30 Days After Introducing Programming: Will My Students Fail?

Márcio Ribeiro[a], Rodrigo Paes[a,*], Rohit Gheyi[b]

[a] *Federal University of Alagoas, Maceió, Brazil*
[b] *Federal University of Campina Grande, Campina Grande, Brazil*

Abstract

Predictors to identify whether a student will succeed or fail in introductory programming courses have been provided by previous research. However, these predictors rely on big, complicated, and time-consuming aptitude tests and surveys, bringing subjectiveness due to the interview process. This way, setting, executing, and replicating these studies is hard and increase the professor effort. To minimize this problem, this article proposes a strategy to automatically predict the potential failing students during introductory programming courses, reducing effort and allowing professors to use it in every course. By having this set of students in the first days of the course, professors and mentors would have time to act and potentially avoid such failings. To evaluate our strategy, we conduct an empirical study regarding 227 freshmen students of 7 introductory programming courses (3.5 years in total; 7 semesters). We apply our strategy considering the first 30 days of the course (22% of each semester). The same professor taught all courses using the C language. The study reveals that, from the group of students our strategy points as "likely to fail," 72% of the students indeed fail with 95% standard confidence level. In addition, despite missing 28%, this set seems still interesting, since at least 33% of these students reach the final exam. Therefore, although they pass, we still consider they are good candidates to give special attention as well, which may avoid final exams and

[*] Corresponding author

Email addresses: `marcio@ic.ufal.br` (Márcio Ribeiro), `rodrigo@ic.ufal.br` (Rodrigo Paes), `rohit@dsc.ufcg.edu.br` (Rohit Gheyi)

lead to better grades. We also conclude that the efficacy of our strategy depends on the number of students in the course.

Keywords: Predictors, Introductory Programming Courses, Dropout Rate.
2010 MSC: 00-01, 99-00

1. Introduction

Introductory programming courses are often perceived by the students as problematic [1]. This understanding seems to be based on the high dropout rates [2, 3], which has been observed in universities around the world, such as at the United States [4], Finland [5], and ours in Brazil.

To predict the students likely to succeed or fail, previous studies provide aptitude tests regarding programming activities. These studies focus on identifying qualitatively the student skills and consider many different variables from big, complicated, and time-consuming surveys. Also, they might bring subjectiveness due to the interview process. To perform the predictions, they use past academic achievement [6, 7], diagnostic and math/logic-based tasks [8, 9], mental models [2], games [10], and even programming languages [11]. Thus, although we can predict the potential failing students by using these studies, setting, executing, and replicating them is hard, time consuming and require extra effort from the professors.

In this context, there is a lack of an automatic approach capable of predicting a set of students that will fail. This way, professors and mentors would be able to act and consequently help them. To achieve promising results, however, this approach must accomplish two main requirements. First, it must correctly predict as soon as possible, otherwise there will be not enough time to act and the student would drop out the course anyway. Also, due to the strong prerequisites of understanding previous classes to understand the current one in programming courses (e.g., to understand loops, students must understand conditional structures), the situation gets worse, even when acting just a bit late. Second, the approach must be automatic, requiring almost no effort from professors and mentors and allowing them to use it in every course they teach.

To minimize this problem, in this article we propose an automatic strategy to early predict failing students in introductory programming courses. Our strategy consists of three simple steps. The first one is to make students use an online judge system. This kind of system executes an online submitted solution to a given problem against a set of predefined test cases to check whether the solution is correct or not. Then, we collect metrics of each student by using such a system. Finally, we execute a clustering algorithm [12] to form groups so that we are able to separate the potential failing students from the other ones.

To evaluate our strategy, we conduct an empirical study regarding 7 introductory programming courses—3.5 years; 7 semesters—with, in total, 227 freshmen students (32 per course, on average). The courses focused on the C language and have been given by the same professor at the Federal University of Alagoas in Brazil. We apply our strategy considering the first 30 days of the programming course (22% of each semester). The results suggest that our strategy can early predict the majority of the failing students within only 30 days. In particular, from the group of students our strategy points as "likely to fail," 72% of the students indeed fail with 95% standard confidence level. Moreover, we observe that the 28% our strategy misses is still important to take into account, since at least 33% of these students have difficulties to pass and reach the final exam. So, although they pass, this set has good candidates that need special attention as well. Also, we conclude that the efficacy of our strategy can strongly depend on the number of students in the course. Because the strategy is automatic, the effort to predict the potential failing students in a course with hundreds [4] can significantly reduce. In case the course has few students (e.g., 10 students), we report that our strategy adds little, since the own professor can identify the failing candidates.

In summary, this article provides the following contributions:

- A strategy to early predict automatically the potential failing students in introductory programming courses;

- An empirical study assessing the potential of our strategy. We evaluate our strategy by using 227 students from 7 courses during 3.5 years, demonstrating significant potential.

We organize the remainder of this article as follows. Section 2 discusses in detail the problem we address in this work. Then, Section 3 introduces our strategy and details the three steps we consider. Section 4 presents the evaluation we perform whereas Section 5 discusses the results and findings. We then discuss the related work in Section 6. Last but not least, we present the concluding remarks in Section 7.

2. Problem

Professors often are not aware of what actually hinders particular students during the learning process. Another difficulty consists of pointing out the exactly students that need help, specially when considering courses with hundreds of enrolled students [4]. With no support, students might have no enthusiasm regarding the classes and get frustrated and disappointed, leading them to fail the course.

In this context, there is no straightforward way to predict the potential failing students during introductory programming courses. When professors and mentors do not early identify such students, they may act too late—or even may not act, since there is no available time anymore—and the students drop out the course anyway. In case they act just a bit late, they still face the hard task of recovering such students, which happens to be even harder in programming courses. Here, to understand the next class there is a strong prerequisite of understanding the previous ones (e.g., to understand loops, students must understand conditional structures).

So, not identifying failing students early is a critical problem, specially when we consider that programming is one of the first courses that students face in their computer science university program. If these courses have as high failure rates as claimed, they could be one of the factors influencing the declining number of students taking a degree in computer science [4].

Previous research introduce predictors to identify these students. However, they use aptitude tests and surveys to qualitatively identify the student skills and decide whether the student will succeed or fail [7, 8, 2], which might bring subjectiveness due to the interviews. They also consider lots of variables, which makes the studies hard to set, execute, and replicate, being time consuming and increasing the professor effort. So, it is difficult to gather and analyze this information at the beginning of the semester [11] to determine the students candidates to fail.

To minimize this problem, we present in the next section a strategy to predict automatically the potential failing students in introductory programming courses. Our strategy consists of three simple steps and it is automatic and objective (relying on metrics), reducing effort and allowing professors to use it in their courses.

3. Strategy to early predict potential failing students

Predicting the potential failing students early is very important in the sense that professors still have enough time to act and avoid students to fail the course. Notice that in case professors act based on the results of our strategy, we can indirectly help on reducing the high rate of failing students. However, in this article, we focus exclusively on identifying these students. This way, evaluating and reporting the rate after applying our strategy is outside the scope of this article.

Our strategy is automatic and objective, consisting of three simple steps: the use of a online judge system, metrics collection, and execution of a well-known clustering algorithm [12]. Because the strategy is automatic, professors can replicate in their courses, reducing their effort.

Next, we detail the three steps of our strategy to predict the failing students.

3.1. Online judge system

To assess the students performance during a course, it is important to closely monitor them. In this context, metrics represent an alternative to measure their performance. However, retrieving metrics regarding each student is a difficult and time-consuming task. To minimize this problem, one might rely on online learning tools, since these systems are not only a source for these metrics, but allow their automatic retrieval.

A popular category of this kind of system is the online judges [13, 14]. These systems provide a set of programming problems so that students can submit their solutions. After submitting, the online judge executes the solution against a set of predefined test cases. In case the solution passes over all the tests, the system evaluates the solution as correct. Otherwise, the system evaluates the solution according to the error type (e.g., wrong answer, compilation error, time limit exceeded, runtime error, etc) and even might give important tips so that students can successfully solve the problem afterwards.

Since students only learn how to program by programming [15], the online judges consists of an important environment in the sense students can constantly practice and improve their learning process. In addition, online judges play an important role regarding rapid, constructive, and corrective feedback, once professors are often overwhelmed with their daily activities[1] and sometimes are not able to help each student separately [16]. The support is even harder in large class sizes [16], so an online tool helps on this front as well.

Given all these advantages, the use of an online judge system is the first step of our strategy. We then can monitor students to automatically identify the candidates to fail the course.

[1] For instance, in Brazil professors very commonly deal with bureaucratic activities and have no support from administrative assistants.

3.2. Metrics

The second step of our strategy consists of metrics retrieval. In particular, we retrieve two metrics by using the online judge system. We explain them in what follows:

- **Number of submissions:** this metric represents the number of submissions a student do during the course. Online judge systems commonly provide many problems. To solve a particular one, a student needs to submit at least one correct solution.
- **Number of correct submissions:** if a student solves one problem, we increase this metric by one.

Depending on the level of difficulty of a problem, students may submit several times to solve it. However, notice that this is not necessarily a bad thing regarding the learning process. For example, submitting many times means that students are somehow practicing and studying continuously. Thus, although she is not hitting a right solution, she is trying hard and will eventually hit one. Thus, the number of submissions metric is a good indicator of the amount of practice, which is very important in programming activities [17] and is frequently associated with the level of engaging and learning. However, when considered isolated, a high number of submissions may also mean that the student is getting frustrated due to so many wrong answers she receives. This way, we also consider the number of correct submissions to compensate and help us on studying such cases as well.

3.3. Clustering algorithm

To identify potential failing students, we use an existing clustering algorithm [12] to define groups of students. Our idea consists of identifying different groups of students so we can clearly separate students susceptible to fail from the other ones. To compute the groups, the algorithm takes into account the metrics we present in Section 3.2. To make the strategy parameterizable in terms of number of groups, we use the well-known clustering algorithm k-means [18], which takes such a number as input.

Notice that the number of groups plays an important role on identifying potential failing students. In this article, we set the algorithm to compute two and three groups and evaluate both cases. For two groups, we have students who will either fail or pass. For three groups, we have fail, pass, and students in which the strategy will not conclude anything about them, which we name "inconclusive." This is reasonable since our study focuses on the very first 30 days, which means our drawings regarding the inconclusive group might be completely wrong: these students can either improve themselves and pass the course or fail due to several reasons.

We focus on two and three groups basically for two reasons: (i) using one group is useless; and (ii) more than three only brings more inconclusive groups to the table which, given our context, is pretty much the same of having only one inconclusive group.

3.4. Summary

Figure 1 combines all steps of our strategy. Notice that our strategy is general in the sense we can use any online judge system, as long as we can collect the metrics we use from the system.

To instantiate our strategy, we use the online judge system named *Huxley* [19]. The system is available online only in portuguese at `http://www.thehuxley.com/`. There are almost 100 registered professors and more than 100 registered courses. Also, more than 20 institutions around Brazil use the system. Huxley provides a database composed by more than 400 problems classified according to level of difficulty and programming topics. This allows the students to choose the next problem in accordance to their corresponding levels. Notice that this helps on avoiding misleading in our metrics, e.g., many incorrect submissions to a problem that the student is not able to solve yet.

Next, we collect the metrics we detail in Section 3.2. To identify potential failing students *early*, we collect the metrics for the first 30 days (representing 22% of the semester) of the introductory programming courses. Then, we execute the k-means clustering algorithm.

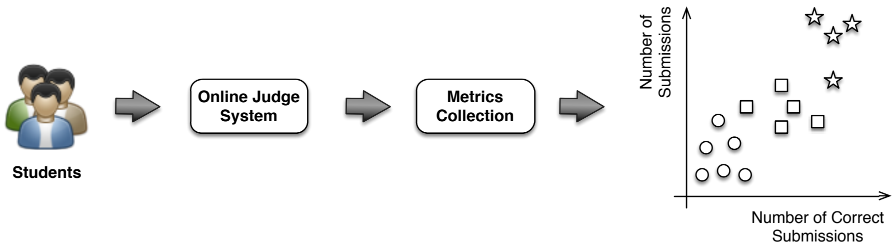


Figure 1: Summary of our strategy to identify potential failing students.

We illustrate the k-means result in a two-dimensional plot (we represent one dimension for each metric we consider). Also, we illustrate the groups using shapes in Figure 1. In this particular case, we set k-means to compute three different groups. The circles represent the students candidates to fail the course. Notice that they submitted few solutions and few of them are correct.

4. Evaluation

To evaluate our strategy, in this section we present the empirical study we conduct. The objective of this study is to evaluate to what extent our strategy is capable of identifying potential failing students in only 30 days. To explain our evaluation design, we first present the general settings in Section 4.1. Then, we detail the procedure we use during the evaluation in Section 4.2.

4.1. Settings

The participants of our study are freshmen students of introductory programming courses at the Federal University of Alagoas in Brazil. We collected the metrics we detail in Section 3.2 of each student by using Huxley. Table 1 distributes the number of participants per semester.

Course	**Number of enrolled students**
2010.02	32
2011.01	38
2011.02	35
2012.01	34
2012.02	29
2013.01	28
2013.02	31
TOTAL:	227

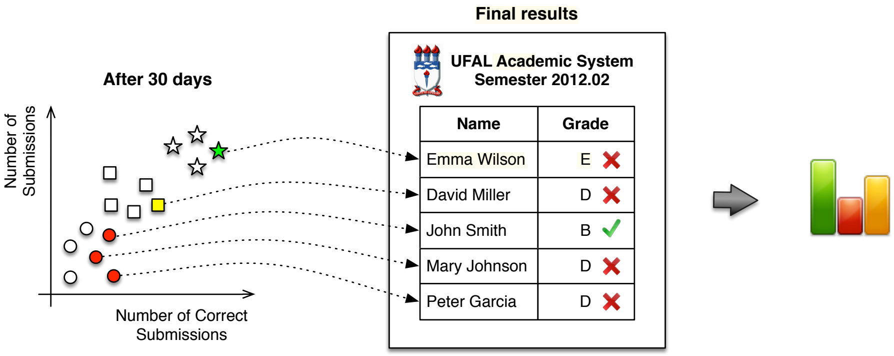
Table 1: Participants per course.

The same professor (Paes, one of the authors) taught the 7 semesters. The classes happened in a lab and some exercises available in Huxley were solved during the class. The professor strongly encouraged all students to submit solutions for the problems available in Huxley during extra-class activities, once there is absolutely no penalty in case of wrong submitted solutions. However, during the first 30 days, the use of Huxley was mandatory for approximately 6 exercises (used as a small percentage of the final grade) and for the first formal exam (held closely to the 30th day).

The programming language was C and the program curriculum for the first 30 days (22% of the semester) was the following: *introduction to algorithms; hello world in C; variables; identifiers; constants; basic structure of a C program; input/output; operators; data types; assignments; and conditional structures.*

4.2. Procedure

Figure 2 illustrates how we perform our evaluation. The result of executing our strategy consists of groups of students according to the clustering algorithm. Now, according to the groups, we have the potential failing students (see the left-hand side in Figure 2). These students are the ones that have a small number of submissions and correct submissions (represented by circles). Notice we also have students that are potential candidates to successfully pass (represented by stars): due to the high number of submissions to Huxley after 30 days, they seem to be practicing programming really hard. We represent the inconclusive group by squares.



Name	Grade
Emma Wilson	E ✗
David Miller	D ✗
John Smith	B ✓
Mary Johnson	D ✗
Peter Garcia	D ✗

Figure 2: Checking the strategy results against the actual grades. "After 30 days" represents the graphic label. The graphic is based on two dimensions, one for each metric we use.

After applying the strategy, we now need to check whether it correctly predicts the failing students after 30 days. To do so, we use the academic system of the Federal University of Alagoas to look for grades and check whether the students failed or not. For example, for the detached circles, the strategy successfully identified that, after 30 days, two students would not pass and they indeed did not. Notice, however, that we may face false positives. Despite indicating the student *John Smith*[2] as a failing one after 30 days, such a student seemed to improve himself during the semester and he has been approved. In addition, we may also have false negatives. For instance, our strategy does not point the students *Emma Wilson* and *David Miller* as failing ones. Although *Emma Wilson* seems to be one of the best students after 30 days, she failed the course. The several reasons why this happened is out of the scope of this article.

[2] All names we consider in this article are fictitious.

To better structure and analyze our results and, at the same time, take false positives and false negatives into account, we consider the following two metrics: precision and recall. Precision is the fraction of retrieved students that are relevant, i.e., the students pointed by the strategy that indeed failed the course. We have a perfect precision, 1.0, when every student retrieved by the strategy is relevant (i.e., a failing student), which means we have no false positives. Precision focuses on *quality* and *accuracy*. However, the precision says nothing about whether *all* relevant students were indeed retrieved.

$$Precision = \frac{|\{relevant\_students\} \cap \{retrieved\_students\}|}{|\{retrieved\_students\}|}$$

Recall, in its turn, is the fraction of relevant students that are retrieved, i.e., it is the probability of retrieving a failing student. A perfect recall, 1.0, means that we retrieve all failing students, which means we have no false negatives. In this context, recall focuses on *completeness* and *quantity*. Notice that recall says nothing about how many irrelevant students (students that will pass) the strategy retrieved.

$$Recall = \frac{|\{relevant\_students\} \cap \{retrieved\_students\}|}{|\{relevant\_students\}|}$$

To better explain these metrics, consider the detached students in Figure 2 as our set (three circles, one square, and one star). The relevant set is {*Emma, David, Mary, Peter*}, whereas the retrieved set is {*John, Mary, Peter*}. The intersection set is {*Mary, Peter*}. This way, we have

$$Precision = \frac{|\{Mary, Peter\}|}{|\{John, Mary, Peter\}|} = 67\%; \quad Recall = \frac{|\{Mary, Peter\}|}{|\{Emma, David, Mary, Peter\}|} = 50\%.$$

Our strategy pointed two out of three students as failed ones and they indeed failed. Therefore, we have 67% of accuracy when identifying potential failing students, raising one false positive. On the other hand, only two out of four students have been pointed as failed ones. This means that the strategy was not able to identify all failing students, raising two false negatives.

Last but not least, after confronting the strategy results with the academic system and summarizing precision and recall, we apply a statistical test to check for significance. Here, we rely on the proportion statistical test based on the Bernoulli distribution [20] so that we have a binary distribution: fail or pass. In this article, we follow the convention of considering a factor as being significant to the response variable when *p-value* < 0.05 [21].

5. Results and Discussion

In this section, we describe the results and test our hypotheses before discussing their implications. All data, materials, and R scripts [22] we use are available at `http://www.ic.ufal.br/`.

5.1. Results

In our evaluation, we use data of 7 courses (3.5 years; 7 semesters) totalling 227 students. We apply our strategy by setting k-means to compute two and three groups. We now proceed separately, reporting the results considering both cases.

5.1.1. Two groups

When considering two groups, we set the strategy to consider all students in two categories: fail or pass. Figure 3 illustrates the results for all courses. Notice that Figure 3(b) contains an outlier. In this particular case, the strategy pointed that all students but one would fail the course. Clearly this result is a consequence of such outlier. This way, the presence of one outlier might represent a problem when considering two groups.

To better analyze our results, we remove this outlier and execute our strategy again. By using two groups and properly removing the outlier, we achieve the following results for precision and recall:

$$Precision = \frac{92}{145} = 63\%; \qquad Recall = \frac{92}{115} = 80\%.$$

Here we observe a high recall, i.e., 80%. This means we only miss 20% of the failing students. We have few false negatives, but this result says nothing about how many false positives (irrelevant students) we retrieved. However, notice that this result (80%) represents our particular sample. We now need to find the population proportion, enabling us to generalize our findings. To do so, we need to check for statistical significance by executing the proportion hypothesis test. In this context, the test reveals that the population recall is 73% with a confidence level of 95% ($p\text{-}value = 0.045$). Regarding precision, our results show a sample precision of 63%. To generalize, we again execute the test and find that the population precision would be 56% ($p\text{-}value = 0.035$).

5.1.2. Three groups

Analogously, we set our strategy to consider three groups. Here, besides the failing and passing groups, there is one extra group where the strategy cannot conclude anything about it. Figure 4 shows the results for three groups. Differently from two groups, here one outlier does not completely compromise the strategy.

We present the precision and recall for three groups in what follows:

$$Precision = \frac{73}{91} = 80\%; \qquad Recall = \frac{73}{115} = 63\%.$$

Now the strategy returns a precision of 80% for the sample proportion. To generalize our results, we again execute the proportion hypothesis test. Regarding the population, our results reveal that the precision is 72% with a confidence level of 95%. In other words, from the set we retrieve, we can identify with statistical significance ($p\text{-}value = 0.04$) 72% of the students that indeed will fail the course after 30 days. We repeat this process for recall and find 63% for the sample and 55% for the population, $p\text{-}value = 0.033$.

Notice that with three groups the results of precision and recall are inverted when compared to two groups. Our strategy uses k-means, which takes into account two metrics—number of submissions and number of correct submissions—and the number of groups. Both executions are totally independent and the algorithm is not aware of precision, recall, and the academic system. Therefore, we take these inverted results as a coincidence.

5.2. Discussion

In this section we discuss the results. We first discuss the number of groups we should set when using our strategy and then we discuss our achievements regarding final exams, number of enrolled students, and applying one exam right in the 30th day.

5.2.1. Two or Three Groups?

When applying our strategy considering two groups, the results indicate a higher recall when compared to three groups. This means that the strategy can identify the majority of the failing students (raising only few false negatives). Although this is an interesting result, the strategy with two groups yields many false positives. In fact, the precision is lower when compared to three groups, i.e., the set we retrieve contains many students that pass. In this situation, professors and mentors might waste effort trying to recover students that actually do not need recovering. On the other hand, with three groups we have a higher precision, which means professors should give special attention to the students the strategy retrieves, since 72% of them tend to fail. Nevertheless, since the recall is lower than with two groups, we have more failing students not retrieved by the strategy for three groups.

In this context, setting the number of groups to execute our strategy seems to depend on the professors priorities and resources. In case the professor has available time and additional mentors to help her, she can probably use two groups, which consists of a more complete retrieved set (the recall is higher for two groups), even though it contains many false positives. However, notice that applying the strategy with two groups is more likely to outliers problems, such as the one we depict in Figure 3(b).

On the other hand, if the professor wants to avoid false positives because of no available time or few mentors to help, she might prefer to use three groups, despite being aware of potential failing students not retrieved by the strategy (many false negatives, lower recall than with two groups).

5.2.2. Final Exams

Even though the strategy pointed students as potential failing ones in the first 30 days, some of them passed, raising false positives. Table 2 illustrates the precision for two and three groups as well as the false positives. For two and three groups, the precision is 56% and 72%, respectively. This way, we have 44% and 28% of false positives. As mentioned, the number of false positives is greater when considering two groups.

Groups	**Precision**	**False Positives**	**Final Exam**
2	56%	44% (53 students)	35% (19 out of 53 students)
3	72%	28% (18 students)	33% (06 out of 18 students)

Table 2: False positives (students pointed as potential failing ones but passed) that performed the final exam.

To better understand these false positives, we now analyze their grades at the academic system. In our study, we find that some of these students did not pass in the first place. In fact, they needed to perform the final exam to pass. At the university we focus on this article, the final exam represents a second chance and is only available to students with not enough grades to pass.

This result is important in the sense that, although the strategy might raise false positives, it seems worth to follow these students as well. We find that at least 33% of them need help (regardless of the number of groups, two or three), otherwise they reach the final exams. This way, professors and mentors can also give special attention to these students, which may improve their learning process, avoid final exams, and lead to better grades.

5.2.3. Back to our objective

As mentioned, the objective of our study is "to evaluate to what extent our strategy is capable of identifying potential failing students in only 30 days." Given the results we achieve, we believe our strategy is indeed capable of predicting the failing students early. In particular, from the group of students our strategy points as "likely to fail," 72% of the students indeed fail with 95% standard confidence level by using three groups.

5.2.4. Number of students

The efficacy of our strategy can strongly depend on the number of enrolled students in the course. In this context, if the number of students is small (e.g., 10 students), the own professor can identify—also early—the students that need help. This scenario is even more promising when the professor counts on mentors to help. Therefore, here our strategy adds little and professors might not notice the benefits of using it.

On the other hand, despite having courses throughout Brazil with few students, it is common to find courses with 30, 40, 50, or even 60 enrolled students. In USA, for instance, this number can be even larger at some universities. In this context, previous studies [4] considering 63 institutions report introductory programming courses with more than 600 students! The average class size is 116, but only 23% of the courses have less than 30 students [4]. In addition, they found that small classes (with less than 30 students) perform better regarding pass rates than larger ones.

When considering larger classes, the task of predicting the potential failing students becomes more difficult, time consuming, and error prone. Therefore, an automatic approach like our strategy can better support professors and their mentors. Moreover, once professors identify such students earlier, they may have better chances of helping the students before it is too late. So, the application of our strategy can also contribute to the dropout rate reduction. Nevertheless, we need further studies to better support this claim.

5.2.5. Applying the Strategy versus Applying Exams

One might wonder what is the difference between applying our strategy and applying a formal exam after 30 days. In this case, students with bad grades are candidates to fail the course. However, notice that one-day exams represent one specific chance to the student. So, it is not uncommon to confirm that this kind of exam often does not truly evaluate the students knowledge/skills. In this context, applying many exams can minimize such a problem. Nevertheless, when considering the professor perspective, grading many programming exams and reasoning at the same time about the potential failing students are time-consuming and error-prone tasks [17], increasing effort.

In contrast, by using online judges, professors might evaluate the students continuously. This way, they might better identify the students deficiencies automatically. In addition, online judges allow students to solve many problems (either as ordinary exercises or as formal exams), instead of only a couple commonly found in one-day exams. Here, many exams do not necessarily impact negatively on the professors effort.

Our strategy represents a feasible way to identify potential failing students early. We perform the identification in an automatic way, reducing effort of professors and mentors. In addition, our results reveal that the strategy can identify such students with good precision, minimizing false positives.

5.3. Threats to validity

In this section we present the threats to validity of our study. Although our sample is reasonably big (227 students), our study has homogeneities that might pose threats. For example, the same professor for all the 7 courses (at the same university) and the same language used (C language) threats external validity. In this way, it is difficult to extrapolate the results to other contexts. Nevertheless, our strategy uses metrics to identify potential failing students. In this context, we argue that the metrics we use (submissions and correct submissions) do not necessarily depend on factors such as the professors and even less on the adopted languages. However, our claim is not enough and we need further studies to better generalize our conclusions.

The use of Huxley threats internal validity. Students must know how to use the system to submit their solutions. If the students somehow do not get used to the system, they might feel frustrated, hindering their learning and, consequently, biasing our results. We minimize this threat by introducing Huxley in the very first classes as well as by using mentors to help the students on how to use Huxley.

6. Related Work

Previous studies introduce aptitude tests to check whether a student will succeed or fail programming courses. To do so, some studies rely on past academic achievements. For example, Butcher et al. [7] use high school data and ACT scores to predict college performance in computer science courses. However, their goal is to predict students who had successfully completed one year of study in computer science in general and not only in computer programming. Besides, they use data that are only available at the United States, then it is difficult to replicate to other countries. Similarly, Hostetler [6] uses the college grade point average and math background to perform the predictions.

Another category of predictors rely on surveys to better understand the student skills. Hughes et al. [9] introduces a survey based on many math and logical questions. Simon et al. [8] describe a multi-institutional study to predict success in introductory programming courses. To do so, they set a number of diagnostic tasks (such as map sketching and phone book searching) and perform a qualitative analysis based on short interviews. Dehnadi et al. [2] observe the mental models that students use when reasoning about simple assignments and sequence of assignments.

Although the surveys present promising results, applying and replicating these studies is difficult, time consuming, and error prone, specially when considering large class sizes. Differently, the strengths of our study include an automatic predictor of students candidates to fail. Because it is automatic, this reduces significantly the effort of professors interested in predicting such candidates. Despite achieving 72% of precision, we focus only on one programming language and we use the same professor in all 7 courses at the same university, differently from Simon et al. [8] that use data from 11 institutions from three countries.

Lorenzen et al. [10] presents a game-based predictor. They claim it is possible to predict failing students by only making them to play the game (named MasterMind). Unfortunately, the given webpage in the paper is not available anymore, so it is hard to set and replicate the study to better support the claims of the paper. Besides, the game does not give any clues to professors and mentors regarding the deficiencies of each student. Online judges, on the other hand, can help on this task, since it is possible to map particular subjects from the curriculum to each problem available in the judge. So, professors can identify that a particular set of students have difficulties in loops, for instance.

Another predictor consists of introducing an assembly-based language designed in a such way that students do not need to have any programming background [11]. This study correlates the results of tasks implemented in such a language with the midterm exam. They validate their study with 23 students and find a good correlation between the tasks and the midterm exam. In contrast, our evaluation focuses on the first 30 days (22% of the course) and use 7 semesters, totalling 227 students. Therefore, we are able to predict earlier, giving better chances for professors and mentors to recover the students. Like the MasterMind game [10], the given webpage in the paper is no longer available (despite being a very recent work), which makes the replication of the study more difficult.

7. Concluding Remarks

This article presented a strategy able to early predict potential failing students in introductory programming courses. The strategy is automatic, reducing effort and allowing professors to use it in their courses. We focused on the identification, rather than on approaches to avoid the failings, e.g., changing the programming language. Our strategy consists of three simple steps: the use of an online judge system, the collection of metrics from this system; and the application of a clustering algorithm. In an empirical study using 7 courses (representing 3.5 years) with freshmen students, the results suggest that our strategy can early detect (within only 30 days) the failing students, which is a promising result. In particular, from the group of students our strategy points as "likely to fail," 72% of the students indeed fail with 95% standard confidence level. We also found that the remaining set of 28% of students that passed is still interesting: at least 33% of them had difficulties to pass and reached the final exam. This way, some of these students need help as well. In case this happens, they may avoid final exams and achieve better grades at the end of the course. We also concluded that the efficacy of our strategy depends on the number of students enrolled in the course. In case the class size is short (e.g., 10 students), the strategy adds little, since the own professor can identify the failing candidates.

As future work, we intend to use other metrics and clustering algorithms to better define and analyze our strategy. Additionally, by using the results of our strategy during actual semesters, we intend to make professors and mentors aware of the potential failing students. Then, we are ready to evaluate whether the strategy application can somehow help on reducing the high rate of failing students in introductory programming courses.

8. Acknowledgments

We would like to thank CNPq, a Brazilian research funding agency, for partially supporting this work. Ribeiro's work was partially supported by RHAE 453716/2013-0.

9. References

[1] A. Yadin, Reducing the Dropout Rate in an Introductory Programming Course, ACM Inroads 2 (4) (2011) 71–76.

[2] S. Dehnadi, R. Bornat, The camel has two humps - (working title).

[3] L. Porter, M. Guzdial, C. McDowell, B. Simon, Success in Introductory Programming: What Works?, Communications of the ACM 56 (8) (2013) 34–36.

[4] J. Bennedsen, M. E. Caspersen, Failure Rates in Introductory Programming, SIGCSE Bull. 39 (2) (2007) 32–36.

[5] P. Kinnunen, L. Malmi, Why Students Drop out CS1 Course?, in: Proceedings of the Second International Workshop on Computing Education Research, ICER, ACM, 2006, pp. 97–108.

[6] T. R. Hostetler, Predicting Student Success in an Introductory Programming Course, SIGCSE Bull. 15 (3) (1983) 40–43.

[7] D. F. Butcher, W. A. Muth, Predicting Performance in an Introductory Computer Science Course, Communications of the ACM 28 (3) (1985) 263–268.

[8] Simon, S. Fincher, A. Robins, B. Baker, I. Box, Q. Cutts, M. de Raadt, P. Haden, J. Hamer, M. Hamilton, R. Lister, M. Petre, K. Sutton, D. Tolhurst, J. Tutty, Predictors of Success in a First Programming Course, in: Proceedings of the 8th Australasian Conference on Computing Education, ACE, Australian Computer Society, Inc., 2006, pp. 189–196.

[9] J. L. Hughes, W. J. McNamara, IBM Programmer Aptitude Test, http://ed-thelen.org/comp-hist/IBM-ProgApti-120-6762-2.html (October 2014).

[10] T. Lorenzen, H.-L. Chang, MasterMind: a predictor of computer programming aptitude, SIGCSE Bulletin 38 (2) (2006) 69–71.

[11] J. Harris, Testing Programming Aptitude in Introductory Programming Courses, Journal of Computing Sciences in Colleges 30 (2) (2014) 149–156.

[12] J. A. Hartigan, Clustering Algorithms, 99th Edition, John Wiley & Sons, Inc., 1975.

[13] UVa Online Judge, `http://uva.onlinejudge.org/` (October 2014).

[14] Sphere Online Judge (SPOJ), `http://www.spoj.com/` (October 2014).

[15] T. Jenkins, On the Difficulty of Learning to Program, in: 3rd annual Conference of LTSN-ICS, 2002, pp. 53–58.

[16] T. Wang, X. Su, P. Ma, Y. Wang, K. Wang, Ability-training-oriented Automated Assessment in Introductory Programming Course, Computers & Education 56 (1) (2011) 220–226.

[17] B. Cheang, A. Kurnia, A. Lim, W.-C. Oon, On Automated Grading of Programming Assignments in an Academic Institution, Computers & Education 41 (2) (2003) 121–131.

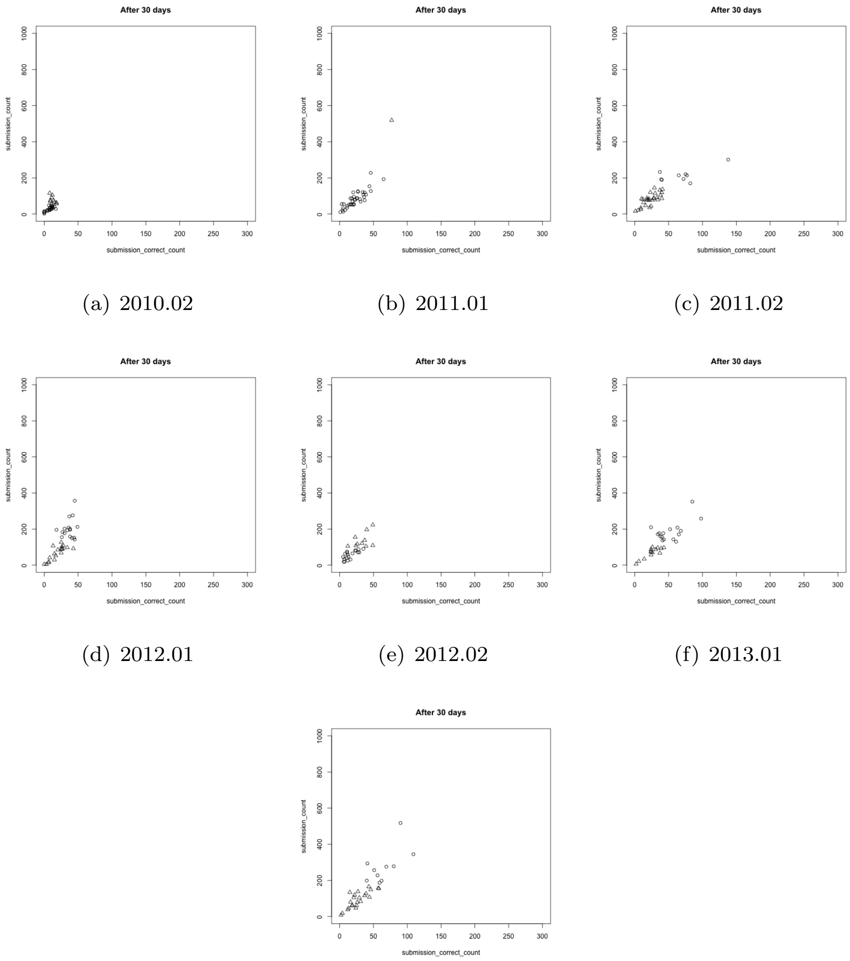
[18] J. A. Hartigan, M. A. Wong, Algorithm AS 136: A K-Means Clustering Algorithm, Journal of the Royal Statistical Society. Series C (Applied Statistics) 28 (1979) 100–108.

[19] R. d. B. Paes, R. Malaquias, M. Guimarães, H. Almeida, Ferramenta para a Avaliação de Aprendizado de Alunos em Programação de Computadores, in: Anais dos Workshops do Congresso Brasileiro de Informática na Educação, Vol. 1, 2013, pp. 203–212, in Portuguese.

[20] P. McCullagh, J. Nelder, Generalized Linear Models, 2nd Edition, Chapman and Hall/CRC, 1989.

[21] G. E. P. Box, J. S. Hunter, W. G. Hunter, Statistics for Experimenters: Design, Innovation, and Discovery, Wiley-Interscience, 2005.

[22] The R Project for Statistical Computing, `http://www.r-project.org/` (October 2014).



(a) 2010.02

(b) 2011.01

(c) 2011.02

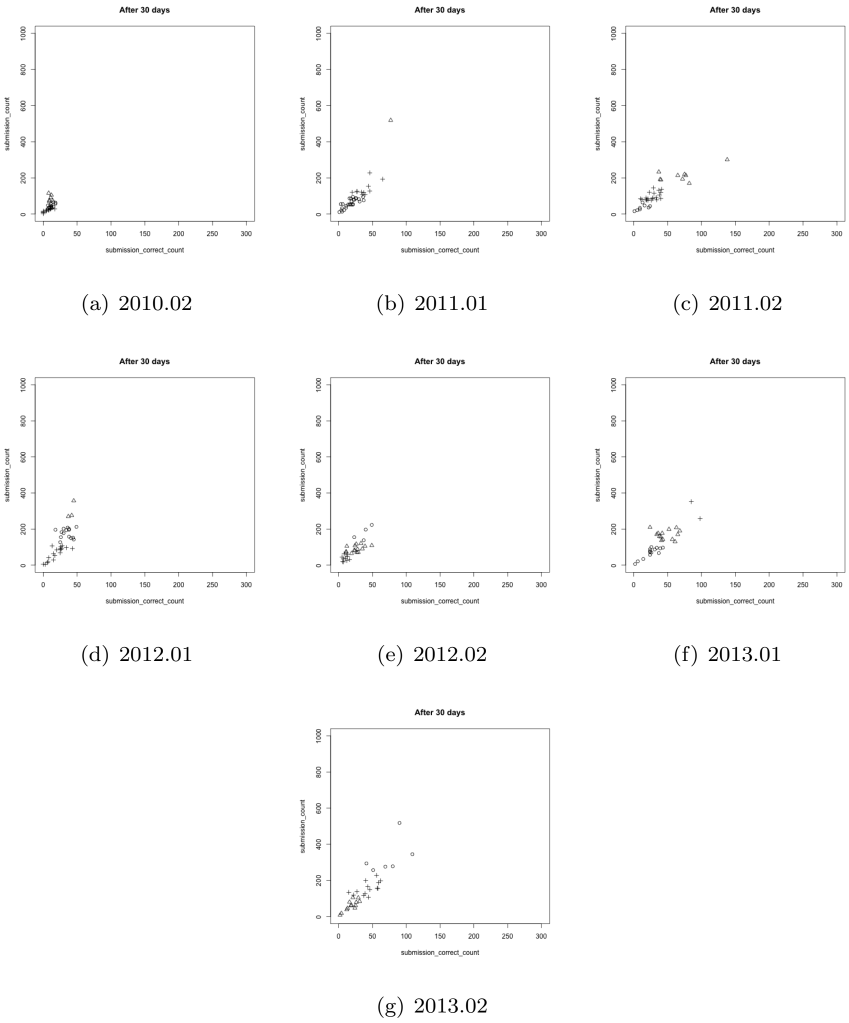
(d) 2012.01

(e) 2012.02

(f) 2013.01

(g) 2013.02

Figure 3: Strategy applied with two groups.



(a) 2010.02

(b) 2011.01

(c) 2011.02

(d) 2012.01

(e) 2012.02

(f) 2013.01

(g) 2013.02

Figure 4: Strategy applied with three groups.