

# Creating a Trajectory for Code Writing: Algorithmic Reasoning Tasks

Shruthi Ravikumar, Margaret Hamilton<sup>a</sup>, Charles Thevathayan<sup>b</sup>, Maria Spichkova<sup>c</sup>,  
Kashif Ali and Gayan Wijesinghe

*School of Computing Technologies, RMIT University, Melbourne Victoria, Australia*

**Keywords:** Learning Trajectory, Programming Fundamentals, Abstract Reasoning Skills, Learning Analytics.

**Abstract:** Many students in introductory programming courses fare poorly in the code writing tasks of the final summative assessment. Such tasks are designed to assess whether novices have developed the analytical skills to translate from the given problem domain to coding. In the past researchers have used instruments such as code-explain and found that the extent of cognitive depth reached in these tasks correlated well with code writing ability. However, the need for manual marking and personalized interviews used for identifying cognitive difficulties limited the study to a small group of stragglers. To extend this work to larger groups, we have devised several question types with varying cognitive demands collectively called Algorithmic Reasoning Tasks (ARTs), which do not require manual marking. These tasks require levels of reasoning which can define a learning trajectory.

This paper describes these instruments and the machine learning models used for validating them. We have used the data collected in an introductory programming course in the penultimate week of the semester which required attempting ART type instruments and code writing. Our preliminary research suggests ART type instruments can be combined with specific machine learning models to act as an effective learning trajectory and early prediction of code-writing skills.

## 1 INTRODUCTION

Code writing requires students to develop solutions for the given programming problems. Students are expected to combine the various constructs, while reasoning about the overall behaviour of the resulting code. A literature review on topics relating to novice teaching and learning has been introduced in (Robins et al., 2003). Many case studies demonstrated that novices who lack reasoning skills struggle to write code, see for example (Denny et al., 2008), (Lister et al., 2009), (Lister et al., 2006), and (Malik et al., 2019). Thus, to help students succeed in learning programming skills, we have to, first of all, support them in developing reasoning skills.

To determine predecessor skills required for code writing, so-called “explain in plain English” (EiPE) questions have been introduced in (Lopez et al., 2008). These questions are classified at the relational level of the *Structure of the Observed Learning Tax-*

*onomy* (SOLO) taxonomy, which can be used to measure and quantify reasoning skills. The SOLO taxonomy classifies learning and assessment tasks based on hierarchical cognitive levels, see (Biggs and Collis, 1982). The work of Lopez et al. was an experiment designed to challenge the results presented in (McCracken et al., 2001). McCracken et al. conducted a study to assess the programming ability of 216 students from four different universities. They claimed that the students’ performances were poor because of their weak capacity in problem solving. The results of their study also demonstrate that when teaching novice programmers (who may be weak in problem solving skills), an educator should consider assessing the students’ precursor skills, which are their code reading skills. Lopez et al. used the students’ performances on the EiPE question to assess the students’ reasoning skills. The study found a significant relationship (a Pearson correlation of 0.5586) between code reading and code writing.

The Neo-Piagetian theory (Teague et al., 2013) also suggested that the novice programmers progress through various stages of learning and they can un-

<sup>a</sup> <https://orcid.org/0000-0002-3488-4524>

<sup>b</sup> <https://orcid.org/0000-0003-2605-1722>

<sup>c</sup> <https://orcid.org/0000-0001-6882-1444>

derstand the overall purpose of the code once they develop abstract reasoning skills. Students might find problem solving difficult, because it requires combining high level thinking, problem abstraction and algorithm development, with language syntax and code tracing.

There has been extensive exploration of approaches to teach programming, comparing languages, intelligent tutoring systems, pedagogical strategies, and conceptual methodologies, software engineering concepts, etc., see (Echeverría et al., 2017), (Silva-Maceda et al., 2016), (Spichkova, 2019), (Xie et al., 2019), (Thevathayan et al., 2017), and (Spichkova, 2022). Despite these advancements, a recent survey across 161 institutions worldwide revealed the failure rate in the introductory programming courses to be 28%, see (Bennedsen and Caspersen, 2019).

Thus, to have an approach to improve students' performance (without grade inflation) would be really helpful. For example, a scientific approach of learning trajectories in the field of mathematics has been shown to improve students performance, see (Izu et al., 2019). Similarly, it is essential to create a trajectory between code tracing and code writing to improve students code writing abilities. Izy et al. defined tentative theoretical learning trajectories that can guide teachers as they select and sequence learning activities in their introductory courses. In our paper, we aim to create a learning trajectory for code writing based on algorithmic reasoning.

*Contributions.* In this study we have developed a combined approach to predict students' performance based on abstract reasoning, with the goal being to improve the students programming skills. The main contribution of our work is the *Algorithmic Reasoning Tasks (ART)* framework to assess the students reasoning skills. The ART framework includes three different types of questions:

- *ART Detection Type questions*, which require the in-depth study of an algorithm to determine its overall effect;
- *ART Comparison Type questions*, which require identifying different algorithms producing the same effect;
- *ART Algorithm analysis type questions*, which require reasoning about behaviour for specified criteria such as performance.

The framework allows automatic prediction of student performance on code writing, based on their performance on ART type questions.

## 2 BACKGROUND AND RELATED WORKS

In this section, we discuss related work from two research areas, which both provide background for our study. We start with an analysis of existing approaches to assess the programming skills of novice programmers in their early stages of learning. Then we introduce recent work on the application of machine learning for prediction of student performance.

### 2.1 Approaches to Assess Programming Skills in Early Stages of Learning

Lack of progress in the early stages of the learning process can create negative momentum, eventually leading to high failure rates. Alternatively, steady progress can lead to positive momentum as each new concept can help reinforce the earlier foundations. Some multi-institutional studies, e.g. (Fincher et al., 2006), have used map drawing styles to predict the success of programming among novice programmers, also suggesting that problem solving, and logical thinking are important skills necessary to succeed in the course. Results of many studies have highlighted the need to assess reasoning skills forming the basis for code writing early, through appropriate tasks. Some exhaustive descriptive studies, e.g., (Simon et al., 2012). have been conducted to understand the types of questions used in the examinations in programming course. The study of Lopez et al. indicates that there exists a loose hierarchy of skills which the students progress through while learning introductory programming concepts, see (Lopez et al., 2008). Activity Diagrams and Parson's Puzzles have been proposed as instruments that to assess the programming ability because they might correlate better with problem solving than code tracing, see (Harms et al., 2016) and (Parsons et al., 2015). Parson's puzzle tasks were designed to ease novices into code writing by allowing students to piece together code fragments interactively, see (Denny et al., 2008). The Spearman ranking coefficient for code writing also showed closer correlation with Parson's puzzle questions when compared to tracing. Parson's puzzle questions however, limit students' freedom in arriving at a solution. Multiple-choice questions (MCQs), when appropriately designed, can be really effective for testing intermediate levels of programming skills. MCQs can be easily automated, which is a critical advantage of this assessment type. However, to design MCQs really well, is not a trivial task. MCQs were found to be the most preferred assessment types in many different domains, see (Furnham et al., 2011)

and (Kuechler and Simkin, 2003). Students in general felt such tests can improve their exam performance, as they felt more relaxed, see (Abreu et al., 2018).

An approach using Activity diagrams (ADs) was introduced to assess the programming ability of students, see (Parsons et al., 2015). ADs are used to present visually the logical flow of a computer program, and have a notation for sequence, conditional statements, and loops. The Pearson-product moment correlation between the exam questions and class project mark was used to measure the student ability to write the code. However, this measure may not be accurate as students may collude in outside class activities. The importance of developing tasks to help students in their self-assessment was also highlighted in (Cutts et al., 2019).

## 2.2 Application of Machine Learning for Prediction of Students' Performance

Students' performance prediction is one of the earliest and most valuable applications of Educational Data Mining (EDM). A systematic literature review (SLR) on the solutions to predict student performance using data mining and learning analytics techniques has been presented in (Namoun and Alshantiri, 2020). Random Forest (RF) and Linear Regression (LR) algorithms have particularly been used in predicting students' academic performance, where the RF algorithm is in the top 5 algorithms with an accuracy of 98%. According to (Sandoval et al., 2018), in an attempt to find a low-case predictive model using the Learning management system data, RF was found to have higher precision in predicting students who are at risk of failing the course i.e., poor performing students. A similar result was presented in (Chettaoui et al., 2021), where the results of the study demonstrated that, of the five classification algorithms, RF outperformed with 84% accuracy.

The prediction of student performance was applied in the context of e-learning (Abubakar and Ahmad, 2017). The study focused on comparing the state-of-the-art classifier algorithms to identify the most suitable for creating a learning support tool. Two comparative studies were conducted using different data sets comprising 354 and 28 records. Findings of that study revealed that the Naive Bayes algorithm achieved 72.48% accuracy, followed by LR with 72.32%. To predict student's success in electronics engineering licensure exam, 500 student's data over different cohorts from 2014 - 2019 was used, see (Maaliw, 2021). The study used RF algorithm to predict the student exam outcome by using 33 different

features and had a prediction accuracy of 92.70%.

An extensive evaluation of machine learning algorithms such as Decision Tree, Naive Bayes, Random Forest, PART and Bayes Network was conducted on 412 postgraduate students' data to predict their academic performance in the current semester. The study (Kumar and Singh, 2017) demonstrated that Random Forest (RF) gave the best performance with precision (1) which is essential in identifying students that are likely to fail at the early stages of the course.

In a study presented in (Meylani et al., 2014), Neural Network (NN) and LR algorithms were evaluated with respect to prediction of students' performance in mathematics examination. The LR models (Linear, Multi-nominal and Ordinal) outperformed the NN models. The study used the student's performance to in-class mathematics tests to predict if the students would pass, fail or excel in the final exam.

## 3 METHODOLOGY

For the study reported in this paper, data was collected through an in-class test facilitated through Google forms. The test consisted of two parts where the first part included objective questions, and the second part included code writing questions. The first part had 12 objective questions which included tracing questions and three different types of ART questions, explained below. Students were advised to spend about 30 minutes on the objective questions (Tracing and Algorithmic Reasoning Tasks) and the remaining 80 minutes on three code-writing questions.

### 3.1 Design of an Abstract Reasoning Task (ART) Type Question

The ART type questions were designed to get relational level responses from students by making tracing difficult or impossible within allocated time. The ART-Detection question shown in Figure 1 was used in the study, where students had to extract the purpose of the algorithm and express it by writing the correct output for different inputs. Given 6 different input arrays with 7 or more values in each, students who have not extracted the overall effect of the algorithm and apply it to different inputs are unlikely to get all the outputs correct. The students are only awarded the mark if they get all the outputs correct.

In the past getting relational responses from students primarily relied on the code-explain instrument. The main drawback of code-explain is the need for manual marking. The ART-detection instrument we have designed requires a similar relational response

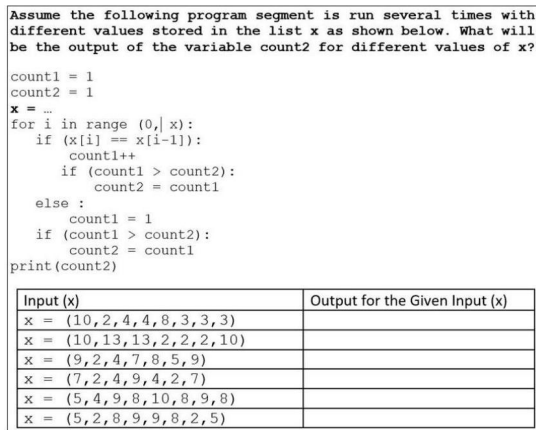


Figure 1: ART Detection Type question used in the study.

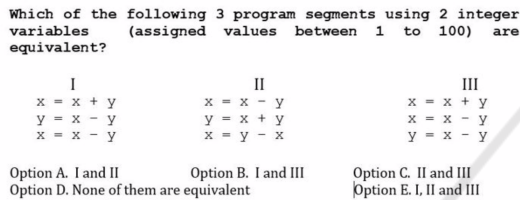


Figure 2: ART Comparison Type question used in the study.

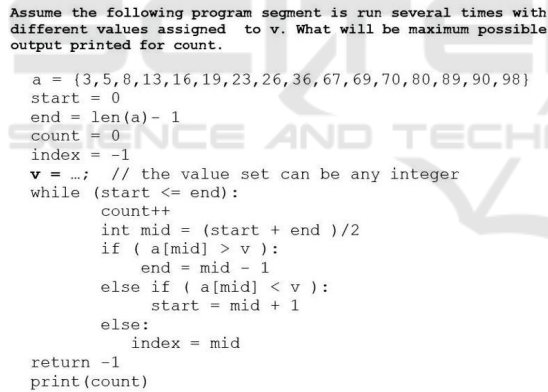


Figure 3: ART Analysis Type question used in the study.

from students who firstly need to understand the overall effect of the algorithm before applying it repeatedly under the time constraint. Similarly, the ART-comparison instrument requires the students to identify the algorithms that have similar behaviour while the ART-analysis instrument requires students to analyse the algorithm and identify the best- and worst-case scenarios. The SOLO classification of the objective questions used in this study and their purpose are given in the below Table 1. The main advantage with all three ART instruments is that it can be automated and that the marking will not be subjective. The ART-Comparison question and ART-Analysis ques-

tion used in the study are shown in Figures 2 and 3.

### 3.2 Code Writing

This test involved three code writing tasks. We consider that the process of writing or creating a block of code to answer the given code writing question involves:

- Problem Analysis, which requires analysing and understanding the problem domain;
- Solution Planning, which requires creating the steps in coming up with a viable solution (algorithm) to solve the problem;
- Coding, which involves converting the problem domain into program domain by combining the different constructs (for-loop or while-loop, etc.) and syntax, to write programming code which computes the required answer(s).

### 3.3 Marking

A positive grading scheme was used for the objective questions:

- 1 mark was awarded for a correct answer,
- 0 marks were awarded for an incorrect answer (an answer which is only partially correct was labelled as incorrect), and
- no negative marks were awarded for incorrect answers.

For the code writing tasks, partial marking was applied depending on whether the given code was partially correct or fully correct.

Each Code Writing question carried 3 marks. Grading was done by an experienced lecturer.

### 3.4 Choice of Machine Learning Model

Because we are predicting the student performance based on their score for an in-class assessment, it is important to ensure the algorithm is outlier resistant and scalable. We analysed several automated approaches to predict the students' performance, see Section 2.2. Their results highlighted that application of the Random Forest algorithm is the most promising direction for prediction of students' performance. We trained the model using both Random Forest (RF) and Ordinal Logistic Regression (LR) algorithms because of the inherent nature of the variables. Therefore, in this study we report the performance of both these models in predicting the student performance in code writing.



Table 1: Classification of Objective Questions using SOLO Taxonomy.

Question Type	SOLO Level	Purpose
Tracing	Multi-Level	Requires the students to understand each statement and trace each line in the in the given piece code and determine the output
Detection	Relational level	Requires abstraction skills to detect what the role of the algorithm. Students are expected to apply cognitive skills at relational level to analyze how the behavior will change for different inputs.
Comparison	Relational level	Students are expected to identify algorithms which will display the same collective or composite behavior considering different input values.
Analysis	Relational level	Students are expected to analyze an algorithm including working out worst case scenarios considering all possible paths.

### 3.5 Data Exploration and Training

In this study we have used the students' performance on the objective questions (tracing and ART questions) to predict students' success in code writing. Data used in this study consists of 243 students' performances on 15 programming questions including both objective and code writing questions. The students' data was loaded into Google drive and mounted on to Google Colab Notebook.

We performed the data profiling using the python panda library to visualise the data. Data pre-processing was performed to identify any duplicates, missing values and data types. The data was encrypted to hide any student's identifying information to ensure privacy and confidentiality. The data encoding of the independent variables was performed to ensure all the data in the dataset are of the same datatype. The unique code writing marks in the dataset 0, 0.5, 1, 1.5, 2, 2.5 and 3 were encoded to 0, 1, 2, 3, 4, 5 and 6 respectively.

The next step in the data preparation was to separate the dependent and independent variables:

- The *dependent variables* are the features which we train the model on. In our study, the features are the students' scores to the objective questions.
- The *independent variables* are also called the labels. In our study, these are the students score to code writing programming questions.

From our data exploration using the data profiling in Python we found that the data set had a class imbalance. To overcome the issue of class imbalance in the dataset the following steps were performed:

- Step 1. Firstly, we identified the class labels (0, 1, 2, 3, 4, 5, 6) whose frequency in the dataset is less than 10 and these class labels were removed from the dataset.
- Step 2. The dataset is then split into training and testing dataset using the *train\_test\_split\_function* from Scikit-learn (sklearn) model selection.
- Step 3. The class labels whose frequencies were less than 10 (identified in Step 1) were added to

the training split of the dataset to ensure that the machine learning algorithm had enough data on these labels to train on. The dataset is now ready to be trained by the chosen machine learning algorithms.

The training split of the dataset was then trained using the Scikit-learn (sklearn) ensemble *Random-Forestregresso* and *LIBLINEAR* Python library. The 10-fold (k-fold) cross validation technique was then applied to both the models using the *GridSearchCV* function from sklearn. 10-fold cross validation involves randomly dividing the training data into 10 folds. The first fold was considered as the testing set and the model was fitted on the remaining 9 folds of the data.

We have compared the effectiveness of both the trained data models using the following metrics shown in below Table 2. In addition to the above metrics, different stratified training and test splits were used to evaluate the performance of the models.

## 4 RESULTS

Table 3 presents a performance comparison of the Random Forest (RF) and Logistic Regression (LR) models for different training and test splits in predicting students' performance to code writing questions. RF attained the highest cross-validated (10-fold) accuracy score of 85.45%. In comparison to the LR model, the RF model has a high precision of predicting students who are at the risk of failure (getting 0 or 0.5 marks in code writing).

Figures 4 and 5 demonstrate the precision, recall and F1-score of both models in predicting students that are at risk of failure. According to our analysis, RF algorithm performs better compared to LR in precision, recall and F1-measure: RF has a high precision of 0.90 and 1 in predicting students that are to score 0 and 0.5 marks in code writing tasks. This precision is vital in identifying students that are at risk to fail in the programming course. Based on our analy-

Table 2: Performance metrics to evaluate the *modelPerformance* metrics to evaluate the model.

Metric	Definition	Formula
<i>Accuracy</i>	It is the proportion of correction predictions, i.e., both true positives (TP) and true negatives (TN), among total number of cases examined	$= (TP + TN) / (TP + TN + FP + FN)$
<i>Recall</i>	The ability of a model to find all the relevant cases within a data set	$= (TP / (TP + FN))$
<i>Precision</i>	Precision, also known as positive predictive value, is the fraction of relevant instances among the retrieved instances	$= (TN / (TN + FP))$
<i>F - Measure</i>	It is the harmonic mean of precision and recall	$= (2 * Precision * Recall) / (Precision + Recall)$

sis it was clear that RF was more suitable models for predicting the student performance, which also correlated with the results presented in the related works, see Section 2.2.

To further evaluate the RF model for applying the ART type questions to predict the students code writing performance, we have analysed the *feature importance*: The RF model computes which feature contributes the most to decrease the weighted impurity. The feature with highest value contributes the most in predicting the variable i.e, code writing. We have used the Scikit-learn to obtain the feature importance.

Figure 6 shows the feature importance computed by the Random Forest model. Based on our analysis, we found that the student’s performance on ART Comparison type questions (0.11) had a significant influence on the student code writing results, followed by ART Detection type questions (0.079) and ART Analysis questions (0.074). The least contribution was from the tracing question grades (0.072). This confirms that the RF model uses the ART type questions to predict the students code writing abilities.



Figure 4: Performance metrics of RF and LF models in predicting the students scoring 0 marks.

Table 3: Comparison of accuracy of RF and LF models.

Train-Test split	Accuracy	
	RF	LR
75 – 25	85.45%	46.45%
70 – 30	81.82%	41.81%

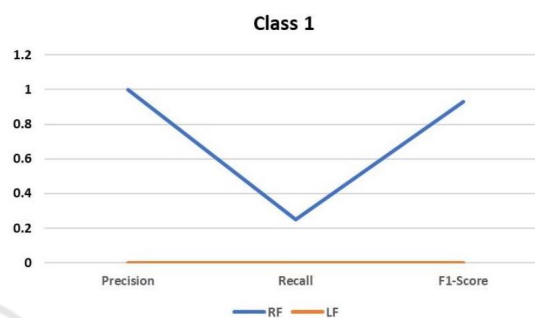


Figure 5: Performance metrics of RF and LF models in predicting the students scoring 0.5 marks.

Table 4: Spearman Rank Correlation with Code Writing for tracing, comparison (C), detection (D), analysis (A) and average correlation for all ART instruments.

Non-ARTs	ARTs			
	C	D	A	average
Tracing	0.69	0.68	0.74	0.70

Table 5: Comparison of Pearson correlations of ART, Activity Diagrams, and Parson’s Puzzle with code writing (the data on both Activity Diagrams and Parson’s Puzzle have been provided in (Parsons et al., 2015)).

ART	Activity Diagrams	Parson’s Puzzle
0.37	0.26	0.12

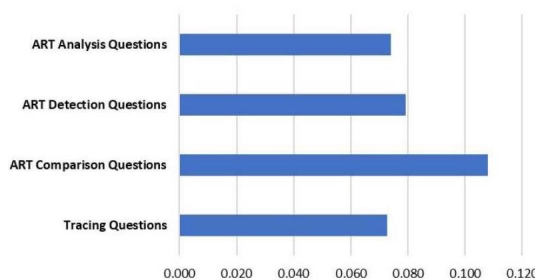


Figure 6: Feature Importance of Random Forest.

The Spearman correlation coefficients, presented in Table 4, affirm a very strong positive relationship between ART Analysis type questions and the code writing question at 0.74. In addition, ART Comparison (0.69) and ART Detection (0.68) have a positive

correlation with code writing. The ART type instruments as a whole have higher positive correlations of 0.70 compared to what we can get for tracing (non-ART instrument), which was only 0.63.

Based on our findings summarized in Tables 3 and 4 as well as Figure 6, it is clear that the ART instrument types have strong connection to students' code writing performance. Thereby confirming ART instruments type can assess the needed relational skills required to excel in coding. This confirms that ART instruments can be used to assess the programming abilities of the students and to identify the students who are at the risk of failure in the early stages of the course. Moreover, comparing our results with the results on Activity Diagrams and Parson's Puzzles from (Parsons et al., 2015), we found that the ART instruments have a slightly higher Pearson correlation (0.37) compared to Activity Diagrams (0.26) and Parson's Puzzles (0.12), see Table 5.

Based on the above findings, our study affirms that ART type instruments can be used to assess the students' programming abilities at the early stages of the course to reduce failure rates and the RF model is more suitable compared to LR to automatically identify students who might require additional support for learning code writing skills.

## 5 CONCLUSION AND FUTURE WORK

In many cases, students struggle with programming mainly through the lack of problem-solving ability. Problem solving ability is also central for students having to apply scientific and mathematical principles to solve real world problems. In recent years, learning trajectories resulting from a research-based curriculum development approach have benefited student learning mathematics, by modelling their thinking process, see (Izu et al., 2019). However, there has been little research done on how such an approach can foster problem solving and code writing skills in novice programmers.

Our research has therefore focused on supplementing traditional and well-researched instruments such as program tracing with tasks demanding more relational thinking gradually. We have developed ART-type questions such as comparison, detection, and analysis which require students to map from problem domain to the solution domain (coding). We are following a scientific approach by using machine learning models that identify tasks which are progressively more complex but lead gradually to skills needed for problem solving and code writing. The

main novelty of our approach is the ability to automate the process of assessment feedback by creating a trajectory of tasks which require no manual intervention and by predicting the students programming abilities based on these trajectory tasks.

The data collected with students have clearly revealed tasks demanding relational level responses which better correlate with code writing when compared to tracing. Our approach involved developing a trajectory of tasks rooted in a multidimensional framework that combined different levels of the SOLO taxonomy with multiple domains. It is our belief such an approach can lead to better learning outcomes in coding as code writing requires analytical ability to understand the problem domain, as well as abstraction skills and the ability to come up with algorithms which can be coded and implemented.

ART type instruments designed to get relational level responses showed greater correlation with code writing when compared to tracing. The Random Forest regression model had an accuracy of 84.5% in predicting student success in code writing based on algorithmic reasoning tasks. The Spearman Rank Correlation coefficient was substantially higher for ART types when compared to tracing. Within the Algorithmic Reasoning Tasks, comparison questions showed substantially higher feature importance when compared to tracing, ART-detection and ART-analysis. Our preliminary results show new types of instruments that gather relational responses can be developed resulting in greater similarity to the reasoning skills needed in code writing. We have shown that these ART type instruments form a loose trajectory starting from Code Tracing Questions to ART Analysis questions, to ART Detection questions, to ART Comparison questions to code writing. By classifying such instruments using machine learning, novices can be provided with a learning trajectory that equips them better with the cognitive skills needed for code writing.

Our preliminary studies with tasks combining multiple domains (problem domain and coding) suggest these tasks demand even greater cognitive depth. To further evaluate we would consider collecting more data over the next few semesters. We also aim to use the final exam results and predict the student success in passing or failing the exam based on how well they perform in ART questions in their mid-semester. We also would like to identify more instruments that can be used to assess the students' programming abilities.

## REFERENCES

- Abreu, P. H., Silva, D. C., and Gomes, A. (2018). Multiple-choice questions in programming courses: Can we use them and are students motivated by them? *TOCE*, 19(1):1–16.
- Abubakar, Y. and Ahmad, N. B. H. (2017). Prediction of students' performance in e-learning environment using random forest. *IJIC*, 7(2).
- Bennedson, J. and Caspersen, M. E. (2019). Failure rates in introductory programming: 12 years later. *ACM inroads*, 10(2):30–36.
- Biggs, J. and Collis, K. (1982). A system for evaluating learning outcomes: The solo taxonomy.
- Chettaoui, N., Atia, A., and Bouhleb, M. S. (2021). Predicting student performance in an embodied learning environment. In *MIUCC*, pages 1–7. IEEE.
- Cutts, Q., Barr, M., Bikanga Ada, M., Donaldson, P., Draper, S., Parkinson, J., Singer, J., and Sundin, L. (2019). Experience Report: Thinkathon—Countering an "I Got It Working" mentality with pencil-and-paper exercises. In *ITiCSE*, pages 203–209.
- Denny, P., Luxton-Reilly, A., and Simon, B. (2008). Evaluating a new exam question: Parsons problems. In *ICER*, pages 113–124.
- Echeverría, L., Cobos, R., Machuca, L., and Claros, I. (2017). Using collaborative learning scenarios to teach programming to non-cs majors. *CAEE*, 25(5):719–731.
- Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., de Raadt, M., Haden, P., Hamer, J., Hamilton, M., Lister, R., et al. (2006). Predictors of success in a first programming course. In *ACE*, pages 189–196.
- Furnham, A., Batey, M., and Martin, N. (2011). How would you like to be evaluated? the correlates of students' preferences for assessment methods. *Personality and Individual Differences*, 50(2):259–263.
- Harms, K. J., Chen, J., and Kelleher, C. L. (2016). Distractors in parsons problems decrease learning efficiency for young novice programmers. In *ICER*, pages 241–250.
- Izu, C., Schulte, C., Aggarwal, A., Cutts, Q., Duran, R., Gutica, M., Heinemann, B., Kraemer, E., Lonati, V., Mirolo, C., et al. (2019). Fostering program comprehension in novice programmers-learning activities and learning trajectories. In *ITiCSE-WGR*, pages 27–52.
- Kuechler, W. L. and Simkin, M. G. (2003). How well do multiple choice tests evaluate student understanding in computer programming classes? *ISE*, 14(4):389.
- Kumar, M. and Singh, A. (2017). Evaluation of data mining techniques for predicting student's performance. *MECS*, 9(8):25.
- Lister, R., Fidge, C., and Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *ACM SIGCSE Bulletin*, 41(3):161–165.
- Lister, R., Simon, B., Thompson, E., Whalley, J. L., and Prasad, C. (2006). Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. *ACM SIGCSE Bulletin*, 38(3):118–122.
- Lopez, M., Whalley, J., Robbins, P., and Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. In *ICER*, pages 101–112.
- Maaliw, R. R. (2021). Early prediction of electronics engineering licensure examination performance using random forest. In *AIIoT*, pages 41–47. IEEE.
- Malik, S. I., Mathew, R., Al-Nuaimi, R., Al-Sideiri, A., and Coldwell-Neilson, J. (2019). Learning problem solving skills: Comparison of e-learning and m-learning in an introductory programming course. *EIT*, 24(5):2779–2796.
- McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. In *ITiCSE-WGR*, pages 125–180.
- Meylani, R., Bitter, G. G., and Castaneda, R. (2014). Predicting student performance in statewide high-stakes tests for middle school mathematics using the results from third party testing instruments. *Journal of Education and Learning*, 3(3):135–143.
- Namoun, A. and Alshantiti, A. (2020). Predicting student performance using data mining and learning analytics techniques. *Applied Sciences*, 11(1):237.
- Parsons, D., Wood, K., and Haden, P. (2015). What are we doing when we assess programming? In *ACE*, volume 27, page 30.
- Robins, A., Rountree, J., and Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer science education*, 13(2):137–172.
- Sandoval, A., Gonzalez, C., Alarcon, R., Pichara, K., and Montenegro, M. (2018). Centralized student performance prediction in large courses based on low-cost variables in an institutional context. *IHE*, 37:76–89.
- Silva-Maceda, G., Arjona-Villicana, P. D., and Castillo-Barrera, F. E. (2016). More time or better tools? *Transactions on Education*, 59(4):274–281.
- Simon, Chinn, D., de Raadt, M., Philpott, A., Sheard, J., Laakso, M.-J., D'Souza, D., Skene, J., Carbone, A., Clear, T., et al. (2012). Introductory programming: examining the exams. In *ACE*, pages 61–70.
- Spichkova, M. (2019). Industry-oriented project-based learning of software engineering. In *ICECCS*, pages 51–60. IEEE.
- Spichkova, M. (2022). Teaching and learning requirements engineering concepts: Peer-review skills vs. problem solving skills. In *RE*, pages 316–322. IEEE.
- Teague, D., Corney, M., Ahadi, A., and Lister, R. (2013). A qualitative think aloud study of the early neo-piagetian stages of reasoning in novice programmers. In *ACE*, pages 87–95. ACS.
- Thevathayan, C., Spichkova, M., and Hamilton, M. (2017). Combining agile practices with incremental visual tasks. In *ACE*, pages 103–112.
- Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H., Tan, A. H., Hwa, L., Li, M., and Ko, A. J. (2019). A theory of instruction for introductory programming skills. *CSE*, 29(2-3):205–253.