Iyer, S. et al. (Eds.) (2022). Proceedings of the 30th International Conference on Computers in Education.
Asia-Pacific Society for Computers in Education

Detection of At-Risk Students in Programming Courses

**Ikkei IGAWA[a*], Yuta TANIGUCHI[a] , Tsubasa MINEMATSU[a] ,
Fumiya OKUBO[a] & Atsushi SHIMADA[a]**
[a] *Kyushu University, Japan*
*igawa@limu.ait.kyushu-u.ac.jp

Abstract: Since the demand for programmers is increasing, programming courses are being offered widely. In this context, students' motivation can be damaged by difficulties they encounter in their programming courses. Although teachers' support is necessary to prevent such an issue, it is impossible for teachers to directly monitor all students' programming activities at the same time and determine which students have troubles with programming. Therefore, several studies have been conducted to help teachers monitor students. However, these studies do not provide an understanding of the activities of students who do not run their code, which may lead researchers to miss students who are in trouble. In this paper, we propose an indicator for detecting students who need coding support by analyzing programming logs that are recorded even when the students do not run their code. This gives teachers deeper insight into the students' programming performance. Although further work remains, the validation of this indicator shows that it could detect those students who are in trouble.

Keywords: Programming education, educational data analysis, clustering, teacher support

1. Introduction

Recently, the demand for programmers has been rapidly increasing, and the lack of programmers has become serious. Therefore, developing programmers is an urgent need, based on which various changes are being implemented in the field of education. For example, programming education became mandatory in Japanese elementary schools in 2020.

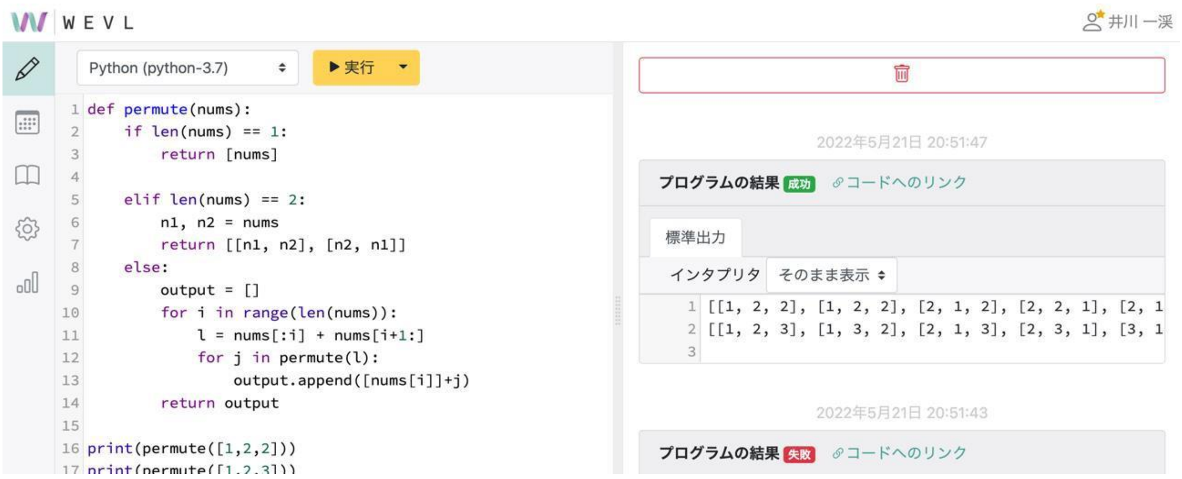
Jenkins (2001) and Pilkington (2018) have reported that students' motivation to learn is important in order for them to succeed in programming courses. However, their motivation is often affected by the difficulties they encounter in coding, which may lead students to drop out (Martins et al., 2010; Feldgen et al., 2004). To prevent this, teachers must monitor students' programming activities and determine which students have troubles with programming (hereafter, at-risk students). However, the larger the number of students, the more difficult it is to monitor all of them at the same time.

To help the teachers monitor students, various studies have been conducted (e.g., Fonseca et al., 2018; Fonseca et al., 2021; Penmetsa et al., 2021). Many of them include the creation of a dashboard. Fu et al. (2017) developed a dashboard that enables teachers to identify the patterns of errors students frequently make by obtaining error messages from the compiler. Furthermore, Diana et al. (2017) analyzed programming logs to predict the students' final grade in a programming course. In those works, however, the researchers only used students' programming logs that were recorded when they ran their code, which means that they could not collect programming logs for analysis from students who wrote their code without execution. This leads to some problems in monitoring students. One such problem is the lack of logs to analyze. For students who seldom run their codes, researchers cannot collect a sufficient number of logs. This leads us to miss at-risk students due to the reduced accuracy of detection. If we can obtain programming logs while the students are coding regardless of execution, we can increase our sample size and deepen our understanding of students' programming activities.

In this paper, we design a better indicator to detect at-risk students by calculating the frequencies of execution errors and autosaves. Autosave is an event that automatically records the students' programming logs such as time and code even if they do not execute their code. This enables us to get a sufficient number of programming logs and identify students' programming activity while they are coding without execution. In addition, we validate how effective the frequencies of the execution errors and autosaves are at finding at-risk students.

2. Programming Log Collecting Tool

The WEVL is an online coding application currently being developed in our laboratory as well as being used in some programming courses in our university (Taniguchi et al., 2022). Figure 1 shows the user interface of WEVL. If a student writes a code in the text box on the left side and clicks the yellow execution button, the execution result of the code is displayed on the right side. While writing a code on WEVL, various pieces of information such as name, time, and code are automatically recorded to the WEVL database. This recording event occurs twice. The first time is when students run the code. In this case, the result of the execution is also saved. The second is when the student stops typing for three seconds. However, if the code is the same as the one that was saved previously, this event does not occur.



```python
def permute(nums):
    if len(nums) == 1:
        return [nums]

    elif len(nums) == 2:
        n1, n2 = nums
        return [[n1, n2], [n2, n1]]
    else:
        output = []
        for i in range(len(nums)):
            l = nums[:i] + nums[i+1:]
            for j in permute(l):
                output.append([nums[i]]+j)
        return output

print(permute([1,2,2]))
print(permute([1,2,3]))
```

```
[[1, 2, 2], [1, 2, 2], [2, 1, 2], [2, 2, 1], [2, 1
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1
```

Figure 1. The User Interface of WEVL.

By automatically recording the programming logs, we can obtain a sufficient number of logs for analysis. For example, in the case of a certain 90-minute programming course in our university consisting of 87 students, the database includes 1,069 autosave logs and 245 execution logs.

If we line up the programming logs in chronological order, we can obtain one time series set for each student. In this paper, we call these series' elements "save-points" and use them to detect at-risk students.

3. Detection of At-Risk Students

3.1 *Indicator for Detection*

We hypothesized that there are some situations that lead a student to be in an at-risk situation; one example situation is repeatedly getting errors when they run their code. In this case, the students could be in an at-risk situation. Another situation is when students get stuck while writing their code. For example, when a student is unsuccessfully trying to solve a task in a programming course, s/he may frequently stop writing code. In other words, s/he would be repeating the trial-and-error process.

Next, we detected students in such situations based on their activity. Students who are repeatedly getting errors when running their code can be detected simply by checking the frequency of execution errors in their save-points. However, it is more difficult to detect students who are repeating trial-and-error processes. As mentioned in section 2, autosave happens when students stop typing a code for a short time. Therefore, we can say that autosave is a good candidate for the detection of trial and error.

We then designed an indicator to detect at-risk students based on the idea that the frequencies of certain kinds of save-points have fruitful information.

3.2 *Designing the Indicator*

We designed an indicator for detecting at-risk students by using students' save-points. As mentioned in section 2, there are three kinds of save-points when students are coding on WEVL: run and success, run and fail, and autosave. Figure 2 shows example save-points of a certain student.

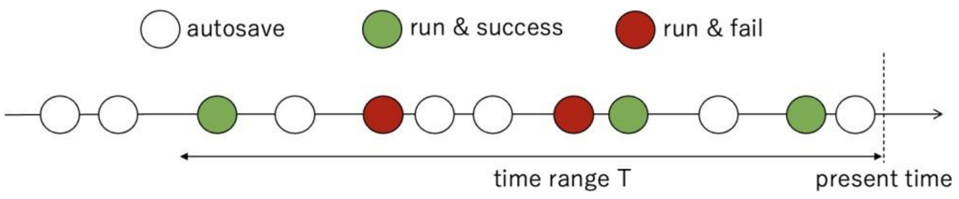


Figure 2. An Example of a Student's Save-points.

In Figure 2, the white points represent autosaves, the green points represent run and success, and the red points represent run and fail. As noted in section 3.1, at-risk students can be found by checking the frequencies of failures and autosaves. When calculating these frequencies, we only take account of recent save-points. As shown in Figure 2, we set the time range T, which covers the past for a certain period to the present time. We calculated the frequencies of the number of save-points in this time range. When we set time range T to T minutes at the present time t, we defined the ratio of errors from time t-T to time t as "failure ratio" and the ratio of autosaves as "autosave ratio." In the case of Figure 2, the number of the save-points in the time range T is 10. There are five autosaves and two failures in the time range T, so the autosave ratio is 50% and the failure ratio is 20%. In this research, we use these ratios as an indicator for detecting at-risk students and validate whether they can detect such students via a classification experiment.

3.3 *Identifying Coding Situations*

In this section, we identify students' coding situations based on the ratio of failures and autosaves. Table 1 shows four possible groups classified by the size of the students' failure and autosave ratio.

Table 1. *Four Groups Classified by Failure and Autosave Ratio*

		autosave ratio	
		high	**low**
failure ratio	**high**	Group A	Group B
	low	Group C	Group D

Group A consists of students who have a high failure and autosave ratio. As mentioned before, a high autosave ratio represents a high probability of repeating trial and error. Therefore, students in this group can be identified as those who are trying to run their code but not finding success. Thus, they can be at-risk students.

Group B is a group of students who have a high failure ratio and a low autosave ratio. A low autosave ratio represents that the student can smoothly write a code. Given that such students have a high failure ratio, this group indicates students who are making several errors occur but are writing code to successfully solve them.

In group C, students have a high autosave ratio and a low failure ratio. In other words, they are repeating trial-and-error without making frequent errors. In general, students with a low failure ratio are less likely to be at-risk. However, we hypothesized that high frequency of trial-and-error leads a student to be in an at-risk situation in section 3.1. Therefore, we cannot easily say that they are low-risk students.

Finally, group D includes students with a low failure and autosave ratio. Naturally, this ratio represents a high frequency of success. Therefore, students in this group can be identified as those who run their code several times, typically successfully, which is a low-risk situation.

To sum up, group A includes at-risk students while group B and D does not. In group C, it seems difficult to say whether the students are at risk. For this reason, we conducted a classification experiment and validated the indicator.

4. Validation of Indicator

4.1 Dataset

In this research, we conducted an experiment using the programming logs collected from the WEVL database. These logs were recorded in the database during a Python programming exercise course offered in our university in 2020, and we used the logs for three sessions of the course.

In Figure 2, we calculated the failure and autosave ratio from save-points in the time range T. If this time range is long enough, a sufficient number of save-points can be collected. If it is too long, however, both the failure and autosave ratios are affected by past data, which may lead to missing students that are currently having troubles with programming. In this experiment, we set the time range T to 30 minutes. Whether this length of the range is appropriate may require discussion. The course was offered from 8:40 am to 10:10 am, and the latter part of the class was used for exercises. Thus, we collected students' programming logs that were recorded from 9:30 am to10:00 am, obtaining 165 pieces of time series data of save-point logs. These save-point logs comprise 6,647 programming logs (1,255 execution logs and 5,392 autosave logs).

The purpose of this research is to find at-risk students. However, we cannot know whether a given student had trouble in a class period based solely on the logs in the database. Therefore, we used the assignment's scores to evaluate the classified students. This assignment was imposed every week in the programming course, and a full score is 5. We conducted an experiment based on the idea that this score is related to the performance in the course.

4.2 Setting the Thresholds

In section 3.3, we created a table divided by whether the ratio was high or low. To do so, a threshold is required for both the failure and autosave ratios.

First, we set the thresholds of failure and autosave ratio at 50%. We then calculated the average score of the assignments for each group. Finally, we maximized the differences between groups A and C, which consist of students with a high autosave ratio, and groups B and D, which consist of students with a low autosave ratio.

Figure 3 shows how the differences of the average score changes according to the thresholds. The vertical axis represents the threshold of the autosave ratio, and the horizontal axis shows the threshold of the failure ratio; the gradation represents the magnitude of the difference. The greater the difference, the brighter the color of the point. According to this figure, the ratios that maximize the differences of the scores are 20% for the autosave ratio and 70% for the failure ratio. Thus, we set these as the thresholds.

4.3 Results of the Experiment

After setting the thresholds, we classified the students into four groups. Table 2 shows the statistics of each group's assignment score; 5 represents the highest score. In this section, we verify how the failure and autosave ratio are effective to detect at-risk students by examining the relationship between the identified situation and the weekly assignment score.

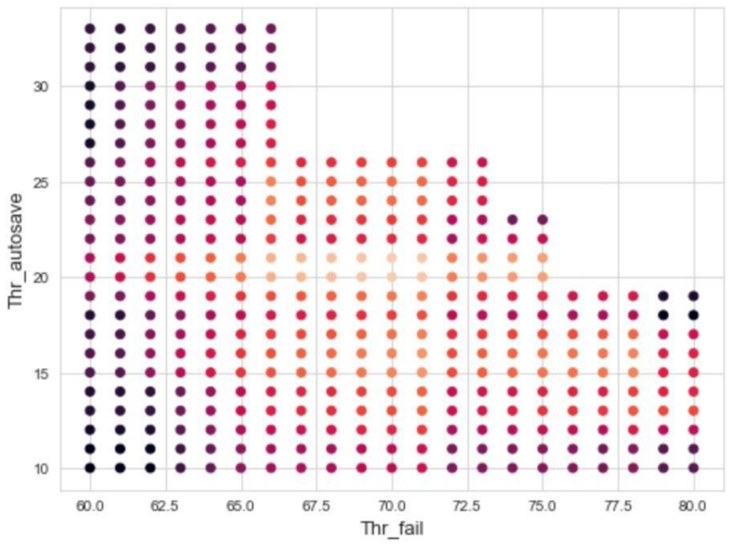


Figure 3. The Differences of the Average Assignment Score.

Table 2. *The Statistics of Each Group's Assignment Score*

	Group A	Group B	Group C	Group D	all
number of data	11	25	114	15	165
average	3.38	3.88	3.92	4.38	3.92
std.	0.83	0.92	1.13	1.10	1.09

Students in group A have high failure and autosave ratios, which suggests that they try to run their code successfully but it does not work well. Table 2 shows that this group's average score is the lowest of the four groups' scores. Therefore, we can say that they are likely to be at-risk students.

In group B, students have a high failure ratio and a low autosave ratio. They are identified as getting several coding errors, but being able to solve them smoothly. Although they have as high a failure ratio as group A, Table 2 shows that they received scores that were not so low compared to the average across the groups.

It seemed difficult to judge whether the students in group C are successful in coding. Table 2 shows that students in this group scored almost the same as the average of all the students, and about 70% of the data points were classified into this group.

Finally, students in group D exhibited low failure and autosave ratios, suggesting that they are coding successfully without getting errors. This group's average score is the highest of the four. Therefore, we can say that they are not likely to be at risk.

4.4 Discussion

Students frequently getting coding errors seem to be at risk in general. However, our results shows that a high error frequency does not always lead to a poor grade. If we classify the students only by failure ratio, groups A and B are classified together. That group's average assignment score is 3.73 out of 5, which does not differ much from the overall average score. Therefore, we can say that failure ratio is not sufficient to classify students as having a poor grade, and we can extract at-risk students from those who have a high failure ratio by using the autosave ratio. Thus, the autosave ratio is an important value for detecting at-risk students. The same applies to groups C and D, as those groups' joint average assignment score is 3.97. From the above, we can also say that the autosave ratio is useful for detecting low-risk students.

The results for Group C show that they received a similar assignment score as the overall average, and many more students were classified into Group C than any other group; both successful and unsuccessful students were included in this group. Thus, more details should be used to classify this group.

5. Conclusion

In this paper, we proposed an indicator to detect at-risk students in programming courses to help teachers comprehend students' programming activities and detect those who are at risk. By making good use of WEVL, which can record students' codes even if they are not executed, we can analyze many more programming logs and capture students' activities while they are coding. We then designed an indicator calculating the frequencies of execution errors and autosaves. We classified students into four groups and validated how important the ratios are to detect at-risk students. The results showed that in addition to failure ratio, autosave ratio, or the amount of trial and error in coding, is valuable for detection. Group A, which had high failure and autosave ratios, are at-risk students. However, about 70% of all students were classified into group C, and the score distribution is similar to the overall average. Thus, this group should be classified in more detail. In this research, we used only two parameters, autosave ratio and failure ratio, to classify students. Therefore, we are conducting further experiments using other parameters such as the length of the code. We are conducting further research to improve detection accuracy and hope that it will be of great help for teachers in the field of programming education.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number JP21K17863 and JP22H00551, Japan.

References

Fu, X., Shimada, A., Ogata, H., Taniguchi, Y., & Suehiro, D. (2017). Real-time learning analytics for C Programming Language courses. *Proceedings of the Seventh International Learning Analytics & Knowledge Conference*. doi:10.1145/3027385.3027407

Diana, N., Eagle, M., Stamper, J., Grover, S., Bienkowski, M., & Basu, S. (2017). An instructor dashboard for real-time analytics in interactive programming assignments. *Proceedings of the Seventh International Learning Analytics & Knowledge Conference*. https://doi.org/10.1145/3027385.3027441

Jenkins, T. (2001). The motivation of students of programming. *ACM SIGCSE Bulletin*, *33*(3), 53–56. https://doi.org/10.1145/507758.377472

Taniguchi, Y., Minematsu, T., Okubo, F., & Shimada, A. (2022). Visualizing Source-Code Evolution for Understanding Class-Wide Programming Processes. *Sustainability 2022*, *14*, 8084. https://doi.org/10.3390/su14138084

Fonseca, N. G., Macedo, L., Marcelino, M. J. & Mendes, A. J., (2018). Augmenting the teacher's perspective on programming student's performance via permanent monitoring, *IEEE Frontiers in Education Conference (FIE)*, 2018, pp. 1-9, doi: 10.1109/FIE.2018.8658924.

Penmetsa, P., Shi, Y., & Price, T. (2021). Investigate Effectiveness of Code Features in Knowledge Tracing Task on Novice Programming Course. *Work-In-Progress Track, CSEDM Workshop @ EDM'21*

Fonseca, N. G., Macedo, L. & Mendes, A. J., (2021). The importance of using the CodeInsights monitoring tool to support teaching programming in the context of a pandemic, *IEEE Frontiers in Education Conference (FIE)*, pp. 1-8, doi: 10.1109/FIE49875.2021.9637292.

Pilkington, C. (2018). A Playful Approach to Fostering Motivation in a Distance Education Computer Programming Course: Behaviour Change and Student Perceptions. *The International Review of Research in Open and Distributed Learning, 19*(3), –. doi:10.19173/irrodl.v19i3.3664

Martins, S. W., Mendes, A. J. & Figueiredo, A. D., (2010). A strategy to improve student's motivation levels in programming courses, *IEEE Frontiers in Education Conference (FIE)*, pp. F4F-1-F4F-7, doi: 10.1109/FIE.2010.5673366.

Feldgen, M. & Clua, O., (2004). Games as a motivation for freshman students learn programming, *34th Annual Frontiers in Education*, pp. S1H/11-S1H/16 Vol. 3, doi: 10.1109/FIE.2004.1408712.