Computer Science Education

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/ncse20

Reevaluating the relationship between explaining, tracing, and writing skills in CS1 in a replication study

Max Fowler, David H. Smith IV, Mohammed Hassan, Seth Poulsen, Matthew West & Craig Zilles

To cite this article: Max Fowler, David H. Smith IV, Mohammed Hassan, Seth Poulsen, Matthew West & Craig Zilles (2022): Reevaluating the relationship between explaining, tracing, and writing skills in CS1 in a replication study, Computer Science Education, DOI: 10.1080/08993408.2022.2079866

To link to this article: https://doi.org/10.1080/08993408.2022.2079866

COMPUTER SCIENCE EDUCATION
https://doi.org/10.1080/08993408.2022.2079866

Routledge
Taylor & Francis Group

Reevaluating the relationship between explaining, tracing, and writing skills in CS1 in a replication study

Max Fowler[1], David H. Smith IV[1], Mohammed Hassan[1], Seth Poulsen[1], Matthew West[2] and Craig Zilles[1]

[1]Department of Computer Science, College of Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA; [2]Department of Mechanical Science and Engineering, College of Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA

ABSTRACT

Background and Context: Lopez and Lister first presented evidence for a skill hierarchy of code reading, tracing, and writing for introductory programming students. Further support for this hierarchy could help computer science educators sequence course content to best build student programming skill.

Objective: This study aims to replicate a slightly simplified hierarchy of skills in CS1 using a larger body of students (600+ vs. 38) in a non-major introductory Python course with computer-based exams. We also explore the validity of other possible hierarchies.

Method: We collected student score data on 4 kinds of exam questions. Structural equation modeling was used to derive the hierarchy for each exam.

Findings: We find multiple best-fitting structural models. The original hierarchy does not appear among the "best" candidates, but similar models do. We also determined that our methods provide us with correlations between skills and do not answer a more fundamental question: what is the ideal teaching order for these skills?

Implications: This modeling work is valuable for understanding the possible correlations between fundamental code-related skills. However, analyzing student performance on these skills at a moment in time is not sufficient to determine teaching order. We present possible study designs for exploring this more actionable research question.

ARTICLE HISTORY
Received 15 November 2020
Accepted 17 May 2022

KEYWORDS
Introductory programming; programming skills; skill hierarchy; replication; structural equation modeling

1. Introduction

Since the work of McCracken et al. (2001) and Lister et al. (2004), there have been multiple studies investigating the relationship between different sets of programming skills and the potential hierarchy they form. Researchers sought to establish the

CONTACT Max Fowler mfowler5@illinois.edu Department of Computer Science, College of Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Max Fowler: CONTACT, exam writer, and project lead. Email: mfowler5@illinois.edu
David H. Smith IV: exam writer and lead literature reviewer
Mohammed Hassan: question pool homogeneity and exam writer
Seth Poulsen: SEM analysis lead
Matthew West: project advisor
Craig Zilles: exam writer and project advisor

existence of a hierarchy, as knowing the relation between programming skills would help educators structure their course content and scaffold their students through the learning process. For example, perceptions of such a hierarchy motivates a recent theory that that suggests students should understand how to read code before students can write code (Xie et al., 2019). Lopez et al. (2008) are notable for first presenting potential evidence for a skill hierarchy, most notably between tracing, reading, and writing code. Their work in investigating the potential hierarchy of skills in introductory programming set off a flurry of follow up work investigating whether a hierarchy exists, what that hierarchy is, and how that hierarchy can be statistically supported (see, Section 2.3).

While the original Lopez et al. (2008) paper is of great importance given the line of work it has inspired, it presents some methodological concerns. We began the project aiming to solve what we considered the primary concern: a sample size that was too small to strongly support the claims about the hierarchy, which was also a concern in subsequent work looking at the hierarchy or the relationships between programming skills (Corney et al. 2014; Kikuchi & Hamamoto, 2016; Lister et al., 2009; Murphy, Fitzgerald et al., 2012; Murphy, McCauley et al., 2012; Venables et al., 2009). Multiple studies brought stronger structural equation modeling (SEM) methods to bear, but retained problems with sample size (Yamamoto et al., 2012, 2011).

We performed a quantitative, conceptual replication of Lopez et al. (2008)'s programming skill hierarchy using a large, non-major introductory Python course. To the original study, we bring a larger student pool (600+ students vs 38) and the more powerful modeling technique of SEM. In order to guarantee the presence of sufficient data to present our results with confidence, our analysis uses a simplified hierarchy to focus on four key latent variables (*Sequence, Tracing, Explaining, Writing*).

With this replication, we ask the following question:

RQ1: Will the hierarchy between *Sequence, Tracing, Explaining*, and *Writing* programming skills replicate with structural equation modeling?

To this research question, there are two potential and dueling hypotheses. We present these hypotheses below:

H1: The results of our study will conceptually replicate the original order of skills from the Lopez hierarchy of skills with a 95% confidence interval.

H2: The results of our study will suggest a hierarchy of fundamental programming skills (i.e. *Sequence, Tracing, Explaining*, and *Writing*) that is better supported.

We, like many others in the community (Corney et al. 2014; Kikuchi & Hamamoto, 2016; Lister et al., 2009; Murphy, Fitzgerald et al., 2012; Murphy, McCauley et al., 2012; Teague et al., 2012; Venables et al., 2009; Yamamoto et al., 2012, 2011), find the idea that a programming hierarchy may exist compelling and highly relevant to the development of practices to teach introductory programming. As such, we were particularly interested

whether or not our replication supports **H1**. At the same time, given the amount of data we will have available, we find it important to also judge the hierarchy against other potential options to determine what the true relationship between these skills is.

However, in the process of completing this replication, we came to a realization: we can determine only the strength of correlational relationships with this approach and data. We discuss how we may move past these limitations in Section 6. Ultimately, work investigating skill hierarchies should not stop at the correlations between skills, but determine the *ordering of* skills in how they should be taught. This is not a question that can be answered with our experimental design, but is the motivation for work on programming skill hierarchies.

The rest of the manuscript is organized as follows. We review prior work with respect to skill hierarchies in Section 2. In Section 3, we discuss our methods and instrument and how we differ from the original study. In Section 4, we discuss the original analysis and then lay out our analysis and its statistical power. In Section 5 we discuss our findings and elaborate on what they can and cannot say about these programming skills. Section 6 provides a discussion of the implications of our findings, as well as possible study designs for truly determining the order skills should be taught. We discuss the limitations present in our study design Section 7. Finally, we briefly conclude in Section 8.

2. Literature review

2.1. The early days of novice programming skills research

2.1.1. Early studies in novice knowledge acquisition

Studies investigating how students progress through learning the fundamentals of programming has been a topic of interest since the emergence of computing education research as a discipline (Soloway & Spohrer, 1989). Though the concepts of a skills hierarchy in programming is relatively new, the idea of there being different categories of skills that students must master is one that is long standing. One example of this is a 1986 article by Soloway in which the process of "chunking" a program into multiple subproblems through stepwise decomposition was formalized (Soloway, 1986). The method of decomposition is defined as being dependent on a student's ability to draw relationships between the set of subproblems they are currently presented with and those with which they are already familiar. Similarly, program composition is defined as a student's ability to break a large problem into smaller ones, solve those, and build a larger program up from those subproblems. Though not explicitly stated by Soloway, both stepwise decomposition and program composition are implied to be dependent on a student's ability to trace through code and extract a more abstract purpose. Winslow (1996) offered similar suggestions stating that programming is an act of mastering many basic problems and gaining the ability to solve and recognize problems through pattern recognition.

Perkins and Martin (1986) conducted investigations into the problems faced by beginner programmers enrolled in their second semester of a course taught in BASIC. The key finding they present is that beginner knowledge is "fragile", meaning it is difficult for students in the formative stages of learning to program to draw relationships between concepts and leverage what they have previously learned to solve new problems. They state that providing students with explicit problem-solving strategies was sufficient to

scaffold learners and lead to a measurable improvement in problem-solving skills. Findings by Soloway et al. (1983) reinforce the idea of introductory students' knowledge as being fragile with findings showing that a significant portion of students were unable to solve even the simplest of problems. Kurland et al. (1986) provided even further reinforcement with a study showing that high school students with two years of experience still had difficulty with the fundamentals of programming.

2.1.2. Early steps towards proposing a hierarchy

These early studies on student knowledge acquisition and the fundamental hurdles introductory programming students face motivated a 2001 ITiCSE working group that has generally been referred to as "The McCracken Group" (McCracken et al., 2001). This multi-national, multi-institutional study developed and administered an assessment on student grasp of programming fundamentals. Despite all students having prior programming experience at a level commensurate with the problems administered, the average score was around 20%. Their findings suggest that students lack a concrete understanding of how to take a large problem statement, decompose it into a number of manageable sub-problems, and construct a complete solution to the original problem.

These findings prompted a series of studies and working groups, one of which included the "Leeds Group". This group performed a multi-national study seeking to answer one question: "To what degree did students perform poorly in the McCracken study because of poor problem-solving skills or because of fragile knowledge and skills that are a precursor to problem-solving?" (Lister et al., 2004). This question was formed on the idea that problem-solving and programming are fundamentally interrelated in that one's programming abilities must be proportionate to the complexity of the problem at hand. That is, if a student fails to solve a problem despite having command of all tools necessary to do so then this indicates a failure of problem solving ability rather than a student simply needing to solidify their fragile knowledge through further practice. The findings of this study indicate that in most cases students experience difficulty in problem-solving due to deficiencies in skills that precede the ability to write programs.

These findings spurred the subsequent investigations by the BRACElet project aimed at further investigating the issue of poor student performance on basic code writing tasks (Clear et al., 2011; Tan & Venables, 2010). The first study to come from this project was a multi-institutional investigation on the existence of a relationship between code reading and writing skills amongst introductory programmers (Whalley et al., 2006). The findings of this study indicate that CS1 students, upon completion of their course, are not yet able to work at a fully abstract level and those who cannot describe code are less likely to be able to write functional code of their own. Lister et al. (2006) described their use of the SOLO taxonomy to evaluate solutions to code reading problems and presented the argument that students should first be taught how to read basic code prior to attempting to write it.

2.2. *"Relationships between reading, tracing and writing skills in introductory programming" – Lopez* et al.

Of the BRACElet studies that were performed Lopez et al. (2008) was among the most influential in terms of the line of work it inspired. Their study sought to investigate the existence of a learning hierarchy in introductory programming courses through statistical analysis of student performance on specific question categories in a comprehensive, end-of-first semester exam. The question categories they employed are:

(1) **Basics**: Questions that emphasize recall memory by requiring students to identify terms and recognize syntax errors.
(2) **Sequence**: These questions included recognizing where lines of code were missing and Parsons problems (Parsons & Haden, 2006).
(3) **Non-iterative Tracing (Tracing 1**): Simple tracing tasks that do not involve looping and ask students to predict program state at some state during runtime.
(4) **Iterative Tracing (Tracing 2**): Tracing tasks that involve iteration and ask students to predict program state at some point during runtime.
(5) **Exceptions**: Questions that test for a conceptual understanding of exceptions by proposing situations where errors might occur and how those errors might be handled by exceptions.
(6) **Data**: Tested students on their knowledge of data types present in Java (e.g. ArrayList, Boolean, double, int, String) and scoping.
(7) **Writing**: Tested students on their ability to write simple programs given a list of specifications.
(8) **Explain**: Tested students on their ability to explain the functionality of given code at an abstract level.

Their study included a group of 38 programming novices that had completed one semester of introductory Java prior to taking the exam. In preprocessing their assessment data, they employed the use of a polytomous Rasch Model to transform the ordinal grading grades into an interval scale (Andrich, 1978; Rasch, 1993). Following this, stepwise regression via backward elimination was used to construct a path diagram (see, Figure 1) representing the relationship between performance on different types of questions. Their most significant findings is that individually Tracing 2 and Explain only explain 15% and 7% of the variance in Writing respectively. However, when combined they explain 46% of the variance. These findings indicate a possible hierarchy in programming skills wherein

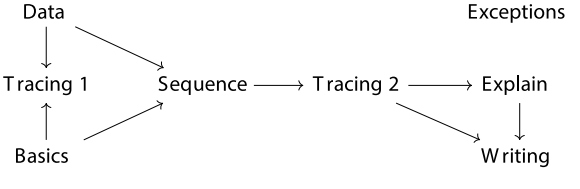


Figure 1. The skill hierarchy proposed by Lopez et al. (2008).

both the ability to trace non-trivial code and convey its purpose in plain English precede the ability to write code. However, their limited sample size prevents proper verification of the model at which they ultimately arrived (Figure 1).

2.3. Further investigations into hierarchies

Following the study by Lopez et al. (2008) in 2008, the BRACElet project continued to study the relationship between categories of programming skills. Lister et al. (2009) performed a replication of Lopez et al. (2008) in an introductory programming course taught in Python. Their replication was successful, suggesting that if such a hierarchy could be said to exist, it may exist independent of programming language. Venables et al. (2009) performed a more direct replication of Lopez's study and, though their findings were consistent with those of Lopez, they noted the results were particularly sensitive to the types of questions asked. Sheard et al. (2008) also found a positive correlation between students' performance on code reading and writing tasks. Teague et al. (2012) performed a replication of the hierarchies across two separate institutions. Though the majority of their findings were consistent with those of Lopez et al. (2008), they did find some variability in the structure of the hierarchy's intermediary components suggesting the hierarchy suggested by Lopez et al. (2008) may not be as consistent as other studies seem to suggest.

Some studies have sought to investigate code writing's position in the hierarchy at a finer granularity and with more sophisticated tools, namely structural equation modeling (SEM). Yamamoto et al. (2011) performed a conceptual replication of Lopez's study using SEM and extended the hierarchy it presented to include two new levels: 1) Modification – The ability of a student to modify a piece of code given a specification and 2) Programming – The ability of a student to write a program *from scratch* given some set of specifications. Each of these levels was further divided into two subcategories, one that involved questions with loops and the other which did not. They concluded that both modification and programming are dependent on the ability of a student to explain their code abstractly and write code that has been scaffolded in some way (Yamamoto et al., 2012, 2011). Similar to Lopez et al. (2008), they lacked the required amount of data to properly falsify models with the amount of complexity that they proposed.

Given the suggestion that within this supposed programming hierarchy, the ability to trace through a program precedes the ability to write one, one would expect that explicitly teaching code tracing would improve a student's ability to write a program. Kumar (2013) performed a study wherein students were first given a code writing pre-test, then performed a problem-solving session with code tracing problems, and finally given a post-test on code writing. The results of this study show that students performed significantly better on the post-test specifically with regard to the use of correct syntax. In a follow-up study, Kumar (2015) performed a study aimed at determining whether the code-tracing practice could improve code writing skills with regard to semantics and found that the ability to trace code is strongly correlated with the ability to write code. Mendoza and Zavala (2018) implemented an intervention plan aimed at facilitating the development of fundamental programming understanding towards improving the ability of students program writing skills. Their findings indicate that these early interventions have a positive impact on the ability of students to write programs.

Murphy, McCauley et al. (2012) investigated the relationship between "Explain in plain English" (EipE) responses and code writing and, in keeping with prior works, noted that students' fundamental knowledge is fragile. A student's ability to complete EipE questions was strongly correlated with their ability to write code later in the course. They additionally discuss the implications such findings have for teaching with their primary suggestion being that instructors should provide students with instruction and assessment on the basics of programming much later into the term than they might expect. In a follow-up study, Murphy, Fitzgerald et al. (2012) compared using paper-based EipE question to computer-based ones and achieved the same outcomes as their previous study across both mediums. They suggest that there is an underlying skill on which writing code and explaining it at a high-level is dependent. In a study investigating the relationship between the ability of students to abstractly describe algorithms and their ability to write the algorithms. Their findings show that the greater the ability of a student to write code the more likely they were to abstract the functionality of an implementation when attempting to describe the functionality of the algorithm they were implementing (Sudol-DeLyser, 2015).

Though the vast majority of investigations on the relationship between code tracing, writing, and explaining has been done with respect to CS1 courses, there have been several studies with investigations pertaining to CS2. Two studies, Harrington and Cheng (2018) and Corney et al. (2014), both investigated these relationships but reached contradictory conclusions regarding the relationship between tracing and writing. Harrington and Cheng (2018) investigated the degree to which the tracing-writing gap still existed in an introductory data structures course (CS2). Their findings indicate that the directional gaps between tracing and writing had largely disappeared by CS2 as students had mastered the fundamentals of programming. Corney et al. (2014) presented an analysis of CS2 students with a specific focus on data structure problems that included recursion in which they showed a strong correlation between the ability to abstractly explain a segment of code and the abilities of code writing and tracing. Given the contradictory conclusions reached by the former two studies, Pelchen et al. (2020) sought to further investigation the presence of a hierarchy in CS2. Their findings corroborate those of Corney et al. (2014) and they make the suggestion that the contradiction may have occurred due to a failure to maintain an even level of problem difficulty across categories on the part of Harrington and Cheng (2018).

Unique among these studies, Simon et al. (2009) failed to find any evidence either for or against a correlation between code reading and writing skills, due in large part to their small data set and use of questions that were not carefully designed for their research purposes. Despite this, they perform an in-depth and insightful discussion of the issues related to attempting to create a pure hierarchy of programming skills, the most notable of which is that there is no formal and reliable method for determining the difficulty of code reading or writing questions. Similar to Venables et al. (2009) they discuss the sensitivity of such findings to the questions that are used in the measurement instrument. This makes it particularly difficult to engage in any objective comparison between reading and writing as questions can only be categorized into different levels of difficulty based on some arbitrary heuristics. Similarly, Clear (2005) states that our current understanding of code comprehension is comparable to attempting to study reading comprehension without the concept of reading level.

As it stands, no study to date has performed a replication of the original hierarchy by Lopez et al. (2008) with *both* a sufficiently large dataset and appropriate methods (e.g. SEM). The majority of studies have instead opted to investigate the relationships between sets of skills (Corney et al. 2014; Harrington & Cheng, 2018; Murphy, Fitzgerald et al., 2012; Sheard et al., 2008) or a simplified hierarchy (Venables et al., 2009). The studies by Yamamoto et al. are a notable exception to this as they employed the use of Structural Equation Modeling (SEM; Yamamoto et al., 2012, 2011) to investigate a hierarchy similar to that which was proposed by Lopez et al. (2008). However, their design added additional latent variables to their model which increased both the complexity of their model and the number of data points required to validate it. In both of their studies, the number of data points required to validate the models they suggest significantly exceeds the number they had access too, and thus draws the conclusions of their study into question. We sought to remedy this gap in the literature by performing a conceptual replication of the hierarchy presented by Lopez et al. (2008) using both SEM and a dataset that is sufficiently large to validate the model(s).

3. Methods

Our aim is a conceptual replication of the Lopez et al. (2008) analysis of the relationship between reading, tracing, and writing skills in introductory programming. Our measure differs from the original, in part due to course context and in part to collect enough data to be confident of our findings' validity. This paper was a preregistered study, with our plan of analysis and the differences in assessment formats between our study and Lopez et al.'s study, discussed later in Sections 4.2 and 7.1, registered before study execution. The analysis procedure from Section 4.2 was updated with details on model counts for added clarity, but modeling was otherwise the same as was preregistered. The use of post hoc power analysis was made after preregistration to verify our original power calculations. Additionally, Section 7.2 was added after the preregistration to address validity concerns related to the item difficulty balance in the question pools used by the exam form generators in the course. Below, we present the ways in which our methods differ from the original study.

3.1. Course context differences

The Lopez et al. (2008) study was conducted in an introductory Java course using the course's final exam. Of the 78 students who completed the exam, 38 gave approval for their data to be used (Lopez et al., 2008). The original study was also conducted using paper-based examinations. Our replication study was conducted in an introductory Python course for non-technical majors during Spring 2021. Due to the COVID-19 pandemic, all lectures, labs, and exams were hosted online. This course uses a computer-based platform (PrairieLearn) to host all homework and exams (West et al., 2015). We also note that the student population we study is larger than the original study (Spring 2021 Final Enrollment of 252 women/399 men), with a total of 674 students taking exam 1, and 612 students taking the final exam.

3.2. Instrument differences

The instruments we used differ from those used in the original study. Lopez et al. used a paper-based final exam with 3 years of previous research-based refinement. Their exam had 13 questions assigned to different statistical variables for later analysis: *Basics, Sequence, Tracing, Data, Writing, Explain, Exceptions*, and *General*.

The scale of the Python course under study necessitates a different format for our instrument. The assessments used in this studied employed the use of PrairieLearn 's autograder for all questions on our instrument *except for* "Explain in plain English" (EipE) questions. These questions are exercises where students are given a code snippet and must provide a high level, natural language description of the code (Corney et al., 2014; Fowler et al., 2021). These questions were manually graded by members of the Python course's staff, including some of the authors on this paper. To guarantee accuracy in grading, all EipE questions were marked as correct or incorrect by two members of course staff. Disagreements were reconciled with the input of a third member of the grading team.

The course had three exams throughout the semester and a final exam, all of which were used independently to model the skill hierarchy. Exams were largely constructed from existing question banks where prior performance data was available. Additionally, new questions were constructed by the course instructor, two graduate teaching assistants, and one graduate research assistant, all with either significant teaching or research experience relating to the course. Due to Covid-19, all exams administered in the Spring 2021 semester took place in a synchronous, online format proctored using video conferencing software (Zoom). In order to mitigate the potential for collaborative cheating while taking the exam, exams were constructed with pools of questions. Each student receives their own exam generated by randomly selecting a question from each of these pools. The homework and exams were predominantly auto-graded using PrairieLearn.

As we later discuss in Section 4.2, creating a hierarchy we can validate requires some changes in the number of latent variables we analyze. Specifically, we analyzed four from the original study: *Sequence, Tracing, Writing*, and *Explain*. We keep the last three as they are of primary interest to both the original authors and other studies in the field. We also retain *Sequence* to validate where it belongs within the hierarchy.

Each latent variable is measured with different kinds of questions. *Writing* is the most straight-forward, with students being tasked with writing small functions in Python to accomplish some task, e.g. summing the numbers in a list. *Explain* questions are structured as short EipE exercises. Students are given a snippet of code and have to write a short, high-level English language description of what the code does. *Sequence* was measured similarly to Lopez et al. (2008)'s method as our exams also use Parson's problems (Parsons & Haden, 2006).

Tracing requires a bit more consideration. In a typical *tracing question*, students are provided with a code snippet and are asked to find the output of the code. As the course's exams are computer-based (and allow IDE usage – see, Section 7.1), students could potentially execute the given code snippet to find the output easily. To discourage this unwanted action, the tracing skill was primarily measured with a new type of question measuring the same knowledge, *Reverse-tracing questions* (Hassan & Zilles, 2021). *Reverse-tracing questions* ask students to input a value for a missing variable (in the code snippet) that would produce a desired output. Figure 2 provides an example of this type of question on PrairieLearn. Since a value of a variable is missing in the given code snippet, students can

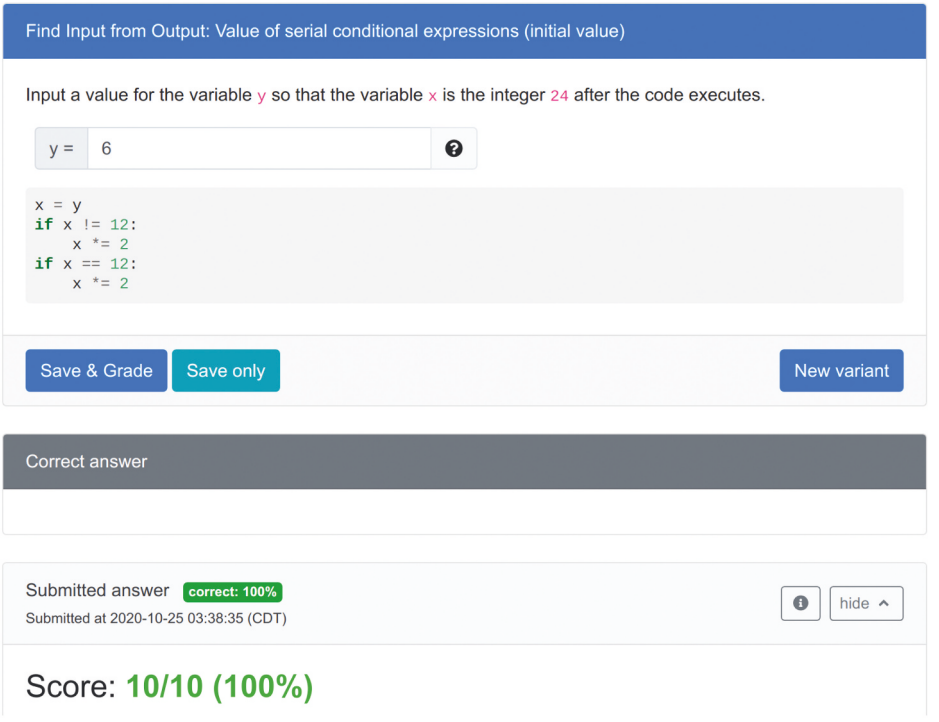


Figure 2. Sample reverse-tracing question on PrairieLearn.

not solve the problem by executing the given code alone, otherwise the student will encounter a variable undefined error. *Reverse-tracing questions* are difficult to brute force with an interpreter (e.g. solve with just randomly guessing inputs and repeatedly executing the code); students perform similarly on them with and without access to an interpreter (Hassan & Zilles, 2021). However, we set a minimum of 3 tracing question pools, overall, for our expected data to fit our hierarchy models. Our decision to prefer reverse-tracing questions was made after many of the exams were already written, leaving a gap in some of the exams. As such, where the number of reverse tracing question pools was 3 or lower, we retained more traditional tracing pools to bolster our measurement of the tracing skill.

In order to match up the question pools to the latent variables, specific question pools on each exam were designed to measure each variable. This did not include *all* of the questions on each exam. Further, as the exams, particularly the final, differed in makeup during the semester, the precise number of question pools per latent variable varied slightly between examinations. The distribution of question pools for the study on each exam is presented in Table 1.

The four exams in the Python course allowed us to take a "snapshot" of student ability at four points in the semester. While the focus of each exam was different (i.e. each exam focused on topics introduced since the previous exam), the four question types under study for a given exam focused on the same set of topics. For example, the topics of focus on the first exam included strings and conditionals, so the model generated from the first exam is built from string and conditional tracing questions, string and conditional

Table 1. Each exam in the course featured pools specifically prepared to measure the latent variables. There was some variance in how many questions measured each variable on each exam based on when in the semester the exam took place. We provide that breakdown for each exam above. Where more traditional tracing questions were retained, they are included in the table in parentheses.

Latent Variable	Exam 1	Exam 2	Exam 3	Final
Sequence	2	2	2	3
Reverse-Tracing (Tracing)	4	3 (2)	2 (2)	4
Writing	6	6	4	6
Explain	2	2	2	3

Parson's questions, string and conditional code-reading questions, string and conditional code-writing questions. A benefit of this approach is that the skills measured by the hierarchy for each exam were in the context of the same topics.

3.3. Data collection

Data is collected automatically by PrairieLearn over the course of students taking exams in the class. No additional data was collected from the students in the course. This data collection was approved by our Institutional Review Board (IRB), protocol number 21780. Consent was opt-out, with all students provided an informed consent document. The opt-out rate was low, with only 4.4% opting-out (thus, 95.6% of students remained in the study).

For each exam, the scores for all the questions were downloaded from the platform. Student identifiers were stripped from the data prior to any analysis via a script. The same script that anonymized student data was also used to label question submissions with the latent variable they measure and which question pool they belong to in order to feed the data into our hierarchy model.

4. Plan of analyses

In this section, we first discuss the analysis used in the original study. We then discuss the differences in our analysis for the replication.

4.1. Original analysis

In the original study by Lopez et al. (2008), they leveraged a polytomous Rasch model to address interval differences between scores by creating new interval marks. The issue they were attempting to address is that, in a rubric that uses whole numbers to signify level of correctness, the difference in ability levels associated with receiving a 1 versus a 2 might not be the same as receiving a 3 over a 2 (Venables et al., 2009). The assumption of unidimensionality was verified with principal component analysis.

Lopez et al. (2008) then used step-wise multiple regression via backward elimination to identify the relationships that existed between the various skills on which students were being tested (i.e. tracing 1 & 2, explain, writing, sequence, data, basics, exception). The regression was performed with each of these variables acting as criterion variables individually with the remaining variables in each case acting as explanatory variables. The process of removing the explanatory variable with the least predictive power was

repeated until all of the three stopping conditions were met: 1) maximum R^2_{adj}, 2) significance of 0.01, and 3) all explanatory variables were significant ($\alpha = 0.05$). This process was used to build the path diagram of associations between skills seen in Figure 1. This was followed up with Pearson correlation analysis between tracing, writing, and explain skills in order to further explore the relationships that exist between them. The assumptions of normality and homoscedasticity associated with these parametric tests were verified with the Jarque-Bera test of normality (Jarque & Bera, 1980).

The main drawback of the literature on the skill hierarchy thus far is that the proposed structural models are very complex and thus cannot be falsified without very large amounts of data. For example, take the skill hierarchy proposed in the original Lopez et al. (2008) paper. Suppose that their model was misspecified, and the correct model did not have the arrow going from "Tracing2" to "Writing". If that was the case, it would take a sample size of 1131 in order to falsify their proposed model–over 1000 data points more than what they had access to. The more complex models proposed by Yamamoto et al. (2012) would take even more data points to confidently falsify than the Lopez models. Thus, they can not say with a high level of confidence that an alternative proposed hierarchy would not be better.

Another drawback of models used in papers thus far is that they convert each of the sections of the test into a single score (either through a Rasch model, or just by summing the points from each question), and use that single score as an input to the model. Using these aggregated scores as inputs to the model requires the assumption that the few questions on the exam are a perfect measurement of each of the students skills (explaining, writing, etc.) without any error, making it impossible to quantify the amount of uncertainty in the final model. We propose to fix this problem by including each of the questions individually into the structural model which we build.

4.2. Our analysis

This study borrowed elements from aforementioned studies towards providing a more complete and robust analysis of any programming learning hierarchies that may emerge. Specifically, we use structural equation modeling (SEM) to model the programming skill hierarchy because it is a currently accepted approach for quantifying relationships between many interconnected measurements in social science settings (Bollen, 1989; Fan et al., 2016) and, as observed by Yamamoto et al. (2012), SEM allows us to fit multiple skill hierarchies and test them against one another to see which model is the best fit for the data.

To enable better handling of measurement error in student testing, we include the student score for each question in our model, rather than collapsing the categories into single scores. To allow proper falsification of incorrect hierarchy specifications, we propose a simplified model of the hierarchy, shown in Figure 3, which contains only the main skills of interest (*Sequencing, Tracing, Explaining*, and *Writing* code):

Using this simplified structure enabled us to our model against alternate structural models at a high level of confidence (95%), while requiring a more feasible number of data points. We used the "semPower" package in R (R Core Team, 2020; Moshagen, 2020) to quantify the data points we need.

For example, suppose that the skill hierarchy is actually a strict hierarchy, as shown in Figure 4. Using our SEM techniques, the "semPower" package's power analysis informs us we are able to rule out this model with only 605 data points, which all of our exams obtained.

As verification of our results following the study, we further used the *postHoc* method of "semPower" to calculate the actual power for the four exams. Degrees of freedom and the effect size as root mean square error of approximation (RMSEA), which is used to assess model fit, were pulled from the run models directly, with RMSEA averaged for the models for each exam to provide an overall view of our results. The power calculations provide the following results: exam 1 models have a power of 0.98 with an average RMSEA of 0.036, exam 2 models have a power of 0.98 with an RMSEA of 0.034, exam 3 models have a power of 0.99 with an RMSEA of 0.042, and exam 4 models have a power of 0.99 with an RMSEA of 0.046. Given this, we achieved over 95% confidence that the models accurately measure the relationships between skills, per the practice of measuring statistical power to determine and verify sample sizes for SEM (Wolf et al., 2013).

We were able to test many other skill hierarchies that involve the *Sequence, Tracing, Explaining*, and *Writing* skills. Specifically, we fit all possible structural models, which corresponds to all possible directed acyclic graphs (DAGs) on 4 nodes. We enumerated these using the connection between DAGs and adjacency matrices set forth in Royle et al. (2004). There are 543 total DAGs on four nodes, but we remove all disconnected graphs (which would correspond to a structural model where at least one variable is not used to predict any others) to get to 458 candidate models. These models are unconstrained, allowing for the *Write* skill to predict other skills in the hierarchy. This allows us to identify if there are alternative relationships between the four skills that are missed if we consider *Writing* as the terminal point of the hierarchy of skills. Equivalent models without *Writing* constrained as the end point could suggest a back-channel where students improve in earlier skills as they get better at *Writing* or could help us better understand the existing proposed hierarchy and how the skills relate.

For one stage of the analysis, we also optionally remove all models that use Write as a predictor of other variables to replicate the methodology used in the original paper, which brought us down to testing among only 151 models. This subset of models not only includes the original hierarchy, simplified down to the four latent variables we measure, but all possible SEM models where the *Writing* skill is not used to predict any others.

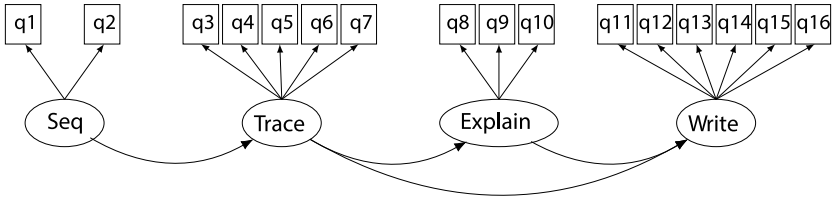


Figure 3. Simplified version of the path diagram, which is more feasible to falsify.

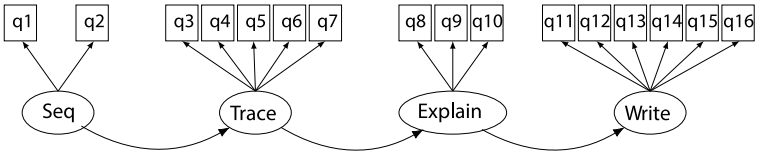


Figure 4. A possible alternative hierarchy with no edge between trace and write. We had enough data to include models like these at 95% confidence.

Our analysis was run on each of the four exams in the introductory Python course. SEM analysis is run on each exam separately to see if the hierarchy replicates regardless of the point in the semester. We then used the Bayesian Information Criteria (BIC) to rank the models for best fit so as to select the models that "best" represent the possible relationships between skills.

4.3. Identifiability of SEM models

As stated, one of the benefits of using SEM is that we can properly model the error in measuring each latent skill based on multiple questions, rather than just assuming no error as previous studies have done. The drawback of this method is that the model will have many unknown variables, and it can be a challenge to prove an estimated model *is* identifiable (Bollen, 1989) – that there exists a unique solution to the system of linear equations necessary to fit an SEM. By making a few reasonable assumptions and utilizing proven theorems regarding identifiability, we are able to show that all of the potential models we fit are, in fact, identifiable.

One method of showing identifiability for SEM with latent variables is to first show the identifiability of the measurement model and then show that the structural model is identifiable, thus showing the identifiability of the model as a whole. This is known as the "two-step rule"(Bollen, 1989, p. 328).

The "two-indicator rule" (Bollen, 1989, p. 244) states that a measurement model is identifiable if there are at least two measurements for each latent variable, each measurement is being predicted by only one latent variable, and all the latent variables have nonzero correlation. In our case, we do have at least two measurements for each latent variable, we only use one latent variable to predict each measured variable, and prior research confirms that the skills are correlated. Therefore, the measurement model is identifiable.

The "recursive rule" (Bollen, 1989, p. 95) states that if (1) the coefficient matrix of the structural model can be written as a lower triangular matrix (equivalent to saying that the graph structure of the model is a DAG) and (2) the error terms for the prediction of each of the latent variables are uncorrelated, then the model is identifiable. As we are only searching through DAGs, the first assumption is clearly true. The second assumption, that error terms are uncorrelated, is essentially the same assumption used in any context where researchers use regression models to predict exam performance, and so we find to be a reasonable assumption. Therefore the structural model is also identifiable and so is the entire model.

5. Findings

5.1. Candidate models for skill hierarchies

The analysis was run separately for each of the exams in an effort to identify if a skill hierarchy, or set of skill hierarchies, remained similar across exams. Additionally, we ran two sets of models for each exam: those constrained to have the code *Writing* skill at the end of the hierarchy (the "write" variable is not used as a regressor for other skills in the structural model) and those without that constraint (the "write" variable is used as a regressor for predicting other variables in the structural model). Of the models, we keep the 29 models from across the four exams that are equivalently explained by the

data. These models are selected as those with the highest, tied BIC score for each exam. BIC[1] was chosen to accommodate the differing number of parameters, represented by arrows in the models, and for its commonality as a model selection metric (Haughton et al., 1997; Lin et al., 2017; Preacher & Merkle, 2012). Together, these graphs are the "best" candidate explanation for the relationship between these four skills. Note that because models are fit per exam, the precise value of the "highest" BIC differs between exams.

Most notably, each exam produces different possible models as the "best" models for the skill hierarchy. Additionally, there is no direct overlap between these models for the various exams. There are some common trends that appear in many of the exams' models. *Sequence* and *Explain, Writing* and *Explain*, and *Writing* and *Tracing* are still often linked, although the direction of the prediction varies across models and not all "best" models feature these links.

The original hierarchy from Figure 3 does not strictly appear in these candidate models. It is instead ranked 112th, 9th, 18th, and 125th out of the 151 possible constrained models for exams 1, 2, 3, and 4 respectively. However, closely similar hierarchies exist within the best model set. These are hierarchies that may have one different link or one missing link. They are: exam 1a, exam 1b, exam 3b, and exam 3c from Figures 6–9. However, these similar hierarchies are not unique in the strength or confidence of the correlations they present between skills compared to the other "best" candidates.

In Figure 5, we use one of the exam 1 graphs as a representative sample of the way the graphs were constructed. Each of the latent traits is estimated by their respective question pools. For space, we will not present the individual weights of the question pools for each latent trait for the remaining 28 equally supported models. We present all the constrained models in Figures 6–9 and the unconstrained models in Figures 10–13.

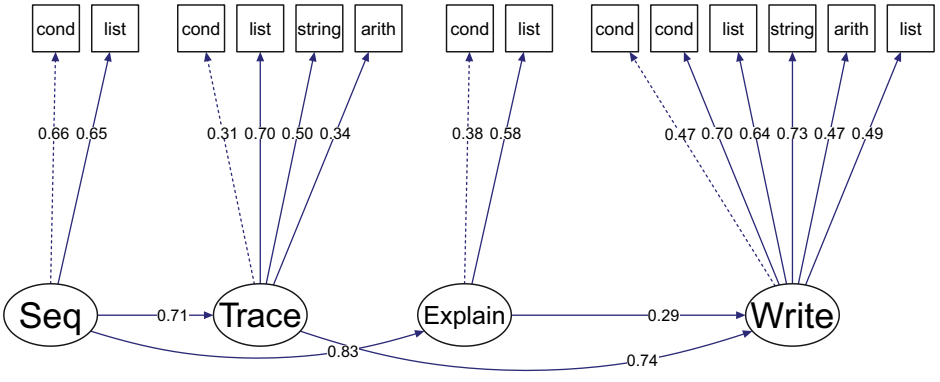


Figure 5. One of the best candidate graphs from Exam 1, with the questions which estimate each latent trait shown. This particular hierarchy is close to, if not the same as, the original hierarchy of skills.

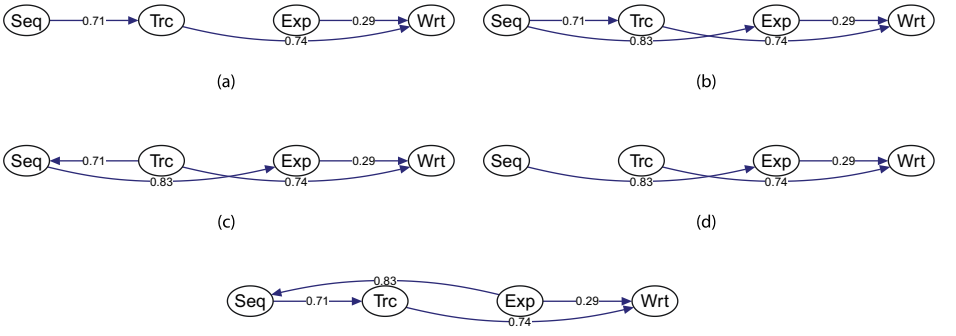


Figure 6. Exam 1: Structural models constrained to write.

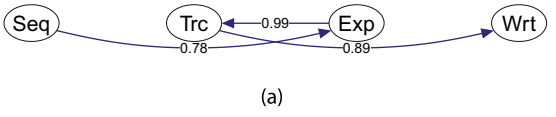


Figure 7. Exam 2: Structural models constrained to write.

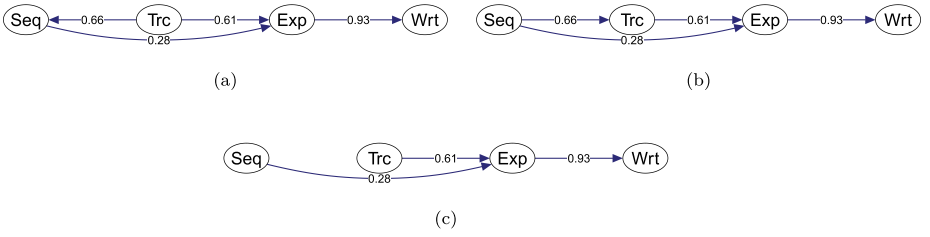


Figure 8. Exam 3: Structural models constrained to write.

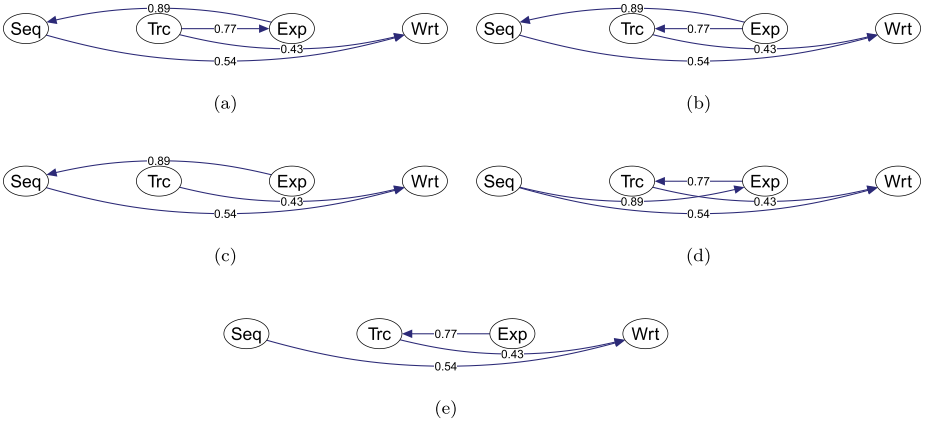


Figure 9. Exam 4: Structural models constrained to write.

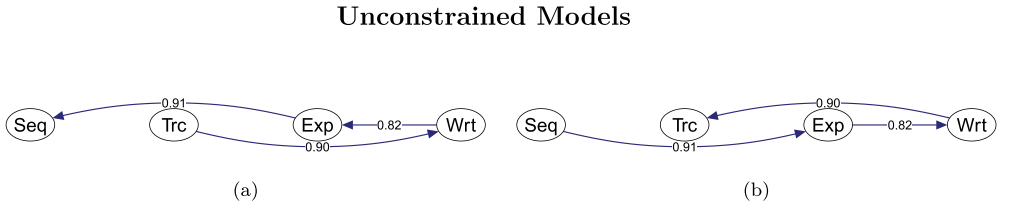


Figure 10. Exam 1: Unconstrained structural models.

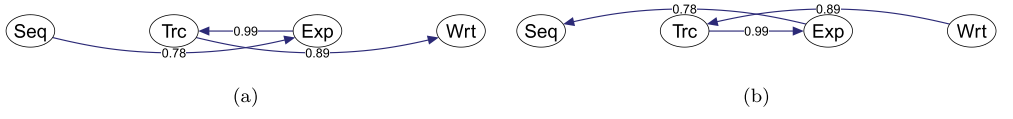


Figure 11. Exam 2: Unconstrained structural models.

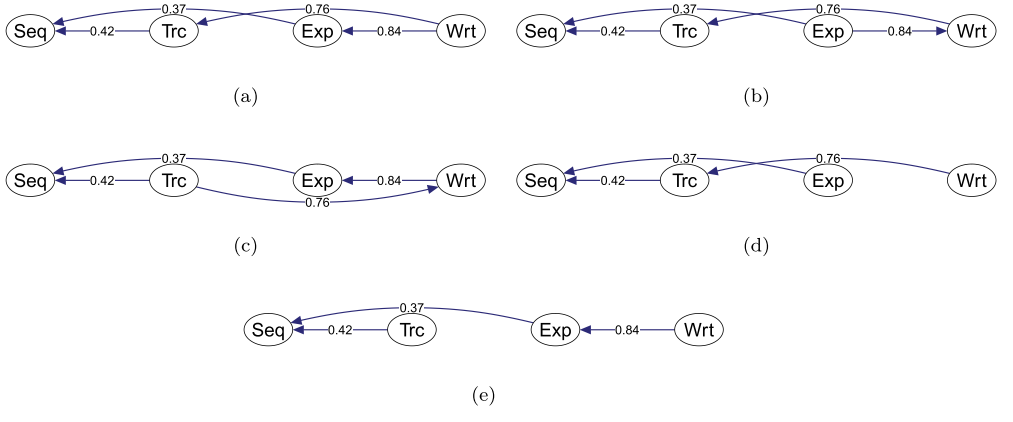


Figure 12. Exam 3: Unconstrained structural models.

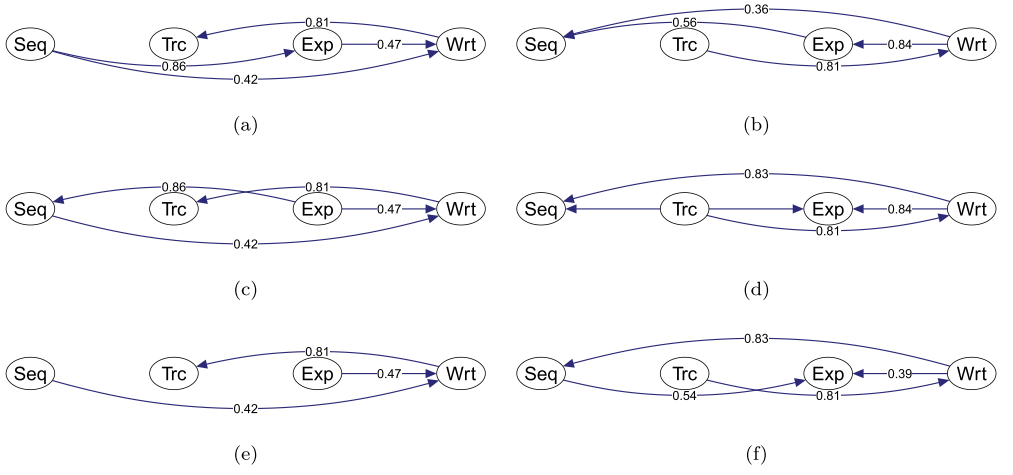


Figure 13. Exam 4: Unconstrained structural models.

With respect to **RQ1**, the hierarchy between *Sequence, Tracing, Explaining*, and *Writing* programming skills does not replicate with structural equation modeling as one of the candidate "best" models. However, the original hierarchy is still within the top 10 BIC scores for exam 2 and notably similar hierarchies appear in the "best" models for exam 1 and 3. The answer to the research question, then, is that our results do not support the specific hierarchy proposed by Lopez et al. (2008) as the strongest hierarchy. However, similar models are among the set of models our results deem the "best", supporting the existence of relationships between these skills in similar configurations. Broadly speaking, we see no support for singling out the model from the original paper as the only hierarchy between these skills.

5.2. On selection of SEM models

Some may find it surprising that there are so many models competing for the best fit in our analysis. We feel that this is an accurate reflection of the true nature of our data, due to the tight association of these skills, and the limitations of finding knowledge in data collected without a true experimental intervention.

All the prior research, and indeed our own data set, has supported that sequencing, tracing, explaining, and writing code all have a very high correlation. This correlation varies a bit from exam to exam due to random chance, question construction, exam topics, and other factors. As such, the best fitting models are different from one exam to the next as some skills appear more highly correlated than others as a result of this variance.

For skills where there is a very large difference in difficulty, and one skill is a hard prerequisite to another, it is possible to detect this by concurrent measurements (Guay & McCabe, 1986). One classic example would be the relationship between calculus and algebra. But for our skills of interest, the data show that none of them reach the point of being a "hard" prerequisite. The skills are too strongly correlated to be able to tell what skill is a prerequisite of another simply by a measurement without an experimental intervention.

Selection of SEM models is known to be a difficult problem (Bollen, 1989; Haughton et al., 1997; Preacher & Merkle, 2012). However, the space of models which are being selected from is usually restricted by some of the variables being independent and some being dependent (Bollen, 1989). Often, an experimental intervention would be an independent variable which would only be used as a predictor (not predicted by other variables) in a SEM model. For example, if students in a class were randomly assigned to either group work or individual work focused lab sections, the section could be limited to use as a predictor. However, in the data collection method used in this study and all prior studies in this vein, there are no independent variables, only dependent variables. Thus, the model space is not constrained, giving many possible models.

5.3. What our data give us – correlation, but not an order of teaching or learning

After performing our analysis, we realized that our chosen data collection and analysis method (along with those of all prior studies in this vein that we were replicating) do not answer the question we truly care about as educators. The models from SEM give us

correlations between skills. The directionality of the correlations does not prove that certain skills are pre-requisites to others nor does it prove that certain skills should be taught before others. The end goal for a hierarchy of skills is determining the optimal order for teaching these skills. What our SEM models show is that students who can succeed at a given skill have a higher chance of succeeding at other, related programming skills, but they do not provide causality. It is unlikely, given the tightly coupled nature of these skills, that more data alone would be able to sufficiently determine the absolute best form of the hierarchy. Instead, even with our data's sufficient statistical power, the tightly coupled nature of these skills makes determining a "best" hierarchy through correlations alone unreliable.

As stated, this is a weakness in the data collection process. We collected snapshots of students' ability through exams. We did not collect sequence data over time, measuring the change in all the skills from their first application in the class. We cannot use our exam data to determine the optimal order for teaching these skills as we lack a progression of students' skills over time and the data is confounded with other factors, such as exam conditions. In the next section, we discuss some ways in which our true interest – what is the correct order to teach programming skills? – could be determined.

6. Implications

6.1. This approach to understanding the skill hierarchy is limited to (helpful) correlations

Per our findings and the existing work we build upon, the hierarchy as it exists is reasonably supported but may not be the "best" possible hierarchy. However, all of this work is largely a set of correlations. This is helpful from a course design and assessment point of view, as it supports that practice for all of these skills contributed to a student's development as a programmer and allow for multiple different kinds of activities to assess a student's programming skills. Regardless, the message from these findings is limited to correlation between students' abilities on various skills in the set of programming skills we teach. By using directed arrows, these skill hierarchy diagrams suggest causality – e.g. that the *Explaining* skill improves the *Writing* skill but not vice-versa – that is not supported by these methods.

This is particularly important to keep in mind given the hopes for the skill hierarchy (Lister, 2020). In Lister's 2020 keynote at the 9th Computer Science Education Research Conference, he expanded on his hopes for the original hierarchy. In brief, the hierarchy was part of establishing a neo-Piagetian model of how students learn to code. Lister proposes students move through a pre-tracing stage, to a tracing stage, to a post-tracing, deductive reasoning stage. This alone does not mean the student has become an expert programmer, but rather that they can now "proceed to coding to learn", having gone through the process of "learning to code (Lister, 2020)". The verification of a hierarchy would be a powerful step toward structuring course content to guide students through this process.

In practice, we should hedge the claims we make about these skill hierarchies. They remain a valuable picture of the relationships between skills and different ways we reason about code. They also help us understand how some of the ways we reason about code are more closely connected to others. However, they do not provide us with actionable proof of the order we should teach skills.

6.2. Possible future work: experiments to test competing skill trajectories and determine directionality

The correlation between these programming skills is well established and now supported by sufficient statistical power. As such, there may be more fruitful directions for research to pursue beyond further correlation studies for these skills. In order to truly understand how these skills influence each other and the order in which we should teach programming skills, we must compare the effectiveness of teaching the skills in different orders. Determining the optimal instructional choices without a rigorous testing framework is largely intractable (Koedinger et al., 2013). We suggest testing the possible hierarchies of skills as a set of A-B tests, similar to instructional methods testing in massive online courses (Z. Chen et al., 2016). In general, A-B tests are proficient at allowing for controlled experiments with large amounts of participants and numerous options to explore.

Xie et al. (2019) showed an example for a skill hierarchy based, multi-step learning process for their proposed theory of instruction. In their study, students received instruction, practice, and feedback on each skill in the hierarchy. This pattern of instruction could be used to design a suite of exercises testing different skill orderings. Specifically, interventions based on different candidate skill hierarchies could be developed for course lab or discussion sections where students receive instruction, some form of practice, and feedback on their performance. This feedback could use automatic test cases, such as the exams in our study, or could be provided by course staff. Weinman et al. (2021) provide one format used for computerized testing of different skills via different problem interfaces in their work on Faded Parsons Problems, which may provide a starting point for building out the interventions in a scalable way. Of course, similar formats can be deployed using the same platform our study used, PrairieLearn.

These A-B tests could be run in the context of a semester long course by teaching different sections or across semesters of the same course using multiple different skill orderings. One section could teach skills in the order of *Sequence* → *Tracing* → *Explaining* → *Writing*, another section could teach *Tracing* → *Sequence* → *Explaining* → *Writing*, and so on. Differences in how students learn as they progress through different orderings could be measured and the most effective hierarchy for learning by the end determined. The large number of possible orderings makes an exhaustive survey somewhat untenable: with 24 orderings of skills into a four-step chain alone, that is 24 different sections.

An alternative approach would be to have students use an intelligent online tutor in a course to learn concepts. This approach is beneficial in that a true experiment is possible. Students could be randomly assigned to a skill order, and with a large enough community (e.g. Kahn Academy, Code4All) testing all orderings would be feasible.

7. Limitations

The primary limitations related to the findings of this study are the differences in exam platform and content we use and those used by Lopez et al. (2008). Because of the scale of the course that we studied, there are differences between how exams were conducted compared to the original study. We highlight the limitations that these differences pose on this study as a replication in the following subsection. While these differences should not be trivialized, we feel that if the skill hierarchy is sufficiently important to guide pedagogy in introductory programming courses, these superficial differences should not prevent our measurement of it. Additionally, past studies that have performed conceptual replications of the skills hierarchy initially suggested by Lopez et al. (2008) have made similar modifications to the skill hierarchies they investigated while preserving the core structure and pedagogical utility of the original hierarchy (see, Section 2.3 for further details).

7.1. Assessment format differences

The experience of our students taking exams on PrairieLearn differed from Lopez et al.'s (2008) paper-based exams in five ways: 1) students had access to an Integrated Development Environment (IDE) during exams, 2) exams provided students with automated instant feedback, 3) student access access to sketching space differed, 4) students had access to practice exam generators, 5) and exams featured item generators that create parameterized question variants (Gierl & Haladyna, 2012).

7.1.1. Access to an IDE

Exams in this course allow students access to the repl.it website, a web-based IDE featuring syntax error underlining, code-execution, and "help" documentation. These features may help students solve code tracing, reading, writing and Parson's problems. In addition, students can execute any code in repl.it as many times as desired, and it does not count toward the allowed number of submissions on problems. These could be seen as advantages not found in paper-based formats. Prior work found no significant differences in student performance semantic-wise across paper- and computer-based formats on easier code writing problems, but on more difficult code writing problems, the computer-based group significantly outperformed the paper-based group (Corley et al., 2020; Grissom et al., 2016; Lappalainen et al., 2017).

There are small risks related to tracing questions given the computer-based testing environment. Traditional tracing questions may be vulnerable to students simply copying code to run it rather than properly tracing (Hassan & Zilles, 2021). Per our discussion in Section 3.2, our expectation is that *reverse-tracing* problems can assess a similar skill to on-paper tracing problems.

7.1.2. Access to automatic feedback during the exam

The PrairieLearn exams used in this study permit students to interactively grade their answers on a per-question basis. When a question is graded, the student is provided (nearly) immediate feedback. Most questions support multiple attempts with partial credit

being assigned based on how many attempts were required to provide a correct answer. Code writing and Parson's problems also automatically assign partial credit based on the degree of correctness of the submitted solution.

The availability of immediate feedback and assigning partial credit automatically instead of manually likely affects both the scores students achieve and their test taking strategy. Corbett and Anderson (2001) found that immediate feedback can lead students to complete exams more quickly with no negative impact on scores. Schooler and Anderson (1990) found the opposite, with immediate feedback increasing error rates. It is unclear, however, if these differences would materially affect the outcomes of this study.

7.1.3. Sketching

While students are allowed to use scratch-paper while taking our computer-based exams, their inability to write directly on the test may affect student behavior, especially with respect to tracing questions. We discuss sketching's impact on tracing briefly below.

Past sketching studies found that students who correctly kept track of all variables in code tracing problems tend to perform better than students who do not. These studies found students writing the values of variables and their change per-iteration on paper (Cunningham et al., 2017, 2019; Vainio & Sajaniemi, 2007; Xie et al., 2018). This improvement in performance is due to the distributed cognition framework, where cognitive tasks can be off-loaded onto physical artifacts (e.g. scratch-paper) as notes or diagrams for memory aids and can be manipulated to help solve programming problems (Cunningham et al., 2017). Some sketching methods can be more successful in off-loading cognition than others. For instance, some students may keep track of only one variable on paper, which has lower cognitive load than keeping track of all variables but is much more error-prone (Cunningham et al., 2017; Vainio & Sajaniemi, 2007). In Cunningham et al. (2017)'s study, students who sketched on fixing code problems kept track of the value of only one variable and as a result performed worse than students who did not sketch.

For the Parson's problems given on PrairieLearn, we use an interactive drag-n-drop interface. This interface includes highlighting of correctly placed blocks and the first incorrectly placed block, as part of the automatic instant feedback with partial credit, as

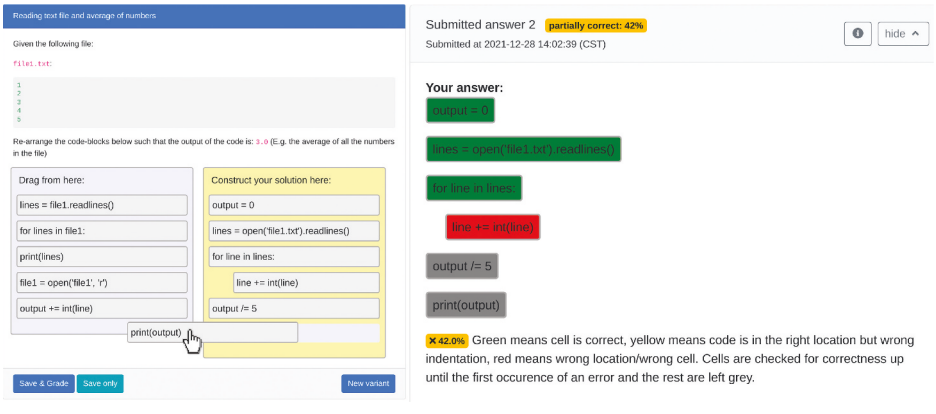


Figure 14. Sample Parson's question on PrairieLearn.

shown in Figure 14. This allows students to view their solution as a complete code snippet after dragging and arranging the blocks. This may impact student behavior differently compared to Parson's problems given on paper-based exams.

7.1.4. Automatic practice exam generation

In the course under study, we provided students access to a practice exam generator through PrairieLearn. Practice exams are automatically generated by randomly selecting subsets of questions from pools. Each pool consisted of a set of questions on a specific topic, where that topic would be on the corresponding actual exam. The tracing and reverse-tracing pools of the practice exams had some overlapping questions to the pools of the actual exams due to the varying parameters of each auto-generated question (covered in Section 7.1.5), but the pools of the code-reading, writing, and Parson's problems were different (completely unique sets of questions) from the actual exams. Since students had the ability to generate practice exams similar in topic to the actual exams, this may impact student performance relative to the Lopez et al. (2008) study.

7.1.5. Item generators

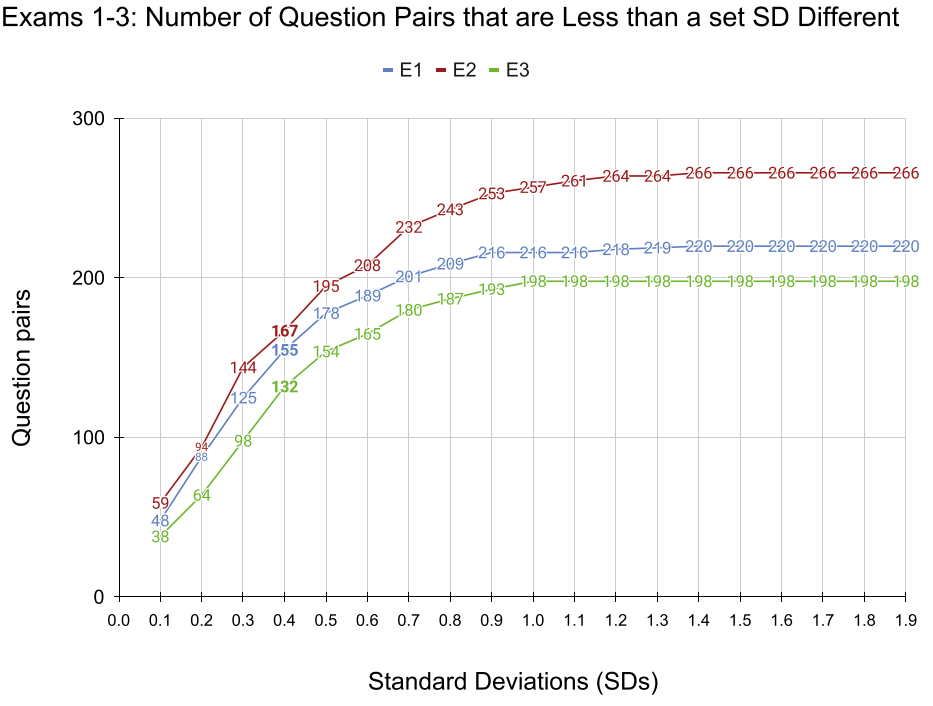
Each tracing and reverse-tracing question on the exams and practice exams consist of problems with varying parameters, also known as isomorphic variants. These varying parameters include different operators (e.g. greater than vs. less than), initialized integer values (e.g. variables), similar constructs (e.g. string upper vs lower), and so on. Chen found that, out of 378 problems that auto-generated isomorphic variants in introductory engineering courses, only 20 problems produced variants with statistically significantly different difficulty levels (B. Chen et al., 2019). For code-reading, Parson's, and programming problems, students were given a random selection from a set of completely unique questions (i.e. question pools).

In the following subsection, we present an analysis on our exams' question pools to determine how balanced the pools were and if any existing imbalance would significantly impact our results.

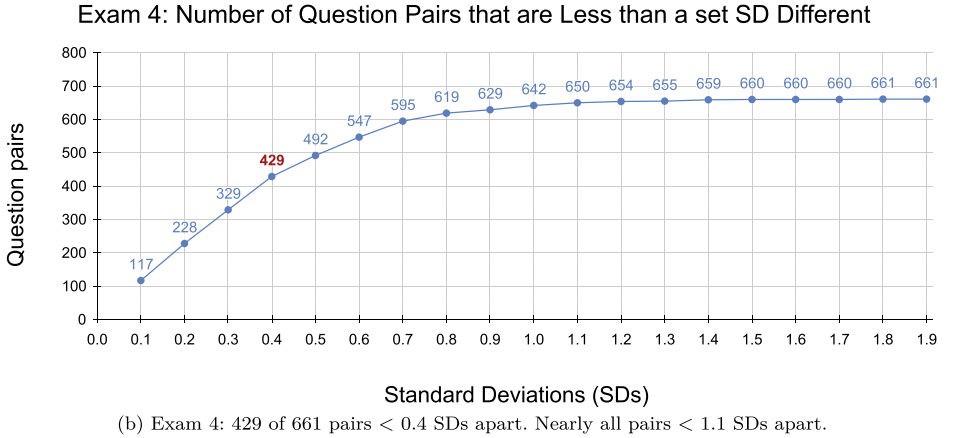
7.2. The balance of the question pools

A potential problem of randomly selecting questions from pools as compared to each student receiving the same questions is whether each question in the pool is of equivalent difficulty. If the difficulty of the questions within pools are not similar, this might cause a problem with the validity of our analysis, as all questions within each pool are supposed to measure the same latent variable. To understand the degree to which our question pools were appropriately balanced, we compared all possible pairs of questions within each pool and determined the difference in standard deviations between each possible pair. A cumulative distribution of these differences is shown in Figures 15(a,b).

While the majority of the questions on the exams were reasonably balanced, there were still some questions of concern. We set a threshold at 0.4 standard deviations, as this generally represents a mean difference of 10% in the score received for a question. For all of the exams, more than half of the question pairs were less than 0.4 standard deviations



(a) Exam 1: 155 of 220 pairs < 0.4 SDs apart. Nearly all pairs < 0.9 SDs apart.
Exam 2: 167 of 266 pairs < 0.4 SDs apart. All pairs < 1 SDs apart.
Exam 3: 132 of 198 pairs < 0.4 SDs apart. Nearly all pairs < 1.1 SDs apart.



(b) Exam 4: 429 of 661 pairs < 0.4 SDs apart. Nearly all pairs < 1.1 SDs apart.

Figure 15. For all of the exams, the number of pairs at or below .4 standard deviations apart in students' scores was above 60%.

different. We further investigated how many pools on each exam had unbalanced questions within them. Specifically, this equates to 4 questions on Exam 1 from 4 pools, 10 questions on Exam 2 from 3 pools, 10 questions on Exam 3 from 6 pools, and 19 questions on Exam 4 from 7 pools.

While a small number of unbalanced question pairs were unlikely to harm our analysis due to sample size, we still wanted to investigate their possible impact. To measure this impact, we dropped each of the 43 problematic questions, one at a time, and reran the analysis. We then determined, on average, how many of the top 5 SEM models from Figure 9 stayed in the top 10 after removing a problematic question. For constrained graphs, where write is not a predictor, on average 4.58 (95% CI [4.32, 4.48]) of the top 5 models remain in the top 10. For all possible models, the average is 4.35 (95% CI [3.93, 4.72]). This gives us confidence that even with some imbalance in some question pools, the impact of individual imbalances was not so large as to threaten our study's results.

8. Conclusion

In this paper, we conducted a quantitative, conceptual replication of Lopez et al. (2008)'s hierarchy of programming skills in introductory computer science. We leveraged both our significantly larger number of data points and SEM to more rigorously explore the existence of this hierarchy of skills. While we did not find strong evidence for the hierarchy suggested by Lopez et al. (2008), our findings do not outright dispute the spirit of those findings. Additionally, among the "best" hierarchies we identify, near neighbors of the original skill hierarchy do appear. Broadly, the body of work on skill hierarchies supports the inter-connected nature of these skills and the value of teaching and assessing multiple skills in the hierarchy. We believe, however, that the community's true interest is understanding the order these skills should be taught in, which we conclude cannot be discovered through this method. Instead, optimal order of instruction should be studied directly as the next step of work in this area.

Note

1. For simplicity of exposition, we maintain BIC over other selection methods, such as bootstrapping (Preacher & Merkle, 2012). Initial explorations with bootstrapping suggest the overall conclusions of our work would not change with another selection criteria.

Disclosure statement

The authors have no relevant financial or non-financial competing interests to report.

Funding

This work was supported by the National Science Foundation [DUE 21-21424].

ORCID

David H. Smith IV http://orcid.org/0000-0002-6572-4347
Matthew West http://orcid.org/0000-0002-7605-0050

References

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika*, *43*(4), 561–573. https://doi.org/10.1007/BF02293814

Bollen, K. A. (1989). *Structural equations with latent variables*. Wiley.

Chen, Z., Chudzicki, C., Palumbo, D., Alexandron, G., Choi, Y.-J., Zhou, Q., & Pritchard, D. E. (2016, April). Researching for better instructional methods using AB experiments in MOOCs: Results and challenges. *Research and Practice in Technology Enhanced Learning*, *11*(1), 9. Retrieved December 30, 2021 from https://doi.org/10.1186/s41039-016-0034-4

Chen, B., Zilles, C., West, M., & Bretl, T. (2019). Effect of discrete and continuous parameter variation on difficulty in automatic item generation. In S. Isotani, E. Millán, A. Ogan, P. Hastings, B. McLaren, & R. Luckin, (Eds.), *Artificial intelligence in education* (pp. 11625). Lecture Notes in Computer Science, Springer, Cham. https://doi.org/10.1007/978-3-030-23204-7-7

Clear, T. (2005). Comprehending large code bases-the skills required for working in a" brown fields" environment. *ACM SIGCSE Bulletin*, *37*(2), 12–14. https://doi.org/10.1145/1083431.1083439

Clear, T., Whalley, J., Robbins, P., Philpott, A., Eckerdal, A., & Laakso, M.-J. (2011 ()). *Report on the final bracelet workshop: Auckland university of technology*. CITRENZ, September 2010.

Corbett, A. T., & Anderson, J. R. (2001). Locus of feedback control in computer-based tutoring: Impact on learning rate, achievement and attitudes. In *Proceedings of the Sigchi conference on human factors in computing systems* (pp. 245–252). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/365024.365111

Corley, J., Stanescu, A., Baumstark, L., & Orsega, M. C. (2020). Paper or ide? the im- pact of exam format on student performance in a cs1 course. In *Proceedings of the 51st ACM technical symposium on computer science education* (pp. 706–712). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3328778.3366857

Corney, M., Fitzgerald, S., Hanks, B., Lister, R., McCauley, R., & Murphy, L. (2014). 'explain in plain English' questions revisited: Data structures problems. In *Proceedings of the 45th ACM technical symposium on computer science education* (pp. 591–596). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2538862.2538911

Cunningham, K., Blanchard, S., Ericson, B., & Guzdial, M. (2017). Using tracing and sketching to solve programming problems: Replicating and extending an analysis of what students draw. In *Proceedings of the 2017 ACM conference on international computing education re-search* (pp. 164–172). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3105726.3106190

Cunningham, K., Ke, S., Guzdial, M., & Ericson, B. (2019). Novice rationales for sketching and tracing, and how they try to avoid it. In *Proceedings of the 2019 ACM conference on innovation and technology in computer science education* (pp. 37–43). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3304221.3319788

Fan, Y., Chen, J., Shirkey, G., John, R., Wu, S. R., Park, H., & Shao, C. (2016, November). Applications of structural equation modeling (SEM) in ecological studies: An updated review. *Ecological Processes*, *5*(1), 19. Retrieved December 28, 2021 from https://doi.org/10.1186/s13717-016-0063-3

Fowler, M., Chen, B., & Zilles, C. (2021). How should we 'explain in plain English'? voices from the community. In *Proceedings of the 17th ACM conference on international computing education research* (pp. 69–80). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3446871.3469738

Gierl, M., & Haladyna, T. (Eds.). (2012). *Automatic item generation: Theory and practice* (1st ed.). Routledge. https://doi.org/10.4324/9780203803912

Grissom, S., Murphy, L., McCauley, R., & Fitzgerald, S. (2016). Paper vs. computer- based exams: A study of errors in recursive binary tree algorithms. In *Proceedings of the 47th ACM technical symposium on computing science education* (pp. 6–11). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2839509.2844587

Guay, R. B., & McCabe, G. P. (1986). A binomial test for hierarchical dependency. *Psychometrika*, *51*(3), 467–474. http://doi.org/10.1007/BF02294067

Harrington, B., & Cheng, N. (2018). Tracing vs. writing code: Beyond the learning hierarchy. In *Proceedings of the 49th ACM technical symposium on computer science education* (pp. 423–428).

Hassan, M., & Zilles, C. (2021). Exploring 'reverse-tracing'questions as a means of assessing the tracing skill on computer-based CS 1 exams. In *Proceedings of the 17th ACM conference on international computing education research* (pp. 115–126).

Haughton, D. M., Oud, J. H., & Jansen, R. A. (1997). Information and other criteria in structural equation model selection. *Communications in Statistics-Simulation and Computation*, *26*(4), 1477–1516. https://doi.org/10.1080/03610919708813451

Jarque, C. M., & Bera, A. K. (1980). Efficient tests for normality, homoscedasticity and serial Independence of regression residuals. *Economics Letters*, *6*(3), 255–259. https://doi.org/10.1016/0165-1765(80)90024-5

Kikuchi, S., & Hamamoto, K. (2016). Investigating the relationship between tracing skill and modification skill for different programming statements. *Proceedings of the School of Information and Telecommunication Engineering*, *9*(1), 8–14. Tokai University.

Koedinger, K. R., Booth, J. L., & Klahr, D. (2013). Instructional complexity and the science to constrain it. *Science*, *342*(6161), 935–937. https://doi.org/10.1126/science.1238056

Kumar, A. N. (2013). A study of the influence of code-tracing problems on code-writing skills. In *Proceedings of the 18th ACM conference on innovation and technology in computer science education* (pp. 183–188).

Kumar, A. N. (2015). Solving code-tracing problems and its effect on code-writing skills pertaining to program semantics. In *Proceedings of the 2015 ACM conference on innovation and technology in computer science education* (pp. 314–319).

Kurland, D. M., Pea, R. D., Clement, C., & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*, *2*(4), 429–458. https://doi.org/10.2190/BKML-B1QV-KDN4-8ULH

Lappalainen, V., Lakanen, A.-J., & Högmander, H. (2017). Towards computer-based exams in CS1. In *Proceedings of the 9th international conference on computer supported education*. SCITEPRESS Science and Technology Publications.

Lin, L.-C., Huang, P.-H., & Weng, L.-J. (2017). Selecting path models in sem: A comparison of model selection criteria. *Structural Equation Modeling: A Multidisciplinary Journal*, *24*(6), 855–869. https://doi.org/10.1080/10705511.2017.1363652

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, *36*(4), 119–150. https://doi.org/10.1145/1041624.1041673

Lister, R., Simon, B., Thompson, E., Whalley, J. L., & Prasad, C. (2006). Not seeing the forest for the trees: Novice programmers and the solo taxonomy. *ACM SIGCSE Bulletin*, *38*(3), 118–122. https://doi.org/10.1145/1140123.1140157

Lister, R., Fidge, C., & Teague, D. (2009). Further evidence of a relationship between explain- ing, tracing and writing skills in introductory programming. *ACM SIGCSE Bulletin*, *41*(3), 161–165. https://doi.org/10.1145/1595496.1562930

Lister, R. (2020). On the cognitive development of the novice programmer: And the development of a computing education researcher. In *Proceedings of the 9th computer science education research conference*. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3442481.3442498

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the fourth international workshop on computing education research* (pp. 101–112).

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., . . . Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. In *Working group reports from iticse on innovation and technology in computer science education* (pp. 125–180).

Mendoza, B., & Zavala, L. (2018). An intervention strategy to hone students' code under- standing skills. *Journal of Computing Sciences in Colleges*, *33*(3), 105–114. http://doi.org/10.5555/3144687.3144714

Moshagen, M. (2020). *sempower: Power analyses for sem [Computer software manual]*. (R package version 1.1.0) https://CRAN.R-project.org/package=semPower

Moubayed, A., Injadat, M., Shami, A., & Lutfiyya, H. (2018, March). Relationship between student engagement and performance in e-learning environment using association rules. In *2018 IEEE World Engineering Education Conference (EDUNINE)* (pp. 1–6). Retrieved December 28, 2021, from http://arxiv.org/abs/2101.02006 (arXiv: 2101.02006)

Murphy, L., McCauley, R., & Fitzgerald, S. (2012). 'explain in plain English'questions: Im- plications for teaching. In *Proceedings of the 43rd ACM technical symposium on computer science education* (pp. 385–390).

Murphy, L., Fitzgerald, S., Lister, R., & McCauley, R. (2012). Ability to'explain in plain English'linked to proficiency in computer-based programming. In *Proceedings of the ninth annual international conference on international computing education research* (pp. 111–118).

Parsons, D., & Haden, P. (2006). Parson's programming puzzles: A fun and effective learning tool for first programming courses. *Proceedings of the 8th Australasian conference on computing education*, *52*, 157–163.

Pelchen, T., Mathieson, L., & Lister, R. (2020). On the evidence for a learning hierarchy in data structures exams. In *Proceedings of the twenty-second Australasian computing education conference* (pp. 122–131).

Perkins, D., & Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers. In *At empirical studies of programmers, 1st workshop* (pp. 213–229). Washington, DC: Ablex Publishing Corp.

Preacher, K. J., & Merkle, E. C. (2012). The problem of model selection uncertainty in structural equation modeling. *Psychological Methods*, *17*(1), 1. https://doi.org/10.1037/a0026804

R Core Team. (2020). *R: A language and environment for statistical computing [Computer software manual]*. https://www.R-project.org/

Rasch, G. (1993). *Probabilistic models for some intelligence and attainment tests*. ERIC.

Royle, G., Mckay, B., Oggier, F., Sloane, N., Wanless, I., & Wilf, H. (2004). Acyclic digraphs and eigenvalues of (0, 1)-matrices. *Journal of Integer Sequences*, *7*(3), online–approx. https://doi.org/10.48550/arXiv.math/0310423

Schooler, L. J., & Anderson, J. R. (1990). *The disruptive potential of immediate feedback*. Proceedings of the Twelfth Annual Conference of the Cognitive Science Society, Cambridge, MA.

Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E., & Whalley, J. L. (2008). Going solo to assess novice programmers. In *Proceedings of the 13th annual conference on innovation and technology in computer science education* (pp. 209–213).

Simon, B., Lopez, M., Sutton, K., & Clear, T. (2009). Surely we must learn to read before we learn to write! *Proceedings of the eleventh Australasian conference on computing education*, *95*, 165–170.

Soloway, E., Bonar, J., & Ehrlich, K. (1983). Cognitive strategies and looping constructs: An empirical study. *Communications of the ACM*, *26*(11), 853–860. https://doi.org/10.1145/182.358436

Soloway, E. (1986, September). Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM*, *29*(9), 850–858. http://doi.org/10.1145/6592.6594

Soloway, E., & Spohrer, J. C. (1989). *Studying the novice programmer*. Psychology Press.

Sudol-DeLyser, L. A. (2015). Expression of abstraction: Self explanation in code production. In *Proceedings of the 46th ACM technical symposium on computer science education* (pp. 272–277).

Tan, G., & Venables, A. (2010). Wearing the assessment 'bracelet'. *Journal of Information Technology Education: Innovations in Practice*, *9*(1), 25–34. http://doi.org/10.28945/707

Teague, D. M., Corney, M. W., Ahadi, A., & Lister, R. (2012). Swapping as the" hello world" of relational reasoning: Replications, reflections and extensions. *Proceedings of conferences in research and practice in information technology (crpit)*, *123*.

Vainio, V., & Sajaniemi, J. (2007). Factors in novice programmers' poor tracing skills. In *Proceedings of the 12th annual Sigcse conference on innovation and technology in computer science education* (pp. 236–240). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1268784.1268853

Venables, A., Tan, G., & Lister, R. (2009). A closer look at tracing, explaining and code writing skills in the novice programmer. In *Proceedings of the fifth international workshop on computing education research workshop* (pp. 117–128).

Weinman, N., Fox, A., & Hearst, M. A. (2021). Improving instruction of programming patterns with faded parsons problems. In *Proceedings of the 2021 chi conference on human factors in computing systems*. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3411764.3445228

West, M., Herman, G. L., & Zilles, C. (2015). *PrairieLearn: Mastery-based online problem solving with adaptive scoring and recommendations driven by machine learning*. Paper presented at the ASEE Annual Conference & Exposition, ASEE Conferences, Seatttle, Washington.

Whalley, J. L., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, P., & Prasad, C. (2006). An Australasian study of reading and comprehension skills in novice programmers, using the bloom and solo taxonomies. *Proceedings of the 8th Australasian conference on computing education*, *52*, 243–252.

Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *ACM Sigcse Bulletin*, *28*(3), 17–22. https://doi.org/10.1145/234867.234872

Wolf, E. J., Harrington, K. M., Clark, S. L., & Miller, M. W. (2013, December). Sample size requirements for structural equation models: An evaluation of power, bias, and solution propriety. *Educational and Psychological Measurement*, *76*(6), 913–934. (25705052[pmid]) https://pubmed.ncbi.nlm.nih.gov/25705052

Xie, B., Nelson, G. L., & Ko, A. J. (2018). An explicit strategy to scaffold novice program tracing. In *Proceedings of the 49th ACM technical symposium on computer science education* (pp. 344–349). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3159450.3159527

Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H., Tan, A. H., Hwa, L., Li, M., & Ko, A. J. (2019). A theory of instruction for introductory programming skills. *Computer Science Education*, *29*(2–3), 205–253. https://doi.org/10.1080/08993408.2019.1565235

Yamamoto, M., Sekiya, T., & Yamaguchi, K. (2011). Relationship between programming concepts underlying programming skills. In *2011 international conference on information technology based higher education and training* (pp. 1–7).

Yamamoto, M., Sekiya, T., Mori, K., & Yamaguchi, K. (2012). Skill hierarchy revised by sem and additional skills. In *2012 international conference on information technology based higher education and training (ithet)* (pp. 1–8).