 points, and clusters 2 and 6 combined 75% of their points to form a single cluster. The within-cluster sum of squares for the Birch algorithm was 83630.75, and for MBK it was 89112.09.

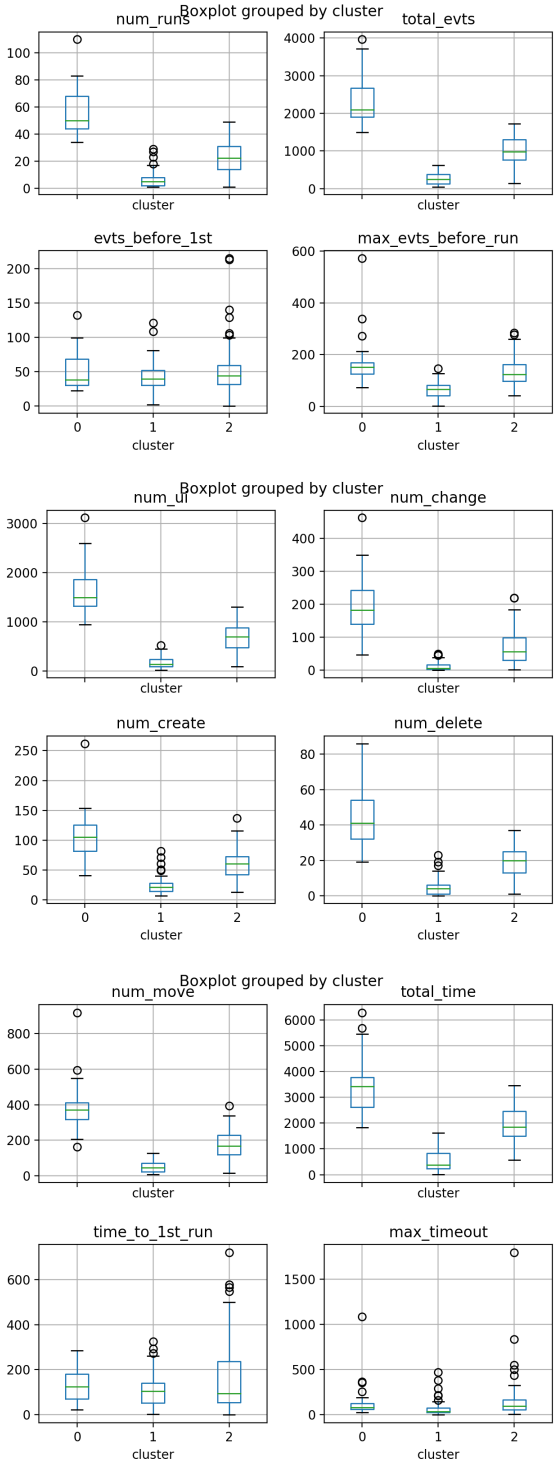


Figure 5.2: Boxplots of magnitude characteristics for three-cluster configuration.

A one-way between-subjects ANOVA was conducted to compare the Birch and mini-batch k-means clusters' differences across the features in the magnitude vectors[2] in the seven-cluster condition. The only features in the vectors in which the clusters did *not* differ significantly at the $p = 0.05$ level were (as in the three-cluster condition) the number of events before the first run attempt ($F(2, 177) = 1.478$, $p = 0.231$), and additionally the time elapsed between level load and the first run attempt ($F(2, 177) = 1.62$, $p = 0.14$). Full statistics across all clusters and features can be found in Appendix B.

For a graphical representation of all seven-cluster characteristics, see Figure 5.3.

In relation to the three-cluster configuration, the Birch algorithm split cluster 3.0 (the slowest cluster) into clusters 7.0 and 7.1.

Cluster 7.0 (Birch $n = 5$) successfully solved the level 100% of the time. This cluster took the longest by far (averaging 5569.88 seconds on level, compared to the overall mean of 1582.79 seconds and even cluster 7.1's mean of 3016.91 seconds) but actually had a very similar average number of runs to cluster 7.1 (54.80 runs for cluster 7.0, vs 57.15 for cluster 7.1).

Cluster 7.1 (Birch $n = 20$) successfully solved the level 70.0% of the time. While they spent the second-longest time on level after cluster 7.0, they had a shorter average time between events (maximum time elapsed between events for cluster 7.1 was an average of 133.84 seconds, compared to cluster 7.0's mean of 212.77 seconds) as well as slightly fewer overall events (mean 2283.85 events vs cluster 7.0 mean of 2678.20 events).

[2]Magnitude vector features were: number of runs, number of events, events before first run, maximum number of events before a run, number of UI-type events, number of change-type events, number of create-type events, number of delete-type events, number of move-type events, total time elapsed on level, time elapsed before first run, maximum time between events, success/failure, time-to-events ratio.

Cluster 3.1, the fastest cluster with respect to time, runs, and events, became clusters 7.3 and 7.5.

Cluster 7.3 (Birch $n = 33$) successfully solved the level 57.6% of the time. This cluster spent an average of 981.69 seconds on the level (or about 16.4 minutes) and ran their code an average of 9.2 times.

Cluster 7.5 (Birch $n = 43$) successfully solved the level 32.6% of the time. On average this cluster spent only 244.67 seconds (or about 4 minutes) on the level and ran their code an average of 3.6 times. Of particular interest is the fact that the 14 pairs in this cluster who did successfully solve the level ran their code an average of 1.6 times and only spent 196.87 seconds (3.28 minutes) on the level; the 29 unsuccessful pairs tried their code an average of 4.6 times and spent 267.74 seconds (4.46 minutes) on the level, even though the option to skip the level was not presented until 10 minutes had elapsed.

Cluster 3.2, the "not too fast and not too slow" cluster, became clusters 7.2, 7.4, and 7.6.

Cluster 7.2 (Birch $n = 17$) successfully solved the level 88.2% of the time. Of these final three clusters, they generally had the lowest mean values for every feature except time spent, both in terms of total time (2680.32 seconds) and as a consequence maximum time between events (an average of 302.63 seconds).

Cluster 7.4 (Birch $n = 42$) successfully solved the level 52.4% of the time, averaging 1487.92 seconds spent on the level with a maximum average downtime of 101.68 seconds. For all other features, their averages landed between those of clusters 7.2 and 7.6.

Cluster 7.6 (Birch $n = 19$) successfully solved the level 78.9% of the time, averaging 2324.06 seconds spent on the level and a maximum average downtime of

124.88 seconds. For all other features, they had the highest averages of the cluster 3.2 subgroups.

With the increase in granularity of the clustering, the agreement between the MBK and Birch algorithms decreased, though similar themes emerged.

With the high significance values of magnitudes related to time and activity count on levels, there is a concern that the groupings are taking into account only the amount of engagement the student has with the tool rather than other qualities of that engagement. This could be mitigated by replacing the summary vectors with scaled values, which I will explore in the next section.

Scaled values: Seven clusters

Scaled vector clustering was much less stable when compared to the corresponding MBK clusters (see Figure 5.1). The most appropriate number of clusters for this data was much less apparent; while the difference between WSS for five and six clusters was minimal, it decreased sharply between six and seven clusters.

A one-way between-subjects ANOVA was conducted to compare the Birch clusters' differences across the features in the scaled vectors[3] in the seven-cluster condition. The only feature in the vectors in which the clusters did *not* differ significantly at the $p = 0.05$ level was the percent of events which were "change" events ($F(2, 177) = 1.23$, $p = 0.293$). Full statistics across all clusters and features can be found in Appendix C.

Birch is a hierarchical clustering algorithm, so decreasing the number of clusters will merge the next closest clusters in its hierarchy. Of the four clusters in the 7-

[3]Scaled vector features were: time-to-events ratio, percent of time elapsed before first run attempt, percent of UI-type events, percent of change-type events, percent of create-type events, percent of delete-type events, percent of move-type events, percent of events occurring before the first run attempt.

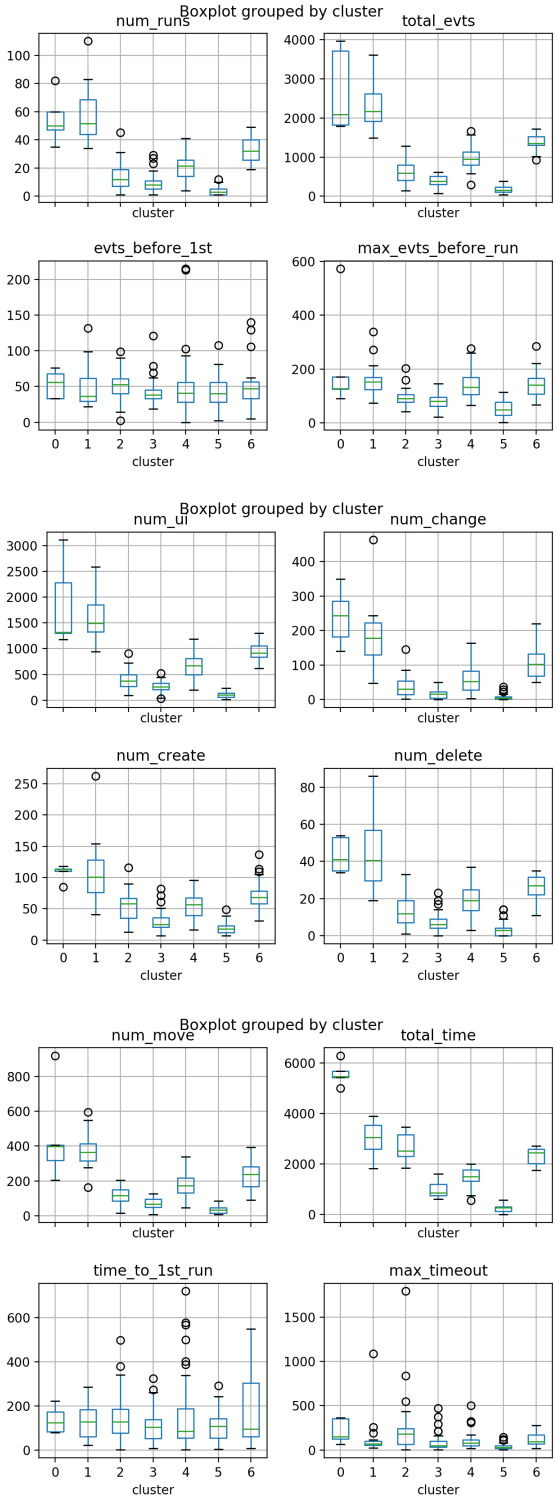


Figure 5.3: Boxplots of magnitude characteristics for seven-cluster configuration.

cluster configuration which contain only a single partnership, two are preserved all the way down to a 3-cluster configuration.

5.3 Discussion

This study's research question, how and when in the programming process does working style become differentially relevant, was addressed in the form of two hypotheses.

The first hypothesis, that transaction log data will demonstrate some patterns indicating distinct working styles, was strongly demonstrated in the clustering analysis. Using either magnitude or scaled summary data produced clusters wherein the differences in the measured features between clusters were nearly all significant with extremely small p values. In fact, the only features which were not significantly different across clusters of partnerships were the time elapsed before attempting a run for the first time, the ratio of time to events, and the percent of events which involved changes to the blocks. All other features were significant across clusters.

The second hypothesis, that successful partners find solutions in observably different ways than unsuccessful partners, was also borne out by the data. Successful partners moved significantly farther through the solution space (as defined by our metric) than unsuccessful partners, with a couple of potential implications: either successful partners are more likely to start small and test code incrementally, or they are more willing to make large changes to their code and not get stuck in one "place" in the graph.

That these differentiations can be made entirely with the observational data collected from the tool and require no subjective input from the participants is another point in favor of our assertion from the previous study: that non-intrusive methods

of partnership assessment have value in terms of evaluating potential partners and creating partnerships.

5.3.1 Impacts and future work

This clustering of student transaction logs reveals some interesting groupings that could be called a quantifiable working style, but not necessarily in the same vein as human-described "working styles". The concept of working style is not new to programming (e.g. [44]), though it is either left un-quantified, discussed in study-specific terms like "copy and paste" vs. "write as you go" [20], or defined by the degree of presence of expert strategies like "beacon" searching [44]. While it may be possible to align these descriptors with characteristics like time before first attempt or ratios of events to runs, like many applications of machine learning to pattern recognition scenarios there may be novel insights for exploration in relation to working style.

For ease of discussion I focus on the characteristics of the three-cluster magnitude clusters in relation to pedagogy. Of the 179 students studied, 25 were in cluster 3.0, 76 were in cluster 3.1, and 78 were in cluster 3.2, and the clusters exhibited strikingly different characteristics:

Cluster 3.0 was the most consistently successful, but spent the most time and effort on task. Students in this cluster may not understand the material when they begin, but are willing to persist at the task and use experimentation to find a solution.

Cluster 3.1 was successful less than half of the time and spent the least time and effort on task. These students get it or they don't; if they get it, great, but if they don't then it's not worth their time to figure it out. I would anticipate many of the students who are only taking an introductory programming course "because it's required" fall into this cluster.

Cluster 3.2 was successful 2/3 of the time, and settled solidly between the other two clusters on most other metrics. These students are likely interested in the material (as evidenced by their success rate), but have not yet mastered the skill. Interestingly, the maximum timeout and time before first run for this cluster was slightly higher than the other two, suggesting that students in this cluster may spend more time thinking through the problem offline without employing the guess-and-check strategy of cluster 3.0.

From an educator's standpoint, it would be useful to classify students into one of these three groups for more targeted pedagogy. The experimenters in cluster 3.0 might benefit from an open-lab style course, where they're given a problem and allowed to experiment toward a solution at their own pace. The students in cluster 3.2 could be better-served by a more traditionally-taught course demonstrating logical strategies and how they translate into programming languages. And particularly the "because it's required" students or students who already possess programming skills could be targeted for a faster-paced practical course, with hacks, tips and tricks, and a focus on utility skills that could complement their existing knowledge or other priorities.

In relation to partner pairings, it would be extremely beneficial to students to match within these clusters. Anecdotally instructors have suggested matching students based on self-reported measures of goals within the course; matching a student who says they are aiming for an A in the course with a student who is just trying not to fail is a recipe for conflict and discontent. However, as with all self-reports students can misrepresent their own goals, whether intentionally or unintentionally. Taking the onus of interpretation off the student and relegating it to machine learning has the potential for improved accuracy while not forcing students to give themselves potentially self-fulfilling labels.

In future work, I would like to replicate these results with another group of students to see whether the characteristics of clusters change or remain relatively constant; additionally, a comparison of paired characteristics to individual characteristics over a series of experiments might prove useful for the eventual pair matching tool.

Chapter 6

Graph-based analysis of student programming behavior

In chapter 5 I described various working styles as characterized by various properties of the recovered website transaction logs from student sessions in a web application completing a maze activity by using a block-based programming tool. Details of the experiment that produced these data are given in chapter 4.

Among the features of working style explored in the previous chapter were the relative rates of success between groups of students with similar working styles. I showed that this feature was significantly different across all variations of clustering, regardless of whether the data was represented with by magnitude or relative frequencies: it appeared that there were clusters of students who were simply more likely to give up.

Based on the observation that working style and success can be related, I turned to a technique used in MMO gameplay analysis to characterize the behavior of players likely to be "successful" in a game (that is, to continue playing and paying for the

service) versus that of players likely to abandon the game: a graph-based analysis of progression through game states [1].

This leads to the research question for this study:

Research question: Which paths through the space of possible code solutions characterize success?

Based on the findings of the previous study, where in general successful partners were those who spent more time on the level, I do not expect to find many instances of unsuccessful pairs going down "rabbit holes". Rather, I hypothesize that successful pairs simply continued to debug the unsuccessful code that caused other pairs to give up – that such "terminal nodes" of unsuccessful partner paths will in fact be contained in the paths of successful partners.

Hypothesis 1: Successful and unsuccessful paths will largely overlap.

Uncovering usage patterns and characteristics is an active field of study in video game design [55]. Users with similar play patterns may become stuck in similar ways, leading to decreased engagement or ultimately giving up on the game (which is highly undesirable to developers of subscription-model online games), and various methods for characterizing and studying the paths which users take through games have been developed.

Hypothesis 2: Successful programmers have observably different behaviors from unsuccessful programmers.

It is worth noting that both hypotheses may hold in this study; an unsuccessful programmer may attempt the same code as a successful programmer but get stuck in a particular state, or give up entirely without continuing along a path that ultimately leads to a successful solution.

6.1 Method

The data used in this study was collected in the same investigation as the previous chapters; information on participants can be found in section 4.1.1, materials can be found in section 4.1.2, procedure can be found in section 4.1.3.

The method of graph construction and analysis used in this study is heavily influenced by that described by Wallner for gameplay analysis in [54].

6.1.1 Graph construction

For this analysis, I focused on the logged versions of the code that was run each time a partnership pressed the Run button in the Blockly tool. The block-based code was automatically translated to JavaScript by Blockly before being passed to the tool's interpreter. Because the JavaScript was auto-generated, no processing needed to be done to reconcile variable names or indentation differences between users.

Before processing, the logged JavaScript code included randomly-generated unique identifiers for each of the blocks created in the tool space (Figure 6.1).

```
if (isPathLeft('block_id_^MY2uNtG**U|j8.CA*WO')) {
    turnLeft('block_id_H1B533{ZlxSu[9UnjkiN');
} else {
    if (isPathForward('block_id_0w|E_[%iIzCJ,puf,5!k')) {
        moveForward('block_id_}[j-}r^#|JumP^GC`D!O');
    } else {
        turnRight('block_id_lExLn71m^,7EMnU=aHqF');
    }
}
```

Figure 6.1: Auto-generated JS code as provided by Blockly.

To allow for more direct comparisons between the generated code, it was processed to remove all block ID information and indentations so that the code from Figure 6.1 is reduced to the code depicted in Figure 6.2.

```
if (isPathLeft()) {
turnLeft();
} else {
if (isPathForward()) {
moveForward();
} else {
turnRight();
}
}
```

Figure 6.2: Auto-generated JS code with block IDs removed.

To determine similarity between code, I created a basic program distance metric inspired by plagiarism detection metrics [18], which provides an integer distance between two programs i and j:

$$\begin{aligned} d(i,j) =& (|L_i| - |L_j|) \\ &+ |l \in L_i, \notin L_j| + |l \in L_j, \notin L_i| \\ &+ \sum_{l \in L_i \cap L_j} |pos_i(l) - pos_j(l)| \end{aligned} \tag{6.1}$$

That is, the distance between two programs i and j is defined as the sum of:

1. $(|L_i| - |L_j|)$ the difference in number of lines $(|L|)$ in programs i and j

2. $|l \in L_i, \notin L_j|$ the number of lines in program i that do not appear in j

3. $|l \in L_j, \notin L_i|$ the number of lines in program j that do not appear in i

4. $\sum_{l \in L_i \cap L_j} |pos_i(l) - pos_j(l)|$ the difference in position of shared lines between the two programs – for example, if `moveForward();` appears at line 1 in i and line 5 in j, the value of $|pos_i(l) - pos_j(l)|$ will be 4.

Any two programs for which $d(i, j) = 0$ were determined to be identical and were merged into a single node in the graph.

To determine graph connectivity, each partnership's sequence of runs was translated into a directed path in the graph. The node representing their first run attempt was labeled an "initial" node, and the node representing their final attempt was labeled a "terminal" node. Terminal nodes were additionally labeled by whether their code successfully solved the level.

The length of each edge was defined to be the $d(i, j)$ of the two nodes which it connects. The weight of each edge was defined to be the number of partnerships whose path through the graph utilized the edge.

6.2 Results

Of 2798 attempted code variations, 656 were run more than once and 310 by more than one partnership. 57 were successful variants and 18 of those successful variants were used by more than one pair. A visual representation of this graph is shown in Figure 6.3, which does not enforce the distance metric defined above.

Across all users there were 4554 connections between nodes (many nodes were connected to each other more than once). The most frequent connection (weight=21) was between the tool-provided default code `moveForward();` and itself; users pressed the "run" button without modifying code multiple times in a row.

The most frequent connection (weight=15) of connections on successful paths (N=2351) was between `moveForward();` and itself. For connections on unsuccessful

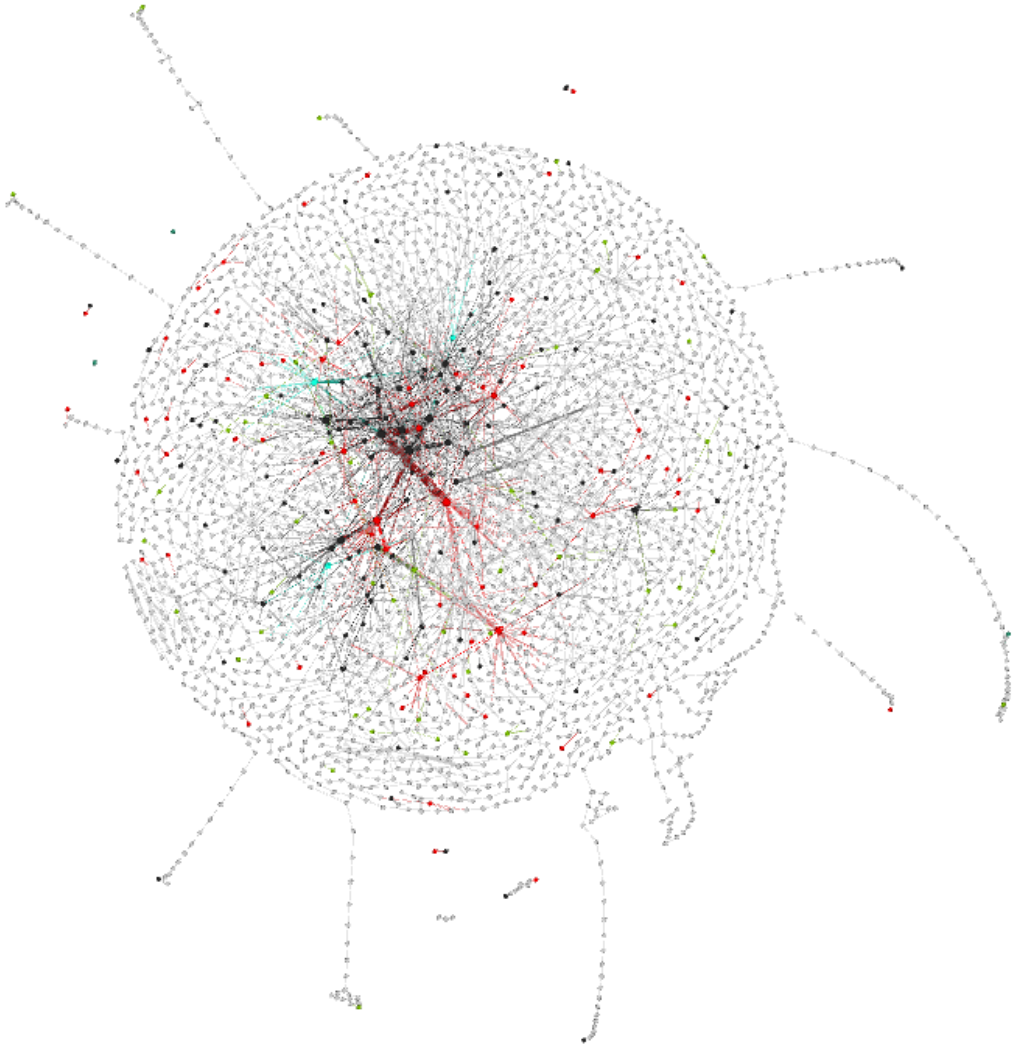


Figure 6.3: A graph representation of the solution state space. Initial nodes are represented in black. Terminal failure nodes are represented in red, terminal success nodes are represented in green. Node size is proportional to number of attempts made using that code.
Each node in this graph represents a unique auto-generated JavaScript program. Nodes have not been condensed in any way to represent identical logic with slightly variant implementations.

```
while (notDone()) {
  moveForward();
  if (isPathForward()) {
    if (isPathLeft()) {
      if (isPathRight()) {
        turnRight();
      } else {
        turnLeft();
      }
    } else {
    }
  } else {
    turnLeft();
    moveForward();
    turnRight();
  }
}
```

Figure 6.4: The most frequently traveled path (weight=13) for unsuccessful coders was from this code to itself. This program appears to successfully solve the maze, but the pegman runs into a wall at the end rather than stopping on the target. Students reported confusion as to why this did not "count".

paths (N=1180), the most frequent connection (weight=13) was between two longer programs, shown in Figure 6.4.

The mean distance traveled by successful partners (per the metric specified in Section 5.1) was 243.45 ($SD = 281.41$), and the mean distance traveled by unsuccessful partners was 160.42 ($SD = 190.03$). A two-tailed t-test revealed that the distances are significantly different between successful/unsuccessful partnerships with $p = 0.028$. Distribution of these distances can be found in Figure 6.5.

Of 72 unique terminal nodes from the the 76 unsuccessful partnerships, 41 were *not* visited in successful partners' traversals. Of the 62 unique initial nodes from unsuccessful partners, 33 were *not* visited by successful partners.

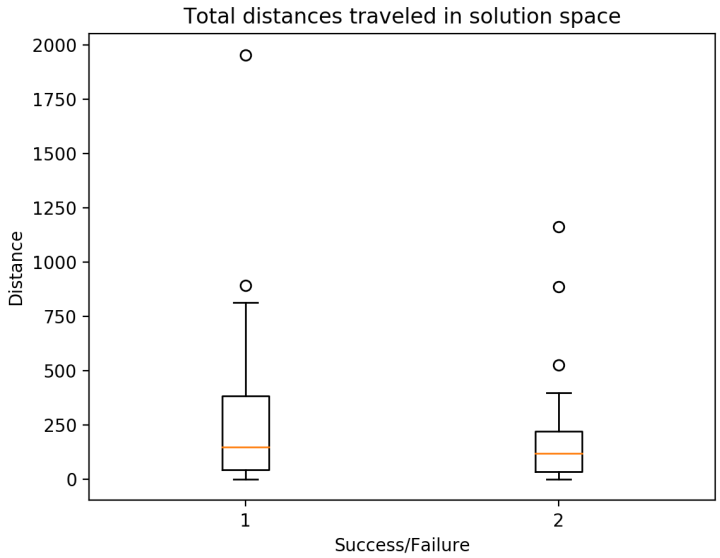


Figure 6.5: Boxplot of distances traveled through solution space, by success (1) or failure (2) on level 10.

6.3 Discussion

As might be expected, the results of this study strongly suggest persistence as a major contributing factor to success in a programming task, and frustration as a major contributing factor to a lack thereof. Paths terminating in a successful solution covered a greater distance from start to finish, and nearly half of the paths that did not *terminate* in a successful solution were connected to one.

The heaviest connection in the unsuccessful space was between a nearly-correct solution and itself, suggesting that students who arrived at this configuration ran it repeatedly without modification — the "why doesn't this work" problem.

Also evident in the results is the sheer variety of logical solutions to what is in theory a relatively simple problem: navigate through a maze. The suggested logical approach was to keep the left hand of the pegman on a wall of the maze, moving left when possible, forward when left was not possible, and right when no

other possibilities presented themselves. Of the 57 successful code variants, more than 2/3 implemented some variety of this logic but in distinct ways. A smaller fraction used the right hand instead, and a handful implemented logic specific to the maze configuration to move the pegman directly to the goal state. This variety in implementation persists even when derived from a block-based language, not taking into account the much greater stylistic variety inherent in written code.

6.3.1 Future work

Despite efforts to collapse paths automatically and simplify the graph space, the representation remained too complex for an intelligible visual representation (see Figure 6.3). Nodes that performed identical logic in a slightly different order remained distinct; a manual analysis of successful node states suggests that of the 57 code variants labeled "successful", there were actually fewer than 10 unique logical variations (including several failing variations that were incorrectly labeled successful).

Future analysis of the graph representation of solution space could include a more context-aware collapsing of the nodes: in the case of this study, a node could be defined not by its source code but rather by the pattern followed by the pegman in the maze space during the code execution.

Chapter 7

Discussion and Implications

The studies and analyses detailed within this dissertation have focused on the characterization of student programming activities using non-intrusive data collection methods as well as an exploration of students' self-reported perceptions of paired and individual work completed over the course of a semester. Applications of the methods described in the previous chapters are intended to contribute to the fields of computer science education research primarily by way of applications of machine learning algorithms to the difficult yet critical task of forming groups for collaborative learning, but there are also broader implications for the field of computer science education in general.

The primary lessons learned from this research are in the groundwork for a program to aid in partnership matchmaking.

7.1 The difficulty of matchmaking

As with any problem of pairing humans, the global optimal solution (the "best" partner for any given person) is not only nontrivial, it is potentially a very personal

decision with many factors that are beyond our modeling capabilities with computers or indeed even with other humans (arranged marriages, for example, remain relatively unpopular in modern society).

Many studies of pair programming, as discussed in the first two chapters, have uncovered that nearly any pairing can be beneficial over individual work in terms of their metrics. Detrimental pairings only arose when students were at wildly disparate levels and the more advanced partner was unable to act as a tutor for the less advanced partner (either because the more advanced student did not want to teach, or the less advanced student did not want to learn, or some combination of the two).

With that guideline, however, narrowing down the pool of all students to merely exclude those candidates who would not be beneficial partners for a particular student is merely a matter of skill level analysis.

7.2 Intellectual merit

The results of the present work have five major implications: less-intrusive observation of programmers (compared to video or audio recording of sessions, or direct observation by a researcher) provides valuable data for the characterization of the quality of that programming session; working style is more a function of a programmer's individual preferences than the culture of a class; partners who are successful at a programming task may help each other recover from states that frustrate or even defeat less successful partners or individuals; allowing students to self-select their partners may be more beneficial than previously suggested; and figuring out how to improve pairing suggestions may be important to supporting under-served students in introductory programming courses.

7.2.1 Less-intrusive observation still provides valuable data

Many studies on pair programming efficacy and programmer working style have relied upon intrusive, in-person observation (e.g. [49]) or video/audio recording (e.g. [34]); or analysis of pre- and post-activity tests as measures of learning (e.g. [7]) or surveys of student perceptions (e.g. [47]). Students who are aware they are being monitored may behave differently from their most routine working style, confounding the analysis. Additionally, these direct observation methods require a nontrivial amount of post-processing, including but not limited to content tagging, transcription, and data entry.

Utilization of web logs requires more initial work for a researcher to set up, and requires that the programmers use a particular tool with which they may not be familiar, but web logs can be collected unobtrusively while students work as they normally would on the internet and can be directly analyzed with little human intervention required. Importantly, these unobtrusive methods produce data which can differentially characterize working styles and success probabilities.

7.2.2 Working style: individual preferences vs class culture

It could potentially be a concern, particularly when examining the working styles of introductory programming students who have not yet developed their own habits and idiosyncrasies, that students will all solve problems in the same way: that is, the way that the instructor of the class has demonstrated. Yet the working style analysis in Chapter 5 demonstrated that even in the third week of a first programming class, utilizing only a block-based programming language, it was possible to cluster students by programming style with many significant differences between the factors used to characterize them.

Additionally, even though all code analyzed was automatically generated from the configuration of blocks for the final program, 57 different solutions were produced by students and only 18 were used by more than one partnership.

7.2.3 Successful partners help each other recover

As discussed in Chapter 6, nearly half of unsuccessful partners abandoned their attempts at the final maze level at a state that at least one successful pair had attempted at some point in their coding process. This aligns in a quantifiable sense with the intuition that a significant contributor to lack of success at early programming tasks is simply fatigue. Well-matched partners will encourage each other to continue, help each other debug, and can recover from states where individuals or less-well-matched partnerships might consider themselves stuck.

Additional analysis of the graph space of student programming attempts could additionally reveal ways in which the programming tool itself could provide hints and feedback to students to help them recover, particularly in the majority of cases where unsuccessful partnerships got stuck in states that were not used by successful pairs. Allowing a backtrack to a previous state that led to a solution, or an automated suggestion of a small modification that would put a student on a successful track, or even automated encouragement of a student for *being* on the right track could all be explored as potentially helping students achieve working results.

7.2.4 Self-selection of partners may be beneficial

It has been suggested that allowing students to self-select partners in a collaborative learning situation is detrimental to students: students who select their friends will have difficulty staying on task, students without existing networks in the class are

increasingly isolated, and in general students are considered to be poor judges of the qualities which would make an optimal working partnership. Approaches such as random assignment are argued to create a more level playing field; instructor assignment allows the intervention of an "expert" (or allegedly at least someone more-informed than the students); tools such as CATME [37] allow for some contributions from students as they self-report some aspects the instructor deems important but ultimately take partner selection out of the students' hands.

In the study detailed in Chapter 3, students were allowed to self-select not only their partners but whether to work with a partner at all. Students who chose a partner and students who chose to work alone exhibited very similar achievement and perception of assignment difficulty across the semester; the outlier group which performed at a lower level and found the assignments more difficult were the students who wanted to work with a partner but could not find one for the given assignment.

7.2.5 Improving pairing may be important for under-served students

Those students who could not find a partner but wanted one are precisely the population the future work aims to support. As a group, these students are more likely to struggle in the course and perceive assignments as more difficult, and helping to bridge that gap could make significant strides toward retention of new majors or at the very least improvements in the DF-drop rate for introductory courses (the proportion of students receiving a grade of D or F, or withdrawing from the course after the initial no-penalty drop deadline), a metric with which the department at Wisconsin at the very least has historically struggled.

Demographic information was not obtained, nor were interviews conducted to uncover any underlying factors leading to these students' inability to find a partner. It can't be said for certain that targeted minorities are over- (or under-) represented within this group. It also cannot be said for certain whether students who don't find partners on their own are attempting and being turned down or are simply shy and overwhelmed by the prospect of picking a partner from the group. However, easing the process of finding a partner and using the information gathered from a web tool to suggest students who are potentially good fits will go a long way toward empowering these students by granting them the agency of self-selection without making their decisions for them.

7.3 Broader impacts

The research described in this dissertation is immediately intended to lay the groundwork for an automated pair matching tool similar to online dating services, where individuals are offered an array of suggested pair programming partners informed by their activities on the site. However, the impacts of this research are broader than just the foundations of the partnering tool: there are implications not only for computer science education researchers, but also for instructors and students in introductory programming courses.

Philosophically, evaluating whether a programming partnership is successful in an introductory course is often simply a matter of asking whether the pair ultimately found a successful solution to the posed problem, which was frequently the metric used in this research. Additional metrics for success (particularly in paired situations) involve more subjective, self-reported measures of success, such as satisfaction with the task, willingness to work with the partner again, or perceived split of the workload;

an observer may evaluate the interactions of the partnership and categorize them on a productivity scale; or partners may complete pre- and post-test evaluations to quantify the learning achieved in the task.

This research suggests that web log analysis may allude to some additional metrics for success. In particular, pairs (or individuals) who find the solution can easily be called successful, but even students who do not find the solution can be monitored in their attempts and rewarded proportionately for progress made. Students who find a solution but do not exhibit behaviors typical to other successful users might merit closer investigation, either for academic misconduct or simply for having a level of programming skill that is very different from the other students in the class – higher, if they find a solution very quickly, or lower and in need of some additional assistance if it takes a very long time and/or an abnormally large number of attempts and actions in the tool.

Being able to differentiate between the behaviors of successful and unsuccessful users in the tool, even without the paired aspect of the activity, is of particular interest for automated tutoring or even a live monitoring dashboard for instructors in a lab setting. Utilizing this unobtrusive monitoring strategy and feeding the data into a live analysis program could help determine when students are more likely to give up and need encouragement, hints, or help from an outside expert.

7.3.1 Broader impacts: CS1 instructors

The value of pair programming in the introductory programming classroom has been widely established [41, 59, 7]. This research underlines that finding, but also establishes the value that can lie in making the decision to partner or work individually elective rather than compulsory (see Chapter 3). While programmers should undoubt-

edly cultivate the ability to work in a group while in school, as their professional lives will almost certainly require them to do so, forcing all students to begin group work at the onset of their programming education may not necessarily be an appropriate choice. *Enabling* all students who want to work with a partner to do so, however, is absolutely a goal that we as instructors should have.

For those students interested in working with a partner, this research aligns with prior results that suggest having a partner is beneficial. As a group, students who wanted partners but could not find them achieved lower scores and perceived assignments as more difficult than students who worked individually by choice or students who did work with partners. Anecdotally, many students reported that allowing them to work with a partner on projects enabled them to implement much more interesting and challenging designs than they would have been able to individually at their novice programming level.

Perhaps of highest value to a wide variety of instructors beyond those teaching introductory programming is the ability to characterize where and how students get stuck in their programming tasks and potential methods in which instructors can assist in getting them back on track. The graph-based analysis of student traversal through the solution space can help instructors diagnose the points at which unsuccessful programmers diverge from the paths of their successful peers, and nudging them back onto a more productive path may provide some assistance. As noted in Chapter 6, the analysis done is very cursory and significant improvements can be made in terms of collapsing the state space and classifying the paths taken by students more generally, which will make significant strides toward supporting instructors in this manner.

7.3.2 Broader impacts: CS education researchers

This dissertation has implications for computer science education researchers in terms of how we regard programming partnerships for CS1 students, its data collection methodology, and its analysis paradigm.

Typically when discussing programming partnerships in CS education research (e.g. [7]), we compare two groups of students: one group in which partnering is compulsory, and one in which work must be completed individually. Students are not allowed the agency to self-select between these groups. However, per my findings in Chapter 3, it may be illustrative to begin allowing students to self-select between these groups. Some introductory students prefer to program alone for various reasons, and others prefer to work with a partner. This complexifies our model of the introductory programming student, and research into the various motivations of these two groups of students could help inform further policies and research on collaborative learning at the CS1 level.

The data collection method for the study detailed in Chapter 4 is a different, less-intrusive approach to observation of programming habits than is typically used in CS education studies. While it measures the pair's interaction with a tool rather than with each other, we can still use the data to derive important characteristics about the partnership as a whole and make significant observations about the working style and success probability of the partnership. This methodology is also extremely useful for making live characterizations of users of a tool as it can be processed and fed into a machine learning algorithm as it is generated, and can therefore give very quick feedback about the styles and success rates of various users of a tool.

Applications of data analysis methods to learning analytics is not in and of itself a novel contribution of this work (see e.g. [5]), but the particular application of graph

analysis (as used in game analytics to map players' paths through the game space) to the problem of analyzing student paths through the solution space is a useful conceit for understanding the contrasting behavior of successful and unsuccessful programmers. While in its raw form the graph space is rather complex, reducing it as is done in game analytics could provide very useful insights to the differing behaviors of programmers and help researchers diagnose the frustrations and issues encountered by students who experience less success in their intro courses.

Chapter 8

Conclusions and Future Work

In practical conversations among instructors of computer science courses, the topic of paired or group work arises frequently. Those who have used it relay their success stories, along with lessons learned of how students are generally initially resistant to the idea and it takes some getting used to. The question almost immediately arises from others, "how do you pair them up?"

In my experience with these conversations, I have heard other instructors cite every method mentioned in Chapter 1: allow the students to choose, assign at random, assign manually, use a survey-based tool like CATME. These recommendations are, however, based only on their anecdotal experience (or at least with CATME, some work showing that it is better than random assignment[37]). But what is really the *best* method, and how do we quantify "best" anyway?

For the purposes of many studies in Pair Programming and other varieties of collaborative learning in the computer science classroom, "best" is defined as the manner which most improves student outcomes, for a variety of outcomes:

- Achievement – quantified as scores on exams and assignments

- Learning – quantified as a difference in performance on pre- and post-activity tests
- Actualization – quantified as student-reported measures of belonging, enjoyment, or other non-academic effects
- Retention – quantified as the rate at which students continue in the major

For instructors in larger classroom settings, the consideration of time investment to create the pairings is also a relevant concern. An ideal answer to the question of how best to pair students would maximize the student outcome improvement while minimizing the required time investment for instructors, thus making all parties as happy as possible.

The question that this research enables us to ask is, then, by using automated logging tools to observe the manner in which each of our students work, what are the best efforts we can make to point them at the students with whom they would be most compatible — with whom they would produce the best work, learn the most, forge the strongest connections within their field, and ultimately be happiest.

8.1 Future Work

Based on the groundwork laid in the investigations detailed in this dissertation, I plan to continue expanding this research into a fully-featured partner matching tool.

While block-based programming activities are useful for base-level working style detection due to their regularity and automatic translation to standardized JavaScript code fragments with no need to account for code style, variable names, or other idiosyncrasies of written languages, the next step will be to replicate and extend the study detailed in Chapter 4 with an in-browser Python interpreter so the study can be

performed longitudinally over the course of a semester, as with the perceptions study in Chapter 3. I anticipate a significant increase in complexity with the transition to a written coding language, but also an increased level of practicality.

Around this extended version of the study I plan to modify the working style machine learning analysis for real-time clustering of working style and tracing of success pathways. I plan to perform an investigation of the quality of the working style partnership suggestions by dividing the study subjects into three groups: providing one group with the algorithm-derived suggestions, one with randomized suggestions and the last with assigned partners. I will track partnerships and perceptions data as in Chapter 3, with a comparison of partnership quality at the end of the semester. The partnership quality I plan to investigate will include both academic achievement for all parties involved, as well as self-reported measures of satisfaction and perception of difficulty relative to other students in the class.

Ultimately what I wish to investigate is the level of effort required from instructors and other experts in matching students for paired programming work: is there a significant utility in determining these matches? Is the determining factor merely the narrowing of the field and agency of the student for self-determination of partnerships? The argument against self-selection of partners has historically been that students will select their friends or people they already know, defeating the networking component of partner work and potentially choosing someone with whom they are likely to get off-topic. But is this actually the case?

Additionally, the majority of partner-based work in the literature is focused on the single paradigm of Pair Programming as officially defined in Chapter 1. Yet, in my work as an instructor, even students exposed to this paradigm and given some training in how to use it were more likely to work in parallel on individual computers, where both students functioned simultaneously as driver and navigator for each

other. Anecdotally students who chose to work like this justified their decisions with the rationale that they would learn better from the physical act of typing the code and running it themselves, even if they were literally copying the exact characters their partner had typed. Further investigation into the efficacy of this paradigm is warranted, given its "natural" adaptation by many intro-level students.

Bibliography

[1] Erik Andersen, Yun-En Liu, Ethan Apter, François Boucher-Genesse, and Zoran Popović. Gameplay analysis through state projection. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, FDG '10, pages 1–8, New York, NY, USA, 2010. ACM.

[2] Elliot Aronson and et Al. *The jigsaw classroom.* Sage, Oxford, England, 1978.

[3] Kent Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, 1999.

[4] Peter Blatchford and Anthony Russell. Class size, grouping practices and classroom management. *International Journal of Educational Research*, 2018.

[5] Paulo Blikstein. Using learning analytics to assess students' behavior in open-ended programming tasks. In *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*, LAK '11, pages 110–116, New York, NY, USA, 2011. ACM.

[6] Grant Braught, L. Martin Eby, and Tim Wahls. The effects of pair-programming on individual programming skill. *ACM SIGCSE Bulletin*, 40(1):200, 2008.

[7] Grant Braught, Tim Wahls, and L. Marlin Eby. The Case for Pair Programming in the Computer Science Classroom. *ACM Transactions on Computing Education*, 11(1):1–21, 2011.

[8] Ann L. Brown. Design Experiments: Theoretical and Methodological Challenges in Creating Complex Interventions in Classroom Settings, 1992.

[9] Ann L. Brown. Transforming Schools into Communities of Thinking and Learning about Serious Matters, 1997.

[10] Sallyann Bryant, Pablo Romero, and Benedict du Boulay. The collaborative nature of pair programming. In Pekka Abrahamsson, Michele Marchesi, and Giancarlo Succi, editors, *Extreme Programming and Agile Processes in Software Engineering*, pages 53–64, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[11] Tracy Camp, W. Richards Adrion, Betsy Bizot, Susan Davidson, Mary Hall, Susanne Hambrusch, Ellen Walker, and Stuart Zweben. Generation cs: The growth of computer science. *ACM Inroads*, 8(2):44–50, May 2017.

[12] Scott Carlson. Wanted: Female computer-science students. *Chronicle of Higher Education*, 52(19), 2006.

[13] Jeffrey C. Carver, Lisa Henderson, Lulu He, Julia Hodges, and Donna Reese. Increased retention of early computer science and software engineering students using pair programming. *Proceedings of the 20th Conference on Software Engineering Education & Training,*, pages 115–122, 2007.

[14] E. A. Chaparro, A. Yuksel, P. Romero, and S. Bryant. Factors Affecting the Perceived Effectiveness of Pair Programming in Higher Education. *17th Workshop of the Psychology of Programming Interest Group*, pages 5–18, 2005.

[15] Kyungsub S. Choi, Fadi P. Deek, and Il Im. Exploring the underlying aspects of pair programming: The impact of personality. *Information and Software Technology*, 50(11):1114–1126, 2008.

[16] Kyungsub Stephen Choi. A comparative analysis of different gender pair combinations in pair programming. *Behaviour and Information Technology*, 34(8):825–837, 2015.

[17] Pierre Dillenbourg and Daniel Schneider. Collaborative learning and the Internet. In *Published at http://tecfasun1.unige.ch/tecfa/tecfa-research/CMC/colla/iccai95 1.html. ICCAI 95*, pages 10–6. Online, 1995.

[18] J. A.W. Faidhi and S. K. Robinson. An empirical approach for detecting program similarity and plagiarism within a university programming environment. *Computers and Education*, 11(1):11–19, 1987.

[19] Silvana Faja. Pair programming as a team based learning activity: A review of research. *Issues in Information Systems*, XII(2):207–216, 2011.

[20] Kathi Fisler and Francisco Enrique Vicente Castro. Sometimes, rainfall accumulates: Talk-alouds with novice functional programmers. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER '17, pages 12–20, New York, NY, USA, 2017. ACM.

[21] Sanjay Goel and Vanshi Kathuria. A Novel Approach for Collaborative Pair Programming. *Journal of Information Technology Education*, 9(1):183–196, 2010.

[22] Keun Woo Han, Eun Kyoung Lee, and Young Jun Lee. The impact of a peer-learning agent based on pair programming in a programming course. *IEEE Transactions on Education*, 53(2):318–327, 2010.

[23] Brian Hanks, Sue Fitzgerald, Renée McCauley, Laurie Murphy, and Carol Zander. Pair programming in education: A literature review. *Computer Science Education*, 21(2):135–173, 2011.

[24] Jo E. Hannay, Erik Arisholm, Harald Engvik, and Dag I.K. Sjøberg. Effects of Personality on Pair Programming. *IEEE Transactions on Software Engineering*, 36(1):61–80, 2010.

[25] Ole R Holsti. *Content analysis for the social sciences and humanities*. Addison-Wesley, Reading, MA, 1969.

[26] Norman Jacobson and Suzanne K. Schaefer. Pair programming in cs1: Overcoming objections to its adoption. *SIGCSE Bull.*, 40(2):93–96, June 2008.

[27] Bernard J Jansen, Isak Taksa, and Amanda Spink. Research and Methodology foundations of transactional log analysis. *Research and Methodology foundations of transactional log analysis*, 2009.

[28] Randall Jensen. A pair programming experience. *CrossTalk*, 16(3):22–24, 2003.

[29] Neha Katira, Laurie Williams, and Jason Osborne. Towards increasing the compatibility of student pair programmers. *Proceedings of the 27th international conference on Software engineering - ICSE '05*, page 625, 2005.

[30] Trupti M Kodinariya and Prashant R Makwana. Review on determining number of cluster in k-means clustering. *International Journal*, 1(6):90–95, 2013.

[31] Ken Kolchier. Exploring synergistic impact through adventures in group pairing. In *Agile Conference, 2009. AGILE'09.*, pages 265–270. IEEE, 2009.

[32] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. In *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

[33] Wilfred W.F. Lau and Allan H.K. Yuen. Exploring the effects of gender and learning styles on computer programming performance: Implications for programming pedagogy. *British Journal of Educational Technology*, 40(4):696–712, 2009.

[34] Colleen M. Lewis and Niral Shah. How Equity and Inequity Can Emerge in Pair Programming. *Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER '15*, pages 41–50, 2015.

[35] Anthony McGregor, Mark Hall, Perry Lorier, and James Brunskill. Flow clustering using machine learning techniques. In Chadi Barakat and Ian Pratt, editors, *Passive and Active Network Measurement*, pages 205–214, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[36] Wes McKinney. Data structures for statistical computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[37] Matthew W Ohland, David J Woehr, Lisa G Bullard, Richard M Felder, Cynthia J Finelli, Richard A Layton, Hal R Pomeranz, Deer Run Associates, and Douglas G Schmucker Consultant. The Comprehensive Assessment of Team Member Effectiveness: Development of a Behaviorally Anchored Rating Scale for Self-and Peer Evaluation. *Academy of Management Learning & Education*, 11(4):609–630, 2012.

[38] Scott E. Page. *The Difference: How the Power of Diversity Creates Better Groups, Firms, Schools, and Societies (New Edition)*. Princeton University Press, 2007.

[39] Erik Pasternak, Rachel Fenichel, and Andrew N Marshall. Tips for creating a block language with Blockly. In *Blocks and Beyond Workshop (B&B), 2017 IEEE*, pages 21–24. IEEE, 2017.

[40] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2012.

[41] Leo Porter, Mark Guzdial, Charlie McDowell, and Beth Simon. Success in Introductory Programming: What Works? *Communications of the ACM*, 56(8):25–28, 2013.

[42] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer science education*, 13(2):137–172, 2003.

[43] Carolyn Rosé, Yi Chia Wang, Yue Cui, Jaime Arguello, Karsten Stegmann, Armin Weinberger, and Frank Fischer. Analyzing collaborative learning processes automatically: Exploiting the advances of computational linguistics in computer-supported collaborative learning. *International Journal of Computer-Supported Collaborative Learning*, 3(3):237–271, 2008.

[44] Mary Beth Rosson. Human factors in programming and software development. *ACM Comput. Surv.*, 28(1):193–195, March 1996.

[45] Norsaremah Salleh, Emilia Mendes, and John Grundy. Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 37(4):509–525, 2011.

[46] Norsaremah Salleh, Emilia Mendes, and John Grundy. The Effects of Openness to Experience on Pair Programming in a Higher Education Context. *2011 24Th Ieee-Cs Conference on Software Engineering Education and Training (Cseet)*, pages 149–158, 2011.

[47] Norsaremah Salleh, Emilia Mendes, and John Grundy. Investigating the effects of personality traits on pair programming in a higher education setting through a family of experiments. *Empirical Software Engineering*, 19(3):714–752, 2014.

[48] Joshua Sennett and Mark Sherriff. Compatibility of partnered students in computer science education. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 244–248, New York, NY, USA, 2010. ACM.

[49] Beth Simon and Brian Hanks. First-year students' impressions of pair programming in CS1. *Journal on Educational Resources in Computing*, 7(4):1–28, 2008.

[50] Jaideep Srivastava, R Cooley, M Deshpande, and P N Tan. Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data. *SIGKDD Explorations*, 1(2):12–23, 2000.

[51] Django Development Team. Django. `https://djangoproject.com`, 2017.

[52] K. Umapathy and A.D. Ritzhaupt. A meta-analysis of pair-programming in computer programming courses: Implications for educational practice. *ACM Transactions on Computing Education*, 17(4):1–13, 2017.

[53] Lev Semenovich Vygotsky. *Mind in society: The development of higher psychological processes*. Harvard University Press, Cambridge, MA, 1978.

[54] Günter Wallner. Play-graph: A methodology and visualization approach for the analysis of gameplay data. In *FDG*, 2013.

[55] Günter Wallner and Simone Kriglstein. An introduction to gameplay data visualization. In Petri Lankoski and Staffan Björk, editors, *Game Research Methods*, pages 231–250. ETC Press, Pittsburgh, PA, USA, 2015.

[56] Armin Weinberger and Frank Fischer. A framework to analyze argumentative knowledge construction in computer-supported collaborative learning. *Computers and Education*, 46(1):71–95, 2006.

[57] Linda L. Werner, Brian Hanks, and Charlie McDowell. Pair-programming helps female computer science students. *Journal on Educational Resources in Computing*, 4(1), 2004.

[58] Laurie Williams, Lucas Layman, Kelli M. Slaten, Sarah B. Berenson, and Carolyn Seaman. On the impact of a collaborative pedagogy on African American millennial students in software engineering. *Proceedings - International Conference on Software Engineering*, pages 677–686, 2007.

[59] Krissi Wood, Dale Parsons, Joy Gasson, and Patricia Haden. It's never too early: Pair programming in CS1. *Conferences in Research and Practice in Information Technology Series*, 136:13–21, 2013.

[60] Stuart Wray. How pair programming really works. *IEEE Software*, 27(1):50–55, 2010.

[61] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, pages 103–114, New York, NY, USA, 1996. ACM.

Appendices

Appendix A

IRB Information

The following information was presented in the Research Participant Information and Consent Form for the OKCoder tool.

Title of the Study: Characteristics of effective pair programming partnerships at the novice level

Principal Investigator: Prof. Matthew Berland [contact phone and email]

Description of the research

You are invited to participate in a research study about effective pair programming partnerships between students learning to program. You have been asked to participate because you are enrolled in a course which is using the OKCoder online programming exercise tool. The purpose of the research is to learn about the individual characteristics of effective pair programming partners, which will support creation of a matching algorithm to suggest effective partnerships to future users of tools like the one you're using today. This study will include students enrolled in courses which use OKCoder. The research will occur online using data generated by OKCoder, and you will have no additional responsibilities beyond those you will be completing for your coursework. **Your instructor will not have any information on whether or not you have chosen to participate in the study.**

If you consent, your data may be used regardless of whether or not your paired partner participates. **Other people in the class will not know whether or not you are participating.**

Your progress in the programming exercises as assigned for lab will be logged, as will your anonymous evaluations of your experiences working with a partner. Only authorized research staff will have access to identifying information, which will not be used in any publications which may be generated from this study. The logged data will be kept on secure, password-protected servers for seven years after the completion of the study. After the seven years have elapsed, all data and consent forms will be destroyed.

What will my participation involve?

If you decide to participate in this research, you will not need to complete any additional tasks beyond those you will already be completing for class. Participation means that you consent to allow the researchers to examine and analyze your use of the tool and your (anonymized) progress in a lab assignment, so that they may improve future performance of the tool.

Are there any risks to me?

We don't anticipate any risks to you from participation in this study; your data will be completely anonymized during the study so there will not be any risk of breach of confidentiality beyond normal usage of the tool.

Are there any benefits to me?

We don't expect any direct benefits to you from participation in this study.

How will my confidentiality be protected?

Your use of this tool will be recorded using a randomly-generated ID. Your identity will not be used in any publications that may result from this study, and will not be known by the researchers. **The instructor of this course will not know your consent status.** There is no possibility that participation in this study could have an effect on your grade in the course, as the instructor will not know who has consented.

Whom should I contact if I have any questions?

You may ask any questions about the research at any time. If you have questions about the research, you should contact the Principal Investigator Prof. Matthew Berland at (608) 263-4600 or mberland@wisc.edu.

If you are not satisfied with the response of the research team, have more questions, or want to talk with someone about your rights as a research participant, you should contact the Education and Social/Behavioral Science IRB Office at (608) 263-2320.

Your participation is entirely voluntary. If you begin participation and change your mind you may end your participation at any time with no effect on your grade.

Your virtual signature below indicates that you have read this consent form, had an opportunity to ask any questions about your participation in this research, and voluntarily consent to participate. You may print a copy of this form for your records.

Appendix B

Magnitude vector clusters

Summary statistics from the magnitude vector cluster analysis begin on the next page.

Magnitude Data

		Cluster 3.0		Cluster 3.1		Cluster 3.2					
	N	25		76		78					
		Cluster 7.0	Cluster 7.1	Cluster 7.3	Cluster 7.5	Cluster 7.2	Cluster 7.4	Cluster 7.6			
	N	5	20	33	43	17	42	19			
Feature	**Statistic**	**Cluster 3.0**		**Cluster 3.1**		**Cluster 3.2**			**Overall**	**F**	**p**
Number of runs	Mean	56.68		6.052631579		22.82051282			20.4301676	205.8660772	8.27E-47
	SD	17.6900424		5.649626762		11.71208404			19.76904034		
	Min	34		1		1			1		
	Max	110		29		49			110		
	Median	50		5		22.5			14		
		Cluster 7.0	**Cluster 7.1**	**Cluster 7.3**	**Cluster 7.5**	**Cluster 7.2**	**Cluster 7.4**	**Cluster 7.6**	**Overall**	**F**	**p**
	Mean	54.8	57.15	9.212121212	3.627906977	14.88235294	21.16666667	33.57894737	20.4301676		
	SD	15.76578574	18.10876859	6.821346067	2.676541119	11.6663536	9.052536258	9.057220379	19.76904034		
	Min	35	34	1	1	1	4	19	1	89.58235177	2.62E-50
	Max	82	110	29	12	45	41	49	110		
	Median	50	51.5	8	3	12	21.5	32	14		
		Cluster 3.0		**Cluster 3.1**		**Cluster 3.2**			**Overall**	**F**	**p**
Number of events	Mean	2362.72		263.4736842		1003.269231			879.0335196		
	SD	650.6119286		160.0844739		375.970596			778.8766578		
	Min	1493		40		140			40	317.6746679	3.93E-59
	Max	3966		616		1719			3966		
	Median	2099		246.5		978.5			653		
		Cluster 7.0	**Cluster 7.1**	**Cluster 7.3**	**Cluster 7.5**	**Cluster 7.2**	**Cluster 7.4**	**Cluster 7.6**	**Overall**	**F**	**p**
	Mean	2678.2	2283.85	385.9393939	169.4883721	626.4117647	984.0714286	1382.894737	879.0335196		
	SD	960.1686102	517.2393329	152.3876515	84.43667562	301.0569731	285.8844583	215.7125023	778.8766578		
	Min	1794	1493	74	40	140	293	927	40	155.6298651	8.67E-67
	Max	3966	3610	616	383	1283	1665	1719	3966		
	Median	2089	2172	381	150	592	955.5	1351	653		
		Cluster 3.0		**Cluster 3.1**		**Cluster 3.2**			**Overall**	**F**	**p**
Events before first run	Mean	50.04		43.52631579		51.82051282			48.05027933		
	SD	26.45219084		21.27914329		37.99991349			30.57188136		
	Min	22		2		0			0	1.4778249	0.2310
	Max	132		121		215			215		
	Median	38		39.5		44			42		
		Cluster 7.0	**Cluster 7.1**	**Cluster 7.3**	**Cluster 7.5**	**Cluster 7.2**	**Cluster 7.4**	**Cluster 7.6**	**Overall**	**F**	**p**
	Mean	53.2	49.25	43.39393939	43.62790698	52.29411765	50.97619048	53.26315789	48.05027933		
	SD	17.6793665	28.1671351	18.93199856	22.91746897	23.35345977	43.86477494	34.39223001	30.57188136		
	Min	33	22	19	2	2	0	5	0	0.50565465	0.8035
	Max	76	132	121	108	99	215	140	215		
	Median	56	36.5	38	40	53	41	47	42		
		Cluster 3.0		**Cluster 3.1**		**Cluster 3.2**			**Overall**	**F**	**p**
Maximum # of events before a run	Mean	171.52		64.43421053		133.4871795			109.4804469		
	SD	99.20972533		31.1529907		51.08006375			67.63072099		
	Min	73		2		42			2	49.67057839	7.88E-18
	Max	573		146		284			573		
	Median	151		65		123.5			99		
		Cluster 7.0	**Cluster 7.1**	**Cluster 7.3**	**Cluster 7.5**	**Cluster 7.2**	**Cluster 7.4**	**Cluster 7.6**	**Overall**	**F**	**p**
	Mean	217.2	160.1	81.09090909	51.65116279	97.29411765	142.2857143	146.4210526	109.4804469		
	SD	179.6578971	59.84889306	27.04918487	27.8833815	37.66690653	49.03851048	51.14305611	67.63072099		
	Min	91	73	22	2	42	65	66	2	21.76195688	5.89E-19
	Max	573	338	146	113	203	277	284	573		
	Median	127	153	80	48	90	132	140	99		

Magnitude Data

		Cluster 3.0		Cluster 3.1		Cluster 3.2			Overall	F	p
Number of UI events	Mean	1628.32		173.8421053		678.1153846			596.7206704		
	SD	475.613517		116.5057094		280.8583447			546.9674244		
	Min	945		15		91			15	279.0362072	2.63E-55
	Max	3111		523		1298			3111		
	Median	1488		136		691			421		
		Cluster 7.0	Cluster 7.1	Cluster 7.3	Cluster 7.5	Cluster 7.2	Cluster 7.4	Cluster 7.6	Overall	F	p
	Mean	1837.4	1576.05	261.3636364	106.6744186	406.2941176	668.3333333	942.9473684	596.7206704		
	SD	751.4361184	357.6806502	115.6024583	57.83043414	208.3432588	231.1393794	174.4596089	546.9674244		
	Min	1177	945	34	15	91	202	618	15	132.4730146	8.51E-62
	Max	3111	2588	523	235	910	1184	1298	3111		
	Median	1315	1493.5	257	105	372	667.5	913	421		
		Cluster 3.0		Cluster 3.1		Cluster 3.2			Overall	F	p
Number of Change events	Mean	193.64		11.10526316		68.93589744			61.79888268		
	SD	88.41668621		12.35716173		49.77934381			76.00936284		
	Min	47		0		1			0	139.3298821	5.35E-37
	Max	463		49		220			463		
	Median	183		6		56.5			30		
		Cluster 7.0	Cluster 7.1	Cluster 7.3	Cluster 7.5	Cluster 7.2	Cluster 7.4	Cluster 7.6	Overall	F	p
	Mean	240.4	181.95	16.78787879	6.744186047	38.82352941	62.0952381	111	61.79888268		
	SD	74.29024162	87.80004271	14.61787527	7.880187175	35.56474675	40.88998569	51.58437136	76.00936284		
	Min	140	47	0	0	1	2	49	0	60.15251765	1.03E-39
	Max	350	463	49	37	145	164	220	463		
	Median	244	178	16	4	30	52	102	30		
		Cluster 3.0		Cluster 3.1		Cluster 3.2			Overall	F	p
Number of Create events	Mean	107.24		23.61842105		59.38461538			50.88268156		
	SD	43.17988421		13.95384036		24.49711169			37.35340331		
	Min	41		7		13			7	115.0919541	1.09E-32
	Max	262		82		137			262		
	Median	105		21.5		61			41		
		Cluster 7.0	Cluster 7.1	Cluster 7.3	Cluster 7.5	Cluster 7.2	Cluster 7.4	Cluster 7.6	Overall	F	p
	Mean	108	107.05	30.48484848	18.34883721	53.94117647	55.5952381	72.63157895	50.88268156		
	SD	11.78134118	47.91395934	15.9907572	9.162318486	26.79435978	20.30137167	26.00852194	37.35340331		
	Min	85	41	7	7	13	16	31	7	42.20360948	2.16E-31
	Max	118	262	82	49	116	96	137	262		
	Median	113	100.5	25	18	58	56.5	68	41		
		Cluster 3.0		Cluster 3.1		Cluster 3.2			Overall	F	p
Number of Delete events	Mean	43.44		4.907894737		19.76923077			16.76536313		
	SD	16.69510108		4.76328511		8.710328495			15.64054058		
	Min	19		0		1			0	175.7413731	1.12E-42
	Max	86		23		37			86		
	Median	41		4		20			13		
		Cluster 7.0	Cluster 7.1	Cluster 7.3	Cluster 7.5	Cluster 7.2	Cluster 7.4	Cluster 7.6	Overall	F	p
	Mean	43.4	43.45	7.090909091	3.23255814	13.64705882	19.4047619	26.05263158	16.76536313		
	SD	8.593020424	18.16445705	5.253360462	3.529524702	8.838027305	7.739999766	5.986596293	15.64054058		
	Min	34	19	0	0	1	3	11	0	68.34327338	5.64E-43
	Max	54	86	23	14	33	37	35	86		
	Median	41	40.5	6	3	12	19	27	13		

Magnitude Data

		Cluster 3.0		Cluster 3.1		Cluster 3.2			Overall	F	p
Number of Move events	Mean	390.08		50		177.0641026			152.8659218	189.5965429	1.24E-44
	SD	142.3724468		32.4256234		76.85656633			135.8609785		
	Min	164		7		15			7		
	Max	918		127		394			918		
	Median	370		46		168			116		
		Cluster 7.0	Cluster 7.1	Cluster 7.3	Cluster 7.5	Cluster 7.2	Cluster 7.4	Cluster 7.6	Overall	F	p
	Mean	449	375.35	70.21212121	34.48837209	113.7058824	178.6428571	230.2631579	152.8659218	78.99463	7.85E-47
	SD	245.4008965	95.90165536	32.4212403	22.30483532	56.21677027	63.80879975	77.42217968	135.8609785		
	Min	205	164	8	7	15	47	91	7		
	Max	918	595	127	86	204	338	394	918		
	Median	398	365	67	33	116	172	237	116		
		Cluster 3.0		Cluster 3.1		Cluster 3.2			Overall	F	p
Total time elapsed on level	Mean	3527.505076		564.6888082		1951.479223			1582.789697	203.6030835	1.63E-46
	SD	1169.075608		427.4852723		634.8704247			1213.109387		
	Min	1834.367985		3.272269		565.054721			3.272269		
	Max	6280.278928		1613.946525		3458.943324			6280.278928		
	Median	3415.838871		385.118515		1853.449421			1443.396828		
		Cluster 7.0	Cluster 7.1	Cluster 7.3	Cluster 7.5	Cluster 7.2	Cluster 7.4	Cluster 7.6	Overall	F	p
	Mean	5569.879859	3016.91138	981.6863728	244.6674213	2680.321999	1487.921488	2324.06331	1582.789697	305.6210057	5.84E-89
	SD	416.7545843	601.2250112	296.5101407	140.195607	498.1609564	339.3519923	298.227146	1213.109387		
	Min	5003.628208	1834.367985	618.888191	3.272269	1847.102266	565.054721	1748.648474	3.272269		
	Max	6280.278928	3890.104085	1613.946525	582.919716	3458.943324	2002.743403	2712.390467	6280.278928		
	Median	5456.30911	3063.587169	855.480616	279.706653	2518.671162	1513.502813	2449.25184	1443.396828		
		Cluster 3.0		Cluster 3.1		Cluster 3.2			Overall	F	p
Time elapsed before first run	Mean	131.476998		106.6162559		166.6856373			136.2639112	4.63336626	0.0109
	SD	78.80579808		69.61192227		165.1392559			124.8516497		
	Min	22.623017		2.36283		0.275099			0.275099		
	Max	285.196857		324.631187		720.828329			720.828329		
	Median	124.232421		104.9785735		95.1744225			103.504907		
		Cluster 7.0	Cluster 7.1	Cluster 7.3	Cluster 7.5	Cluster 7.2	Cluster 7.4	Cluster 7.6	Overall	F	p
	Mean	135.8514378	130.3833881	111.7262307	102.6946474	161.7301206	160.5012874	184.7901891	136.2639112	1.62241349	0.1436
	SD	54.61688396	83.73297646	70.78421511	68.44042425	131.1583301	173.5920105	171.8025154	124.8516497		
	Min	78.399284	22.623017	6.450816	2.36283	0.472392	0.275099	6.611459	0.275099		
	Max	221.512399	285.196857	324.631187	292.20025	498.472502	720.828329	549.114025	720.828329		
	Median	124.232421	128.2383035	103.389585	106.858364	128.504416	85.8781875	94.487617	103.504907		
		Cluster 3.0		Cluster 3.1		Cluster 3.2			Overall	F	p
Maximum time between events	Mean	149.6247336		63.47760466		151.1277303			113.7032361	5.17178121	0.0066
	SD	211.1352326		79.49694865		228.5737815			183.1317257		
	Min	24.398735		0.997116		5.091897			0.997116		
	Max	1086.094759		472.394066		1795.531721			1795.531721		
	Median	77.766381		36.438315		93.1089615			65.845314		
		Cluster 7.0	Cluster 7.1	Cluster 7.3	Cluster 7.5	Cluster 7.2	Cluster 7.4	Cluster 7.6	Overall	F	p
	Mean	212.7684986	133.8387924	97.32014067	37.50542586	302.6336761	101.6788704	124.8777852	113.7032361	5.26148074	5.30E-05
	SD	124.6167774	224.931752	105.5163149	32.73891596	428.7151454	89.27624161	76.04814386	183.1317257		
	Min	62.136236	24.398735	1.564261	0.997116	5.091897	20.955241	17.76416	0.997116		
	Max	364.721307	1086.094759	472.394066	147.175862	1795.531721	500.476151	279.066935	1795.531721		
	Median	152.522634	69.0574135	50.095051	26.636106	182.310697	78.0801375	95.007613	65.845314		

Magnitude Data

		Cluster 3.0		Cluster 3.1		Cluster 3.2			Overall	F	p
Success	Mean	0.76		0.4342105263		0.6666666667			0.581005586	6.5321071	0.0018
	SD	0.4270831301		0.4956528474		0.4714045208			0.493394461		
	Min	0		0		0			0		
	Max	1		1		1			1		
	Median	1		0		1			1		
		Cluster 7.0	Cluster 7.1	Cluster 7.3	Cluster 7.5	Cluster 7.2	Cluster 7.4	Cluster 7.6	Overall	F	p
	Mean	1	0.7	0.575757575	0.325581395	0.882352941	0.523809523	0.789473684	0.581005586	5.00232718	9.47E-05
	SD	0	0.458257569	0.494227467	0.468591667	0.322189739	0.499432784	0.407682457	0.493394461		
	Min	1	0	0	0	0	0	0	0		
	Max	1	1	1	1	1	1	1	1		
	Median	1	1	1	0	1	1	1	1		
		Cluster 3.0		Cluster 3.1		Cluster 3.2			Overall	F	p
Time-to-Events Ratio	Mean	0.7262462413		1.439159884		0.5959417205			0.972155091	1.67445558	0.1904
	SD	0.2539484802		4.468116614		0.3898787289			2.952003789		
	Min	0.3207933669		0.05194487994		0.06439418775			0.051944879		
	Max	1.352751436		33.00462156		2.640452233			33.00462156		
	Median	0.762916548		0.5496434511		0.5342693639			0.583716392		
		Cluster 7.0	Cluster 7.1	Cluster 7.3	Cluster 7.5	Cluster 7.2	Cluster 7.4	Cluster 7.6	Overall	F	p
	Mean	0.477921325	0.788327470	0.427460177	2.215580589	0.227771847	0.738456620	0.610323933	0.972155091	1.8044325	0.1008
	SD	0.159540267	0.234475199	0.215000892	5.819064966	0.095827054	0.439623841	0.148980214	2.952003789		
	Min	0.320793366	0.427903089	0.051944879	0.221696172	0.064394187	0.194868580	0.375314694	0.051944879		
	Max	0.730032914	1.352751436	0.940692115	33.00462156	0.406020508	2.640452233	0.983044920	33.00462156		
	Median	0.417497046	0.803058109	0.378435999	0.772627856	0.230287803	0.620588258	0.591839024	0.583716392		

Appendix C

Scaled vector clusters

Summary statistics from the scaled vector cluster analysis are on the next page.

Scaled Data

		Cluster 7.0	Cluster 7.1	Cluster 7.2	Cluster 7.3	Cluster 7.4	Cluster 7.5	Cluster 7.6			
	N	29	1	1	1	1	7	140			
Feature	**Statistic**	**Cluster 7.0**	**Cluster 7.1**	**Cluster 7.2**	**Cluster 7.3**	**Cluster 7.4**	**Cluster 7.5**	**Cluster 7.6**	**Overall**	**F**	**p**
Time-to-events ratio	Mean	0.851818827	33.00462156	21.31797176	9.571810029	2.244214585	1.728422742	0.507914744	0.966935485	4.24E+03	7.27E-185
	SD	0.241287413	0	0	0	0	0.469604790	0.228590377	2.944620549		
	Min	0.266940223	33.00462156	21.31797176	9.571810029	2.244214585	1.270192251	0.032625939	0.032625939		
	Max	1.281879511	33.00462156	21.31797176	9.571810029	2.244214585	2.640452233	1.022792537	33.00462156		
	Median	0.878081984	33.00462156	21.31797176	9.571810029	2.244214585	1.562772916	0.501051783	0.579803824		
Feature	**Statistic**	**Cluster 7.0**	**Cluster 7.1**	**Cluster 7.2**	**Cluster 7.3**	**Cluster 7.4**	**Cluster 7.5**	**Cluster 7.6**	**Overall**	**F**	**p**
Percent of time elapsed before first run attempt	Mean	0.483794014	1	1	0.651975093	1	0.022193695	0.108775361	0.183699488	3.56E+01	7.17E-28
	SD	0.313587760	0	0	0	0	0.019045771	0.114573729	0.241414823		
	Min	0.037531177	1	1	0.651975093	1	0.000486853	0.000255747	0.000255747		
	Max	1	1	1	0.651975093	1	0.061116938	0.748649932	1		
	Median	0.406043054	1	1	0.651975093	1	0.016511737	0.068413586	0.079145915		
Feature	**Statistic**	**Cluster 7.0**	**Cluster 7.1**	**Cluster 7.2**	**Cluster 7.3**	**Cluster 7.4**	**Cluster 7.5**	**Cluster 7.6**	**Overall**	**F**	**p**
Percent UI events	Mean	0.574450865	0.870370370	0.870370370	0.655172413	0.770491803	0.764496882	0.668820169	0.660065455	6.30E+00	5.19E-06
	SD	0.122965211	0	0	0	0	0.078712641	0.098186716	0.111351866		
	Min	0.322033898	0.870370370	0.870370370	0.655172413	0.770491803	0.693093093	0.295774647	0.295774647		
	Max	0.778284671	0.870370370	0.870370370	0.655172413	0.770491803	0.894736842	0.893617021	0.894736842		
	Median	0.579399141	0.870370370	0.870370370	0.655172413	0.770491803	0.716897506	0.689186641	0.681713221		
Feature	**Statistic**	**Cluster 7.0**	**Cluster 7.1**	**Cluster 7.2**	**Cluster 7.3**	**Cluster 7.4**	**Cluster 7.5**	**Cluster 7.6**	**Overall**	**F**	**p**
Percent Change events	Mean	0.056007071	0	0	0.103448275	0	0.052257949	0.058076213	0.056800715	1.23E+00	2.93E-01
	SD	0.041572583	0	0	0	0	0.036111279	0.040342859	0.040780592		
	Min	0.009259259	0	0	0.103448275	0	0	0	0		
	Max	0.190789473	0	0	0.103448275	0	0.102575107	0.223001402	0.223001402		
	Median	0.053103448	0	0	0.103448275	0	0.042512690	0.049837305	0.049303188		
Feature	**Statistic**	**Cluster 7.0**	**Cluster 7.1**	**Cluster 7.2**	**Cluster 7.3**	**Cluster 7.4**	**Cluster 7.5**	**Cluster 7.6**	**Overall**	**F**	**p**
Percent Create events	Mean	0.119486186	0.064814814	0.064814814	0.120689655	0.114754098	0.039871712	0.072960670	0.079576492	6.65E+00	2.38E-06
	SD	0.059778838	0	0	0	0	0.013024939	0.036285111	0.044463585		
	Min	0.022810218	0.064814814	0.064814814	0.120689655	0.114754098	0.018025751	0.023468803	0.018025751		
	Max	0.275	0.064814814	0.064814814	0.120689655	0.114754098	0.057640750	0.225352112	0.275		
	Median	0.107317073	0.064814814	0.064814814	0.120689655	0.114754098	0.042659279	0.065874415	0.068124811		
Feature	**Statistic**	**Cluster 7.0**	**Cluster 7.1**	**Cluster 7.2**	**Cluster 7.3**	**Cluster 7.4**	**Cluster 7.5**	**Cluster 7.6**	**Overall**	**F**	**p**
Percent Delete events	Mean	0.020837661	0	0	0	0	0.012754049	0.019030091	0.018654351	2.89E+00	1.03E-02
	SD	0.011764302	0	0	0	0	0.009185017	0.009734228	0.010451869		
	Min	0	0	0	0	0	0	0	0		
	Max	0.038732394	0	0	0	0	0.024798927	0.054945054	0.054945054		
	Median	0.02	0	0	0	0	0.013813813	0.018415092	0.018415092		
Feature	**Statistic**	**Cluster 7.0**	**Cluster 7.1**	**Cluster 7.2**	**Cluster 7.3**	**Cluster 7.4**	**Cluster 7.5**	**Cluster 7.6**	**Overall**	**F**	**p**
Percent Move events	Mean	0.229218215	0.064814814	0.064814814	0.120689655	0.114754098	0.130619406	0.181112854	0.184902983	5.37E+00	4.15E-05
	SD	0.069563879	0	0	0	0	0.053917088	0.057957075	0.064481839		
	Min	0.093065693	0.064814814	0.064814814	0.120689655	0.114754098	0.042944785	0.056451612	0.042944785		
	Max	0.389830508	0.064814814	0.064814814	0.120689655	0.114754098	0.189008042	0.366197183	0.389830508		
	Median	0.224543080	0.064814814	0.064814814	0.120689655	0.114754098	0.151800554	0.177782541	0.179288876		
Feature	**Statistic**	**Cluster 7.0**	**Cluster 7.1**	**Cluster 7.2**	**Cluster 7.3**	**Cluster 7.4**	**Cluster 7.5**	**Cluster 7.6**	**Overall**	**F**	**p**
Percent of events occurring before the first run attempt	Mean	0.297790058	0.129629629	0.129629629	0.344827586	0.229508196	0.042122279	0.100585372	0.132479527	8.37E+00	5.61E-08
	SD	0.226422722	0	0	0	0	0.047804745	0.122510941	0.160737603		
	Min	0.02	0.129629629	0.129629629	0.344827586	0.229508196	0	0	0		
	Max	0.833333333	0.129629629	0.129629629	0.344827586	0.229508196	0.122699386	0.824324324	0.833333333		
	Median	0.248780487	0.129629629	0.129629629	0.344827586	0.229508196	0.011587982	0.058890807	0.070379453		