*mathematics*

MDPI

*Article*

# Exploring the Predictive Potential of Complex Problem-Solving in Computing Education: A Case Study in the Introductory Programming Course

**Bostjan Bubnic, Marjan Mernik * and Tomaž Kosar**

Faculty of Electrical Engineering and Computer Science, University of Maribor, Koroška Cesta 46, 2000 Maribor, Slovenia; bostjan.bubnic@student.um.si (B.B.); tomaz.kosar@um.si (T.K.)
* Correspondence: marjan.mernik@um.si

**Abstract:** Programming is acknowledged widely as a cornerstone skill in Computer Science education. Despite significant efforts to refine teaching methodologies, a segment of students is still at risk of failing programming courses. It is crucial to identify potentially struggling students at risk of underperforming or academic failure. This study explores the predictive potential of students' problem-solving skills through dynamic, domain-independent, complex problem-solving assessment. To evaluate the predictive potential of complex problem-solving empirically, a case study with 122 participants was conducted in the undergraduate Introductory Programming Course at the University of Maribor, Slovenia. A latent variable approach was employed to examine the associations. The study results showed that complex problem-solving has a strong positive effect on performance in Introductory Programming Courses. According to the results of structural equation modeling, 64% of the variance in programming performance is explained by complex problem-solving ability. Our findings indicate that complex problem-solving performance could serve as a significant, cognitive, dynamic predictor, applicable to the Introductory Programming Course. Moreover, we present evidence that the demonstrated approach could also be used to predict success in the broader computing education community, including K-12, and the wider education landscape. Apart from predictive potential, our results suggest that valid and reliable instruments for assessing complex problem-solving could also be used for assessing general-purpose, domain-independent problem-solving skills in computing education. Likewise, the results confirmed the positive effect of previous programming experience on programming performance. On the other hand, there was no significant direct effect of performance in High School mathematics on Introductory Programming.

**Keywords:** computational thinking; introductory programming; predictors of performance; problem-solving; structural equation modeling

**MSC:** 97P30

check for updates

**Citation:** Bubnic, B.; Mernik, M.; Kosar, T. Exploring the Predictive Potential of Complex Problem-Solving in Computing Education: A Case Study in the Introductory Programming Course. *Mathematics* **2024**, *12*, 1655. https://doi.org/10.3390/math12111655

Academic Editor: José Luis Galán-García

Received: 21 April 2024
Revised: 20 May 2024
Accepted: 22 May 2024
Published: 24 May 2024

**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

## 1. Introduction

Programming is considered a fundamental skill in Computer Science education. Apart from Computer Science, the demand for Introductory Programming Courses is growing for non-CS majors [1]. Likewise, in the recent decade, the scope of teaching programming has been extended to K-12 [2]. Despite the significant efforts of researchers and educators to improve teaching methods and learning techniques, learning programming has been identified as difficult, regardless of the educational level. To improve student learning and address reportedly high drop-out rates, much research has been devoted to Introductory Programming (CS1) Courses [3]. Predictors of performance tend to be associated closely with drop-out rates in CS1. The purpose of predictors is to identify potentially struggling students who are at risk of failing or dropping out of the CS1 course. Early identification of non-progressing students based on predictors allows implementation of the appropriate

teaching interventions to support identified students along the CS1 course [4]. However, Luxton-Reilly et al. [5] and Watson and Li [6] argued that the computing education community has generally accepted empirically unsupported claims of high drop-out rates in CS1. Likewise, researchers assert that little empirical evidence was presented in the past decades regarding the potentially high failure rates in CS1 [5,7]. Recently, studies by Bennedsen and Caspersen [8], Watson et al. [9], and Bennedsen and Caspersen [7] found CS1 average pass rates ranging between 67.7% and 72%. Accordingly, they suggest that "an average CS1 failure rate of 28% does not seem alarmingly high" (p. 30, [7]). Nevertheless, the consensus exists that the predictors have an important role throughout computing education. For instance, in the investigations of students' knowledge, their personal characteristics and behavior, and in developing teaching interventions to support students and improve their learning in introductory programming [4,10].

Problem-solving has been at the core of the discipline of Computer Science since its inception. For almost as many years, researchers from computing education and educational psychology have been investigating various aspects of problem-solving in computing, including teaching methodologies and student learning [11]. Historically, the vast majority of problem-solving studies in computing education have examined problem-solving in relation to computer programming, targeting students in introductory programming classes. Among the reasons for this predominance, Deek [12] emphasized the similarities between the processes involved in solving problems and program development. In programming, problem-solving processes have been modeled through various problem-solving frameworks that have emerged within the discipline (e.g., [13,14]), or have been based on George Polya's [15] four-step problem-solving process (e.g., [12,16]). In these frameworks, problem-solving has been observed primarily as a discipline-specific set of cognitive skills employed by the problem solver while performing computer programming tasks. Concerning the relationship between problem-solving and programming, Lishinski et al. [17] identified two clusters of research. In the first, the researchers were particularly interested in problem-solving as a predictor of programming performance. In the second, the area of interest has been the potential causal relationship between computer programming instruction and problem-solving skills. The authors argued that previous studies have focused largely on the potential causal relationship, while research on problem-solving as a predictor of programming performance has been less common. During the past decade, emerging educational movements, including computer science unplugged, computational thinking, and coding in block-based programming environments, have broadened the scope of problem-solving research in computing education from introductory programming to K-12 (e.g., [18–20]). Moreover, there is a great interest of researchers and educators to extend these initiatives to STEM [21]. One of the main driving forces behind these initiatives is that learning programming tends to foster problem-solving skills. Furthermore, computational thinking is considered a problem-solving skill for solving problems in everyday life and work [22]. In line with this perspective, researchers are now particularly interested in how teaching and learning computational thinking influence general-purpose, domain-independent problem-solving skills (e.g., [23,24]). However, some researchers argue that the computing education community lacks empirical investigations that would identify and support the potential causal relationship between learning programming and proficiency in solving problems clearly [13,25,26].

As a result, the primary motivation of the present study was to fill the gap identified by Lishinski et al. [17] concerning the predictors of performance. Namely, the authors advocated that the research focus should be directed toward exploring problem-solving as a predictor of performance in Introductory Programming Courses. Furthermore, this study could also fill the gap identified by Palumbo [13] and Robins et al. [25]. The authors argued that additional research effort is needed concerning the investigation of the potential causal relationship between learning programming and proficiency in solving domain-specific problems in computing, as well as solving domain-independent problems in every life and work.

In this context, the primary objective of this case study is to evaluate if the complex problem-solving (CPS) abilities of introductory programming students could be used as a predictor of their success at the end of the CS1 course. In addition, our study explores whether previous programming experience and performance in High School mathematics have effects on performance in the CS1 course. Finally, we investigate the potential correlation of students' performance in experiments with their final grades in the CS1 course.

According to the results of this exploratory study, 64% of variance in programming performance is explained by complex problem-solving. In addition, our results show that performance in the experiment correlates with a student's final grade positively, and that previous programming experience has a significant effect on programming performance. On the other hand, we could not confirm performance in High School mathematics having a significant effect on introductory programming performance.

Following the results, it seems reasonable to conclude that complex problem-solving performance has the potential to be considered as a significant, cognitive, dynamic predictor, applicable at the beginning of the Introductory Programming Course. This result also has potential implications for teaching and learning in classrooms. Namely, an easy-to-administer complex problem-solving instrument could be applied at the beginning of the course to identify potentially struggling students, as well as high-achievers. In line with the results of the identification, teaching and learning activities in the classroom could be aligned promptly.

The rest of the paper is structured as follows. Section 2 presents a literature review of the predictors of performance and complex problem-solving. Section 3 presents a conceptual model and hypotheses. In Section 4, we present the experiment design, with details regarding the participants, instruments, and measures used in the study. In addition, this section presents the methodology for hypothesis testing and provides details about the sample size. Section 5 presents the experimental results and data analysis. Section 6 discusses the results of the study and also presents the implications in the respective subsection. Section 7 presents potential threats to validity, including construct, internal, and external threats. In the final Section, we summarize our work and broaden the horizons for some future directions.

## 2. Literature Review

The following Section reviews the literature on the predictors of performance in programming, as well as in the broader education landscape. In addition, the concept of complex problem-solving is introduced.

### *2.1. Predictors of Academic Performance*

Predictors of performance have a long tradition in introductory programming research. By tracing the literature on predicting student success in CS1, Evans and Simkin [27] identified three cluster intervals of research, spanning from the mid-1970s to the present day. According to Cafolla [28], the predictors of performance in programming originate from various aptitude tests that were introduced in the early 1970s, including the "Programmers Aptitude Test" and the "Aptitude Test for Programming Personnel" (ATTP). ATTP was designed exclusively for IBM customers and was in use until the late 1980s. During the early years, assessing aptitude was limited to professionals working with computers [28]. However, the ATTP was also employed in one of the earliest empirical investigations concerning predicting performance in computing education. The study was conducted by Bauer et al. [29] in 1968. In addition to ATTP, the authors employed three more independent variables and concluded that grade point average was found to be the best predictor. Furthermore, Cafolla [28] cited additional studies that investigated the predictive power of various abilities in computing education during the mid-1970s and 1980s. They included cognitive style, the ability to organize information, verbal and mathematical reasoning, and grade point average. In the mid-1980s, scholars began to explore the relationships between

programming abilities and general cognitive processes, which led to the introduction of additional predictors, including abstraction ability and cognitive style [27].

Different categorizations have been introduced throughout the history of studying predictors in CS1. Bergin and Reilly [30] categorized predictors into the following: (1) previous academic and computer experience, with emphasis on mathematics and programming; (2) cognitive factors; (3) psychological factors, with emphasis on the comfort-level of the course; (4) self-regulated learning. Watson et al. [9] distinguished between traditional and dynamic predictors. They considered traditional predictors as test-based, relying on psychological tests or background questionnaires that evaluated students' trait-type factors. Dynamic predictors, often referred as to data-driven predictors, evaluate aspects of students' programming behavior by applying statistical analysis continuously on log data collected throughout the CS1 course. While reviewing the literature on learning analytics and educational data mining in programming, Ihantola et al. [31] identified several systems for data collection, including automated grading systems, integrated development environment instrumentation, version control systems, and key logging. In terms of prediction timing, Quille and Bergin [4] categorized predictors into prior CS1, early in CS1, and throughout the CS1 course. In terms of prediction accuracy, they distinguished between significant and non-significant predictors.

#### 2.1.1. Previous Academic Achievement

Previous academic achievement is often cited as an indicator of success in the CS1 course. Researchers have been particularly interested in the influence of mathematics ability and previous exposure to mathematics on performance in CS1. While several studies found mathematics to be a significant predictor (e.g., [32–34]), there are also studies that identified a weak, or no effect (e.g., [9,35]). In addition to mathematics, which tends to be the most extensively studied, studies also investigated the predictive power of achievements in biology, chemistry, physics, natural language (mother and foreign variants), and grade point average (e.g., [9,35,36]).

#### 2.1.2. Previous Programming Experience

The existing literature has largely investigated the impact of previous programming experience on performance in CS1. According to Wiedenbeck [37], this relationship was the most frequently mentioned factor in the literature. Many studies confirmed a significant predictive power of previous exposure to programming (e.g., [9,27,32,38,39]), but some studies failed to demonstrate a significant relationship (e.g., [33,35,40]). Furthermore, Hagan and Markham [38] discovered that the number of programming languages a student had previously engaged with correlated with CS1 performance. On the other hand, Watson et al. [9] argued that, in general, specific aspects, such as years of experience or the number of programming languages studied, have little impact on performance. In addition to the previous programming experience, Bergin and Reilly [33] and Wilson and Shrock [32] provided references to studies that also investigated the relation of previous non-programming computer experiences to performance in CS1, such as experience with playing computer games, using internet applications, and the usage of productivity software.

#### 2.1.3. Psychological Factors

The role of psychological factors in predicting programming performance has also received substantial attention. By drawing from research in Cognitive Psychology, researchers addressed the following: (1) the predictive power of abstraction ability [41,42]; (2) spatial ability [39,43]; (3) cognitive style [27,44]; (4) problem-solving [17,45–48]. In educational psychology, researchers investigated whether performance in programming can be predicted by students' intellectual development based on various versions of Piagetian reasoning tests [49]. In addition, researchers also examined the predictive capacity of students' learning [36,50]. Finally, past research has also been devoted to the predictive potential of noncognitive factors. While studies by Wiedenbeck [37] and Askar and Davenport [51]

evaluated the impact of self-efficacy beliefs, Bergin and Reilly [52] conducted research regarding motivation and comfort-level. Likewise, Bennedsen and Caspersen [53] addressed the predictive potential of self-esteem, attitudes towards perfectionism, affective states and optimism. Last but not least, when considering a broad range of factors, Quille [54] found programming self-efficacy to be the most prominent predictor. Likewise, a study by Watson et al. [9] found programming self-efficacy to be the strongest predictor among the traditional predictors.

### *2.2. Complex Problem-Solving*

Complex problem-solving is a well-established field in Cognitive Psychology [55]. It evolved from research on general intelligence in the 1980s, when researches asserted that traditional intelligence tests were bad predictors of one's performance on partially intransparent, ill-structured complex problems [56]. Complex problems can be characterized as sets of interactive, dynamic, and previously unknown tasks the problem solver needs to solve. In this context, complex problem-solving can be defined as a non-routine analytical skill involving domain-general mental processes that are required to solve diverse complex problems [57].

According to Funke and Frensch [58], two trends can be observed in complex problem-solving research. In Europe, the focus has been on general problems that were investigated in laboratory environments. General problems exist in everyday life and work, where humans are faced with novel, unknown and dynamically changing situations involving ill-structured problems. Solving a complex, ill-structured problem usually incorporates a certain level of uncertainty about necessary rules, principles, and methods (see [59] for details on ill-structured problems). In North America, the focus has been on problem-solving in specific, natural knowledge domains, such as chemistry, physics, and mathematics, and on its transferability to other domains. Nowadays, complex problem-solving is considered one of the most important cross-functional, work-related skills [60]. Furthermore, Phebe Novakovic, a senior Chief Executive Officer of a large US corporation, emphasized how senior leadership teams in corporations are challenged repeatedly with highly complex problems. She pointed out that one's ability to solve those complex problems is an important element of day-to-day businesses [61].

Systematic research of complex problem-solving was introduced in Germany in the 1970s, where cognitive abilities for solving complex problems were modeled in microworlds [55]. A microworld is a computer-simulated, dynamic system, designed for assessing one's general problem-solving abilities. In order to solve a problem, the problem solver needs to: (1) gather the knowledge from within the microworld environment; (2) apply the acquired knowledge successfully [62]. The complex problem-solving literature often cites the "Lohhausen" microworld as one of the first simulated environments. The "Lohhausen" required participants to manage a small town by manipulating city policies, such as taxes, working conditions, and leisure time activities. In "Lohhausen", these activities were implemented by more than 2000 highly related variables that a problem solver needs to alter [63].

Recent advances in complex problem-solving research have led to several complex problem-solving instruments that measure cognitive abilities within interactive, computer-based environments. According to Greiff et al. [64], they can be categorized as: *classical complex problem-solving measures* and *multiple complex systems*. Classical problem-solving measures are implemented in microworlds. They simulate real-life problem situations, such as managing a retail business (*Tailorshop*), controlling a power plant (*Powerplant*), governing a small town (*Lohhausen*), or coordinating fire-fighters (*Winfire*) (for review, see [65]). While classical complex problem-solving measures tend to be a good performance predictor in real-world settings (ecological validity), their psychometric properties tend to be weak. On the other hand, multiple complex systems are instrumented through multiple small tasks, integrated into a particular problem-solving assessment. The well-known problem-solving assessments based on the multiple complex systems' paradigm are *Genetics Lab*, *MicroDYN* and *MicroFIN* (for a review, see [65]). While multiple complex systems tend to be a valid

and psychometrically acceptable alternative to classical complex problem-solving measures, they have shortcomings related to ecological validity [65].

## 3. Conceptual Model and Hypothesis Development

The purpose of this study was to explore the predictive potential of complex problem-solving ability in the Introductory Programming Course. In this respect, the students' performance in complex problem-solving is considered as an independent variable, while performance in programming is considered as a dependent variable. In addition, the students' final grades in the Introductory Programming Course are employed as a dependent variable. Previous programming experience and the students' matriculation grades in mathematics are used as control variables. It should be noted that matriculation examination in Slovenia is taken at the end of secondary education, in order to qualify for entry into post-secondary education. Let us also note that, in line with most references, we refer to *High School mathematics* when addressing performance in secondary mathematics throughout this paper.

In a broader problem-solving context, our conceptual model follows a multidimensional characteristic of the term competence proposed by Weinert [66], as well as problem-solving conceptualization by Funke et al. [67]. Funke et al. [67] considered problem-solving competence as an umbrella term for a collection of cognitive and noncognitive skills, abilities or aptitudes, as well as personal characteristics. In this broader context, problem-solving performance is a result of the synergistic interaction of the aforementioned factors. Furthermore, our conceptual model relies on previous studies in the domain of mathematics, where problem-solving has been investigated extensively [68–70]. Accordingly, this study considers problem-solving competence, which is presented in Figure 1, as a collection of cognitive, metacognitive and affective components. The *cognitive* component refers to the cognitive aspects of problem-solving, including knowledge, cognitive operations, and the cognitive processes and skills employed by problem solvers while solving problems [71]. The *metacognitive* component refers to activities and processes employed by problem solvers while setting goals, and while monitoring, regulating and controlling their cognitive-affective processes while solving problems [72,73]. In line with McLeod [74], the *affective* component encompasses attitudes, beliefs, and emotions (for a description and definition of terms, see [75], p. 259). In addition, other researchers have proposed extensions of the initial McLeod's framework that include affective components such as value, motivation, and engagement (e.g., [76]).

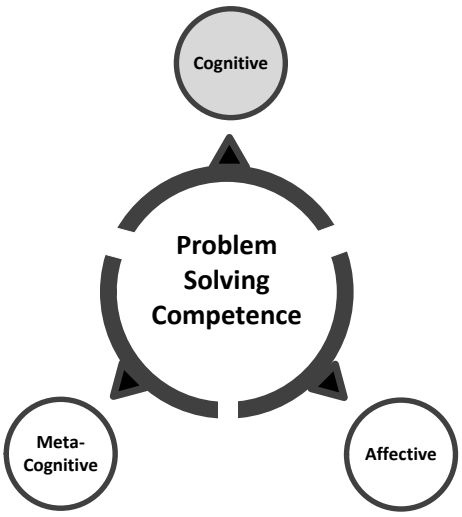


**Figure 1.** Multidimensional model of the problem-solving competence indicating how this study addressed only the cognitive portion of the broader competence.

In line with the aforementioned conceptual model, our study considers complex problem-solving *ability* as a set of cognitive skills, which addresses the cognitive-only portion of problem-solving competence. Likewise, we refer to problem-solving *competence* when addressing the broader set of factors.

Performance in programming is known to be affected by several factors, including knowledge, cognitive abilities, previous programming experience, academic performance, attitudes toward programming, beliefs about programming, and emotions [77,78]. In this context, our conceptual model incorporates knowledge, cognitive abilities, previous programming experience, and performance in High School mathematics as relevant measures of performance in programming.

*Hypotheses*

The present study is guided by the following hypothesis:

- H1. *Complex problem-solving ability has a direct positive effect on performance in introductory programming*.
  Problem-solving has been recognized as one of the core competencies in Computer Science since its inception. For almost as many years, researchers from Computer Science education and from Educational Psychology have been investigating various educational aspects of problem-solving in computing, including teaching methodologies and differences in student learning [11]. In a recent literature review on the teaching and learning of introductory programming, Medeiros et al. [79] argued that problem-solving was the most frequently cited necessary skill in addition to mathematical ability. However, some researchers argue that the computing education community lacks empirical investigations that would identify clearly and explain the potential causal relationship between problem-solving ability and programming performance [13,26]. Regarding empirical investigations, mixed results have been reported. A study by Lishinski et al. [17], which implemented a domain-independent approach in the form of paper–pencil tasks, indicated that problem-solving ability correlates significantly with success in CS1 assessment tasks. On the other hand, a study by Veerasamy et al. [48], which implemented a domain-independent approach in the form of a questionnaire, identified a weak relationship between students' perceived problem-solving skills and their performance in CS1 assessment tasks.
- H2. *Students' performance in the experiment correlates with their final grade in the Introductory Programming Course.*
  Recent developments in education research advocate that increased student engagement, motivation and course participation have a positive effect on academic outcomes [80]. Based on their literature review of gamification in education, Zeybek and Saygı [80] argued that the use of gamification, which is defined as the use of game elements in non-game environments, has been investigated mostly in programming, language education, and engineering. Regarding empirical investigations in introductory programming, Imran [81] investigated different levels of gamification, and confirmed that the gamification level was a significant determinant of motivation and performance, but not engagement. Similarly, Kučak et al. [82] identified that students lectured using gamification elements performed significantly better.
  Following the gamification principles, our CS1 course design stimulates students to earn credit points for non-mandatory tasks and programming challenges, for participating in experiments, and for course presence. Whenever possible, the credit points are not given per se, but are calculated based on their performance in a particular activity. As a consequence, the final course grade used in this study is the result of several indicators, including midterm and final exam grades, performance in two experiments, and course presence. Thus, the aim of testing this hypothesis was to determine to what extent such a broadly defined final grade could be explained by students' performance in our experiments.

- H3. *Previous programming experience has a direct positive effect on performance in the Introductory Programming Course.*
  Research in Educational Psychology has shown that prior knowledge in specific content domains, including mathematics, physics and economy, contributes to students' learning and achievement [83]. In computing education, there is a general agreement that previous exposure to programming influences students' performance in CS1 [84]. Furthermore, Wiedenbeck [37] reported that the relationship between previous programming experience and performance in CS1 was the most frequently mentioned factor in the literature. Mixed results have been reported regarding empirical investigations. While Bockmon et al. [39] and Watson et al. [9] confirmed a significant relationship, studies by Ventura Jr [35] and Kangas et al. [40] failed to confirm a significance.
- H4. *Performance in High School mathematics has a direct positive effect on performance in the Introductory Programming Course.*
  Research involving mathematics as a predictor of success incorporates various aspects, including previous exposure to mathematics courses, performance in those courses, as well as achievements in mathematics [35]. In addition to performance, Trujillo-Torres et al. [85] investigated factors of "Mathematics Learning", including affinity, teaching, study time, didactic resources employed, study aids, and motivation. Moreover, Toland and Usher [86] advocated that mathematics self-efficacy tends to be the most extensively studied predictor in mathematics education. Regarding empirical investigations, Shaffer [87] provided references to four studies, where the grade point average in High School mathematics was a significant predictor of performance in CS1. Likewise, Bergin and Reilly [52] discovered a strong correlation between High School mathematics matriculation grade and CS1 performance. However, Fan and Li [88] found average scores in High School mathematics to be a weak predictor of performance in CS1.

## 4. Research Design

The following Section highlights the experiment procedure, reports on sample characteristics and sample size estimation, and describes the instruments used in our empirical study.

### *4.1. Experiment Procedure*

The experiment procedure consisted of two parts. In the first part, Genetics Lab [89], a valid and reliable problem-solving assessment, was conducted at the beginning of the Introductory Programming Course. Students were asked to participate voluntarily and enroll in a designated group based on their scheduling constraints. Each group was administered individually in the research lab. In total, 13 groups were administrated. All the students' activities in the lab were completed using computers. Before entering the Genetics Lab, the students were asked to fill out a background questionnaire about their demographic information, achievement in High School mathematics, and prior programming experience. The Genetics Lab is a custom application that incorporates instruction and assessment episodes. During the interactive instruction episode, the students became familiar with the assessment environment. Such a procedure ensured that each group received the same instructions. The instruction episode was not limited in time. After all the students finished the instruction episode, an instructor was available for any additional explanations. After that, a password was provided by the instructor for the students to enter the assessment episode simultaneously. In the assessment episode, the students were presented with 12 interactive problem-solving tasks. The assessment episode was limited to 35 min. The students spent approximately 47 min completing both episodes (M = 46.59; SD = 9.25).

In the second part of the experiment, a Second CS1 Assessment (SCS1) [90] was planned to be administered at the end of the CS1 course. However, due to the COVID-19 restrictions, the Faculty was closed in the middle of the first semester. Therefore, the CS1 course switched to fully online for the remainder of the semester. As it was expected that

COVID-19 restrictions might be lifted at the beginning of the second semester, the SCS1 was rescheduled to the beginning of the second programming (CS2) course. Unfortunately, the restrictions were not lifted. Accordingly, SCS1 was administered remotely, at the beginning of the second semester, in the form of a single-session online delivery. The unplanned interval extension between Genetic Lab and SCS1 was approximately six weeks (including the four-week winter break). SCS1 was implemented as an online questionnaire in the Qualtrics Survey Software [91], and therefore, the students were able to access it via a web browser on computers, smartphones, or tablets. Twenty-seven assessment tasks in SCS1 are provided in the pseudocode. Before beginning, the students were provided with pseudocode examples in a 10-min demonstration session. The session was conducted by the instructor, who presented predefined examples from the SCS1 suite. The duration of the SCS1 was limited to 120 min. Students spent approximately 85 min completing the SCS1 (M = 85.33; SD = 19.90).

The experiment procedure is presented in Figure 2.

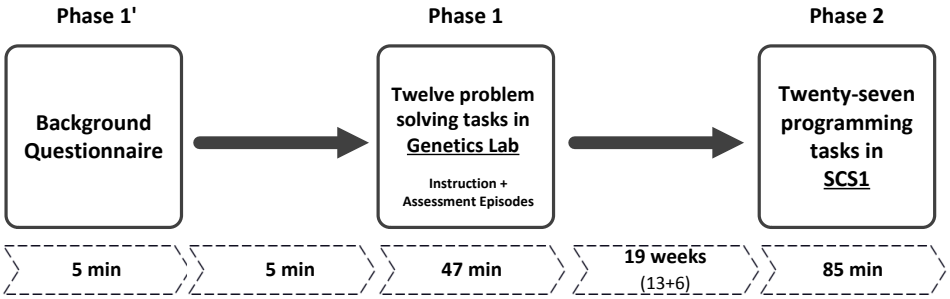


**Figure 2.** Experiment conduction protocol.

### *4.2. Participants*

The study was carried out in the Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia, in the academic year 2020–2021. In total, 122 students (N = 122; 11 females; 0 non-binary; age: M = 18.97; SD = 1.82), enrolled in a first-year Introductory Programming Course, and participated in this study voluntarily. Before the study, the students were provided with a consent form explaining the purposes of the experiment. Following practices identified by Sjøberg et al. [92], the students received course credit for their participation. In particular, based on their overall performance in the experimental tasks, the students could earn up to 10% of their CS1's final grade.

### *4.3. Instruments and Measures*

The Genetics Lab and SCS1 assessments were employed in the present study. Both were presented to students in their native language.

#### 4.3.1. Instrument Adjustments and Pilot Study

Genetics Lab is available in English, German and French language versions. It was translated from English and German to Slovenian. The instrument is only available in the form of a custom application in Adobe Flash technology. Because Adobe Flash [93] is not supported by recent versions of web browsers, some technical adjustments were necessary. The initial data collection process was straightforward, but extracting results from data files into spreadsheets required a basic knowledge of R language [94]. For potential large-scale executions, additional adjustment efforts may be necessary: (1) because of Adobe Flash; (2) because R script has to be executed for every participant.

Originally available in the English language, SCS1 was also translated to Slovenian. The instrument was implemented as a Qualtrics Online Survey [91]. While the data collection process was straightforward, converting data from the survey system into spreadsheets required a paid subscription. As an alternative, the authors provided assessment tasks as a

PDF file, which opens the possibility for full customization. No limitations were identified for the potential large-scale execution.

In order to avoid technical difficulties during the experiment, and to examine and refine the experiment procedure, a pilot study was conducted in the Computer Science lab a week before each part of the experiment. Both assessments were piloted by Computer Science teaching staff (N = 5). In addition, special attention was given to the accuracy of translations during the pilot studies.

#### 4.3.2. Complex Problem-Solving

We employed Genetics Lab to measure complex problem-solving ability in this study. Genetics Lab has been recognized as one of the most prominent computer-based, complex problem-solving performance assessments in Cognitive Psychology [95]. From a theoretical perspective, Genetics Lab falls into the category of multiple complex systems, where general problem-solving ability is measured through multiple small problem-solving tasks. In particular, the ability is evaluated through knowledge acquisition and knowledge application processes (see [57,96] for details). Multiple complex systems tend to be valid and psychometrically acceptable complex problem-solving assessments [95,97]. In the Genetics Lab, complex problem-solving processes are represented by a three-dimensional model, where exploration behavior, knowledge acquisition and knowledge application are the relevant problem-solving performance measures [89].

Twelve problem-solving tasks are incorporated in the Genetics Lab application, where participants are engaged with the genes of fictitious creatures and their influence on the creatures' physical characteristics (see [89] for details). A sample task is presented in Figures 3–5. Each task is divided into two phases. In the first phase, the participants explore the relation of genes and the creature's characteristics (investigation process), as well as document the results of the investigation process (documentation process).

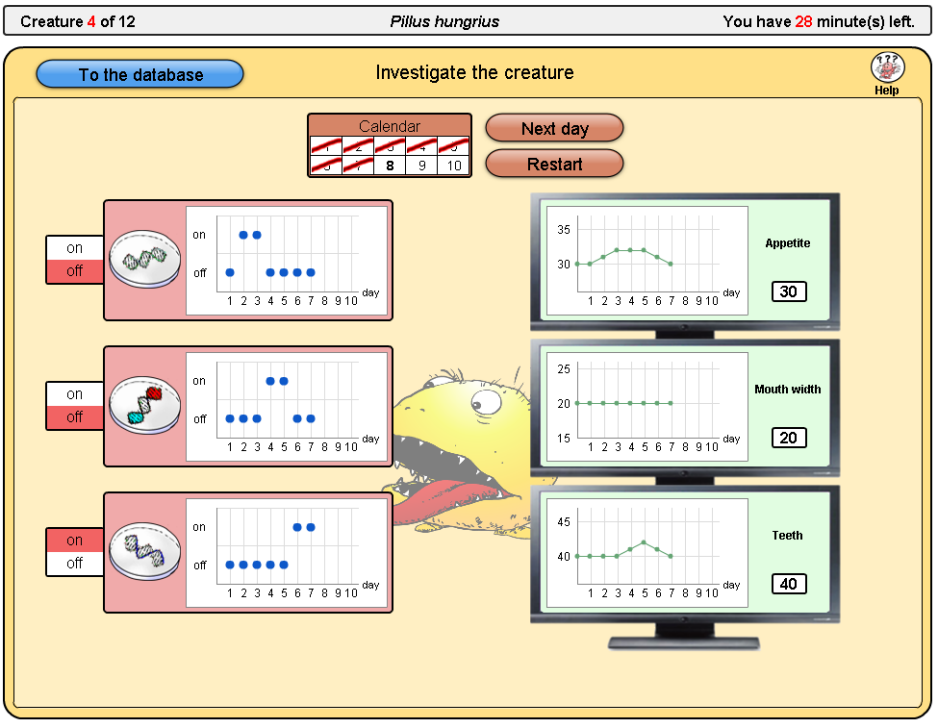


**Figure 3.** Screenshot of the Genetics Lab [89] user interface dedicated to the investigation process.

In the investigation process, the participants explore the impact of genes on the creature's characteristics. For instance, the participants investigated the impact of three genes on appetite, mouth width, and teeth length in the investigation process presented in Figure 3. The impact of each gene is manipulated with "on" and "off" switches, while the

effect on specific characteristics can be observed graphically. A particular gene can have a positive, negative, or no effect on a creature's characteristics. For instance, none of the genes affect the creature's mouth width in Figure 3. To identify as many relations as possible, an unlimited number of steps are available within the investigation process. After a relation has been identified, it has to be recorded in the database during the documentation process. In this process, the participants register relations between the genes and characteristics as a set of arrows pointing from a gene to a characteristic. In addition, the direction of the effect has to be indicated (positive or negative). The result of the documentation process is presented in Figure 4.

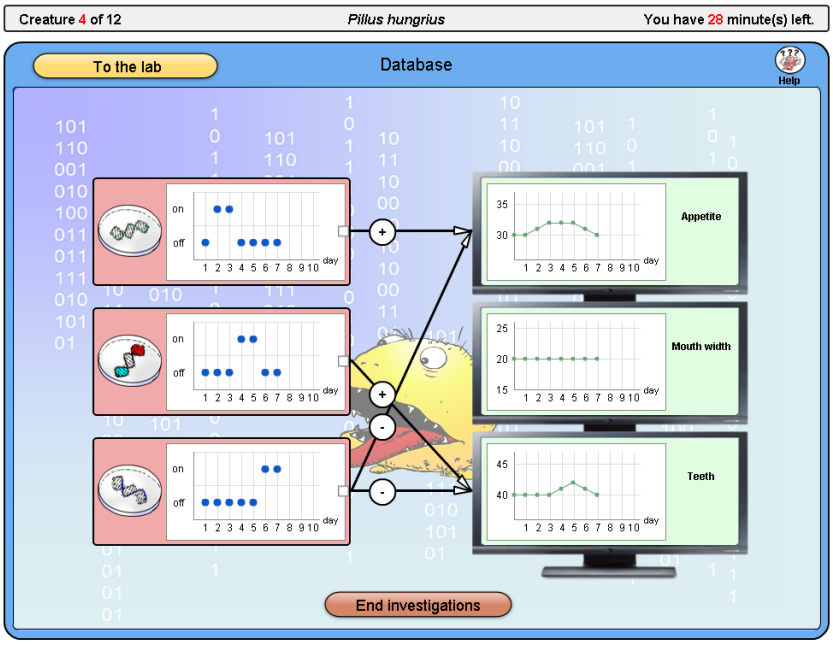


**Figure 4.** Screenshot of the Genetics Lab [89] user interface dedicated to the documentation process.

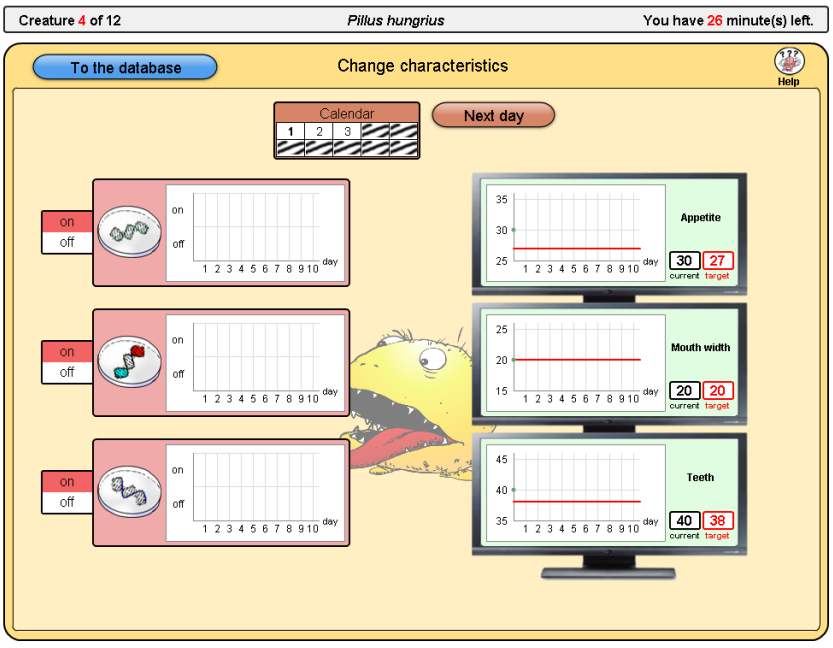


**Figure 5.** Screenshot of the Genetics Lab [89] user interface dedicated to solving the task.

In the final phase, knowledge about previously identified relations has to be applied in order to solve the task. In particular, the participants must manipulate the genes in order

to achieve the target values. For instance, the genes in Figure 5 have to be manipulated in such a way that appetite would decrease from 30 to 27, while teeth length would decrease from 40 to 38. Likewise, the mouth width has to remain unchanged. All the target values have to be achieved within only three steps. During the final phase, the students can utilize previously recorded relations between the genes and physical characteristics as support while solving the task. The problem-solving performance score is presented at the end of each task. Likewise, the overall performance is presented at the end of the assessment.

During problem-solving activities, the participants' behavior is recorded continuously into log-files. In the end, the problem-solving performance scores can be calculated based on a log-file analysis. The log-file analysis is performed by script in the R language, and the script is part of the Genetics Lab suite. The script transforms log-files into text formatted results (problem-solving performance measures), which can be manipulated further with spreadsheet software.

In line with the Genetics Lab documentation, the relevant problem-solving performance measures in our experiment were rule identification (*GL.RI*), rule knowledge (*GL.RK*), and rule application (*GL.RA*) (see [95] for details).

Several studies have been employed to investigate the psychometric properties of the Genetics Lab. Let us highlight some of the initial examinations. The instrument was initially evaluated in 2012, when two studies were conducted. The internal consistency measured by Cronbach's $\alpha$ coefficient for the first study ranged from 0.80 to 0.94 (RI = 0.94, RK = 0.89, RA = 0.80, N = 43), while the Cronbach's $\alpha$ for the second study ranged from 0.61 to 0.88 (RI = 0.88, RK = 0.77, RA = 0.61, N = 61) [89]. The properties were assessed further on a larger sample in 2013, when the Cronbach's $\alpha$ ranged from 0.79 to 0.91 (RI = 0.91, RK = 0.90, RA = 0.79, N = 563). According to Sonnleitner et al. [95], all three problem-solving measures showed satisfactory reliabilities.

#### 4.3.3. Computer Programming Performance

We employed a Second CS1 Assessment (SCS1) to measure the computer programming performance in this study. SCS1 falls into the category of concept inventory assessments. A concept inventory notion originates from physics, where the Force Concept Inventory was designed to measure student knowledge regarding the force concept from Newton's Mechanics [98]. More generally, the idea of a concept inventory is to measure the students' understanding of the most crucial concepts of the course, instead of assessing everything that is covered in the particular course summatively [99]. According to Webb et al. [100], concept inventory is delivered as a standardized, multiple-choice assessment that is easy to administer and grade. Likewise, it must be demonstrated that it measures the intended components (validity), and that these components can be measured consistently under the different conditions (reliability). Motivated by the success of Force Concept Inventory, numerous concept inventory assessments have been designed across the STEM disciplines, including Computer Science, mathematics, physics, chemistry, and biology, as well as in statistics and nursing [99,100].

SCS1 tends to be the most developed, validated instrument for evaluating CS1 knowledge across multiple programming languages [101]. To be independent of a programming language or pedagogical paradigm, SCS1 was built in pseudocode. It was designed to include a broad spectrum of CS1 concepts, including programming fundamentals, logical operators, conditionals, loops, arrays, functions, and recursion. These concepts were incorporated in three different types of questions: definitional (D), code tracing (T), and code completion (C) (see [90] for details on SCS1).

SCS1 has been validated for a first-semester undergraduate Computer Science course and tends to be used best when administrated at the end of the course [102]. It incorporates 27 questions in which participants chose from one of the five possible answers in a multiple-choice format. A sample question is not presented, because the instrument was not released publicly. For the computing education community, it is available upon request [102]. The recommended duration is 60 min. Due to the recognized difficulty of the assessment,

several approaches have been identified in the literature to mitigate this issue, including shortening the assessment or duration extension [102]. Accordingly, duration extension was utilized in our study.

Several studies have investigated the psychometric properties of SCS1, which employed different measures of statistical validity. Regarding the internal consistency, the initial validation study resulted in a Cronbach's $\alpha$ coefficient of 0.59 (N = 183), which was below the acceptable level of 0.7 [103]. More recently, Parker et al. [104] analyzed data from a larger portion of different SCS1 executions. They found a Cronbach's $\alpha$ coefficient of 0.78 (N = 547). Xie et al. [105] extended the validation of SCS1 by employing the Item Response Theory [106]. According to their findings, four questions were too difficult for the students in their sample (N = 489), while three questions measured knowledge that was unrelated to the rest of SCS1. The proposed measures to tackle the recognized difficulty included question refinement, shortening of the assessment, or duration extension [102,105].

SCS1 is implemented in a Qualtrics Online Survey [91]; therefore, the data collection process was very straightforward. The students' scores are instantly available in various formats, including text and spreadsheets. Based on the students' scores, and in line with the question categorization in the SCS1 documentation, the relevant programming performance measures in our experiment were definitional (*SCS1.D*), code tracing (*SCS1.T*) and code completion (*SCS1.C*). *SCS1.D* explores a student's general understanding of the construct. *SCS1.T* examines the student's ability to predict the execution of a particular code. *SCS1.C* evaluates the student's ability to write code [90].

#### 4.3.4. Background Variables

A background questionnaire was employed to gather the students' demographic information, including gender and age. In addition, the questionnaire also collected the students' previous programming experience and achievements in High School mathematics as predictors of performance in CS1. The predictors were selected from a list of traditional predictors of programming performance compiled by Watson et al. [9]. Regarding previous programming experience, the students rated their agreement with statements concerning the number of projects they had worked on previously, regardless of the programming languages used. The questionnaire is presented in Figure 6. Regarding mathematics, the students reported their final scores from the matriculation exam. In a separate form, we asked the students for consent to access their CS1 course results.

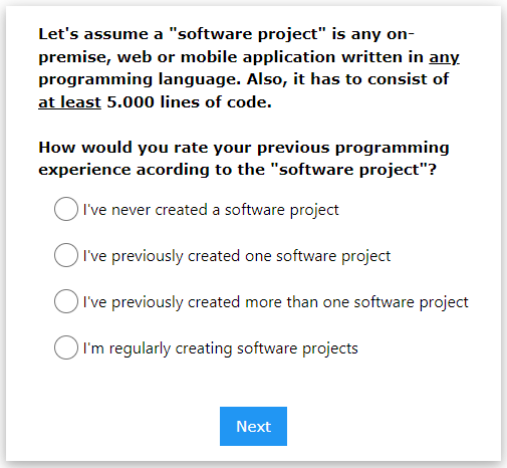


**Figure 6.** Screenshot of the self-report questionnaire where students rated their previous programming experiences.

### *4.4. Methodology for Hypothesis Testing*

Covariance-based structural equation modeling (CB-SEM) with a statistical software package AMOS 26 (Analysis of Moment Structures) was used for assessing a measurement

model and for hypotheses testing [107]. A two-step approach was used in line with Anderson and Gerbing [108]. In the first step, a measurement model (confirmatory factor analysis) was proposed, where complex problem-solving (*CPS*) was constructed as a latent variable with three manifest variables, namely, rule identification (*GL.RI*), rule knowledge (*GL.RK*), and rule application (*GL.RA*). Likewise, performance in the Introductory Programming Course (*P1*) was constructed as a latent variable with three indicators, namely, definitional (*SCS1.D*), code tracing (*SCS1.T*), and code completion (*SCS1.C*). The final grade in the Introductory Programming Course (*GRADE*) was introduced into the model as a latent variable with a single indicator, where the residual variance was set to 0. Previous programming experience (*PProg*) and achievements in High School mathematics (*PMath*) were introduced as controls, and were measured with a single variable name. In the second step, a structural equation model was proposed, its model fit was assessed, and the proposed hypotheses were tested.

### *4.5. Sample Size*

During the experiment planning, a Monte Carlo simulation was utilized for sample size estimation [109,110]. The simulation was implemented using a simsem package for R [111]. The factor loadings for constructing a data generation model were gathered from previous studies by Sonnleitner et al. [95] (*GL.RI* = 0.68, *GL.RK* = 1, *GL.RA* = 0.94; cluster 1: N = 300, 146 females, M = 15.6, SD = 0.75; cluster 2: N = 263, 138 females, M = 17.4, SD = 0.75) and Xie et al. [105] (*SCS1.D* = 0.41, *SCS1.T* = 0.34, *SCS1.C* = 0.37; N = 118, 58 females, 87 students aged 18–22 years, 31 students older than 22). According to the results of the simulation, 112 participants tend to be an adequate sample size to ensure proper model convergence and accurate parameter estimations.

The sample was gathered from a population of nearly 200 introductory programming students, where the participation was voluntary. One hundred and thirty-two students (N = 132) participated in the first part of the experiment, while we had 164 participants (N = 164) in the second part. Only students who attended both parts (N = 122) were included in the analysis.

## 5. Experimental Results and Data Analysis

A latent variable approach was used to examine the relations between *CPS*, *P1* and *GRADE*. During the data analysis, we followed the Brunswick symmetry principle [112], which suggests studying relations between latent constructs at the same level of aggregation or abstractness. In their study, the authors demonstrated that an aggregated performance measure was best predicted by an aggregated knowledge measure. Thus, *CPS* and *P1* were introduced as global measures, and were examined on the same level of aggregation.

### *5.1. Measurement Model*

We used confirmatory factor analysis (CFA) to investigate the validity of the measurement model [108,113]. The model parameters (factor loadings, latent correlations) to be estimated, as well as their estimates, are presented in Figure 7.

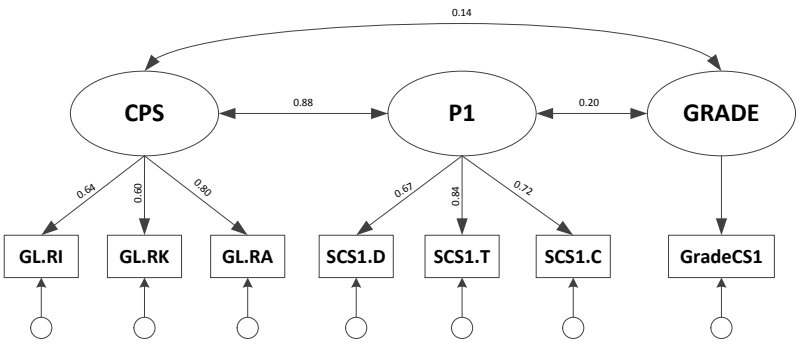


**Figure 7.** Standardized parameter estimates for the measurement model using CFA.

The results from the CFA revealed that the proposed model fitted the data well. This is reflected in the $\chi^2$ value ($\chi^2(12) = 14.132$), which is insignificant at $p > 0.05$, and indicates a good fit. Furthermore, additional fit indices were used, including GFI = 0.97, RMR = 0.005, NFI = 0.949, IFI = 0.992, TLI = 0.985, CFI = 0.992, and RMSEA = 0.038. GFI, NFI, IFI, TLI, and CFI were all well above 0.90. The RMSEA was lower than 0.08, and the RMR was lower than 0.1. In line with the representative literature (e.g., [114,115], such values also pointed toward a good fit of the model. The measurement model fit indices are presented in Table 1.

The standardized factor loadings, average extracted variance (AVE) and composite reliabilities (CR) are presented in Table 2. The factor loading estimates were all higher than 0.6, varying from 0.60 to 0.84. Different factor loading thresholds have been proposed in the representative literature (e.g., [116]. Given the exploratory nature of this study and considering our sample size (N = 122), the factor loading threshold of 0.6 is acceptable. The AVE was above 0.5 for computer programming, and slightly below 0.5 for complex problem-solving. According to Fornell and Larcker [117], AVE should not be lower than 0.5 to demonstrate an acceptable level of convergent validity. While an AVE of 0.560 indicates the appropriate convergent validity of computer programming, the potential consequences for complex problem-solving (AVE = 0.472) are discussed in Section 7.3. Finally, CR was higher than the suggested threshold of 0.7 for the exploratory studies [118], pointing toward the reliability of the constructs.

**Table 1.** Model fit indices for both the structural and measurement models.

| | $\chi^2$ | *df* | *p* ($\chi^2$) | GFI | RMR | NFI | IFI | TLI | CFI | RMSEA |
|---|---|---|---|---|---|---|---|---|---|---|
| Measurement model | 14.132 | 12 | 0.292 | 0.970 | 0.005 | 0.949 | 0.992 | 0.985 | 0.992 | 0.038 |
| Structural model | 33.867 | 23 | 0.067 | 0.946 | 0.034 | 0.898 | 0.965 | 0.942 | 0.963 | 0.062 |

df degrees of freedom, GFI goodness of fit index, RMR root mean square residual, NFI normed fit index, IFI incremental fit index, TLI Tucker-Lewis index, CFI comparative fit index, RMSEA root mean square error of approximation.

**Table 2.** Means, standard deviations, loadings, latent variables composite reliabilities (CR), and average variances extracted (AVE) for the measurement model.

| Latent | Indicator | M | SD | Loadings | CR | AVE |
|---|---|---|---|---|---|---|
| CPS | GL.RI | 0.829 | 0.182 | 0.640 | 0.725 | 0.472 |
| | GL.RK | 0.354 | 0.164 | 0.601 | | |
| | GL.RA | 0.755 | 0.187 | 0.804 | | |
| P1 | SCS1.D | 0.622 | 0.176 | 0.671 | 0.791 | 0.560 |
| | SCS1.T | 0.648 | 0.220 | 0.842 | | |
| | SCS1.C | 0.546 | 0.254 | 0.721 | | |
| GRADE | GradeCS1 | 3.908 | 1.107 | 1.000 | | |

### *5.2. Structural Model*

After the validity and reliability of our model were confirmed, a structural equation model was used to test our hypotheses. Structural paths were proposed from *CPS* to *P1* and from *P1* to *GRADE*. Regarding the control variables, structural paths were proposed from *PProg* and *PMath* to *P1*. The structural equation model is presented in Figure 8.

The overall model fit was assessed before the structural path analysis. As presented in Table 1, $\chi^2$ was again non-significant ($\chi^2(23) = 33.867$, $p > 0.05$). Likewise, the other fit indices, with the exception of NFI, which is sensitive to low sample sizes, were inside the suggested intervals, pointing toward a good model fit.

The results of the structural path analysis are presented in Table 3. The findings indicated that *CPS* had a strong positive impact on *P1* ($\gamma_1 = 0.80$, $p < 0.001$), which confirmed our hypotheses H1. The findings also showed that *P1* had a positive and significant impact

on *GRADE* ($\beta_1$ = 0.21, $p < 0.05$). Hence, we can support our H2. Regarding the control variables, we could confirm a positive impact of *PProg* on *P1* at $p < 0.05$ (H3). However, we could not confirm *PMath* impacting *P1* (H4).

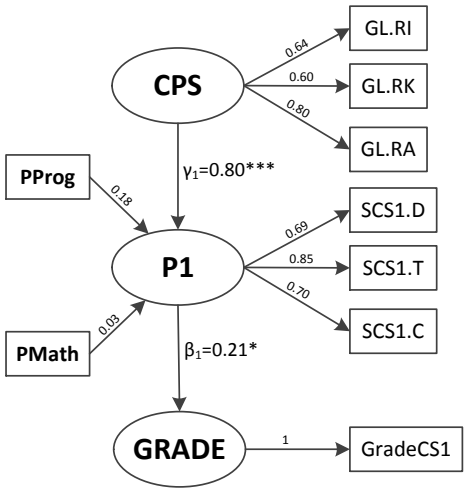


**Figure 8.** Structural equation model illustrating latent correlations between complex problem-solving (*CPS*), performance in the Introductory Programming Course (*P1*), and final grade in the CS1 course (*GRADE*). Previous programming experience (*PProg*) and performance in High School mathematics (*PMath*) were introduced as control variables. $\gamma_1$ is significant at $p < 0.001$ (***), while $\beta_1$ is significant at $p < 0.05$ (*).

**Table 3.** Results of the structural model.

| H# | Proposed Relationship | Type | Estimate | Significance |
|---|---|---|---|---|
| H1 | CPS → P1 | Latent | 0.800 | $p < 0.001$ |
| H2 | P1 → GRADE | Latent | 0.211 | $p < 0.05$ |
| H3 | PProg → P1 | Control | 0.184 | $p < 0.05$ |
| H4 | PMath → P1 | Control | 0.032 | n.s. |

CPS complex problem-solving, P1 performance in the Introductory Programming Course, GRADE final grade in the CS1 course, PProg previous programming experience, PMath performance in High School mathematics.

## 6. Discussion

The aim of the present case study was to explore if complex problem-solving ability could potentially be used as a predictor of success in computing education. The purpose of the predictors of success in education is to identify potentially struggling students who are at risk of underperforming or academic failure. To evaluate this relationship empirically, an experiment was conducted in the undergraduate Introductory Programming Course, where predictors of performance have been investigated extensively. Genetics Lab, a valid and reliable complex problem-solving assessment was used, to evaluate problem-solving performance at the beginning of the CS1 course. Programming performance was assessed by SCS1 at the end of the CS1 course. SCS1 is a valid instrument for assessing CS1 programming knowledge. In general, this study indicated that the ability to solve complex problems has a strong positive effect on performance in introductory programming. Accordingly, it seems reasonable to conclude that CPS has the potential to be considered as a significant, cognitive, dynamic predictor, applicable at the beginning of the CS1 course. It should be noted that the dynamicity of our predictor originates from the dynamic nature of problem-solving situations in microworlds. In terms of the predictor categorizations presented in Section 2, problem-solving predictors generally fall into the category of traditional predictors.

Although our findings are in line with several studies that have identified the predictive power of problem-solving in Introductory Programming (see [119] for details), the present study may be the first that explored a domain-independent predictor in the

form of a dynamic system. Namely, previous studies that applied a domain-independent problem-solving approach have been relying on self-report questionnaires (e.g., [48]) or paper–pencil tasks. For instance, Lishinski et al. [17] used a domain-independent approach in the form of paper–pencil tasks from the "Programme for International Student Assessment" (PISA) problem-solving suite. In addition to predictors, the present study also contributes to broadening the scope of assessing problem-solving from domain-specific programming tasks towards a domain-independent context. This is aligned with emerging educational movements, including computer science unplugged, computational thinking, and block-based programming environments, which advocate that learning programming tends to foster general problem-solving skills [120].

Regarding our first hypothesis, the results revealed that complex problem-solving (*CPS*) has a strong positive effect on the performance in the Introductory Programming Course (*P1*). Likewise, the findings suggest that 64% ($R^2 = 0.64$) of the *P1* score's variance is explained by *CPS*. This result was well above our expectations. In this regard, we consulted the author of the Genetics Lab and the SEM expert before interpreting the results. Even after a joint meeting, no flaws were detected. Accordingly, our results could have the following implications for teaching and learning in the classroom. First, there is a CPS assessment available that takes roughly one hour and is easy to administer and grade. Second, by assessing the students' problem-solving performance at the beginning of the CS1 course, teachers could identify potentially struggling students who are at risk of non-progression or dropping out of the CS1 course. Third, high-achieving students could also be identified. Finally, teaching and learning activities in the classroom could be aligned promptly, in order to support students with potential difficulties. Quille and Bergin [4] emphasized the importance of prompt identification of non-progressing students, because lecturers may not be aware that students are struggling due to the very high student-lecturer ratios.

It has been shown that CPS is related highly to general intelligence [57,95]. Despite the relation, both have been considered as distinct constructs [121]. Accordingly, the results supporting our H1 should be considered as the cognitive-only portion of the broader problem-solving competence. Namely, our conceptual model, which is described in Section 3, considers problem-solving competence as a broad, multi-component construct. The model incorporates a collection of cognitive and noncognitive skills, abilities or aptitudes, as well as personal characteristics. In this broader context, performance in problem-solving is affected not only by cognitive skills but also by how individuals adapt and apply metacognitive strategies, as well as by their motivation [122]. As a result, in addition to cognitive skills, metacognitive and affective factors should also be considered when predicting academic outcomes in computing education.

Regarding our second hypothesis, the results showed a positive and significant correlation between *P1* and the students' final grades in the CS1 course (*GRADE*). Our intention was to introduce *GRADE* as a latent variable with several indicators, including the results from intermediate or final exams, extra credits for course presence, for non-mandatory tasks, as well as a credit for participating in the experiments. Namely, our CS1 course design follows recent developments in education research, which advocate that increased engagement, motivation and course participation have a positive effect on academic outcomes [80]. In line with these findings, students can earn credits for course presence, for non-mandatory tasks, and for participating in experiments. Wherever possible, the credit is not given per se, but is calculated based on their performance in a particular activity. Unfortunately, due to legal restrictions, we were not able to access the structure of the students' final grades. As a consequence, *GRADE* was introduced as a latent with the students' final grades as a single indicator. Given the broad definition of *GRADE* in our model, it is not surprising that it could not be explained fully by *P1* and *CPS*.

Concerning control variables, H3 confirms the positive effect of previous programming experience (*PProg*) on *P1*. This is in line with our anticipations, as well as with many previous studies that have identified the same (e.g., [9,39]). Likewise, there is a general agreement in the computing education community that *PProg* influences students'

performances in CS1 [84]. Less consensus exists, however, about how the *PProg* should be evaluated. Researchers have previously relied on various questionnaires, asking students about their years of programming experience, number of languages previously studied, or longest program written [9]. In this context, the present study may be the first that introduced a questionnaire where students rated their previous programming experience. Their experiences were self-assessed in terms of the number of projects they have worked on previously, regardless of the programming languages used. Despite the supportive results, we support the proposal by Duran et al. [84], who suggested that a common measure of *PProg* is needed.

Regarding H4, we could not confirm performance in High School mathematics (*PMath*) having an effect on *P1*. This is not in line with our expectations, as well as with previous studies, which mostly confirmed a significant effect of *PMath*. However, Ventura Jr [35] also examined the mathematics portion of SAT scores (math SAT), which tends to be an equivalent of the matriculation scores that were used in our study. The correlation between *PMath* and performance in a CS1 course of more than 300 students was found to be weak [35]. In addition, Watson et al. [9] also found a non-significant correlation between math SAT and CS1 performance, while they also reported "a total lack of correlation between discrete math and programming performance" (p. 471).

*Implications*

Although this exploratory small-scale study has clear limitations, our findings provide opportunities for extending the demonstrated approach to a broader computing education community. Namely, assessing problem-solving skills in the form of computer-simulated microworlds has been a previously reported method for evaluating domain-independent skills across post-secondary education, including STEM, as well as humanities and social sciences, law, and economics (e.g., [123–125]). Furthermore, the same approach has also been used in grades 6 to 8 (e.g., [126,127]). Likewise, Sonnleitner et al. [128] addressed some challenges and opportunities when applying CPS instruments in the classroom. In this context, complex problem-solving ability tends to be a useful predictor in the broader computing education community. This applies particularly to emerging educational movements, such as coding in block-based programming environments and computational thinking, which advocate that learning programming fosters general problem-solving skills. As these movements have recently broadened the scope of computing education into K-12, the approach demonstrated in the present paper could also be extended into K-12. For instance, MicroDYN [129], a variant of a valid and reliable problem-solving assessment, could be administered at the beginning of the Scratch course. As a result, students with potential difficulties could be identified, allowing teaching and learning activities in the classroom to be aligned promptly.

While reviewing the literature on performance predictors in education, Hung et al. [130] identified several gaps, including the lack of common predictors identified for both the K-12 and post-secondary education environments. Furthermore, they argued that the majority of studies related to performance prediction were focused on post-secondary education. In this context, our approach could potentially fill the identified gaps in the broader education community. For instance, assessing general problem-solving performance in Genetics Lab or MicroDYN could be a useful predictor in the broader education landscape, targeting K-12 and post-secondary levels.

## 7. Threats to Validity

This section discusses the potential construct, internal, and external validity threats [131] of our experimental study.

### *7.1. Construct Validity*

Construct validity refers to the extent to which we measure the concept under examination accurately. In our experiment, our aim was to gauge participants' complex

problem-solving abilities, and assess their potential as predictors in the Introductory Programming Course.

Two primary concerns influenced the construct validity of this experiment: the selection of instruments for phases 1 and 2, and the specific modifications we implemented in these instruments.

Initially, we utilized Genetics Lab to assess complex problem-solving abilities and SCS1 to evaluate computer programming performance. One could raise questions regarding the quality of these chosen instruments and the assessments within them. Nevertheless, both instruments underwent independent evaluations, and empirical evidence exists supporting their satisfactory reliability (see Sections 4.3.2 and 4.3.3 for details). However, the study's outcome might have varied had alternative instruments been employed for both purposes.

Secondly, some adjustments (translation to native language, technical adjustments, online execution, etc.) were made to Genetics Lab and SCS1. While these adaptations mitigated several validity threats, they also introduced another potential concern. We lack insight into the outcomes had we retained the original settings. Nonetheless, a pilot study was conducted to rectify any instrument construction errors and mitigate technical issues. Therefore, we have confidence in the results and quality of this research.

### *7.2. Internal Validity*

Internal validity is the degree of confidence that accidental or other confounding factors do not influence the relationship that is subject to investigation.

The present study was conducted during the COVID-19 pandemic. As anticipated during the final stages of experiment planning, student participation was affected by the pandemic and was actually lower. Before the beginning, we decided to switch from a random to a convenience sample, which presented additional validity concerns. To mitigate the situation somehow, an additional control variable (performance in High School mathematics) was introduced into the experiment design. Accordingly, the impact of this threat was minimized.

The unexpected strength of *CPS* on *P1* may have been influenced by the sample size. In general, the sample size of our study (N = 122) was above the threshold estimated by the Monte Carlo simulation (N = 112). Moreover, the size of the sample was sufficient to ensure the proper model convergence and accurate parameter estimations. However, for the model presented in Section 5.1, better fit indices were estimated by Monte Carlo for larger sample sizes (N = 175, N = 238, N = 300, based on the respective fit indices cutoffs).

Regarding the sample, there was a discrepancy in the sample sizes between the two phases of the experiment. In the first phase, which was conducted in the CS lab, the participation rate was low (N = 132), because students were not present due to COVID-19 infections. On the other hand, the second part of the experiment had to be conducted remotely, because the university was closed due to high COVID-19 infection rates. The participation rate in the remote experiment was higher (N = 164).

Because the second part of the experiment was administered remotely, the students' SCS1 scores might have been impacted by cheating [132]. To mitigate this issue, the students were asked to keep their video and audio devices on during the experiment, while the CS teaching staff monitored their behavior. In addition, a data cleaning procedure was implemented and executed on the results' dataset. The aim was to detect and remove clusters of scores where the probability of group cheating could be considerable. In particular, our focus was on detecting scores that could be a result of collaborative cheating via platforms like Discord or WhatsApp. Noorbehbahani et al. [133] considered collaborative cheating as a subcategory of the broader category of group cheating. As a result, five records were removed from the dataset. However, even if the aforementioned procedures were applied, it is not possible to exclude the impact of cheating completely.

Due to the pandemic, there was an unplanned interval extension between the experiments that could also have an impact on the results. In particular, the second part of the experiment was conducted six weeks (including the four-week winter break) after the end

of the CS1 course. Finally, online learning during the lockdown of the COVID-19 pandemic could also affect the general level of the students' CS1 knowledge.

The potential internal validity threat also exists because of student dropouts between both phases of the experiment (mortality), as well as because of missing values. Regarding mortality, the students who actually participated in both phases of the experiment could be more motivated. Regarding missing values, two students did not finish the first phase of the experiment, while six students did not answer all the questions from the second phase of the experiment. All were excluded from the analysis, which represents a selection threat. However, since the missing values represent less than 4% of all the participants, the potential level of implication is acceptable.

### *7.3. External Validity*

External validity examines whether the results of an experiment can be generalized to other contexts.

The results of our study are limited primarily to the Introductory Programming Course. As a consequence, an external validity threat exists, because we do not know what the results would be if these experiments were part of courses in the same or other school years.

The generalizability of the results was also limited because the experiment was conducted in a specific environment. First, the convenience sample for this study was drawn from a single institution. Second, the results could be the subject of influence by demographic and cultural characteristics. Third, due to the COVID-19 pandemic, students' inclusion based on random sampling was not possible. Instead, convenience sampling was applied.

Another external validity concern is associated with latent variable *CPS*, which had a value of average variances extracted (AVE) lower than 0.5 (AVE = 0.472). If the AVE is lower than the suggested threshold, the convergent validity of the measurement model might be weaker than expected. Convergent validity indicates how much variance in the observed variables is explained by the latent construct. However, due to the exploratory nature of this study, as well as the AVE of *CPS* was slightly below the suggested threshold (5.6%), the potential level of implication is acceptable.

Overall, we would like to point out that exploratory studies have their own place in experimental research, especially in the segments where previous research has been lacking. However, they require replications to confirm the level of confidence before drawing general conclusions. Moreover, the potential confirmation should be the subject of replicated studies in multi-institutional (and multi-national) environments.

## 8. Conclusions

This case study explored the predictive potential of complex problem-solving ability in the Introductory Programming Course. Following the results, the study opens a path to a novel predictor and potentially fills the gap in predicting performance in introductory programming. In addition, our results show that performance in the experiment was positively correlated with the students' final grades. In line with most of the previous studies, we were also able to confirm that previous programming experience has a significant and positive effect on programming performance. However, we were not able to confirm a significant effect of performance in High School Mathematics on performance in the Introductory Programming Course. The results of our study could potentially have implications for teaching and learning in classrooms. Namely, an easy-to-administer complex problem-solving instrument could be applied at the beginning of the course, to identify potentially struggling students, as well as high-achievers. Likewise, our results suggest that complex problem-solving performance could also be a useful predictor of academic performance in the broader computing education community. Moreover, it could also be applied across education, targeting K-12 and post-secondary levels. In line with our conceptual model, the results reflect only a cognitive portion of the broader problem-solving competence. In this broader context, problem-solving performance depends not only on cognitive skills,

but also on noncognitive factors, including metacognition, emotions, and motivation. In addition to predictors, the application of complex problem-solving assessments in the form of computer-simulated microworlds could also be used to examine the potential causal relationship between learning programming and proficiency in solving problems. This applies particularly to the investigation of proficiency in solving general-purpose, domain-independent problems, which tends to be fostered by learning programming and computational thinking.

*Future Directions*

Our study employed an exploratory method. Therefore, additional validations are required to confirm the level of confidence suggested by our results, particularly because the strength of the effect of *CPS* on *P1* was well above our expectations. On the other hand, we could not confirm *PMath* having an effect on *P1*. *PMath* was introduced as a control variable, and it was anticipated to be confirmed.

In this regard, we encourage the computing education community to replicate our study and re-evaluate the level of confidence suggested by our results. In the case of a replication study, a larger sample (N > 300) would be recommended for effect size confirmation. In line with the results of the Monte Carlo simulation, such a sample size will not only result in a proper model convergence, accurate estimates of parameters and standard errors, but also in higher levels of statistical power (see also Sections 7.2 and 4.5)

Regarding the possible alternative settings in the domain of complex problem-solving, we believe that the premise should be twofold. First, it would be interesting to see the effect of *CPS* on *P1* if the Genetics Lab is substituted with another CPS instrument from the family of multiple complex systems, such as MicroDYN [134] or MicroFIN [135]. Second, further studies should also consider employing a CPS instrument from the family of classical problem-solving measures, such as Tailorshop [136].

In a broader problem-solving context our future research focus on computing education should be directed towards exploration of the predictive potential of metacognition and emotions. Both tend to have a multi-component structure, consisting of static (off-line) and dynamic (on-line) portions. It would be interesting to explore the overall predictive potential of the main components, as well as the potential differences on the sub-component level.

**Author Contributions:** Conceptualization, B.B.; methodology, B.B.; software, B.B. and T.K.; validation, T.K. and M.M.; investigation, B.B., T.K. and M.M.; writing—original draft preparation, B.B., T.K. and M.M.; writing—review and editing, B.B., T.K. and M.M. All authors have read and agreed to the published version of the manuscript.

**Funding:** The second and third authors acknowledge the financial support from the Slovenian Research Agency (Research Core Funding No. P2-0041).

**Institutional Review Board Statement:** Ethical review and approval were waived for this study because the tests had the form of a midterm exam.

**Informed Consent Statement:** Informed consent was obtained from all subjects involved in the study.

**Data Availability Statement:** Dataset available on request from the authors.

**Acknowledgments:** The authors wish to thank the whole team of the Programming Methodologies Laboratory at the University of Maribor, Faculty of Electrical Engineering and Computer Science, for their help and fruitful discussions during the execution of the controlled experiment. In addition, the authors would like to acknowledge Borut Milfelner for his contribution to initial data analysis and suggestions concerning structural equation modeling. The authors would also like to acknowledge Valerie Shute and Philipp Sonnleitner for their manuscript revision and their suggestions in order to improve this work.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References


1. Dawson, J.Q.; Allen, M.; Campbell, A.; Valair, A. Designing an introductory programming course to improve non-majors' experiences. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education, Baltimore, MD, USA, 21–24 February 2018; pp. 26–31.
2. Sun, L.; Guo, Z.; Zhou, D. Developing K-12 students' programming ability: A systematic literature review. *Educ. Inf. Technol.* **2022**, *27*, 7059–7097. [CrossRef]
3. Luxton-Reilly, A.; Simon; Albluwi, I.; Becker, B.A.; Giannakos, M.; Kumar, A.N.; Ott, L.; Paterson, J.; Scott, M.J.; Sheard, J.; et al. Introductory programming: A systematic literature review. In Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, Larnaca, Cyprus, 2–4 July 2018; pp. 55–106.
4. Quille, K.; Bergin, S. CS1: How will they do? How can we help? A decade of research and practice. *Comput. Sci. Educ.* **2019**, *29*, 254–282. [CrossRef]
5. Luxton-Reilly, A.; Ajanovski, V.V.; Fouh, E.; Gonsalvez, C.; Leinonen, J.; Parkinson, J.; Poole, M.; Thota, N. Pass rates in introductory programming and in other STEM disciplines. In Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education, Aberdeen Scotland, UK, 15–17 July 2019; pp. 53–71.
6. Watson, C.; Li, F.W. Failure rates in introductory programming revisited. In Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education, Uppsala, Sweden, 21–25 June 2014; pp. 39–44.
7. Bennedsen, J.; Caspersen, M.E. Failure rates in introductory programming: 12 years later. *ACM Inroads* **2019**, *10*, 30–36. [CrossRef]
8. Bennedsen, J.; Caspersen, M.E. Failure rates in introductory programming. *ACM Sigcse Bull.* **2007**, *39*, 32–36. [CrossRef]
9. Watson, C.; Li, F.W.; Godwin, J.L. No tests required: Comparing traditional and dynamic predictors of programming success. In Proceedings of the 45th ACM Technical Symposium on Computer Science Education, Atlanta, GA, USA, 5–8 March 2014; pp. 469–474.
10. Petersen, A.; Craig, M.; Campbell, J.; Tafliovich, A. Revisiting why students drop CS1. In Proceedings of the 16th Koli Calling International Conference on Computing Education Research, Koli, Finland, 24–27 November 2016; pp. 71–80.
11. Tedre, M.; Malmi, L. Changing aims of computing education: A historical survey. *Comput. Sci. Educ.* **2018**, *28*, 158–186. [CrossRef]
12. Deek, F.P. The software process: A parallel approach through problem solving and program development. *Comput. Sci. Educ.* **1999**, *9*, 43–70. [CrossRef]
13. Palumbo, D.B. Programming language/problem-solving research: A review of relevant issuess. *Rev. Educ. Res.* **1990**, *60*, 65–89. [CrossRef]
14. Winslow, L.E. Programming pedagogy—A psychological view. *ACM Sigcse Bull.* **1996**, *28*, 17–22. [CrossRef]
15. Polya, G. *How to Solve It*; Princeton University Press: Princeton, NJ, USA, 1945.
16. Barnes, D.J.; Fincher, S.; Thompson, S. Introductory problem solving in computer science. In Proceedings of the 5th Annual Conference on the Teaching of Computing, Dublin, Ireland, August 1997; pp. 36–39.
17. Lishinski, A.; Yadav, A.; Enbody, R.; Good, J. The influence of problem solving abilities on students' performance on different assessment tasks in CS1. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education, Memphis, TN, USA, 2–5 March 2016; pp. 329–334.
18. Demir, Ü. The effect of unplugged coding education for special education students on problem-solving skills. *Int. J. Comput. Sci. Educ. Sch.* **2021**, *4*, 3–30.
19. Fessakis, G.; Gouli, E.; Mavroudi, E. Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Comput. Educ.* **2013**, *63*, 87–97. [CrossRef]
20. Park, K.; Mott, B.; Lee, S.; Gupta, A.; Jantaraweragul, K.; Glazewski, K.; Scribner, A.J.; Ottenbreit-Leftwich, A.; Hmelo-Silver, C.E.; Lester, J. Investigating a visual interface for elementary students to formulate AI planning tasks. *J. Comput. Lang.* **2022**, *73*, 101157. [CrossRef]
21. Li, Y.; Schoenfeld, A.H.; diSessa, A.A.; Graesser, A.C.; Benson, L.C.; English, L.D.; Duschl, R.A. On computational thinking and STEM education. *J. Stem Educ. Res.* **2020**, *3*, 147–166. [CrossRef]
22. Grover, S.; Pea, R. Computational thinking: A competency whose time has come. *Comput. Sci. Educ. Perspect. Teach. Learn. Sch.* **2018**, *19*, 19–38.
23. Ezeamuzie, N.O.; Leung, J.S.; Garcia, R.C.; Ting, F.S. Discovering computational thinking in everyday problem solving: A multiple case study of route planning. *J. Comput. Assist. Learn.* **2022**, *38*, 1779–1796. [CrossRef]
24. Standl, B. Solving everyday challenges in a computational way of thinking. In Proceedings of the Informatics in Schools: Focus on Learning Programming: 10th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2017, Helsinki, Finland, 13–15 November 2017; pp. 180–191.
25. Robins, A.; Rountree, J.; Rountree, N. Learning and teaching programming: A review and discussion. *Comput. Sci. Educ.* **2003**, *13*, 137–172. [CrossRef]
26. Unuakhalu, M. Effect of Computer Programming Instruction on the Problem Solving Capability of College Level Introductory Computer Students. Ph.D. Thesis, University of Kentucky, Lexington, KY, USA, 2004.
27. Evans, G.E.; Simkin, M.G. What best predicts computer proficiency? *Commun. ACM* **1989**, *32*, 1322–1327. [CrossRef]
28. Cafolla, R. The Relationship of Piagetian Formal Operations and Other Cognitive Factors to Computer Programming Ability (Development). Ph.D. Thesis, Florida Atlantic University, Boca Raton, FL, USA, 1986.

29. Bauer, R.; Mehrens, W.A.; Vinsonhaler, J.F. Predicting performance in a computer programming course. *Educ. Psychol. Meas.* **1968**, *28*, 1159–1164. [CrossRef]
30. Bergin, S.; Reilly, R. Predicting introductory programming performance: A multi-institutional multivariate study. *Comput. Sci. Educ.* **2006**, *16*, 303–323. [CrossRef]
31. Ihantola, P.; Vihavainen, A.; Ahadi, A.; Butler, M.; Börstler, J.; Edwards, S.H.; Isohanni, E.; Korhonen, A.; Petersen, A.; Rivers, K.; et al. Educational data mining and learning analytics in programming: Literature review and case studies. In Proceedings of the 2015 ITiCSE on Working Group Reports, Vilnius, Lithuania, 4–8 July 2015; pp. 41–63.
32. Wilson, B.C.; Shrock, S. Contributing to success in an introductory computer science course: A study of twelve factors. *ACM Sigcse Bull.* **2001**, *33*, 184–188. [CrossRef]
33. Bergin, S.; Reilly, R. Programming: Factors that influence success. In Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education, St. Louis, MO, USA, 23–27 February 2005; pp. 411–415.
34. Quille, K.; Culligan, N.; Bergin, S. Insights on Gender Differences in CS1: A Multi-institutional, Multi-variate Study. In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, Bologna, Italy, 3–5 July 2017; pp. 263–268.
35. Ventura, P.R., Jr. Identifying predictors of success for an objects-first CS1. *Comput. Sci. Educ.* **2005**, *15*, 223–243. [CrossRef]
36. Byrne, P.; Lyons, G. The effect of student attributes on success in programming. In Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education, Canterbury, UK, 24–30 June 2001; pp. 49–52.
37. Wiedenbeck, S. Factors affecting the success of non-majors in learning to program. In Proceedings of the First International Workshop on Computing Education Research, Seattle, WA, USA, 1–2 October 2005; pp. 13–24.
38. Hagan, D.; Markham, S. Does it help to have some programming experience before beginning a computing degree program? In Proceedings of the 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, Helsinki, Finland, 11–13 July 2000; pp. 25–28.
39. Bockmon, R.; Cooper, S.; Gratch, J.; Zhang, J.; Dorodchi, M. Can Students' Spatial Skills Predict Their Programming Abilities? In Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, Trondheim, Norway 15–19 June 2020; pp. 446–451.
40. Kangas, V.; Pirttinen, N.; Nygren, H.; Leinonen, J.; Hellas, A. Does creating programming assignments with tests lead to improved performance in writing unit tests? In Proceedings of the ACM Conference on Global Computing Education, Chengdu, China, 9–19 May 2019; pp. 106–112.
41. Kurtz, B.L. Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class. In Proceedings of the Eleventh SIGCSE Technical Symposium on Computer Science Education, Kansas City, MO, USA, 14–15 February 1980; pp. 110–117.
42. Bennedssen, J.; Caspersen, M.E. Abstraction ability as an indicator of success for learning computing science? In Proceedings of the Fourth International Workshop on Computing Education Research, Sydney, Australia, 6–7 September 2008; pp. 15–26.
43. Jones, S.; Burnett, G. Spatial ability and learning to program. *Hum. Technol. Interdiscip. J. Hum. ICT Environ.* **2008**, *4*, 47–61. [CrossRef]
44. Mancy, R.; Reid, N. Aspects of cognitive style and programming. In Proceedings of the 16th Workshop of the Psychology of Programming Interest Group, Carlow, Ireland, 5–7 April 2004.
45. Shute, V.J. Who is likely to acquire programming skills? *J. Educ. Comput. Res.* **1991**, *7*, 1–24. [CrossRef]
46. Schuyler, S.T.; Skovira, R.J. Is the Problematic in CS1 a Student's Problem Solving Ability. *Issues Inf. Syst. IIS* **2007**, *8*, 112–119.
47. Barlow-Jones, G.; Westhuizen, D. Problem solving as a predictor of programming performance. In Proceedings of the Annual Conference of the Southern African Computer Lecturers' Association, Magaliesburg, South Africa, 3–5 July 2017; pp. 209–216.
48. Veerasamy, A.K.; D'Souza, D.; Lindén, R.; Laakso, M.J. Relationship between perceived problem-solving skills and academic performance of novice learners in introductory programming courses. *J. Comput. Assist. Learn.* **2019**, *35*, 246–255. [CrossRef]
49. Lister, R. Concrete and other neo-Piagetian forms of reasoning in the novice programmer. In Proceedings of the Thirteenth Australasian Computing Education Conference, Perth, Australia, 17–20 January 2011; Volume 114, pp. 9–18.
50. Lau, W.W.; Yuen, A.H. Modelling programming performance: Beyond the influence of learner characteristics. *Comput. Educ.* **2011**, *57*, 1202–1213. [CrossRef]
51. Askar, P.; Davenport, D. An investigation of factors related to self-efficacy for Java Programming among engineering students. *Turk. Online J. Educ. Technol.* **2009**, *8*, 3.
52. Bergin, S.; Reilly, R. The influence of motivation and comfort-level on learning to program. In Proceedings of the 17th Workshop of the Psychology of Programming Interest Group, Sussex University, Brighton, UK, 29 June–1 July 2005.
53. Bennedsen, J.; Caspersen, M.E. Optimists have more fun, but do they learn better? On the influence of emotional and social factors on learning introductory computer science. *Comput. Sci. Educ.* **2008**, *18*, 1–16. [CrossRef]
54. Quille, K. Predicting and Improving Performance on Introductory Programming Courses (CS1). Ph.D. Thesis, National University of Ireland, Dublin, Ireland, 2019.
55. Dörner, D.; Funke, J. Complex problem solving: What it is and what it is not. *Front. Psychol.* **2017**, *8*, 1153. [CrossRef] [PubMed]
56. Süß, H.M.; Kretzschmar, A. Impact of cognitive abilities and prior knowledge on complex problem solving performance–Empirical results and a plea for ecologically valid microworlds. *Front. Psychol.* **2018**, *9*, 626. [CrossRef]
57. Funke, J. Complex problem solving: A case for complex cognition? *Cogn. Process.* **2010**, *11*, 133–142. [CrossRef]

58. Funke, J.; Frensch, P.A. Complex problem solving: The European perspective—10 years after. In *Learning to Solve Complex Scientific Problems*; Jonassen, D.H., Ed.; Routledge: London, UK 2007; pp. 25–48.
59. Jonassen, D.H. Toward a design theory of problem solving. *Educ. Technol. Res. Dev.* **2000**, *48*, 63–85. [CrossRef]
60. Forum, W.E. *The Future of Jobs: Employment, Skills and Workforce Strategy for the Fourth Industrial Revolution*; World Economic Forum: Geneva, Switzerland, 2016.
61. Rubenstein, D.; Novakovic, P. The David Rubenstein Show: General Dynamics CEO Phebe Novakovic. 2021. Available online: https://www.bloomberg.com/news/videos/2021-08-26/the-david-rubenstein-show-phebe-novakovic-video (accessed on 15 May 2024).
62. Greiff, S.; Wüstenberg, S.; Molnár, G.; Fischer, A.; Funke, J.; Csapó, B. Complex problem solving in educational contexts—Something beyond g: Concept, assessment, measurement invariance, and construct validity. *J. Educ. Psychol.* **2013**, *105*, 364. [CrossRef]
63. Buchner, A. Basic topics and approaches to the study of complex problem solving. In *Complex Problem Solving: The European Perspective*; Frensch, A., Funke, J., Eds.; Erlbaum: Hillsdale, NJ, USA, 1995; pp. 27–63.
64. Greiff, S.; Fischer, A.; Wüstenberg, S.; Sonnleitner, P.; Brunner, M.; Martin, R. A multitrait–multimethod study of assessment instruments for complex problem solving. *Intelligence* **2013**, *41*, 579–596. [CrossRef]
65. Greiff, S.; Stadler, M.; Sonnleitner, P.; Wolff, C.; Martin, R. Sometimes less is more: Comparing the validity of complex problem solving measures. *Intelligence* **2015**, *50*, 100–113. [CrossRef]
66. Weinert, F.E. *Concept of Competence: A Conceptual Clarification*; Hogrefe & Huber Publishers: St. Newburyport, MA, USA, 2001.
67. Funke, J.; Fischer, A.; Holt, D.V. Competencies for complexity: Problem solving in the twenty-first century. In *Assessment and Teaching of 21st Century Skills*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 41–53.
68. Stanic, G.; Kilpatrick, J. Historical perspectives on problem solving in the mathematics curriculum. In *The Teaching and Assessing of Mathematical Problem Solving*; Charles, R., Silver, E., Eds.; National Coundl of Teachers of Mathematics: Reston, VA, USA, 1988; pp. 1–22.
69. Schoenfeld, A.H. Learning to think mathematically: Problem solving, metacognition, and sense-making in mathematics. In *Handbook for Research on Mathematics Teaching and Learning*; Grouws, D., Ed.; Macmillan: New York, NY, USA, 1992; pp. 334–370.
70. Lester, F.K.; Cai, J. Can mathematical problem solving be taught? Preliminary answers from 30 years of research. In *Posing and Solving Mathematical Problems*; Felmer, P.L., Pehkonen, E., Kilpatrick, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 117–135.
71. Bahar, A. The Influence of Cognitive Abilities on Mathematical Problem Solving Performance. Ph.D. Thesis, The University of Arizona, Tucson, AZ, USA, 2013.
72. Garofalo, J.; Lester, F. Metacognition, cognitive monitoring, and mathematical performance. *J. Res. Math. Educ.* **1985**, *16*, 163–176. [CrossRef]
73. Wilson, J.; Clarke, D. Towards the modelling of mathematical metacognition. *Math. Educ. Res. J.* **2004**, *16*, 25–48. [CrossRef]
74. McLeod, D.B. Research on affect in mathematics education: A reconceptualization. *Handb. Res. Math. Teach. Learn.* **1992**, *1*, 575–596.
75. Philipp, R.A. Mathematics teachers' beliefs and affect. In *Second Handbook of Research on Mathematics Teaching and Learning*; Lester, F.K., Ed.; NCTM: Reston, VA, USA, 2007; pp. 257–315.
76. DeBellis, V.A.; Goldin, G.A. Affect and meta-affect in mathematical problem solving: A representational perspective. *Educ. Stud. Math.* **2006**, *63*, 131–147. [CrossRef]
77. Lishinski, A. Cognitive, Affective, and Dispositional Components of Learning Programming. Ph.D. Thesis, Michigan State University, East Lansing, MI, USA, 2017.
78. Malmi, L.; Sheard, J.; Kinnunen, P.; Sinclair, J. Theories and models of emotions, attitudes, and self-efficacy in the context of programming education. In Proceedings of the 2020 ACM Conference on International Computing Education Research, Virtual Event, New Zealand, 1–5 August 2020; pp. 36–47.
79. Medeiros, R.P.; Ramalho, G.L.; Falcão, T.P. A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Trans. Educ.* **2018**, *62*, 77–90. [CrossRef]
80. Zeybek, N.; Saygı, E. Gamification in Education: Why, Where, When, and How?—A Systematic Review. *Games Cult.* **2024**, *19*, 237–264. [CrossRef]
81. Imran, H. An empirical investigation of the different levels of gamification in an introductory programming course. *J. Educ. Comput. Res.* **2023**, *61*, 847–874. [CrossRef]
82. Kučak, D.; Pašić, Đ.; Mršić, L. An Empirical Study of Long-Term Effects of Applying Gamification Principles to an Introductory Programming Courses on a University Level. In Proceedings of the International Conference on Intelligent Computing & Optimization, Hua Hin, Thailand, 27–28 October 2022; pp. 469–477.
83. Thompson, R.A.; Zamboanga, B.L. Academic aptitude and prior knowledge as predictors of student achievement in introduction to psychology. *J. Educ. Psychol.* **2004**, *96*, 778. [CrossRef]
84. Duran, R.S.; Rybicki, J.M.; Hellas, A.; Suoranta, S. Towards a common instrument for measuring prior programming knowledge. In Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education, Aberdeen, Scotland, UK, 15–17 July 2019; pp. 443–449.

85. Trujillo-Torres, J.M.; Hossein-Mohand, H.; Gómez-García, M.; Hossein-Mohand, H.; Hinojo-Lucena, F.J. Estimating the academic performance of secondary education mathematics students: A gain lift predictive model. *Mathematics* **2020**, *8*, 2101. [CrossRef]
86. Toland, M.D.; Usher, E.L. Assessing mathematics self-efficacy: How many categories do we really need? *J. Early Adolesc.* **2016**, *36*, 932–960. [CrossRef]
87. Shaffer, D.O. Predicting Success in the Undergraduate Introductory Computer Science Course Using the Theory of Planned Behavior. Ph.D. Thesis, The University of Texas at Austin, Austin, TX, USA, 1990.
88. Fan, T.S.; Li, Y.C. Is math ability beneficial to performance in college computer science programs. *J. Natl. Taipei Teach. Coll.* **2002**, *15*, 69–98.
89. Sonnleitner, P.; Brunner, M.; Greiff, S.; Funke, J.; Keller, U.; Martin, R.; Hazotte, C.; Mayer, H.; Latour, T. The Genetics Lab. Acceptance and psychometric characteristics of a computer-based microworld to assess complex problem solving. *Psychol. Test Assess. Model.* **2012**, *54*, 54–72.
90. Tew, A.E. Assessing Fundamental Introductory Computing Concept Knowledge in a Language Independent Manner. Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2010.
91. Qualtrix Online Survey Software. Available online: https://www.qualtrics.com/au/core-xm/survey-software/ (accessed on 15 May 2024).
92. Sjøberg, D.I.; Hannay, J.E.; Hansen, O.; Kampenes, V.B.; Karahasanovic, A.; Liborg, N.K.; Rekdal, A.C. A survey of controlled experiments in software engineering. *IEEE Trans. Softw. Eng.* **2005**, *31*, 733–753. [CrossRef]
93. Adobe Flash Player. Available online: https://www.adobe.com/products/flashplayer/end-of-life.html (accessed on 15 May 2024).
94. The R Project for Statistical Computing. Available online: https://www.r-project.org/ (accessed on 15 May 2024).
95. Sonnleitner, P.; Keller, U.; Martin, R.; Brunner, M. Students' complex problem-solving abilities: Their structure and relations to reasoning ability and educational success. *Intelligence* **2013**, *41*, 289–305. [CrossRef]
96. Wüstenberg, S.; Greiff, S.; Funke, J. Complex problem solving—More than reasoning? *Intelligence* **2012**, *40*, 1–14. [CrossRef]
97. Greiff, S.; Fischer, A.; Stadler, M.; Wüstenberg, S. Assessing complex problem-solving skills with multiple complex systems. *Think. Reason.* **2015**, *21*, 356–382. [CrossRef]
98. Hestenes, D.; Wells, M.; Swackhamer, G. Force concept inventory. *Phys. Teach.* **1992**, *30*, 141–158. [CrossRef]
99. Taylor, C.; Zingaro, D.; Porter, L.; Webb, K.C.; Lee, C.B.; Clancy, M. Computer science concept inventories: Past and future. *Comput. Sci. Educ.* **2014**, *24*, 253–276. [CrossRef]
100. Webb, K.C.; Zingaro, D.; Liao, S.N.; Taylor, C.; Lee, C.; Clancy, M.; Porter, L. Student performance on the BDSI for basic data structures. *ACM Trans. Comput. Educ. (TOCE)* **2021**, *22*, 1–34. [CrossRef]
101. Luxton-Reilly, A.; Becker, B.A.; Cao, Y.; McDermott, R.; Mirolo, C.; Mühling, A.; Petersen, A.; Sanders, K.; Whalley, J. Developing assessments to determine mastery of programming fundamentals. In Proceedings of the 2017 ITiCSE Conference on Working Group Reports, Bologna, Italy, 3–5 July 2018; pp. 47–69.
102. Parker, M.C.; Guzdial, M.; Tew, A.E. Uses, Revisions, and the Future of Validated Assessments in Computing Education: A Case Study of the FCS1 and SCS1. In Proceedings of the 17th ACM Conference on International Computing Education Research, Virtual Event, USA, 16–19 August 2021; pp. 60–68.
103. Parker, M.C.; Guzdial, M.; Engleman, S. Replication, validation, and use of a language independent CS1 knowledge assessment. In Proceedings of the 2016 ACM Conference on International Computing Education Research, Melbourne, Australia, 1–12 September 2016; pp. 93–101.
104. Parker, M.C.; Davidson, M.J.; Kao, Y.S.; Margulieux, L.E.; Tidler, Z.R.; Vahrenhold, J. Toward CS1 Content Subscales: A Mixed-Methods Analysis of an Introductory Computing Assessment. In Proceedings of the 23rd Koli Calling International Conference on Computing Education Research, Koli, Finland, 13–18 November 2023; pp. 1–13.
105. Xie, B.; Davidson, M.J.; Li, M.; Ko, A.J. An item response theory evaluation of a language-independent CS1 knowledge assessment. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education, Minneapolis, MN, USA, 27 February–2 March 2019; pp. 699–705.
106. De Ayala, R.J. *The Theory and Practice of Item Response Theory*; Guilford Publications: New York, NY, USA, 2013.
107. Blunch, N.J. Introduction to structural equation modeling using IBM SPSS statistics and AMOS. In *Introduction to Structural Equation Modeling Using IBM SPSS Statistics and AMOS*; Sage Publications Ltd.: London, UK, 2012; pp. 1–312.
108. Anderson, J.C.; Gerbing, D.W. Structural equation modeling in practice: A review and recommended two-step approach. *Psychol. Bull.* **1988**, *103*, 411. [CrossRef]
109. Muthén, L.K.; Muthén, B.O. How to use a Monte Carlo study to decide on sample size and determine power. *Struct. Equ. Model.* **2002**, *9*, 599–620. [CrossRef]
110. Wolf, E.J.; Harrington, K.M.; Clark, S.L.; Miller, M.W. Sample size requirements for structural equation models: An evaluation of power, bias, and solution propriety. *Educ. Psychol. Meas.* **2013**, *73*, 913–934. [CrossRef] [PubMed]
111. Jorgensen, T.D.; Pornprasertmanit, S.; Miller, P.; Schoemann, A.; Quick, C. simsem: SIMulated Structural Equation Modeling. 2018. Available online: https://CRAN.R-project.org/package=simsem (accessed on 15 May 2024).
112. Wittmann, W.W.; Süß, H.M. Investigating the paths between working memory, intelligence, knowledge, and complex problem-solving performances via Brunswik symmetry. In *Learning and Individual Differences: Process, Trait, and Content Determinants*; Ackerman, P.L., Kyllonen, P.C., Roberts, R.D., Eds.; American Psychological Association: Washington, DC, USA, 1999; pp. 77–108.

113. Ding, L.; Velicer, W.F.; Harlow, L.L. Effects of estimation methods, number of indicators per factor, and improper solutions on structural equation modeling fit indices. *Struct. Equ. Model. Multidiscip. J.* **1995**, *2*, 119–143. [CrossRef]
114. MacCallum, R.C.; Browne, M.W.; Sugawara, H.M. Power analysis and determination of sample size for covariance structure modeling. *Psychol. Methods* **1996**, *1*, 130. [CrossRef]
115. Hu, L.; Bentler, P.M. Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives. *Struct. Equ. Model. Multidiscip. J.* **1999**, *6*, 1–55. [CrossRef]
116. What Thresholds Should I Use for Factor Loading Cut-Offs? 2013. Available online: https://imaging.mrc-cbu.cam.ac.uk/statswiki/FAQ/thresholds (accessed on 15 May 2024).
117. Fornell, C.; Larcker, D.F. Evaluating structural equation models with unobservable variables and measurement error. *J. Mark. Res.* **1981**, *18*, 39–50. [CrossRef]
118. Hair, J.F.; Black, W.C.; Babin, B.J; Anderson, R.E *Multivariate Data Analysis*, 7th ed.; Prentice Hall: Upper Saddle River, NJ, USA, 2009.
119. Santos, J.S.; Andrade, W.L.; Brunet, J.; Melo, M.R.A. A Systematic Literature Review on Predictive Cognitive Skills in Novice Programming. In Proceedings of the 2022 IEEE Frontiers in Education Conference (FIE), Uppsala, Sweden, 8–11 October 2022; pp. 1–9.
120. Popat, S.; Starkey, L. Learning to code or coding to learn? A systematic review. *Comput. Educ.* **2019**, *128*, 365–376. [CrossRef]
121. Stadler, M.; Becker, N.; Gödker, M.; Leutner, D.; Greiff, S. Complex problem solving and intelligence: A meta-analysis. *Intelligence* **2015**, *53*, 92–101. [CrossRef]
122. Mayer, R.E. Cognitive, metacognitive, and motivational aspects of problem solving. *Instr. Sci.* **1998**, *26*, 49–63. [CrossRef]
123. Molnár, G.; Alrababah, S.A.; Greiff, S. How we explore, interpret, and solve complex problems: A cross-national study of problem-solving processes. *Heliyon* **2022**, *8*, e08775. [CrossRef] [PubMed]
124. Csapó, B.; Molnár, G. Potential for assessing dynamic problem-solving at the beginning of higher education studies. *Front. Psychol.* **2017**, *8*, 2022. [CrossRef] [PubMed]
125. Krieger, F.; Stadler, M.; Bühner, M.; Fischer, F.; Greiff, S. Assessing complex problem-solving skills in under 20 minutes. *Psychol. Test Adapt. Dev.* **2021**, *2*, 80–92. [CrossRef]
126. Greiff, S.; Molnár, G.; Martin, R.; Zimmermann, J.; Csapó, B. Students' exploration strategies in computer-simulated complex problem environments: A latent class approach. *Comput. Educ.* **2018**, *126*, 248–263. [CrossRef]
127. Wu, H.; Molnár, G. Logfile analyses of successful and unsuccessful strategy use in complex problem-solving: A cross-national comparison study. *Eur. J. Psychol. Educ.* **2021**, *36*, 1009–1032. [CrossRef]
128. Sonnleitner, P.; Keller, U.; Martin, R.; Latour, T.; Brunner, M. Assessing complex problem solving in the classroom: Meeting challenges and opportunities. In *The Nature of Problem Solving*; Csapó, B., Funke, J., Eds.; OECD: Paris, France, 2017.
129. Molnár, G.; Csapó, B. The efficacy and development of students' problem-solving strategies during compulsory schooling: Logfile analyses. *Front. Psychol.* **2018**, *9*, 302. [CrossRef]
130. Hung, J.L.; Shelton, B.E.; Yang, J.; Du, X. Improving predictive modeling for at-risk student identification: A multistage approach. *IEEE Trans. Learn. Technol.* **2019**, *12*, 148–157. [CrossRef]
131. Feldt, R.; Magazinius, A. Validity threats in empirical software engineering research-an initial survey. In Proceedings of the Seke, San Francisco, CA, USA, 1–3 July 2010; pp. 374–379.
132. Janke, S.; Rudert, S.C.; Petersen, Ä.; Fritz, T.M.; Daumiller, M. Cheating in the wake of COVID-19: How dangerous is ad hoc online testing for academic integrity? *Comput. Educ. Open* **2021**, *2*, 100055. [CrossRef]
133. Noorbehbahani, F.; Mohammadi, A.; Aminazadeh, M. A systematic review of research on cheating in online exams from 2010 to 2021. *Educ. Inf. Technol.* **2022**, *27*, 8413–8460. [CrossRef] [PubMed]
134. Greiff, S.; Wüstenberg, S.; Funke, J. Dynamic problem solving: A new assessment perspective. *Appl. Psychol. Meas.* **2012**, *36*, 189–213. [CrossRef]
135. Neubert, J.C.; Kretzschmar, A.; Wüstenberg, S.; Greiff, S. Extending the assessment of complex problem solving to finite state automata. *Eur. J. Psychol. Assess.* **2014**, *31*, 181–194. [CrossRef]
136. Danner, D.; Hagemann, D.; Holt, D.V.; Hager, M.; Schankin, A.; Wüstenberg, S.; Funke, J. Measuring performance in dynamic decision making. *J. Individ. Differ.* **2011**, *32*, 225–233. [CrossRef]


**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.