Leinonen, Juho; Castro, Francisco Enrique Vicente; Hellas, Arto

Time-on-Task Metrics for Predicting Performance

Published in:
SIGCSE 2022: Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1

DOI:
10.1145/3478431.3499359

Published: 22/02/2022

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Leinonen, J., Castro, F. E. V., & Hellas, A. (2022). Time-on-Task Metrics for Predicting Performance. In *SIGCSE 2022: Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1* (pp. 871-877). ACM. https://doi.org/10.1145/3478431.3499359

Time-on-Task Metrics for Predicting Performance

Juho Leinonen
juho.2.leinonen@aalto.fi
Aalto University
Espoo, Finland

Francisco Enrique Vicente Castro
fcastro@cs.umass.edu
University of Massachusetts Amherst
Amherst, Massachusetts, USA

Arto Hellas
arto.hellas@aalto.fi
Aalto University
Espoo, Finland

ABSTRACT

Time-on-task is one key contributor to learning. However, how time-on-task is measured often varies, and is limited by the available data. In this work, we study two different time-on-task metrics—derived from programming process data—for predicting performance in an introductory programming course. The first metric, *coarse-grained time-on-task*, is based on students' submissions to programming assignments; the second, *fine-grained time-on-task*, is based on the keystrokes that students take while constructing their programs. Both types of time-on-task metrics have been used in prior work, and are supposedly designed to measure the same underlying feature: time-on-task. However, previous work has found that the correlation between these two metrics is not as high as one might expect. We build on that work by analyzing how well the two metrics work for predicting students' performance in an introductory programming course. Our results suggest that the correlation between the fine-grained time-on-task metric and both weekly exercise points and exam points is higher than the correlation between the coarse-grained time-on-task metric and weekly exercise points and exam points. Furthermore, we show that the fine-grained time-on-task metric is a better predictor of students' future success in the course exam than the coarse-grained time-on-task metric. We thus propose that future work utilizing time-on-task as a predictor of performance should use as fine-grained data as possible to measure time-on-task if such data is available.

CCS CONCEPTS

• **Social and professional topics** → *Computing education.*

KEYWORDS

time-on-task, predicting performance, programming process data, keystroke data, educational data mining, learning analytics

ACM Reference Format:
Juho Leinonen, Francisco Enrique Vicente Castro, and Arto Hellas. 2022. Time-on-Task Metrics for Predicting Performance. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2022), March 3–5, 2022, Providence, RI, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3478431.3499359

SIGCSE 2022, March 3–5, 2022, Providence, RI, USA

ACM ISBN 978-1-4503-9070-5/22/03…$15.00
https://doi.org/10.1145/3478431.3499359

1 INTRODUCTION

Predicting how students will perform in the future is a common topic in learning analytics and educational data mining as well as in the computing education research domain [1, 4, 7, 11, 14, 16, 26, 31, 32]. Predicting students' future success can, for example, allow instructors to proactively deploy interventions to help struggling students. Many contemporary courses collect a variety of data about students' working processes, which can be utilized for predicting performance. For example, collecting log data from learning management systems and integrated development environments has become popular in recent years [16].

One aspect of learning that has been found to correlate with performance is *time-on-task*, defined as the time a student spends actively engaged in a learning task. It has been suggested that increasing students' time-on-task could lead to improved learning results [34]. However, there is no widely agreed-upon solution or practice on how time-on-task should be calculated, which is typically dependent on the data available to researchers conducting time-on-task-related studies. There has been some prior work that has evaluated different ways of measuring time-on-task [19, 23, 24, 27], where the authors point out the issue of different measurement methods possibly leading to dissimilar results. For example, Kovanović et al. [19] found that the approach taken to compute time-on-task affected the fit of models used to predict performance. Similarly, Nguyen [27] found that taking into account individual and task-specific differences as well as the stage of the learning process can increase performance of prediction models that utilize time-on-task.

In this work, we investigate the relationship between time-on-task and students' performance in an introductory programming course. Our work is inspired by earlier work comparing two granularities for estimating time-on-task, *coarse-grained time-on-task* and *fine-grained time-on-task* [23]. The fine-grained one takes breaks students take within their learning into account while the coarse-grained metric does not. Both approaches (taking and not taking breaks into account) have been used in other prior work (e.g. [9, 10, 24, 29]). In this article, we contribute to the earlier research by studying how the two time-on-task metrics correlate with both students' exam points and weekly exercise points, and by evaluating to what extent the two time-on-task metrics work for predicting whether students will get a passing grade in the course exam.

This work is organized as follows. In the next section, we provide background on time-on-task and predicting students' performance. We then outline the context of the study as well as our research questions and approach in Section 3. We present our results in Section 4, which we further discuss in Section 5. Lastly, in Section 6, we finish with a recap of our research questions and their answers.

2 BACKGROUND

In this section, we first present background on time-on-task and how it has been calculated in previous work. Then, we go over prior work that has predicted academic performance with data collected from integrated development environments (IDEs) as the data used in this study also comes from an IDE. Lastly, we go over prior work that has utilized time-on-task for predicting performance which is the main aspect of the present work.

2.1 Time-on-Task

Prior work has used time-on-task to understand student affect, how students engage with learning materials, and its impact on learning outcomes. We describe in this subsection different ways that time-on-task has been studied and/or defined from varying perspectives such as in education research, and more specifically in CS education.

The early studies on time-on-task often relied on external observers who manually kept track of whether students are actively engaged on the task. In Good and Beckerman's study, for example, coders observed and categorized US sixth-grade students' *involvement* in-class activities (e.g. *definitely involved*, *inattention*, *misbehavior*, etc.) [12]. They found that students were more involved (i.e. spent more time-on-task) in activities that are more structured and demanded active responses, as opposed to instances of individual activities or assignments. This partly aligns with Anderson and Scott's findings [3]: they found that teaching methods that delegate students to manage their use of time are associated with high levels of time-on-task, but only for those students with more positive academic self-concepts. In contrast, methods that foster "two-way" communication (teacher-directed but with student participation) are associated with high levels of time-on-task for all types of students.

Kounin and Gump [18] identified the impact of the *continuity* of *signal systems* on time-on-task. They define signal systems as information, materials, or behavior settings that guide students' behaviors and activities in classrooms. They found from observations of classroom activities that high-continuity behavior settings (e.g. engaging in construction activities) have "holding power", enable a variety of behaviors, contain clear indicators of student accomplishment, and are also impacted by pacing. Classroom activities and teaching behaviors that have holding power, are paced by the teacher and promote continuity of signal systems are associated with high levels of time-on-task.

Advances in educational technology have enabled the now-widespread practice of utilizing log data collected in educational software used in classrooms or technology-augmented learning activities (e.g. Intelligent Tutoring Systems (ITS)) to calculate time-on-task. This is perhaps even more common within computer science education research compared to other education research as using the computer—which allows easy logging of learning activities—is at the core of learning to program. In studies related to learning to program, a typical approach is to log user interactions within integrated development environments (IDEs) [16].

Some examples of prior studies that have calculated time-on-task from IDE log data include, for example, Edwards et al.'s work on studying behaviors of high and low-performing students [9]. They found that the difference in the total elapsed time between first and last submissions of high and low-performing students was generally small. Leinonen et al. [24] compared different time metrics—including IDE-based ones—and studied their relationships. They found that data from one source (e.g. the IDE) tends to correlate strongly with other data from the same source, and less strongly with data from other sources. Fagerholm and Hellas [10] studied differences in productivity, as measured by time-on-task, between students in a programming course and found large differences in productivity of students. Part of the difference was explained by previous programming experience, however.

2.2 Predicting Academic Performance with IDE Data

Research on predicting academic success is on the increase [14]. One major goal of such research is to allow instructors to detect students who are at risk of performing poorly (e.g. dropping out of the course) so that the instructor could deploy an intervention [35] to help those students.

As mentioned in the previous subsection, in modern programming courses, it is common that log data about students' learning processes is collected [16, 21]. This can facilitate predicting students' performance. To this end, Hundhausen et al. [15] propose a framework for IDE-based learning analytics, where data about students' programming process is first collected in the IDE and analyzed for patterns, and then used to design and deliver interventions to students directly within the IDE.

Indeed, other previous work has found that data collected from IDEs can be used to predict how students will perform [1, 8, 13, 17, 22, 25]. Ahadi et al. [1] used data collected from an introductory programming course to predict students' success and found that using data from just the first week of the course was enough to accurately predict future success. Another approach employed by previous work has been deriving more complex metrics from IDE data to predict success; for example, Jadud's Error Quotient (EQ) [17] is a measure of how much students struggle on syntax while constructing programs and can be calculated from IDE logs. EQ has been found to correlate with exam scores [17], although its performance for predicting success seems to be context-specific [30]. Similarly, Leinonen et al. [25] constructed typing profiles based on IDE data to predict students' success and found that students' typing patterns in course assignments were somewhat indicative of their future performance in the course exam.

2.3 Using Time-on-Task for Predicting Performance

In addition to the IDE metrics in the previous subsection, researchers have calculated time-on-task from log data for performance prediction. For example, Kovanović et al. [19] examined different approaches for calculating time-on-task based on learning management system logs, and studied how the different approaches compare for predicting student performance. The authors found that the choice of how time-on-task is calculated can have significant effects on how well it explains performance. Similarly, Nguyen [27] utilized time-on-task to predict academic performance and found that taking individual, time, and task differences into account can

increase the performance of prediction models. Alamri et al. [2] found that time-on-task was a good predictor of performance, and emphasize that it is lightweight and thus easy to implement on many different types of courses.

The relationship between time-on-task and performance has been studied also within introductory programming. Leinonen et al. [24] explored the correlations between different time metrics and found that the time that students' were actively engaged on the task correlated with exam scores. Similarly, Pereira et al. [29] found that time spent in the IDE in a CS1 course was one of the most important features for predicting performance. Carter et al. [6] developed a predictive model they coined the "Normalized Programming State Model" (NPSM). The model represents different possible states of students' programming process: for example, whether students are currently debugging or editing the program. They examined how the amount of time spent in the different states of the model relate to students' performance and found that this explained 36% of the variance in students' grades. In a later study they found that students' paths in the model also correlate with performance [5].

3 METHODOLOGY

3.1 Context and Data

The course under study is an introductory programming course (CS1) organized at a research-oriented university in Finland. The course uses Java as the programming language and expects that students have no prior programming experience. The main goal of the course is to teach students the principles of object-oriented programming. Students are first familiarized with variables, conditional statements, and loops; then functions, function definitions, and parameters; after which students start practicing using objects and dividing responsibilities to different classes such as separating the (textual) UI and logic of the program.

The course lasts for seven weeks and utilizes a many-small-exercises approach. New exercises are released weekly and students have approximately a week to work on the exercises of a specific week. There were a total of 147 exercises over the duration of the seven-week course. The course has two exams: a midterm exam on the third week and a final exam at the end of the course. Both exams are computer-based and include tasks that are similar to the exercises students have completed over the course. The course grade emphasizes the weekly exercises with 70% of the grade coming from successfully completed exercises and 30% of the grade coming from the exams (10% from the midterm and 20% from the final exam). In order to successfully pass the course, students have to receive at least 70% of the combined points available from the exercises and the exams as well as get at least 50% of the points from the final exam.

Within the course, students use an Integrated Development Environment (IDE) with the TestMyCode plugin [36] to download exercises and to submit them for automated assessment. The exercises are accompanied by a unit test suite that students can freely use to evaluate their progress in the exercise. Once all the tests pass in the test suite, students are prompted to submit their work for grading. Students are allowed to submit exercises multiple times; however, this rarely happens due to students preferring to evaluate progress with the local test suite. In addition to handling exercise downloads and submissions, the IDE collects data about the students' programming process. The IDE collects every keystroke students take while they are working on course exercises, the timestamp of those keystrokes, as well as a timestamped snapshot of the program when students submit their work for assessment. In this study, we used data from 132 students who agreed for their data be used for research, had completed at least ten exercises in the course, and had data available from both the exam and the IDE.

3.2 Metrics and Research Questions

In this work, we examine two metrics derived from programming process data: *coarse-grained* time-on-task and *fine-grained* time-on-task. Additionally, we study the relationship between these two metrics, weekly exercise points, and exam points. Below are the definitions of the variables used in this study:

- **Coarse-grained time-on-task**: The time elapsed between the first keystroke and the first submission of the exercise. This is calculated separately for each exercise.
- **Fine-grained time-on-task**: The sum of latencies between keystrokes until the first submission of the exercise with all breaks of ten minutes and longer removed. This is calculated separately for each exercise.
- **Exercise points**: The points that students received for exercises of a specific week of the course. This is calculated separately for each week of the course.
- **Exam points**: The points that students received for the final exam at the end of the course. Students had to receive at least 10 out of 20 points to pass the course (regardless of exercise points).

The only difference between the *coarse* and the *fine-grained* time-on-task metrics is that the *fine-grained* time-on-task metric accounts for breaks that students took in their programming process as it does not include any breaks students took that lasted for ten minutes or more. The choice of setting the threshold for not including a break at ten minutes is based on previous work [23]. The time-on-task is only calculated until the first submission since in our context, students typically have a single submission for each exercise as they are allowed to evaluate their progress on the exercise by running local unit tests; thus, any subsequent submissions typically are result of some rare outlier situation.

3.3 Research Questions

Our research questions (RQs) for this study are the following:

RQ1. How do the fine-grained and the coarse-grained time-on-task metrics correlate with students' exam points and weekly exercise points?

RQ2. To what extent can students' performance in the course exam be predicted based on the fine-grained and the coarse-grained time-on-task metrics?

3.4 Research Methods

For research question 1, we computed the Pearson correlation coefficients between the different metrics. We calculated correlations separately for individual weeks of the course and over the whole

course. In the weekly correlation analysis, we summed the times-on-task of that week's assignments together to compute the correlation between the sums and weekly exercise points. Similarly, for the correlation analysis of the whole course, we summed the times-on-task of all the exercises of the course and computed the correlation between the sums and exam points.

For research question 2, we examined three different machine learning models to predict students' performance in the course using either coarse or fine-grained times-on-task as predictors. The three chosen machine learning models are random forest, logistic regression, and a majority classifier as a baseline to which we can compare the performance of the other two approaches. We chose random forest and logistic regression as they have been used in prior student performance prediction studies [14], typically perform quite well, and are from two different machine learning model families with random forest being a decision tree-based ensemble model and logistic regression being a regression-based model.

For the prediction task, the features used as predictors are the exercise-specific times-on-task with min-max normalization. The variable being predicted is a binary variable that represents whether the student received at least half of the points from course final exam, which was a requirement to pass the course. 103 students out of the 132 received at least half of the points. This means that the baseline majority classifier will always predict that a student will get over half of the points in the exam.

For training the models, we employed nested cross-validation with four folds for both the inner and outer loops using Scikit-learn [28]. In ordinary cross-validation, data is split into k folds, and the performance of the model is evaluated using each fold once as a test set and taking the average of the scores. Nested cross-validation is useful when the hyperparameters of the ML models need to be tuned. In nested cross-validation, the data is first split in an outer loop to k folds. Then, the training data (all folds except the one left as a test set) is again split into k folds (where k can differ from the outer loop) in the inner loop. The inner cross-validation is used to tune the hyperparameters of the models, while the outer cross-validation is used to evaluate the performance of the models. When tuning the hyperparameters, we optimized for accuracy and used a grid search over the hyperparameter space[1].

Since there were 147 exercises in the course, and thus 147 features (time-on-task per exercise), and only 132 students, we used feature selection to prune the number of features used in the prediction task. We used the SelectKBest[2] feature selection from Scikit-learn with chi-squared scoring to select the twelve best features (since $\sqrt{147} \approx 12$). The feature selection selected the same features for both time-on-task metrics. Five exercises from week 1, one from week 2, one from week 3, three from week 4, and two from week 6 out of the 147 total exercises were selected.

To evaluate how well the models performed in the prediction task, we calculated four different evaluation metrics. The four chosen metrics were accuracy, ROC-AUC, F1 score, and Matthews correlation coefficient (MCC). Brief definitions of the metrics are as follows:

- **Accuracy**. Accuracy represents how many students were correctly classified. The value ranges from 0 to 1 with 0 meaning none of the students were correctly classified and 1 meaning that all of the students were correctly classified.
- **ROC-AUC**. ROC-AUC is the area under the receiver operating characteristic curve. It represents how well the model performs with different classification thresholds. The value ranges from 0.5 to 1 where a higher value means that the model performed better.
- **F1 score**. The F1 score is the harmonic mean of precision (fraction of correctly identified students out of all identified students) and recall (fraction of correctly identified students out of all students who should have been identified). The value ranges from 0 to 1 where a higher value means that the model performs better.
- **Matthews correlation coefficient (MCC)**. MCC is calculated with the two-by-two confusion matrix of true and false positives and negatives and works well for imbalanced datasets. The value ranges from -1 to 1 where a value of 0 means that the model's performance was equal to random guessing. The closer that MCC is to 1 (all students classified correctly) or -1 (all students classified incorrectly), the better (since in the negative case, the model's predictions could be simply reversed).

[1]For random forest, the hyperparameter space was: n_estimators: [50, 100, 200], max_depth: [10, 20, None], min_samples_split: [2, 5, 10], min_samples_leaf: [1, 2, 4], and default Scikit-learn values for the rest of the parameters. For logistic regression, the hyperparameter space was: solver: ["lbfgs", "liblinear"], C: [100, 10, 1.0, 0.1, 0.01], and default Scikit-learn values for the rest of the parameters.

[2]https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

Table 1: Pearson correlation coefficients between the weekly metrics. The time-on-task metrics for a week are sums of the times-on-task for that week's exercises.

	Coarse ToT	Fine ToT
Coarse ToT		
Fine ToT	0.31 (p=1.3e-19)	
Exercise points	0.03 (p=0.39)	0.35 (p=1.6e-25)

Table 2: Pearson correlation coefficients between the metrics over the whole course. The time-on-task metrics for the course are sums of the times-on-task for all the exercises of the course.

	Coarse ToT	Fine ToT
Coarse ToT		
Fine ToT	0.50 (p=8.6e-10)	
Exam points	0.23 (p=0.007)	0.51 (p=3.7e-10)

4 RESULTS

4.1 Correlation Analysis

Table 1 shows the correlation between the two time-on-task metrics and weekly exercise points. From the table, we can see that the correlation between the coarse and the fine-grained time-on-task metric when summed for individual weeks is statistically significant (p-value = 1.3e-19) but weak (r = 0.31). Additionally, we can see

Table 3: Prediction results when predicting students' performance with the coarse-grained time-on-task. MCC for the majority vote is not defined due to division by zero.

	Accuracy	ROC-AUC	F1	MCC
Random forest	0.87	0.91	0.66	0.58
Logistic regression	0.79	0.74	0.09	0.10
Majority vote (baseline)	0.78	0.5	0.0	-

Table 4: Prediction results when predicting students' performance with the fine-grained time-on-task. MCC for the majority vote is not defined due to division by zero.

	Accuracy	ROC-AUC	F1	MCC
Random forest	0.93	0.97	0.82	0.80
Logistic regression	0.91	0.96	0.78	0.75
Majority vote (baseline)	0.78	0.5	0.0	-

that the coarse-grained time-on-task does not statistically significantly correlate with the weekly exercise points (p = 0.39), while the correlation between the fine-grained time-on-task and weekly exercise points is statistically significant (p = 1.6e-25) albeit weak (r = 0.35).

Table 2 shows the correlation between the time-on-task metrics and final exam points. We can observe that when summed over the whole course, the correlation between the fine-grained and the coarse-grained time-on-task metric is moderate (r = 0.50) and statistically significant (p = 8.6e-10). The correlation between the coarse-grained time-on-task and exam points is weak (r = 0.23) but statistically significant (p = 0.007). For the fine-grained time-on-task, the correlation with exam points is moderate (r = 0.51) and statistically significant (p = 3.7e-10).

4.2 Prediction Analysis

Table 3 shows the results of the prediction task when using the coarse-grained times-on-task as predictors. The results show that using the coarse-grained time-on-task to predict success with the logistic regression model is not feasible as the scores for all the evaluation metrics (accuracy, ROC-AUC, F1 and MCC) are very low. The random forest model shows some promise, achieving better results compared to logistic regression, although the accuracy of the model is only nine percentage points higher than the accuracy for the baseline majority classifier.

Table 4 shows the results of the prediction task when using the fine-grained times-on-task as predictors. Compared to the previous case of using the coarse-grained times-on-task, we see that the results have improved significantly. The performance of the logistic regression model and the random forest model are now similar. Both achieve very high scores in all of the studied metrics, and perform significantly better compared to the majority vote classifier.

5 DISCUSSION

5.1 Relationship Between Time-on-Task and Performance

In the correlation analysis performed for RQ1, we found that the correlation between the coarse-grained and the fine-grained time-on-task is lower than one might expect. Similar findings of different time-on-task metrics not correlating strongly have been reported in prior work [19, 23]. Both metrics are supposed to measure the same feature, time-on-task, but our results suggest this might not be the case. Both types of time-on-task metrics have been used previously (see e.g. [9, 19, 24]). However, it is possible that the metrics are measuring different things as noted in prior work [9, 19, 24]. Our fine-grained time-on-task metric differs from the coarse-grained metric only in that it takes breaks into account, removing any breaks that last over 10 minutes from the time-on-task. Thus, perhaps the fine-grained time-on-task metric should be referred to as "time-in-IDE" while the coarse-grained time-on-task should be referred to as, for example, "work span". If metrics that actually measure different things fall under the same umbrella, results related to those metrics can be unreliable. Hence, when comparing results between different studies, researchers should carefully consider the metrics used in the studies and ponder whether the studies are truly comparable.

We saw that in both the weekly and the whole course data, the fine-grained time-on-task correlated more strongly with performance compared to the coarse-grained time-on-task. For example, for the weekly data, we found that the correlation for the fine-grained time-on-task and weekly exercise points was an order of magnitude stronger (r = 0.31 vs r = 0.03) and that the correlation between the coarse-grained metric and weekly points was not even statistically significant. Similarly, considering the data for the whole course, while both metrics correlated statistically significantly with exam points, the correlation was two times stronger for the fine-grained metric (r = 0.5 vs r = 0.23). However, while the relationship between performance and time-on-task was stronger for the fine-grained metric, there could be other uses for the coarse-grained metric.

5.2 Predicting Success with Time-on-Task

Based on the results of RQ2, we saw that in addition to correlating more strongly with performance, the fine-grained time-on-task also worked better for predicting students' success in the exam. Interestingly, we saw that logistic regression seemed to suffer more from using the coarse-grained features compared to random forest. This suggests that random forest was able to better leverage the coarse-grained features. However, both models performed considerably better with the fine-grained features.

Overall, the performance of the models for predicting success is impressive. This highlights previous findings on how important time-on-task is for learning [2, 29, 34]. Considering predicting students' success, time-on-task could be useful in at least two ways. Firstly, as prior work has shown, more time engaged in educational activities tends to increase performance [34, 37]. On the other hand, it is possible that some of the performance of the models is due to the models learning that excessive time spent on exercises can be an indicator of struggling. For example, Jadud's Error Quotient [17],

which measures how much students struggle with syntax, has been found to be a good predictor of performance in some contexts [30]; and presumably, students who struggle a lot with syntax will work longer on the task on average compared to students who do not struggle with syntax.

The assignments chosen by the feature selection provide some insight into how the models discriminate between at-risk students and those not at risk of failing the exam. We found that the selected assignments included ones where students practice concepts that some students commonly struggle with. For example, we found that many of the assignments were those that are more mathematical or heavy on logic (e.g. conditions). This could be related to some students having a more extensive background in mathematics than others, or the students' perceptions on mathematics [33]. Similarly, some of the selected assignments involved different exceptions such as the null pointer exception and the index-out-of-bounds exception, which are common struggles for students and require students to carefully read the material before attempting the exercise if they do not have prior background in programming.

5.3 Limitations

There are some limitations to our study, which we outline here. Firstly, we only evaluated machine learning models using time-on-task features as predictors. Thus, our results only shed light on the performance of the two time-on-task metrics studied in this work for predicting performance, but not on how well time-on-task in general works for the prediction task compared to other types of features. It is possible that a simpler model based on, for example, the number of submissions would perform as well or better; although this would be contrary to prior work [20]. Another problem related to the chosen features is the relative simplicity of the predicted target variable. Future work should evaluate model performance with a more complex target variable; for example predicting course grade instead of a binary at-risk status.

Another limitation is the real-world applicability of the machine learning models. We used data from all exercises of the course to build the models, and thus e.g. interventions made based on the models could only be deployed near the end of the course[3], at which point it could be too late to provide at-risk students enough support to help them pass the exam. However, our main point in this work was to compare the two time-on-task metrics for prediction and not build a realistic prediction model that could be used already early in the course, which is part of our future work. In addition, prior work has found that time-on-task from the first week of a course can be a good predictor [2]. The features that were selected by the feature selection in this study included many features from the first weeks of the course; for example, five out of the twelve chosen features were from the first week, and ten out of twelve from the first four weeks, suggesting that the models did not solely focus on predicting success based on exercises that are situated near the end of the course.

Lastly, one of the main findings of this work is that the data used to build time-on-task metrics matters, and thus it is possible that the results of our study are specific to our particular context, and that different results would be observed in other contexts.

[3]In our case, at the end of the sixth week of the seven week course, since two assignments of the sixth week were selected by the feature selection. No assignments from the seventh week were selected.

6 CONCLUSIONS

In this work, we studied the relationship between time-on-task and performance. We examined this with two different time-on-task metrics that were based on prior work [23] that we call *fine-grained time-on-task* and *coarse-grained time-on-task*. The difference between the two metrics is that the fine-grained time-on-task does not include breaks (of ten or more minutes) students take in its time-on-task estimate while the coarse-grained time-on-task metric includes these breaks. Additionally, the fine-grained time-on-task metric requires more fine-grained log data to build whereas the coarse-grained time-on-task metric only requires information on when students begin and end work on an exercise. Next, we revisit our research questions and answer them.

RQ1. How do the fine-grained and the coarse-grained time-on-task metrics correlate with students' exam points and weekly exercise points?

Answer: We found that the fine-grained time-on-task metric correlates more strongly with both weekly exercise points and exam points. The fine-grained time-on-task metric had a weak (r = 0.35) correlation with weekly points and a moderate (r = 0.51) correlation with exam points. For the coarse-grained metric, the correlation with weekly points was not statistically significant and the correlation with exam points was weak (r = 0.23).

RQ2. To what extent can students' performance in the course exam be predicted based on the fine-grained and the coarse-grained time-on-task metrics?

Answer: We discovered that—similar to the correlation analysis—the fine-grained time-on-task metric also worked better for predicting students' success in the course final exam. We predicted whether students will get a passing grade in the exam with a random forest classifier, a logistic regression classifier, and a majority vote classifier (used as a baseline). Using coarse-grained time-on-task features, the logistic regression classifier performed almost on par with the baseline majority classifier, and the random forest classifier had moderate results, while with the fine-grained features, both models achieved results that were significantly better compared to the baseline majority vote classifier.

Altogether, our results suggest that future work that utilizes time-on-task for performance prediction should use as fine-grained data as possible to measure time-on-task if such data is available. Additionally, the high performance of the prediction models provides support for the notion that time-on-task is an important aspect of learning. Future work should further study how time-on-task can best be utilized for predicting students' performance, and especially focus on models that only use data collected early in the course so that interventions such as additional help could be deployed to struggling students promptly. In addition, while the present work focused on predicting performance, an interesting aspect could be to seek to predict future time-on-task based on present time-on-task estimates.

ACKNOWLEDGMENTS

We are grateful for the grant by the Media Industry Research Foundation of Finland which partially funded this work.

REFERENCES

[1] Alireza Ahadi, Raymond Lister, Heikki Haapala, and Arto Vihavainen. 2015. Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. 121–130.

[2] Ahmed Alamri, Mohammad Alshehri, Alexandra Cristea, Filipe D Pereira, Elaine Oliveira, Lei Shi, and Craig Stewart. 2019. Predicting MOOCs dropout using only two easily obtainable features from the first week's activities. In *International Conference on Intelligent Tutoring Systems*. Springer, 163–173.

[3] Lorin W Anderson and Corinne C Scott. 1978. The relationship among teaching methods, student characteristics, and student involvement in learning. *Journal of Teacher Education* 29, 3 (1978), 52–57.

[4] Brett A Becker, Catherine Mooney, Amruth N Kumar, and Sean Russell. 2021. A simple, language-independent approach to identifying potentially at-risk introductory programming students. In *Australasian Computing Education Conference*. 168–175.

[5] Adam Scott Carter and Christopher David Hundhausen. 2017. Using programming process data to detect differences in students' patterns of programming. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 105–110.

[6] Adam S Carter, Christopher D Hundhausen, and Olusola Adesope. 2015. The normalized programming state model: Predicting student performance in computing courses based on programming behavior. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. 141–150.

[7] Karo Castro-Wunsch, Alireza Ahadi, and Andrew Petersen. 2017. Evaluating neural networks as a method for identifying students in need of assistance. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 111–116.

[8] John Edwards, Juho Leinonen, and Arto Hellas. 2020. A study of keystroke data in two contexts: Written language and programming language influence predictability of learning outcomes. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 413–419.

[9] Stephen H Edwards, Jason Snyder, Manuel A Pérez-Quiñones, Anthony Allevato, Dongkwan Kim, and Betsy Tretola. 2009. Comparing effective and ineffective behaviors of student programmers. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop*. 3–14.

[10] Fabian Fagerholm and Arto Hellas. 2020. On the Differences in Time That Students Take to Write Solutions to Programming Problems. In *2020 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–9.

[11] Nickolas JG Falkner and Katrina E Falkner. 2012. A fast measure for identifying at-risk students in computer science. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research*. 55–62.

[12] Thomas L Good and Terrill M Beckerman. 1978. Time on task: A naturalistic study in sixth-grade classrooms. *The Elementary School Journal* 78, 3 (1978), 193–201.

[13] Joonas Häkkinen, Petri Ihantola, Matti Luukkainen, Antti Leinonen, and Juho Leinonen. 2021. Persistence of Time Management Behavior of Students and Its Relationship with Performance in Software Projects. In *Proceedings of the 17th ACM Conference on International Computing Education Research*. 92–100.

[14] Arto Hellas, Petri Ihantola, Andrew Petersen, Vangel V Ajanovski, Mirela Gutica, Timo Hynninen, Antti Knutas, Juho Leinonen, Chris Messom, and Soohyun Nam Liao. 2018. Predicting academic performance: a systematic literature review. In *Proceedings companion of the 23rd annual ACM conference on innovation and technology in computer science education*. 175–199.

[15] Christopher David Hundhausen, Daniel M Olivares, and Adam S Carter. 2017. IDE-based learning analytics for computing education: a process model, critical review, and research agenda. *ACM Transactions on Computing Education (TOCE)* 17, 3 (2017), 1–26.

[16] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, et al. 2015. Educational data mining and learning analytics in programming: Literature review and case studies. *Proceedings of the 2015 ITiCSE on Working Group Reports* (2015), 41–63.

[17] Matthew C Jadud. 2006. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the second international workshop on Computing education research*. 73–84.

[18] Jacob S Kounin and Paul V Gump. 1974. Signal systems of lesson settings and the task-related behavior of preschool children. *Journal of educational psychology* 66, 4 (1974), 554.

[19] Vitomir Kovanović, Dragan Gašević, Shane Dawson, Srećko Joksimović, Ryan S Baker, and Marek Hatala. 2015. Does Time-on-Task Estimation Matter? Implications for the Validity of Learning Analytics Findings. *Journal of Learning Analytics* 2, 3 (2015), 81–116.

[20] Vitomir Kovanović, Dragan Gašević, Shane Dawson, Srećko Joksimović, Ryan S Baker, and Marek Hatala. 2015. Penetrating the black box of time-on-task estimation. In *Proceedings of the fifth international conference on learning analytics and knowledge*. 184–193.

[21] Juho Leinonen. 2019. *Keystroke Data in Programming Courses*. Ph.D. Dissertation. University of Helsinki.

[22] Juho Leinonen, Francisco Enrique Vicente Castro, and Arto Hellas. 2021. Does the Early Bird Catch the Worm? Earliness of Students' Work and its Relationship with Course Outcomes. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. 373–379.

[23] Juho Leinonen, Francisco Enrique Vicente Castro, and Arto Hellas. 2021. Fine-Grained Versus Coarse-Grained Data for Estimating Time-on-Task in Learning Programming. In *Proceedings of the 14th Educational Data Mining Conference*.

[24] Juho Leinonen, Leo Leppänen, Petri Ihantola, and Arto Hellas. 2017. Comparison of time metrics in programming. In *Proceedings of the 2017 ACM conf. on International Computing Education Research*. 200–208.

[25] Juho Leinonen, Krista Longi, Arto Klami, and Arto Vihavainen. 2016. Automatic inference of programming performance and experience from typing patterns. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 132–137.

[26] Soohyun Nam Liao, Daniel Zingaro, Kevin Thai, Christine Alvarado, William G Griswold, and Leo Porter. 2019. A robust machine learning technique to predict low-performing students. *ACM Transactions on Computing Education (TOCE)* 19, 3 (2019), 1–19.

[27] Quan Nguyen. 2020. Rethinking time-on-task estimation with outlier detection accounting for individual, time, and task differences. In *Proceedings of the Tenth International Conference on Learning Analytics & Knowledge*. 376–381.

[28] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[29] Filipe Dwan Pereira, Samuel C Fonseca, Elaine HT Oliveira, Alexandra I Cristea, Henrik Bellhäuser, Luiz Rodrigues, David BF Oliveira, Seiji Isotani, and Leandro SG Carvalho. 2021. Explaining Individual and Collective Programming Students' Behavior by Interpreting a Black-Box Predictive Model. *IEEE Access* 9 (2021), 117097–117119.

[30] Andrew Petersen, Jaime Spacco, and Arto Vihavainen. 2015. An exploration of error quotient in multiple contexts. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*. 77–86.

[31] Leo Porter, Daniel Zingaro, and Raymond Lister. 2014. Predicting student success using fine grain clicker data. In *Proceedings of the tenth annual conference on International computing education research*. 51–58.

[32] Keith Quille and Susan Bergin. 2018. Programming: predicting student success early in CS1. a re-validation and replication study. In *Proceedings of the 23rd annual ACM conference on innovation and technology in computer science education*. 15–20.

[33] Nikki Sigurdson and Andrew Petersen. 2019. A Survey-Based Exploration of Computer Science Student Perspectives on Mathematics. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 1032–1038.

[34] Jane Stallings. 1980. Allocated academic learning time revisited, or beyond time on task. *Educational researcher* 9, 11 (1980), 11–16.

[35] Claudia Szabo, Nickolas Falkner, Antti Knutas, and Mohsen Dorodchi. 2018. Understanding the effects of lecturer intervention on computer science student behaviour. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*. 105–124.

[36] Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Martin Pärtel. 2013. Scaffolding students' learning using test my code. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. 117–122.

[37] Herbert J Walberg. 1988. Synthesis of research on time and learning. *Educational leadership* 45, 6 (1988), 76–85.