Received August 16, 2020, accepted September 8, 2020, date of publication September 14, 2020, date of current version September 24, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3024102

Analysis of Learning Behavior in an Automated Programming Assessment Environment: A Code Quality Perspective

HSI-MIN CHEN, BAO-AN NGUYEN, YI-XIANG YAN, AND CHYI-REN DOW
Department of Information Engineering and Computer Science, Feng Chia University, Taichung 40724, Taiwan

Corresponding author: Hsi-Min Chen (hmchen@mail.fcu.edu.tw)

This work was supported by the Ministry of Science and Technology of Taiwan, under Grant MOST 107-2221-E-035 -025 -MY2 and Grant MOST 108-2745-8-035 -005.

ABSTRACT Automated programming assessment systems are useful tools to track the learning progress of students automatically and thereby reduce the workload of educators. They can also be used to gain insights into how students learn, making it easier to formulate strategies aimed at enhancing learning performance. Rather than functional code which is always inspected, code quality remains an essential aspect to which not many educators consider when designing an automated programming assessment system. In this study, we applied data mining techniques to analyze the results of an automated assessment system to reveal unexpressed patterns in code quality improvement that are predictive of final achievements in the course. Cluster analysis is first utilized to categorize students according to their learning behavior and outcomes. Cluster profile analysis is then leveraged to highlight actionable factors that could affect their final grades. Finally, the same factors are employed to construct a classification model by which to make early predictions of the students' final results. Our empirical results demonstrate the efficacy of the proposed scheme in providing valuable insights into the learning behaviors of students in novice programming courses, especially in code quality assurance, which could be used to enhance programming performance at the university level.

INDEX TERMS Automated programming assessment system, code quality, educational data mining, early learning achievement detection, programming education.

I. INTRODUCTION

Assessment and feedback are essential tasks of educational activities that enable students and lecturers to keep track of the performance of the learning process. Through assessments, students can understand their strengths and weaknesses in certain learning objectives, then they can improve them based on provided feedback. In practical courses such as computer programming, these tasks are even more important because students rarely achieve acceptable solutions in some first tries. However, giving frequent assessments and detailed feedbacks significantly increases the workload of lecturers. Consequently, many educators have developed Automated Programming Assessment Systems (APASs) to enhance programming assessments in their courses automatically.

The associate editor coordinating the review of this manuscript and approving it for publication was Laxmisha Rai.

APASs are widely accepted by both students and institutions via their fruitful benefits. First, they reduce the educators' workload in code evaluation as well as provide accurate grading and immediate feedback to learners[1]. Second, the iterative process of resubmission/assessment by APASs provides students learn-by-error experiences, in which students have chances to correct their mistakes by inspecting immediate feedbacks and resubmit their solutions until reaching a successful state. Leveraging APASs frees educators from the constraints imposed by a lack of tutoring resources [2] as well as enables Massive Open Online Courses (MOOCs) frameworks to scale up their capacities. In their deployment environment, APASs often provide statistical reports visualized in dashboards to students and educators [3], [4].

In their natural targets, most of APASs aim to detect syntax errors, run-time errors, and functional errors from students' code. However, it is suggested that novice programmers should be assessed by software engineering metrics rather than grading schemes based solely on functional correctness [5]. The static quality of code also has been recently suggested as criteria that should be considered and optimized in students' programming portfolios [6]. This aspect can be assessed by inspecting adherence of students to coding conventions [7], a set of guidelines aimed at improving the readability and maintainability of software [4]. Nonetheless, many APASs fail to adequately address this issue.

In our previous work [4], we developed an automated assessment tool (referred to as ProgEdu), which was featured by coding quality analysis functions for students' Java assignment. To facilitate students to early understand concepts and tools of software engineering, we employed Git[8] as an assignment submission tool and adopted GitLab[9] as a code repository of ProgEdu, in which students use Git's protocol to commit their homework code into GitLab repository. Embedded code analysis modules are leveraged to check the correctness of submissions in terms of syntax, coding convention, and functionality. Since 2017, this system has been used in courses on Java programming for undergraduate computer science students at Feng-Chia University. Free resubmission policy was adopted to encourage students to pursue clean coding. Students are allowed to receive immediate feedback and resubmit their improved solutions several times. This scheme promotes reflective learning that is a form of education in which students are motivated to make critical thinking on the drawbacks of the current solution and to improve it in the future [10]. As a large number of submissions were made by students during the course, ProgEdu forms a fruitful dataset about programming improvements of students.

The effectiveness of ProgEdu has been analyzed through research on 69 students which was presented in our other report [11]. Those analysis results show high acceptance of students on the usage of ProgEdu and the iterative learning approach. With aiming to understand the learning behavior on code quality improvement on programming courses, we analyzed the log data from ProgEdu to address the following research questions:

- RQ1: How is the ability regarding code quality improvement of students associated with their learning results?
- RQ2: What student groups can be formed based on the learning behavior of students in code quality improvement?
- RQ3: Does effort to improve code quality imply the learning performance of students?
- RQ4: Is it possible to build a prediction model capable of early identifying at-risk students based on their behaviors in code quality improvement?

We answer these research questions by developing an Educational Data Mining (EDM) workflow to facilitate the analysis of student behaviors using APSP log data. EDM is an approach to extracting potential-valuable knowledge from data in the field of education. This multi-disciplinary approach deals with statistics, probability, machine learning, and artificial intelligence [12]. The review article [13] reported that EDM can be used for a range of applications: (i) student modeling, (ii) student behavior modeling, (iii) student performance modeling, (iv) student assessment, (v) student support and feedback, (vi) curriculum, domain knowledge, sequencing, and teacher support. Results of these EDM tasks are often presented as dashboards to students/teachers, in which students can be noticed about their learning performance or at-risk situation [14]. In this research, both descriptive statistics, data visualization, unsupervised and supervised learning were leveraged to answer the research questions.

APASs do not only serve as learning assessment tools but also gather a great deal of data related to student submissions, including the time of submissions, source codes, grading results, code analysis reports, and stack traces. The code analysis reports represent a vast repository of data resulting from syntax checking, coding convention checking, and unit testing. Adopting resubmission policy in APASs has made available a numerous volume of log data that records the learning progress of students. These data could be explored to gain insight into the coding behaviors of novice programmers [15].

A literature review of learning analytics conducted in 2015 reported that 43.4% of inspected papers (33 per 76) used high levels of logged data from students' submissions in their analysis [16]. However, to make a precise picture of students' status, those systems usually require multiple sources of data that are not always available such as demographics information and academic transcript of previous courses [17]. Alternatively, applying EDM based on in-course data is a potential approach to simplifying data acquisition tasks and gains the reproduction ability of conducted works [18]. This approach is suitable for our context with the data acquired by ProgEdu.

In our analysis, only the log data from students' submissions to ProgEdu was employed. Cluster analysis was used to identify groups of students with similar learning behaviors and achievements. Our analysis of cluster profiles revealed several factors that are strongly correlated to final grades, including submission effort, time usage, and behaviors in code quality improvement. We also identified five groups of novice programmers exhibiting obvious differences in their approach to fix programming failures. These differences should be considered when making academic recommendations, adjusting class policies, establishing learning resources, and dealing with at-risk students. This work assumes that learning behavior in the early stages of a course is crucial to ultimate academic success. Thus, we built a prediction model to detect at-risk students based on programming assignments in the first half of the course. The 87% prediction accuracy and 73% F1-measure of the proposed scheme provide further evidence that learning success can be predicted using EDM with limited data. Our results demonstrate the potential benefits of applying EDM to improving the quality of training procedures at the university level. The use of existing tools permits the replication of the proposed workflow by other educators with minimal effort.

The remainder of this paper is structured as follows: Section 2 presents a literature review on the use of EDM in programming courses. Section 3 outlines the proposed methods and results. A discussion and brief conclusion are presented at the end of the paper.

II. RELATED WORK

Despite considerable variation in approaching methods, most instructors share similar expectations in assessing the quality of their students' work when building an APAS [19]. First, APASs satisfy students with timely, accurate, and fair feedbacks. Second, APASs help in reducing educators' workload. Last but not least, the operational database of APASs introduces a programming portfolio of students which can be learned to improve education quality [20].

A wide range of APASs has been developed for MOOCs as well as conventional courses [2]. Almost existing APASs focus on C, Java, and Python languages [21], [22]–[24]. In MOOCs environment, automated assessment plugins integrated within course management systems (e.g., Moodle [25]) are increasingly interested to avoid rebuild course manage features [2], [18]. With standalone courses, educators often build their own APASs with a set of features including assignment delivery and submission interface, code compilers and analyzers, plagiarism detector, feedback interface, and operational database [26]. Recently, the benefits of Git [27] and GitHub were employed in classrooms as a version control system and a tool for collaborative coding activities for students, respectively [28], [29]. Motivated by the potential outcome of using Git and GitHub, we designed ProgEdu, the APAS used in this paper, by integrating open-source tools to facilitate code submissions with extensions to verify the correctness of the code [4].

Since code quality is not popularly considered strictly in almost programming courses, the relationship between code quality and learning achievement is rarely reported. By analyzing over one million Java source files submitted into the automated grading system Web-CAT, the authors of [30] presented interesting statistics. Among types of error, formatting, and documentation are the most common types. The error rate at first submission is six times higher than that in the last submission on average. The mean of durations for students to remove a static error is 20 hours. There is a relationship between the code quality of a specific exercise and its grade. Notably, the presence of coding flaws in final submission associates with critical lower scores. These results announce educators to pay more attention to the code quality assurance aspect in their courses. Otherwise, this ability of students will not be improved over the years, as reported in [31] that there is no significant difference in the code quality of first-year and second-year students.

As ProgEdu was used for homework submission of students, we need to investigate studies about the homework behavior of students. Among factors related to homework which associate with learning achievement, homework effort and homework time are two important aspects of drawing characters of students' homework behavior [25]. Research framework proposed by Trautwein *et al.* leveraged latent profile analysis (LPA [34]) on various aspects of homework to evaluate students' behaviors and predict their academic achievement [25], [26]. While people believe that homework time and achievement are weakly associate, the authors of [26] argued that time spent on homework is a robust predictor of achievement in the combination with homework effort. Based on these two factors, students can be segmented into five possible groups in Figure 1, in which the naming of the groups is based on the relationship between learning behaviors and learning outcomes. We aim to adapt this framework in our context to categorize our students based on homework behavioral patterns.

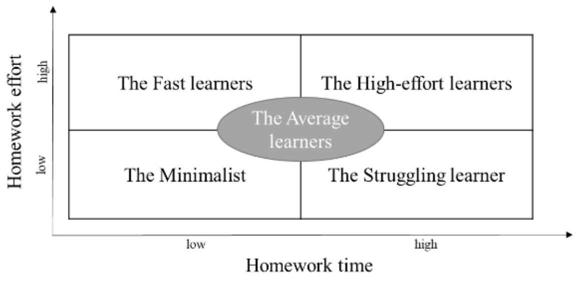

FIGURE 1. **Possible homework learning types.** ***Source: [26].***

Student segmentation is among the most important problems in EDM, wherein students' profiles are created based on partitions using hierarchical clustering techniques. The resulting cluster profiles are then used to highlight learning patterns. With this motivation, the author of [35] examined the relationships among student behavior, code resubmissions, and final grades using data gathered from the APAS TRAKLA [36]. That study identified five groups of students: *Talented, Ambitious, Iterators, Ordinaries,* and *Passers*. Two of the groups (*Talented* and *Passers*) could be early identified from the beginning of the course. The author recommended that the number of resubmissions should be limited to avoid the trial-and-error strategy of some low effort students.

In [37], the authors divided students into four modes of programming based on patterns in their programming results. Those patterns were the product of problem-solving duration, compilation and execution intervals, compilation frequency, number of errors, and final grade. Ineffective characteristics of each mode are clarified in cluster profile analysis which enables teaching assistants to provide more effective supports.

The authors of [38] introduced a recursive clustering approach using students' performance in terms of CGPA, prerequisite courses, co-requisite courses, and current course. Each group of students was assigned specific exercises. After a while, students were re-clustered based on their new performance on solving given problems. The process was repeated three times so that low-performance students can be detected early by in-class assignments.

Similarities in the learning behaviors of students are also means of learning achievement prediction. With an assumption that students having similar learning behaviors share similar achievement, a visual tool for students' performance in theoretical and practical assessed activities were introduced in [18]. In this scheme, a two-dimension scatter plot was used in which position of points in the plot represent the performance of students, the color intensity of a point denotes the similarity of that student to a *"referenced student"*. Based on similarities computed using the Bray-Curtis metric between learning behaviors of students in the current course and referenced students in the previous edition of the course, both failure/success estimation and numeric score estimation can be performed.

The use of machine learning to predict learning performance has attracted considerable attention from educators. Most of this work has been conducted using Decision Trees (DT), Support Vector Machines (SVM), Linear/Logistic regression, Naive Bayes (NB), K-Nearest Neighbors, artificial neural networks (ANN), and random forest (RF) algorithms. These methods can be used for the early identification of at-risk students [39] or for predicting learning outcomes in MOOCs [40]. Ensemble methods based on traditional classifiers were also used to boost the prediction performance [41]. Long short-term memory (LSTM) [42] is a learning model based on a recurrent neural network (RNN) that has also been used to predict learning status in MOOCs [43].

Statistical attributes extracted from programming activities can be used as significant predictors of the final achievements of students. As reported by an analysis of 89,979 assignment submissions of 1,101 students to the Web-CAT grading system, those students who received A/B grades start and finish their programming assignments earlier than students received C/D/F grades [44]. By incorporating the assessment service Test My Code [45] in an introductory programming course with 52 students, the authors of ([46] built prediction models for students' success in the concurrent mathematics course using students' snapshots describing weekly statistics on programming activities such as working hours, minutes to deadline, minutes between sequential snapshots, minutes spent programming, percentage of assignment done during the night, number of compilations and style errors, etc. These researches are consistent in proving the relationship of time-related attributes to student achievements. However, students' behaviors in certain states of failure within each assignment were not considered yet. In this paper, we conduct a deeper analysis of this aspect, to determine the association between the ability of code quality improvement and the learning performance of students.

Early warning or detection at-risk students are interesting and have a high educational impact. The research [47] used ANN on demographics information and learning conditions of students to identify school dropout risk groups of students in Brazil and reported the prediction accuracy of 76%. The article [48] presents two approaches to early predict students' final grades. In the first approach, study-related patterns and social behavior of students were inputted to classification and regression models to improve the prediction from the first quarter of the course. In the second, collaborative filtering was leveraged to predict the final grade of a student based on previous achievements of similar students. By employing previous academic results of students, the author of [49] presented a mining framework to identify potential dropout students.

To achieve the early prediction target, the aforementioned studies require large datasets derived from LMSs or other sources to obtain data on demographics, academic background, and learning behaviors of students. However, the practicality of such studies is limited due to the less availability and accessibility of needed data. In such cases, in-course data mining is a good deal for those educators who aim to understand their students' learning behavior in time without using any external data [18]. The detection model of at-risk students in this paper was implemented based on this approach. Its results and the comparison with similar works are present in section IV.

III. DATA AND METHODOLOGY

A. DATA

This section details the source of background data from which relevant features were extracted.

1) DATA CHARACTERISTICS

The dataset used in this paper was compiled by ProgEdu [8], an APAS developed by and operated in the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. The instructional scheme employed at our institution is based on iterative learning [40], which means that students can submit their code and obtain feedback in a cycle, which repeats until all of the requirements of the assignment have been satisfied or the deadline comes due. We extracted log data of homework submissions from 69 students enrolled in a Java programming course during the spring semester of the 2018-2019 academic year. Originally, 79 students registered for the course. 10 students decided to drop out after the midterm exam. Since they did not engage in all given assignments, we eliminated their data from the dataset.

The course addressed in this study included six homework (HW) assignments, three of which were assigned before mid-term exams and three after. The students were given one week to finish each assignment. The requirements of the assignments were available on the ProgEdu web interface, and the initial sample code was downloadable from the system repository. Upon each submission, the code was immediately analyzed at multiple levels using (i) a compilation test to ensure that the code was free from syntax errors, (ii) a unit test to determine whether all functional requirements were satisfied, and (iii) a code quality test to detect violations of the coding convention. The test results were encoded as follows:

- *CPF*: *Compilation Failure* – The program could not be compiled due to syntax errors.
- *UTF*: *Unit Test Failure* – The program does not me *et al*l assignment requirements in terms of functionality.
- *CSF*: *Coding Style Failure* – Code quality requirements (adherence to established conventions) are not satisfied.
- *S*: *Success* – All tests are passed.

The grading scheme used in that course included three parts with equal weights: homework assignments, midterm exam (writing test), and final exam (writing test). All student assignments were evaluated using a scale of 1 to 100. Students who earned a final grade of not less than 65 passed the course. We obtained log data 1310 records related to feedback on student code, including 4 fields: student ID, homework ID, timestamp of submission, and testing results. The average number of submissions per student was 19 (sd = 7.5), the minimum value was 9 and the maximum value was 44. The density plot of the number of submissions within six homework is shown in Figure 2. Among the 69 students who enrolled for the course, 51 passed and the remaining 18 failed.

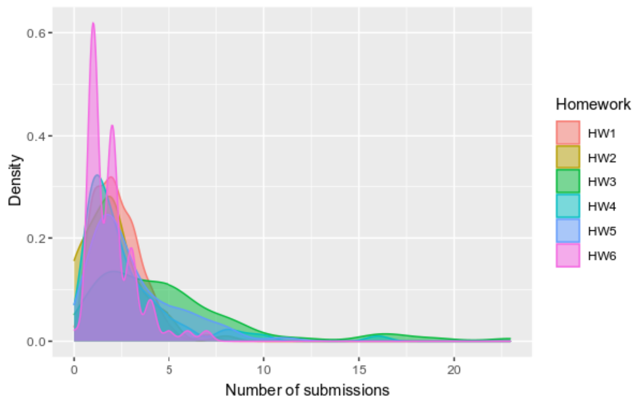

FIGURE 2. Density plot of the number of submissions within the homework.

2) FEATURE SELECTION

We began our analysis of homework performance and academic achievement by addressing one particular question: what features best describe student behavior in terms of homework performance? Trautwein *et al.* [32] reported that the homework behavior of students can be modeled in three dimensions: effort, time, and learning strategies. In the current study, we tailored the model to focus on the first two dimensions. Statistical features extracted from the log dataset were used to adapt the conceptual features proposed in the model presented in [32] to the context of our APAS, resulting in the features listed in Table 1.

B. METHODOLOGY

Multiple data preprocessing techniques were used in the extraction of meaningful features from the log data.

TABLE 1. Homework behavior model of Trautwein *et al.* and adapted features for the proposed model.

Trautwein *et al.* model features	Our adapted features
Homework effort	
Compliance	Number of specific failures
Investment	Number of submissions in total
Concentration	Number of on-time submissions
Number of tasks complete/ Percentage attempted	Not used
Homework time	
Time on homework	Time to first submission
	Time to last submission
	Durations in a specific state of failures
Additional learning	Not used
Learning strategies	
Cognitive	Not used
Metacognitive	Not used

Note that the log data is an unevenly spaced time series, in which observations are captured after student submissions without fixed intervals. Pivoting, aggregation, and other data manipulation methods were used to reshape the data into a tabular format. The final data was organized in a table, where each row denotes all lists of the features on one specific student.

Correlation analysis between the statistical features in Table 1 and the final grade was used to characterize the relationship between learning behaviors and achievements. We then employed K-means clustering to identify groups of students with similarities in learning patterns and achievements. Cluster profile analysis was used to highlight significant characteristics in student behavior/effort in each cluster. These patterns related to academic achievement can be used by instructors to formulate strategies to support students or adjust class policies.

Finally, the features identified in previous steps were then applied to the log data before midterms to train a model to make predictions of final course outcomes. The results of the prediction model are based on selected features (via information gain) in terms of importance in identifying at-risk student profiles. We assessed various machine learning algorithms for our prediction task: K-nearest neighbors (KNN) [50], naive Bayes (NB) [51], support vector machine (SVM) [52], decision tree (DT) [53], random forest (RF) [54] and AdaBoost (AB) [55]. 10-fold cross-validation was used to assess the models. The workflow used in this research is illustrated in Figure 3, and the results are detailed in the following section.

IV. RESULTS

A. DATA EXTRACTION AND EXPLORATORY DATA ANALYSIS

The features are categorized according to learning effort and homework time at two statistical levels: an individual level for each homework assignment (i.e., assignments 1 to 6), and an overall level for the entire set of assignments (Table 2).

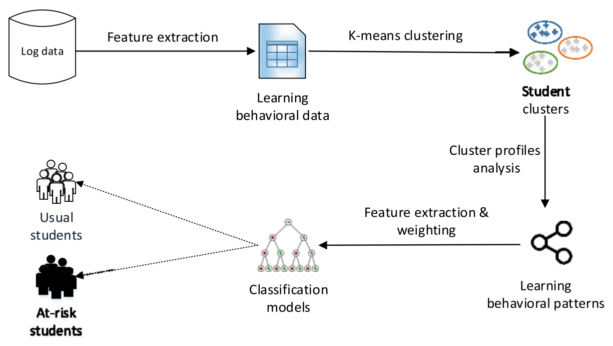

FIGURE 3. Workflow adopted by the current study.

TABLE 2. Behavioral features of students in homework submission.

ID	Individual-level feature name	Overall-level feature name	Description
Homework effort (count)			
F1	CPF.Count$_i$	Total.CPF.Count	Number of submissions given CPF
F2	CSF.Count$_i$	Total.CSF.Count	Number of submissions given CSF
F3	UTF.Count$_i$	Total.UTF.Count	Number of submissions given UTF
F4	Sub.Count$_i$	Total.Sub.Count	Number of submissions
F5	Ontime.Sub. Count$_i$	Total.Ontime.-Sub.Count	Number of on-time submissions
Homework time (minutes)			
F6	Time.To.Start$_i$	Avg.Time.To.Start	Time to first submission
F7	Time.To.End$_i$	Avg.Time.To.End	Time to last submission
F8	CPF.Dur$_i$	Avg.CPF.Dur	Duration of CPF
F9	CSF.Dur$_i$	Avg.CSF.Dur	Duration of CSF
F10	UTF.Dur$_i$	Avg.UTF.Dur	Duration of UTF

1) INDIVIDUAL LEVEL

Features related to homework effort indicated by the number of times the student submitted a particular homework assignment (submission count). Features related to the amount of time the student spent doing homework are measured in minutes, based on the difference between the timestamps of corresponding events (e.g., receiving assignments, getting failures, passing tests and so forth). *Duration in failure* refers to the amount of time the student struggled to surmount or move past a specific type of failure, based on the difference between the timestamps of adjacent submissions.

2) OVERALL LEVEL

Features related to homework effort are computed by summing up the total number of submissions made by a given student completing all the assignments. Features related to homework time are computed by averaging the total amount of time spent in completing each of the assignments. Given n homework assignments, the total values of the features (f_i) in the homework effort category are computed as follows:

$$Total.f = \sum_{i=1}^{n} f_i \tag{1}$$

Average value of the features (f_i) in the homework time category is computed as follows:.

$$Avg.f = \frac{1}{n}\sum_{i=1}^{n} f_i. \tag{2}$$

Exploration data analysis (EDA) is often performed prior to mining operations to gain insight into the nature of the data, including distributions, statistical descriptions, and relationships among properties. In this study, we employed two techniques based on graphical EDA (correlation matrix and correlation network), to elucidate the behavioral patterns exhibited by students in the submission of homework and how the patterns affect their final grades. Figure 4 presents a correlation matrix constructed using Pearson coefficient and Figure 5 is the correlation network containing strong correlated features ($|\rho| \geq 0.50,\ p \leq 0.05$) extracted from Figure 4.

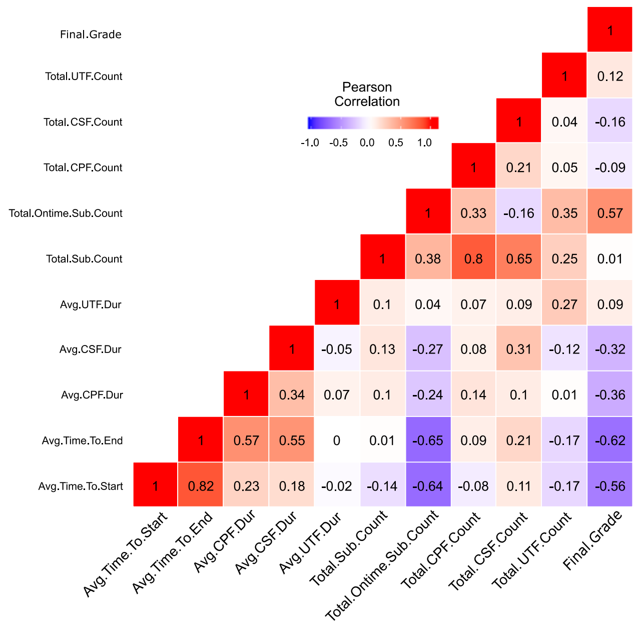

FIGURE 4. Correlation matrix extracted between features.

The strong correlation values in Figure 5 reveal interesting patterns among student effort, student concentration, code quality ensuring and learning results. Three features are strongly correlated with *Final.Grade*: *Total.Ontime.Sub.Count* ($\rho = 0.57,\ p = 0.00$), *Avg.Time-To.Start* ($\rho = -0.56,\ p = 0.00$), and *Avg.Time.To.End* ($\rho = -0.62, p = 0.00$). It is conspicuous that *Avg.Time.To.End* correlates to *Avg.CPF.Dur* ($\rho = 0.57,\ p = 0.00$) and *Avg.CSF.Dur* ($\rho = 0.55,\ p = 0.00$). The correlations can be interpreted as follows:

- The earlier students begin working on their homework, the earlier they finish.
- Students who begin and finish their homework earlier achieve higher grades. Students who make more effort to finish their homework before the deadline will earlier finish their homework and achieve higher grades.

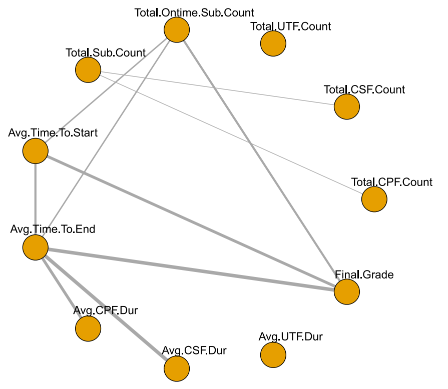

FIGURE 5. The network of strong correlation features.

- Code quality improvement ability of students relatively affects their learning performance. Since students who have higher ability of code quality improvement spend less time to solve CPF and CSF failures and finish their work earlier (smaller value of *Avg.Time.To.End*), they can reach higher final grade. The negative correlation between *Avg.Time.To.End* and *Final.Grade* ($\rho = 0. -62, p = 0.00$) well support this implication.

These findings are related to self-regulated behaviors of students in time management and code quality improvement which help to resolve RQ1: *How is the ability regarding code quality improvement of students associated with their learning results?* The answer is as follows: Those students whose better ability of code quality improvement and tried to make more prompt efforts to overcome coding failures in their homework have better learning results.

B. LEARNING PATTERN ANALYSIS

Student clustering is a usual task done based on the assumption that those students sharing similar learning behaviors are similar in learning performance [18]. Clustering is an effective approach to data segmentation in cases where the analyst has no prior knowledge pertaining to the nature of the data. Clustering is used to categorize data without predefined classes (clusters); however, data instances within a cluster still tend to present similarities. In this study, clustering analysis was used to answer RQ2: *What student groups can be formed based on learning behavior of students in code quality improvement?* We adopted the protocol established in [27], where k = 5 was deemed the optimal number of clusters, to differentiate among various groups of students.

We used overall level features listed in Table 2 as the input of k-means clustering. Since *Total.Sub.Count* partly depends on counts of specific failures, we eliminated this feature in the input set. After considering data distribution, features related to UTF were also excluded because there were too few UTF failures resulted from student submissions. Consequently, selected features for clustering include *{F1, F2, F5, F6, F7, F8, F9}*. Due to the heterogeneity of selected features, we applied Principal Component Analysis (PCA) dimensionality reduction to obtain a better projection of data in a new space with lower dimensionality which maximizes the variance of data. After transformed original data into reduced 3D space, it is conspicuous that homework time features highly correlate with the first principal component *PC1*, homework effort features highly correlate with the second one *PC2*, and *PC3* correlates to *Avg.CPF.Dur* and *Total.CSF.Count*, as depicted in Figure 6.

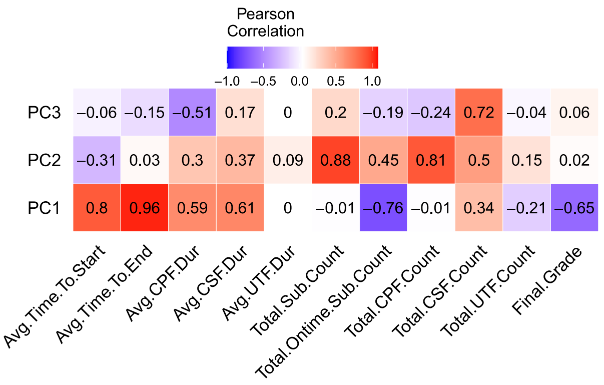

FIGURE 6. Correlation matrix between original features and PCs.

K-means clustering was performed using Euclidean distance in the 3D space from PCA. We performed a detailed analysis of the quality of the code (in terms of adherence to code conventions) as well as specific stages of failure in the students' code. Figures 7a and 7b present the results of K-means clustering in 3D PCA space (a) and in the projected space of three selected attributes which represent the relationship between learning type and learning outcome: *Final.Grade*, *Total.Sub.Count*, and *Avg.Time.To.Start* (b). Note that individual clusters were assigned names according to the relationship between learning patterns and the corresponding outcomes. The naming of clusters is detailed below:

C1. Effective learners appear in the top-left corner of the figures. These individuals achieved high scores within only a few iterations. They appear to be good at resolving programming problems. They begin working on their homework early and finish it quickly.

C2. High-effort learners appear in the top-right of the figures. They achieved final grade scores that are comparable with those of the effective learners. They made a large number of submissions and did not succeed as rapidly as their effective counterparts. They iteratively resolved problems without giving up. They eventually succeeded but did not take the most direct path.

C3. Average learners appear in the middle-left of the figures. The number of submissions in this group was similar to that of effective learners and less than that of high-effort learners. They appear to have made a moderate effort and their final results were acceptable, albeit lower than those of the two groups mentioned above.

C4. Blind-trial learners appear in the bottom-right of the figures. This group used the submission system much more than did the other groups; however, they did not always resolve the problems adequately, even after many

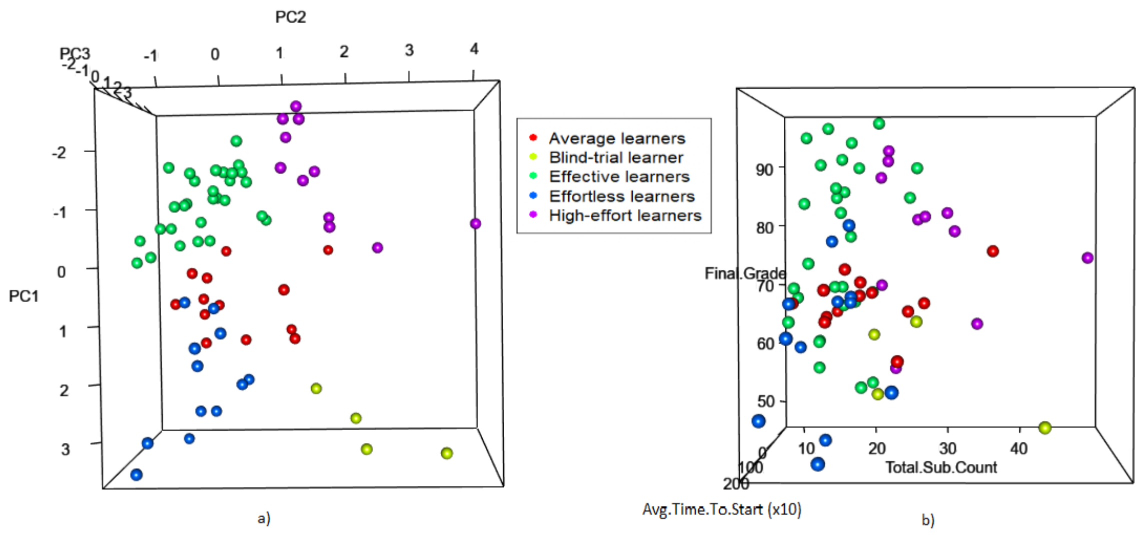

FIGURE 7. Five clusters of students a) in PCA space, and b) in learning behavior and achievement space.

submissions. They failed to achieve results on par with the effective and high-effort learners, and many were unable to pass the course.

C5. Low-effort learners appear in the bottom-left corner of the figures. They received the lowest score after making only a small number of submissions. Many of them ceased submitting code after meeting with initial failures. Many of these individuals failed the course

Further analysis and discussion about differences among the five groups are mentioned in the following sections. To compare learning patterns of students in specific groups, we apply Wilcoxon tests on each pair of clusters and use p-value to conclude about the significance of differences. Wilcoxon test was used in our analysis because the distributions of behavioral features of students do not conform to the normal distribution. Cluster profiles described by normalized mean value (z-score) are shown in Table 3.

TABLE 3. Five cluster profiles described in mean value of features (z-standardized).

Cluster / Feature	Effective learners N= 29 (42%)	High-effort learners N=11 (16%)	Average learners N=13 (19%)	Blind-trial learners N= 4 (6%)	Effortless learners N= 12 (17%)
Avg.Time.To.Start	-0.46	-0.81	0.31	0.16	1.47
Avg.Time.To.End	-0.68	-0.72	0.27	1.59	1.48
Avg.CPF.Dur	-0.48	-0.18	-0.37	2.42	0.92
Avg.CSF.Dur	-0.48	-0.27	0.34	2.50	0.21
Avg.UTF.Dur	-0.17	0.38	-0.18	-0.18	0.32
Total.Sub.Count	-0.44	1.18	0.09	1.11	-0.50
Total.Ontime.Sub.Count	0.18	1.53	-0.76	-0.49	-0.86
Total.CPF.Count	-0.40	1.31	-0.34	0.92	-0.17
Total.CSF.Count	-0.53	0.13	1.01	1.11	-0.30
Total.UTF.Count	0.08	0.46	-0.29	-0.29	-0.19
Final.Grade	0.40	0.49	-0.31	-1.18	-0.68

1) HOMEWORK EFFORT

Figure 8 illustrates the differences among the five groups in terms of homework effort. The high-effort learners made

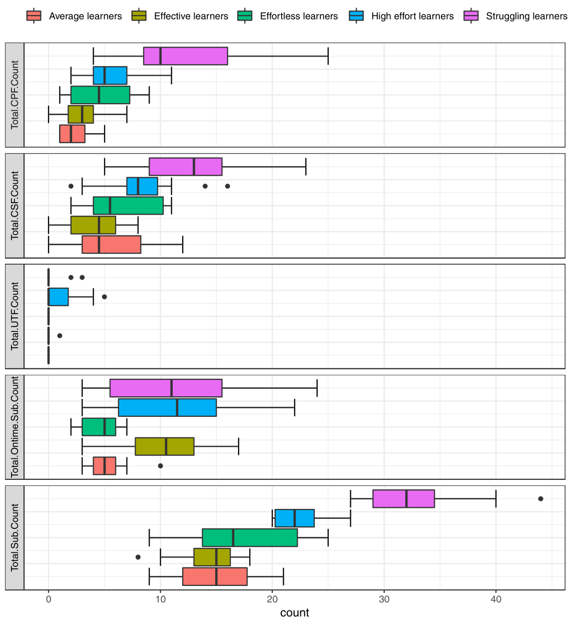

FIGURE 8. Boxplots indicating homework effort of five groups.

the largest number of submissions, followed by the blind-trial learners. The other three groups made roughly the same number of submissions. The most important attribute for this analysis is the number of on-time submissions, which Poženel *et al.* [26] noted was an important indicator of compliance, investment, and concentration. We found it advantageous to combine the effective learners and high-effort learners into a supergroup, called *high achievers*, who made significantly more on-time submissions than did the individuals in the other three groups (*Wilcoxon test, p <0.05*). This is a clear indication that high-performance students make more of an effort to meet submission deadlines, whereas low achievers postpone their efforts until it is too late. This pattern is consistent with the correlation between the number of on-time submissions and final grade ($\rho = 0.57$, $p = 0.00$).

2) HOMEWORK TIME

We observed a strong correlation between the time of the first submission and the time of the last submission as shown in Figure 9. Essentially, the effective and high-effort learners began submitting code earlier and finished the assignments earlier.

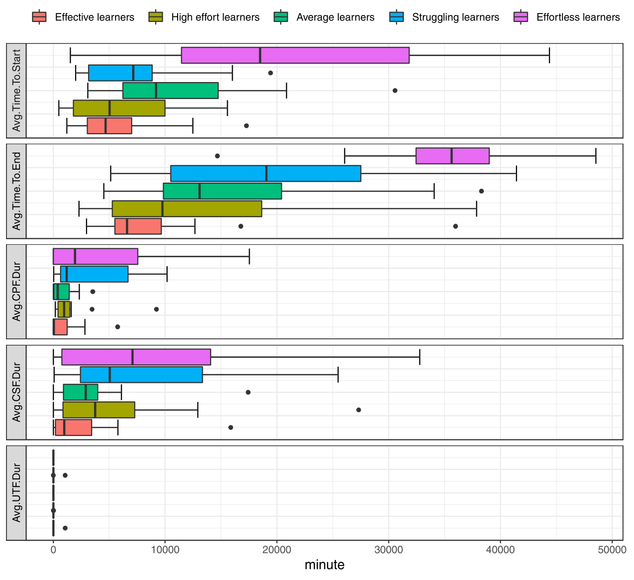

FIGURE 9. **Boxplots indicating homework time of five groups.**

To assess the ability to overcome difficulties, we applied the Wilcoxon test to assess the length of time students remained in CPF or CSF. Our results did not reveal significant differences among the five groups in terms of CPF; however, we observed significant differences in terms of CSF among four of the groups, except the group of Effortless learners (*Wilcoxon test,* $p< 0.05$). Note that many of the students in the low-effort group gave up entirely, sometimes before receiving any CSF feedback. Overall, our results indicate that students who try to maintain code quality generally achieve higher final results. This suggests that code quality should be emphasized in novice programming courses.

3) MAINTAINING CODE QUALITY

The compilation test, unit test, and code quality test revealed considerable variations among the five groups in terms of coding behavior. We recorded only a small number of UTF failures, perhaps due to the fact that most of the students examined the satisfaction of assignment functional requirements locally before submission. Note also that many syntax issues can be resolved quickly and easily using Java integrated development environments (IDEs), such as Eclipse and IntelliJ. Thus, it is not surprising that we did not observe significant differences in CPF duration between the effective, high-effort, and average learners. Overall, we found that a student's ability to maintain code quality can be assessed in terms of code style failures (CSF).

Figure 9 indicates that students who rapidly resolved CSF failures achieved higher final grades. The effective learners received little in the way of CSF feedback and acted quickly to resolve the minor issues they did encounter. They were also attentive to the feedback from ProgEdu and coding convention documents, which no doubt contributed to their high final grades.

The high-effort learners tended to follow a different strategy, in which they focused on a few errors at a time and resubmitted their code iteratively. This resulted in a large amount of CSF feedback; however, they did not have to spend significantly more time on this issue than did the effective learners.

It appears that the average learners were unable to concentrate on completing their assignments. After receiving CSF feedback, they tended to postpone their revisions/resubmissions for a long time, with the result that they were unable to move past the CSF stage without difficulty. Compared to the high-effort and blind-trial learners, they did not resubmit their code many times; however, most of them were able to resolve the issues when provided sufficient time.

Like the high-effort learners, those in the blind-trial group resubmitted code numerous times; however, they displayed an inability to resolve difficulties and conform to syntax rules and coding conventions. They received a large amount of error feedback with the result that they remained in the CPF and CPF states for an extended period.

The low-effort learners tended to give up early, i.e., immediately after receiving a few CPF feedbacks. Most of them were unable to pass syntax checking and therefore remained in the CPF state for a long time. It is possible that this behavior can be attributed to cognitive difficulties, requiring early detection and intervention.

The aforementioned analysis can clearly address the research question RQ3 about the relationship between code quality maintaining effort and learning outcomes. Those students who pay earlier or more effort in solving the CSF failures can achieve higher final results.

Our analysis of student progress in learning to ensure code quality involved applying paired Wilcoxon tests to all pairs of homework assignments to assess differences in the mean number of CSFs (Table 4). Note that CSF was not available for assignment 1; therefore, only the last five assignments were available for comparison. Our results revealed a decrease in the mean number of CSFs after assignment 3,

TABLE 4. **Statistics of CSF failures in specific assignments.**

	CSF. Count_2	CSF. Count_3	CSF. Count_4	CSF. Count_5	CSF. Count_6
Mean	1.35	1.55	1.07	0.88	0.38
SD	1.95	2.26	1.86	1.37	0.95
Max	8	14	9	8	6
Min	0	0	0	0	0

particularly in the final homework assignment *(Wilcoxon tests, $p < 0.05$)*. This demonstrates the advantages of the iterative learning approach employed in our APAS *(ProgEdu)* in promoting adherence to code quality.

C. LEARNING ACHIEVEMENT PREDICTION

There is a crucial relationship between learning patterns and learning outcomes. In this study, we built a prediction model by which to characterize students according to learning behavior in the early stages of the course. The target of the prediction model was the final result (pass or fail). Students who eventually failed should be identified as at-risk as early as possible (preferably before mid-terms) to facilitate behavioral correction.

1) DATA AND FEATURE SELECTION

To achieve the goal of identifying at-risk students before mid-terms, we obtained data for predictions from only the first three assignments. We focused on 10 individual-level features from each assignment, resulting in a total of 30 features. We implemented feature selection using information gain to avoid overfitting due to a large number of features. Information gain is a measurement used to select the best split attribute based on a reduction in entropy [42]. We first ranked all the computed information-gain values. The training and validation steps were repeated using top-weighted features, and feature selection was performed using the hill-climbing method. Table 5 lists the eight weighted features that generated the best validation results, and their corresponding information-gain values.

TABLE 5. **Top-8 features ranked by information gain.**

Feature	Information gain
Time.To.End_3	0.1363
Time.To.Start_3	0.1239
CPF. Dur_2	0.1204
CSF. Dur_3	0.1005
CSF.Count_3	0.0977
$\text{Ontime.Sub.Count}_3$	0.0951
$\text{Ontime.Sub.Count}_2$	0.0858
Time.To.End_1	0.0697

2) CLASSIFICATION EVALUATION

In assessing the performance of the prediction models, it was particularly important to avoid overfitting, due to the fact that our ultimate objective was to build generalizable models capable of generating reliable predictions using unseen data. This was achieved using 10-fold cross validation. The original data was first divided into 10 blocks. In each validation round, 9 blocks were used as training data and 1 block was used for validation. The process was repeated 10 times using different testing blocks in each round. The overall performance of the models was estimated by averaging the measurements derived in all 10 cases of cross validation. Our use of all entries in the original data for training as well as validation helped us to estimate how the machine learning models generalize independent datasets.

In our evaluation of effectiveness, we adopted confusion matrix-based measurements, including Accuracy *(Acc)*, Area Under the Curve *(AUC)*, Precision *(Pr)*, Recall *(Re)*, and *F1-measure*, which are widely used in classification-related problems. A confusion matrix describes summarized results of a binary classifier with four possible prediction cases: True Positive *(TP)*, True Negative *(TN)*, False Positive *(FP)*, and False Negative *(FN)*. Accuracy is the overall correct rate achieved by the classifier. Precision indicates the exactness of the classifier in positive cases. Recall indicates completeness in positive detections. Due to the inverse relationship between precision and recall, the F-measure is used as a harmonic mean between the two. The calculations used in the evaluation metrics were as follows:

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

$$F_1 - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5}$$

A relatively small proportion of the students were unable to pass the course; therefore, the task of predicting at-risk students became an imbalanced classification problem *(pass rate = 73.9% (51/69) and fail rate = 26.1% (18/69))*. To ensure a reliable assessment of prediction performance, we performed evaluations on both classes of the target feature (i.e., at-risk and non-risk students). Table 6 presents the results of 10-fold cross validation.

TABLE 6. **Cross-validation results for identifying at-risk students using the early detection model.**

Method	AUC	Acc	Non-risk			At-risk		
			F1	*Pr*	*Re*	*F1*	*Pr*	*Re*
SVM	0.68	0.71	0.82	0.77	0.86	0.33	0.41	0.28
AB	0.73	0.80	0.86	0.86	0.86	0.61	0.61	0.61
DT	0.80	0.83	0.88	0.88	0.88	0.67	0.67	0.67
NB	0.81	0.75	0.81	0.93	0.73	0.64	0.52	**0.83**
RF	0.85	0.78	0.85	0.86	0.84	0.60	0.58	0.61
KNN	**0.87**	**0.87**	**0.91**	**0.89**	**0.94**	**0.73**	**0.80**	0.67

The evaluation results reveal that the proposed scheme performed reasonably well in the early detection of at-risk students. KNN provided the best performance with an accuracy of 0.87, and F1-measures of 0.914 (non-risk students) and 0.727 (at-risk students). NB was the most sensitive to at-risk students, with a recall of 0.833; however, its precision in this area was only 0.517. The performance of DT was slightly lower than that of KNN, with an accuracy of 0.826 and F1-measures of 0.882 (non-risk students) and 0.667 (at-risk students). The performance of the remaining algorithms was roughly the same, with accuracy varying from 0.710 to 0.797.

When using KNN, we were able to detect 70% of at-risk students based solely on data obtained prior to mid-terms. This finding also answers the *RQ4: Is it possible to build a prediction model capable of early identifying at-risk students based solely on their behaviors in code quality improvement?* A list of important predictors in Table 5 reveals that ability in code quality improvement has strong effects on the success of students in the course.

Our predicting results are comparable to those of the papers [39] and [56]. In [39], the early prediction of at-risk students was produced after the first five weeks of the first-year-engineering course at the Midwestern US university. The robustness of algorithms used in our work and that paper also consistent: KNN outperformed the others in overall prediction accuracy (87% vs 94.9% in accuracy) and NB was the most sensitive algorithm with at-risk students (83% vs 86.9% in recall). Note that in [39], by the end of the fifth week, students already took part in 10 quizzes, five homework with 33 learning objectives, and one written exam which referred to as the midterm. In [56], the authors proposed an APAS for Java course at National Taichung University of Education and use the log data to predict the final grades of students. The best accuracy of forecasting was 77% when using data extracted from pre-midterm exercises and the midterm exercise. We claim that our prediction can be made with fewer data and does not require any additional grade given by manual assessments of the educators such as writing exams or midterm. Although our prediction just can be done until before the midterm using current data, we believe that earlier predictions can be made if the instructors having more frequent assignment schedule, e.g. weekly homework, in future courses.

Our prediction results are also consistent with the investigation of the effectiveness of EDM techniques in early prediction failures of students in introductory programming courses [57]. Both distance learning and on-campus course datasets are used in that study. With an in-campus course, the data of each student including demographics information, amount of exercises done, number of correct exercises, and weekly performance in activities and exams; the first exam was given in the fourth week. With distance learning course, the data was enriched by online activities in quizzes, interactions with the LMS and forum, etc.; the first exam was given in the fifth week. The highest performance of academic failures prediction was produced by SVM when using data from the beginning of the courses to the first exam (F1-measure was 92% in distance learning course and 83% in on-campus course). In the case of the first exam results were excluded, these measures dropped down to 79% and 78%, respectively. Our results again, with fewer input data, are comparable when we only use the log data from three homework. To obtain earlier and more accurate predictions, it is suggested to the instructors of the future courses that homework should be given more frequently to provide enough data for the prediction model.

D. THREATS OF VALIDITY

Although we discovered interesting patterns in homework behaviors which affects students' final achievement in a Java programming course, it is important to note some threats:

First, the data source used for analysis represents information about students of the course Java programming in a university in Taiwan. Consequently, the experimental results are not general.

Second, the performance of early detection of at-risk students may depend on the difficulty levels of individual homework. For example, easy assignments at the beginning of the course do not contribute significantly to the prediction models. In this study, homework 3 required most efforts to be solved and had the most impact on the detection model, consequently. The same prediction results might not be obtained in other courses because of differences in assigning time and difficulty level of given homework assignments.

Third, besides homework performance, the results of the midterm exam are also a strong factor affecting the possibility to be the failure of students. There were 10 students who dropped out after the midterm exam. We leave the dropout prediction problem of students for another study. Therefore, discovered learning patterns and the prediction model of at-risk students in this study do not include dropped out students.

Finally, the volume of data obtained in this study might be governed by the instructors of the course, who are also the authors of this paper. In this first study about the usage of the system, we aimed to understand students' learning behaviors in a freedom mode, so that the number of submissions was not limited. In other scenarios, the behaviors of students may significantly be shifted if any restriction is in place. For example, the groups of students with a high number of submissions will be no longer detected.

V. DISCUSSION

The purpose of this paper is to find out the relationship between student behaviors in code quality improvement and learning achievement. First, we suggest that the homework behavioral model of Flunger *et al.* [32] can be adapted to the context of programming courses to analyze the learning performance of students. Second, we segmented students based on their behavioral characteristics in code quality improvement and describe in detail the differences between groups of students. There are significant differences in homework behavior between high-performance groups and low-performance ones. The homework time attributes have more impact on student segmentation than homework effort attributes. Third, employing attributes related to coding quality improvement provides good signals to understand students' situations. Long durations to overcome CPF and CSF failures are denotations that a student has trouble with their learning. Late starting, early giving up after some CPF failures, and rarely receiving CSF failures are characteristic of effortless learners. Finally, we believe that we can gain more with unlimited resubmission since a high number of submissions is also a good signal of potential struggling learners. In overall, a dashboard containing visualizations of proposed attributes is useful for instructors to give appropriate instructions that improve students' learning performance.

The prediction model for early warning of at-risk students was built only in-course data, especially high level of homework submission logs. Although deeper analyzing system feedbacks and students' code may bring potential fruitful knowledge, we leave those tasks for future research since we aim to maintain the replicability of the current mining flow. The warning made before the midterm does not seem very early. Yet, three assignments are few enough to be arranged in the first quarter of any course. Hence, this is also a notable remark for those educators who want to make earlier predictions.

Acceptance of students on code quality improvement functionalities of ProgEdu was surveyed and presented in the article [11]. More than 70% of asked students agreed with the assessment method given by the system, especially with code quality assessment. The most frequent errors were missing comments or documentation. The students commented that there are some given functions are too simple to be commented on. Hence, they always got CSF when ignoring such comments. Overall, our iterative approach to the improvement of code quality of students is widely acceptable by students. With experiences conducted, we will apply the research flow of this study in future courses to validate it and increase its impact on programming teaching.

VI. CONCLUSION

In this study, the EDM approach was used to characterize the learning behavior of students engaged in a computer programming course using data extracted from the APAS ProgEdu. A set of data mining techniques including EDA, clustering, classification, and data visualization was leveraged. Behavioral features related to the time and effort that went into homework assignments were extracted from system log data. Graphical EDA revealed that students who made a pronounced effort to complete their homework early tended to finish earlier and earn higher final grades.

K-means clustering analysis was used to differentiate students into five groups based on their learning behavior in the coding course. Cluster profile analysis confirmed that the effort students put into their homework (denoted by the number of submissions) was not linearly associated with final grades. In contrast, learning motivation (denoted by early and on-time submissions) was strongly associated with final grades.

From the perspective of code quality, we found that high-performance students had fewer CSF failures and were able to resolve errors more quickly. Our analysis of adherence to coding conventions revealed that throughout the course, most of the students improved in their ability to keep their code clean, such that in the final assignment, the probability of receiving a CSF in a certain submission was less than 0.4.

Concerning the prediction of learning performance, KNN achieved the highest overall accuracy in predicting learning results, whereas the Naive Bayes approach produced the best results in predicting at-risk students. We believe that integrating this prediction model within the APAS system would help to identify at-risk students and provide insight that could prove beneficial to reforming their learning behaviors.

In conclusion, our results demonstrate the efficacy of EDM, even with a relatively small dataset. We believe that the simple workflow proposed in this paper could easily be replicated by educators via referencing the information and tools provided by the repository of ProgEdu.[1] We claim that the information collected from assignment submissions is valuable for multiple educational purposes, for instance, to improve the assignment design for the next versions of the course, and to be featured for learning analytics related to students' behavior. In the future, we will analyze system feedback in conjunction with students' behavior to clarify how students could improve their code based on the feedback which they receive after each submission.

REFERENCES

[1] R. Romli, S. Sulaiman, and K. Z. Zamli, "Improving automated programming assessments: User experience evaluation using FaSt-generator," *Procedia Comput. Sci.*, vol. 72, pp. 186–193, Jan. 2015.
[2] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in *Proc. 10th Koli Calling Int. Conf. Comput. Edu. Res. (Koli Calling)*, 2010, pp. 86–93.
[3] Pieterse and Vreda, "Automated Assessment of Programming Assignments," in *Proc. 3rd Comput. Sci. Educ. Res. Conf. Comput. Sci. Educ. Res.* Heerlen, The Netherlands: Open Universiteit, 2013, pp. 45–56.
[4] H.-M. Chen, W.-H. Chen, and C.-C. Lee, "An automated assessment system for analysis of coding convention violations in java programming assignments," *J. Inf. Sci. Eng.*, vol. 34, no. 5, pp. 1203–1221, 2018.
[5] R. Cardell-Oliver, "How can software metrics help novice programmers?" in *Proc. 13th Australas. Comput. Edu. Conf.*, vol. 114, 2011, pp. 55–62.
[6] A. L. Patton and M. McGill, "Student portfolios and software quality metrics in computer science education," *J. Comput. Sci. Coll.*, vol. 21, no. 4, pp. 42–48, 2006.
[7] M. Smit, B. Gergel, H. J. Hoover, and E. Stroulia, "Code convention adherence in evolving software," in *Proc. 27th IEEE Int. Conf. Softw. Maintenance (ICSM)*, Sep. 2011, pp. 504–507.
[8] S. Chacon and B. Straub, *Pro Git*. New York, NY, USA: Apress, 2014.
[9] GitLab Inc. (2017). *GitLab*. Accessed: Jun. 20, 2020. [Online]. Available: https://about.gitlab.com/
[10] A. Korhonen, L. Malmi, J. Nikander, and P. Tenhunen, "Interaction and feedback in automatically assessed algorithm simulation exercises," *J. Inf. Technol. Educ., Res.*, vol. 2, pp. 241–255, Jan. 2003.
[11] Y.-X. Yan, J.-P. Wu, B.-A. Nguyen, and H.-M. Chen, "The impact of iterative assessment system on programming learning behavior," in *Proc. 9th Int. Conf. Educ. Inf. Technol.*, Feb. 2020, pp. 89–94.
[12] A. Bogarín, R. Cerezo, and C. Romero, "A survey on educational process mining," *WIREs Data Min. Knowl Discov*, vol. 8, p. 1230, Jan. 2018.
[13] A. Peña-Ayala, "Educational data mining: A survey and a data mining-based analysis of recent works," *Expert Syst. Appl.*, vol. 41, no. 4, pp. 1432–1462, Mar. 2014.
[14] K. Verbert, E. Duval, J. Klerkx, S. Govaerts, and J. L. Santos, "Learning analytics dashboard applications," *Amer. Behav. Scientist*, vol. 57, no. 10, pp. 1500–1509, Oct. 2013.

[1]https://github.com/fcumselab/ProgEdu

[15] M. Ahmadzadeh, D. Elliman, and C. Higgins, "An analysis of patterns of debugging among novice computer science students," in *Proc. 10th Annu. SIGCSE Conf. Innov. Technol. Comput. Sci. Edu. (ITiCSE)*, 2005, vol. 37, no. 3, p. 84.

[16] P. Ihantola, A. Vihavainen, A. Ahadi, M. Butler, J. Börstler, S. H. Edwards, E. Isohanni, A. Korhonen, A. Petersen, K. Rivers, and M. Á. Rubio, "Educational data mining and learning analytics in programming: Literature review and case studies," in *Proc. Work. Group Rep. (ITICSE-WGR)*, 2015, pp. 41–63.

[17] M. Potgieter, M. Ackermann, and L. Fletcher, "Inaccuracy of self-evaluation as additional variable for prediction of students at risk of failing first-year chemistry," *Chem. Educ. Res. Pract.*, vol. 11, no. 1, pp. 17–24, 2010.

[18] L. de-La-Fuente-Valentín, A. Pardo, F. L. Hernández, and D. Burgos, "A visual analytics method for score estimation in learning courses," *J. Univers. Comput. Sci.*, vol. 21, no. 1, pp. 134–155, 2015.

[19] M. Joy, N. Griffiths, and R. Boyatt, "The boss online submission and assessment system," *J. Educ. Resour. Comput.*, vol. 5, no. 3, p. 2, Sep. 2005.

[20] L. Attwood and J. Carter, "Semi-automating the marking of a java programming portfolio assessment: A case study from a UK undergraduate programme," in *Higher Education Computer Science*. Cham, Switzerland: Springer, 2018, pp. 161–169.

[21] J. C. Rodríguez-del-Pino, E. R. Royo, and Z. H. Figueroa, "A virtual programming Lab for Moodle with automatic assessment and anti-plagiarism features," in *Proc. Int. Conf. e-Learning, e-Business, Enterprise Inf. Syst. e-Government*, 2012, pp. 80–85.

[22] J. L. F. Aleman, "Automated assessment in a programming tools course," *IEEE Trans. Educ.*, vol. 54, no. 4, pp. 576–581, Nov. 2011.

[23] D. Insa and J. Silva, "Automatic assessment of java code," *Comput. Lang., Syst. Struct.*, vol. 53, pp. 59–72, Sep. 2018.

[24] H. Fangohr, N. O'Brien, A. Prabhakar, and A. Kashyap, "Teaching Python programming with automatic assessment and feedback provision," 2015, *arXiv:1509.03556*. [Online]. Available: http://arxiv.org/abs/1509.03556

[25] J. Cole, H. Foster *Using Moodle: Teaching With the Popular Open Source Course Management System*. Newton, MA, USA: O'Reilly Media, Inc., 2007.

[26] M. Pozenel, L. Furst, and V. Mahnicc, "Introduction of the automated assessment of homework assignments in a university-level programming course," in *Proc. 38th Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, May 2015, pp. 761–766.

[27] J. Loeliger, M. McCullough, *Version Control With Git: Powerful Tools and Techniques for Collaborative Software Development*. Newton, MA, USA: O'Reilly Media, 2012.

[28] J. Lawrance, S. Jung, and C. Wiseman, "Git on the cloud in the classroom," in *Proc. 44th ACM Tech. Symp. Comput. Sci. Edu. (SIGCSE)*, 2013, p. 639.

[29] J. Kelleher, "Employing git in the classroom," in *Proc. World Congr. Comput. Appl. Inf. Syst. (WCCAIS)*, Jan. 2014, pp. 1–4.

[30] S. H. Edwards, N. Kandru, and M. B. M. Rajagopal, "Investigating static analysis errors in student java programs," in *Proc. ACM Conf. Int. Comput. Edu. Res.*, Aug. 2017, pp. 65–73.

[31] D. M. Breuker, J. Derriks, and J. Brunekreef, "Measuring static quality of student code," in *Proc. 16th Annu. joint Conf. Innov. Technol. Comput. Sci. Edu. (ITiCSE)*, 2011, pp. 13–17.

[32] B. Flunger, U. Trautwein, B. Nagengast, O. Lüdtke, A. Niggli, and I. Schnyder, "A person-centered approach to homework behavior: Students' characteristics predict their homework learning type," *Contemp. Educ. Psychol.*, vol. 48, pp. 1–15, Jan. 2017.

[33] B. Flunger, U. Trautwein, B. Nagengast, O. Lüdtke, A. Niggli, and I. Schnyder, "The janus-faced nature of time spent on homework: Using latent profile analyses to predict academic achievement over a school year," *Learn. Instruct.*, vol. 39, pp. 97–106, Oct. 2015.

[34] W. A. Gibson, "Three multivariate models: Factor analysis, latent structure analysis, and latent profile analysis," *Psychometrika*, vol. 24, no. 3, pp. 229–252, Sep. 1959.

[35] V. Karavirta, A. Korhonen, and L. Malmi, "On the use of resubmissions in automatic assessment systems," *Comput. Sci. Edu.*, vol. 16, no. 3, pp. 229–240, Sep. 2006.

[36] L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, and P. Silvasti, "Visual algorithm simulation exercise system with automatic assessment: TRAKLA2," *Informat. Educ.*, vol. 3, no. 2, p. 267, 2004.

[37] T. Kato, Y. Kambayashi, and Y. Kodama, *Data Mining of Students' Behaviors in Programming Exercises*. Cham, Switzerland: Springer, 2016, pp. 121–133.

[38] V. K. Anand, S. K. A. Rahiman, E. Ben George, and A. S. Huda, "Recursive clustering technique for students' performance evaluation in programming courses," in *Proc. Majan Int. Conf. (MIC)*, Mar. 2018, pp. 1–5.

[39] F. Marbouti, H. A. Diefes-Dux, and K. Madhavan, "Models for early prediction of at-risk students in a course using standards-based grading," *Comput. Edu.*, vol. 103, pp. 1–15, Dec. 2016.

[40] R. Al-Shabandar, A. Hussain, A. Laws, R. Keight, J. Lunn, and N. Radi, "Machine learning approaches to predict learning outcomes in massive open online courses," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 713–720.

[41] J. Xu, K. H. Moon, and M. van der Schaar, "A machine learning approach for tracking and predicting student performance in degree programs," *IEEE J. Sel. Topics Signal Process.*, vol. 11, no. 5, pp. 742–753, Aug. 2017.

[42] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[43] Z. Liu, F. Xiong, K. Zou, and H. Wang, "Predicting learning status in MOOCs using LSTM," 2018, *arXiv:1808.01616*. [Online]. Available: https://arxiv.org/abs/1808.01616

[44] S. H. Edwards, J. Snyder, M. A. Pérez-Quiñones, A. Allevato, D. Kim, and B. Tretola, "Comparing effective and ineffective behaviors of student programmers," in *Proc. 5th Int. workshop Comput. Edu. Res. workshop (ICER)*, 2009, pp. 3–14.

[45] A. Vihavainen, M. Luukkainen, and M. Pärtel, "Test my code: An automatic assessment service for the extreme apprenticeship method," in *Proc. 2nd Int. Workshop Evidence Technol. Enhanced Learn.*, 2013, pp. 109–116.

[46] A. Vihavainen, M. Luukkainen, and J. Kurhila, "Using students' programming behavior to predict success in an introductory mathematics course," in *Proc. 6th Int. Conf. Educ. Data Mining*, Memphis, TN, USA, 2013, pp. 300–303.

[47] V. R. C. Martinho, C. Nunes, and C. R. Minussi, "Prediction of school dropout risk group using Neural Network," in *Proc. Federated Conf. Comput. Sci. Inf. Syst.*, 2013, pp. 111–114.

[48] H. Bydžovská, "A comparative analysis of techniques for predicting student performance," in *Proc. 6th Int. Conf. Educ. Data Mining*, 2016, pp. 306–311.

[49] L. M. B. Manhães, S. M. S. Da Cruz, and G. Zimbrão, "WAVE: An architecture for predicting dropout in undergraduate courses using EDM," in *Proc. ACM Symp. Appl. Comput.*, 2014, pp. 243–245.

[50] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 1, pp. 21–27, Jan. 1967.

[51] M. E. Maron, "Automatic indexing: An experimental inquiry," *J. ACM*, vol. 8, no. 3, pp. 404–417, Jul. 1961.

[52] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[53] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, Mar. 1986.

[54] T. Kam Ho, "Random decision forests," in *Proc. 3rd Int. Conf. Document Anal. Recognit.*, vol. 1, 1995, pp. 278–282.

[55] Y. Freund and R. R. E. Schapire, "Experiments with a new boosting algorithm," in *Proc. Int. Conf. Mach. Learn.*, vol. 96, 1996, pp. 148–156.

[56] C.-S. Koong, H.-Y. Tsai, Y.-Y. Hsu, and Y.-C. Chen, "The learning effectiveness analysis of JAVA programming with automatic grading system," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 2, Jul. 2018, pp. 99–104.

[57] E. B. Costa, B. Fonseca, M. A. Santana, F. F. de Araújo, and J. Rego, "Evaluating the effectiveness of educational data mining techniques for early prediction of students' academic failure in introductory programming courses," *Comput. Hum. Behav.*, vol. 73, pp. 247–256, Aug. 2017.

HSI-MIN CHEN received the B.S. and Ph.D. degrees in computer science and information engineering from National Central University, Taiwan, in 2000 and 2010, respectively. He is currently an Associate Professor with the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. His research interests include software engineering, programming education, object-oriented technology, service computing, and distributed computing.

BAO-AN NGUYEN received the B.S. degree from the Hanoi University of Science and Technology, in 2005, and the M.S. degree in information engineering and computer science from Feng Chia University, Taiwan, in 2011, where he is currently pursuing the Ph.D. degree with the Department of Information Engineering and Computer Science. He is also a faculty of the Department of Engineering and Technology, Tra Vinh University, Vietnam. His research interests include data mining, software engineering, and education technology.

YI-XIANG YAN received the B.S. degree from the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan, where he is currently pursuing the master's degree. His research interests include software engineering, mobile application technology, and education technology.

CHYI-REN DOW was born in 1962. He received the B.S. and M.S. degrees in information engineering from National Chiao Tung University, Taiwan, in 1984 and 1988, respectively, and the M.S. and Ph.D. degrees in computer science from the University of Pittsburgh, Pittsburgh, PA, USA, in 1992 and 1994, respectively. He is currently a Professor with the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. His research interests include mobile computing, ad-hoc wireless networks, agent techniques, fault tolerance, and learning technology.