Chang, M. et al. (Eds.) (2019). Proceedings of the 27th International Conference on Computers in Education. Taiwan: Asia-Pacific Society for Computers in Education

# Analysis of Student Behaviors in Programming Exercises in Controlled and Natural Environments

**Thomas James TIAM-LEE[*] & Kaoru SUMI**
*Future University Hakodate, Japan*
*g3117002@fun.ac.jp

**Abstract:** We performed an analysis of the behaviors of novice students while solving programming exercises from data collected in two environments: a controlled laboratory setup and an online system that could be used freely in the wild. We modeled student behavior from raw action sequences using hidden Markov models to capture the sequential information of the problem solving process. We found similar sequential structures between the two environments, with students generally starting from being idle, followed by writing of the code, followed by testing and submission, which may then transition back to an idle state if not successful. While the models were similar, we found evidences of less persistence in solving the problems on students using the online system compared to those in the controlled setup.

**Keywords:** Student modelling, learning analytics, programming

## 1. Introduction

Tutoring is considered to be one of the most effective ways to learn (Bloom, 1984). Because of this, there has been significant research interest in intelligent tutoring systems (ITS) that can replicate aspects of human tutoring. A core part of these systems is being able to keep track of a student model, so that the system is able to respond in a personalized way to the needs of the student. However, many ITS are developed and evaluated on data from controlled laboratory experiments, and it is often difficult to determine if findings are reproducible in natural learning environments.

A recent study on ITS design has found that findings from a controlled laboratory experiments contradicted with those of a same experiment performed in a more natural setting (Kumar, 2019). This is of particular interest to researchers because it is important for findings to be reproducible in order to have a real impact in solving societal problems (Drummond, 2009). Thus, there is value in understanding the differences between controlled and natural settings in various domains.

In this study, we focused on the specific learning domain of programming. In our previous studies, we developed a simple intelligent programming tutor that offers guides and adjusts problems based on the presence of confusion on the student (Tiam-Lee & Sumi, 2018) and made classifier models to predict various student emotions in programming from face features and logs (Tiam-Lee & Sumi, 2019). In these studies, datasets were collected from and evaluated on controlled laboratory setups. Our motivation for this paper is to take a step towards understanding the similarities or differences in how students interact with such systems in controlled and natural programming environments. To achieve this, we modeled student solving behaviors from raw action sequences using hidden Markov models using data from a controlled laboratory setup and a natural online system that could be used freely by students. We present a comparison of the models of the student learning behaviors between the groups, and discuss its implications in future research.

## 2. Data Collection Methodology

In this section, we describe the two programming environments from which we collected our data for analysis. The first environment followed a controlled laboratory setup, while the second environment was an online system that was used by students freely.

### *2.1 Environment 1: Controlled Laboratory Setup*

For the controlled laboratory setup, we used data collected from an observational setup. We recruited 73 students who were all enrolled in a university introductory programming class at the time. Each student was asked to participate in a simulated programming session in which they must solve several coding exercises while a video recording of their face and their activity logs were saved for analysis.

Before the session, each student underwent a briefing phase in which details of the data collection process was explained. The student was also informed that a video camera would be recording their face and all of their activity logs would be saved throughout the duration of the coding session. Thus, in this environment, each student was aware that his or her actions would be observed and analyzed. Each student was asked to sign an informed consent form, indicating that he or she agreed to participate in the session voluntarily.

After the briefing, the student started the programming phase. In this phase, he or she must solve coding exercises. There are a total of nine exercises that increased in difficulty. The exercises covered introductory programming concepts such as variables, expressions, if-else statements, loops, and arrays. All students used the same set of exercise. In each exercise, the student must write the body of a function that performs a specified task. For example, one of the exercises required the function to return the area of a square, given the measurement of one of its sides as an argument.

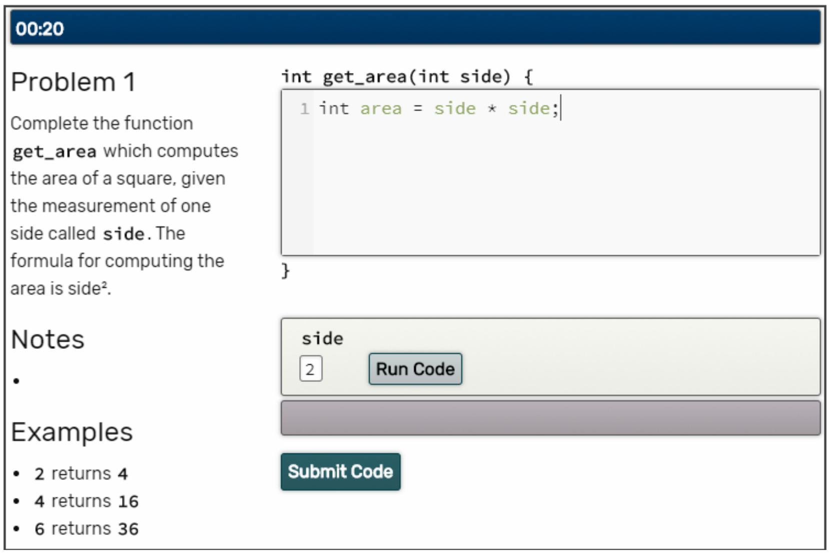
A custom local application was developed to serve as an interface for the students to write their programs. This application provided an interface for the students to view the exercise specifications, write the code, test the code by providing user arguments, and submit the code for checking. Submitted codes can automatically evaluated by running it on a set of pre-defined test cases. In addition to these, the application also records a video of the student's face and saves all activity logs, which included information on document changes, compilations, and submissions. A screen shot of the application is shown in Figure 1. Each student used the system for 45 minutes or until all exercises were solved correctly. Exercises must be solved sequentially with no option to skip.

### *2.2 Environment 2: Online System in the Wild*

To represent data coming from an uncontrolled setup, we used data collected from an online system for programming practice that we have deployed for freshmen students as a supplementary tool for their introductory programming class. The data was collected over a month in the latter part of the semester, from June 2018 to July 2018. The online system was introduced to all students enrolled in the introductory programming class of Future University Hakodate in that month as an online tool for them to hone their programming skills. None of the students were enforced to use the system, so all usage was on a voluntary basis. 96 unique users used the system in total by the end of the period.

Each student was given a unique account that they could use to log into the online system. The online system worked very similarly to the programming session application used in the controlled observational setup discussed in the previous section. In this system, students could solve exercises in which they have to write the body of a function that performs a specified task. The system provides an interface to view the exercise specifications, write code, test code by providing user arguments, and submit code for checking. Submissions are automatically checked as well.

Students could select from various categories of exercises in this system, but all of them follow the same format of writing the body of a function to satisfy specifications. When a user successfully solves a problem, he is taken back to the selection menu and a new one could be selected. Unlike the controlled setup, the student could at any time give up on a problem and select a different one if he wished to. As can be seen in a screen shot of the online system showing the interface for performing the programming exercise in Figure 2, the online system shares an almost identical interface with the application used in the controlled setup. Unlike the controlled setup, the users' video recordings were not saved and they were not conscious their system actions were being logged.


00:20

Problem 1

Complete the function `get_area` which computes the area of a square, given the measurement of one side called `side`. The formula for computing the area is side².

```
int get_area(int side) {
int area = side * side;
}
```

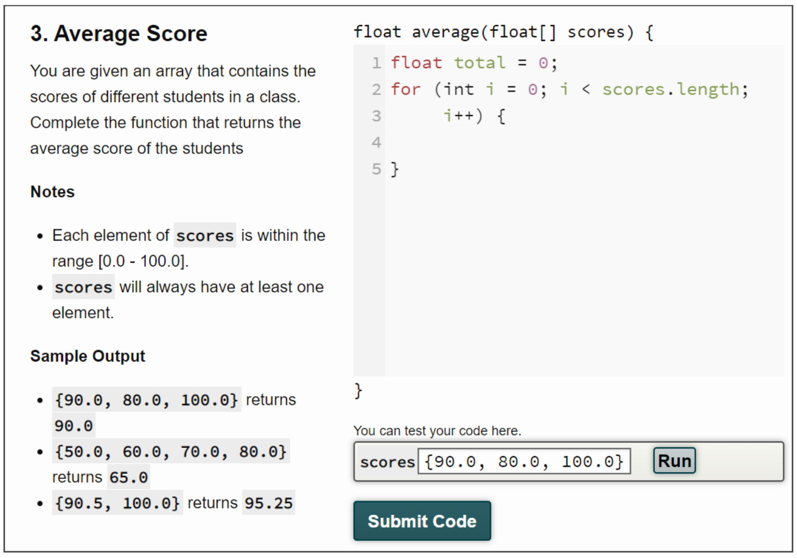
Notes

•

Examples

- 2 returns **4**
- 4 returns **16**
- 6 returns **36**

*Figure 1.* Screen shot of custom application for controlled laboratory environment setup.


**3. Average Score**

You are given an array that contains the scores of different students in a class. Complete the function that returns the average score of the students

**Notes**

- Each element of `scores` is within the range [0.0 - 100.0].
- `scores` will always have at least one element.

**Sample Output**

- `{90.0, 80.0, 100.0}` returns `90.0`
- `{50.0, 60.0, 70.0, 80.0}` returns `65.0`
- `{90.5, 100.0}` returns `95.25`

```
float average(float[] scores) {
float total = 0;
for (int i = 0; i < scores.length;
     i++) {

}
}
```

You can test your code here.

scores `{90.0, 80.0, 100.0}`

*Figure 2.* Screen shot of online application in the natural environment setup.

## 3. Modelling Student Behavior

In this section, we discuss how we modeled the student behaviors in solving programming exercises. We chose a hidden Markov model (HMM) in order to capture the time-sequential nature of coding tasks, which involve actions such as modifying the code, thinking, and compiling to test the program.

### *3.1 Hidden Markov Model*

A hidden Markov model is a statistical Markov model that can be used to describe observable events and hidden events. Observable events are a set of directly observable occurrences in the data, while the hidden events are hidden factors that are considered to be casual factors for the probabilistic model. An HMM is composed of a set of states $Q = \{q_1, q_2, \ldots, q_N\}$, with each state having an emission probability for each observation $O = \{o_1, o_2, \ldots, o_T\}$. In this case, the observations are the individual actions in the Markov chain, while the states are the hidden factors that cause such actions.

As the HMM is based on a Markov chain, it follows the Markov assumption that the probability of a particular state $q_i$ depends only on the previous state. Furthermore, the probability of an observation depends only on the current state and not in any other states or observations in the sequence:

$$P(q_i \mid q_1, ..., q_{i-1}) = P(q_i \mid q_{i-1})$$

$$P(o_i \mid q_1, ..., q_i, ..., q_L, o_1, ..., o_i, ..., o_L) = P(o_i \mid q_i)$$

The HMM also contains a transition probability matrix containing the probabilities of transitioning from one state to another, and an initial probability distribution for which state the model starts in. The transition probabilities and emission probabilities can be estimated from a set of training data where each instance in the set is a sequence of observations from O using an algorithm called the Baum-Welch algorithm. Given an HMM, it is also possible to compute the likelihood that a given observation sequence is generated by the model using the forward algorithm. A good summary of hidden Markov models can be found in Appendix A of Martin & Jurafsky (2009).

## *3.2 Sequence Extraction from Activity Logs*

We represented student coding behaviors as a sequence of actions that are performed in the process of solving a coding exercise. We extracted these actions from the activity logs collected from the programming environments. All events were marked with a timestamp. Figure 3 shows an example of an action sequence extracted from the activity logs.

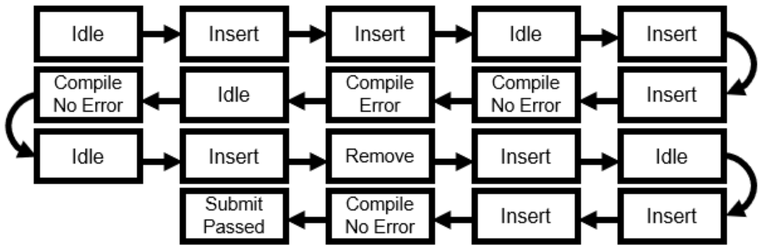


*Figure 3.* Example action sequence from student log data.

Compilations represent points in the session where the student tested the code by providing some arguments. A compilation may result into a successful compilation (i.e., no syntax errors) or an unsuccessful compilation (i.e., with syntax errors). A successful compilation did not necessarily mean that the code was correct, as it was possible to have a syntactically correct code that did not accomplish the target task. Submissions, on the other hand, represent points in the session where the student attempted to submit the code for checking. This may result into either a “submission passed” (i.e., the code is correct and passed all test case) or a “submission failed” (i.e., code failed at least one test case).

Typing information is captured with the “insert” and “remove” actions. We considered a typing sequence as a series of document changes with intervals of at most 3 seconds in between. Typing action was classified as “insert” or “remove” based on how many characters were inserted and deleted within the typing sequence. If there were more characters inserted, we considered the action as the student adding more to the code. If there were more characters deleted, we considered the action as the student removing parts of the code. Ties are resolved in favor of the “insert” classification. Although this metric has limitations as it does not capture the semantic information of the code, it is a good enough heuristic for the general type of action being performed in this analysis.

Finally, an “idle” action is added when the student did not perform any actions in the system for at least 10 seconds. These actions represented long pauses in the action sequence, which may represent the student thinking, reading the problem, or even being unfocused. Only a single “idle” action is added to the sequence regardless of the length of inactivity.

## 3.3 *Model Training*

We trained an HMM with a fully-connected structure for each of the two datasets (laboratory-controlled and natural environments) using the Baum-Welch algorithm. Our HMM model is defined by the set of states Q = {$q_1$, $q_2$, ..., $q_N$} and the set of observations O = {idle, insert, remove, compile error, compile no error, submit failed, submit passed}. The average number of observations in each sequence is 39.09 in the laboratory environment and 11.83 in the online environment.

We selected the number of hidden states N based on a ten-fold cross validation that maximizes the likelihood computed using the forward algorithm. We first divided the set of observation sequences into ten groups. In each fold, one group served as the validation set while the others served as the training set. We trained an HMM using the training set from randomly initialized values. To account for the randomness, each training process was repeated 5 times, each time having different random values, and the model that maximizes the probability of the training set to be generated by the model was selected. Once the model was trained, we computed the likelihood of the validation set being generated by the model, and got the average across all the folds. We computed this for N = 2, 3, ..., 10 hidden states and selected the best number of states based that information. Figure 4 shows the mean log-likelihood score across folds across different values of N.

We selected N = 5 as the number of hidden states for both the laboratory and online datasets since the improvement in the log-likelihood significantly decreases beyond N = 5. We generally prefer to have less states for easier visualization and understanding of the model. From this, we estimated the model parameters from the training data using N = 5 using the Baum-Welch algorithm.

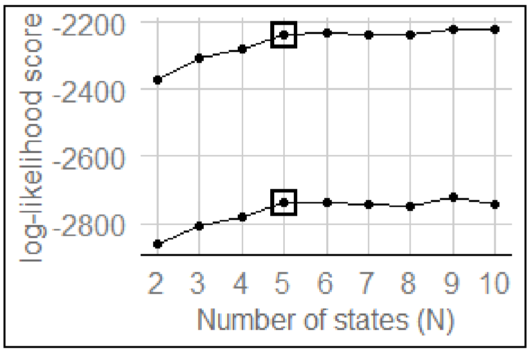


*Figure 4.* Log-likelihood score for different values of N.

## 4. Results

In this section, we discuss the resulting HMMs trained from our datasets. Figures 5 and 6 show graphical representations of the HMMs trained from the action sequences extracted from the controlled laboratory environment setup and the online system natural environment setup, respectively. Each node in the graph represents a single state in the model. The directed edges represent the transition probabilities from one state to another. While an HMM normally has a non-zero probability to transition from every pair of states, only transitions with substantial probabilities over 15% are shown for clarity. In each state, the emission probabilities for each of the coding actions are shown when the model is in that particular state. All values are rounded off to 2 decimal places. We manually placed a loose label to describe each state based on the emission probabilities, shown on the lower-left corner of each node. This helps in getting a contextual understanding of the model as well as for easier discussion in this paper.

## 4.1 *HMM for Controlled Laboratory Setup*

The states in the HMM for the controlled laboratory setup appeared to represent different parts of the coding process. In the "mostly idle" state, the most probable action is being idle (72%). This state represents moments of inactivity in terms of interaction with the system. In the "building code" state, the most probable action is inserting characters into the code (80%), and likely represents moments where the student building modules of the code. On the other hand, the "modifying code" state contains a mixture of character insertions (65%) and removals (18%), with a small chance of idle (8%) actions. This state likely represents modifying parts of the code. The “testing” state has a mixture of compilations and typing actions. It could represent moments of testing the code, or finding and fixing bugs revealed by running tests on the code. Finally, the “testing and submitting” state has relatively high emissions for compilations and submission actions, which could represent moments of verifying the correctness of the code and submitting it.

The initial probability distribution for this HMM puts the probability of starting in the “mostly idle” state at 96%. This was expected, as the students mostly take the time to read the problem specifications first before writing any code. From the “mostly idle” state, students likely transition to the “building code” state. Students may move between the “building code” and “modifying code” states back and forth. Both “building code” and “modifying code” states are relatively persistent states, having a fairly large probability of spanning through several actions (i.e., transitioning to itself). From these states, the student may transition to the “testing” state and eventually the “testing and submitting” state, which represent actions wherein the students are evaluating the correctness of the code that they have built. There are also strong transitions from these states back to the idle state, likely because students take the time to process the result of a compilation or submission (e.g., thinking about why results were unexpected).

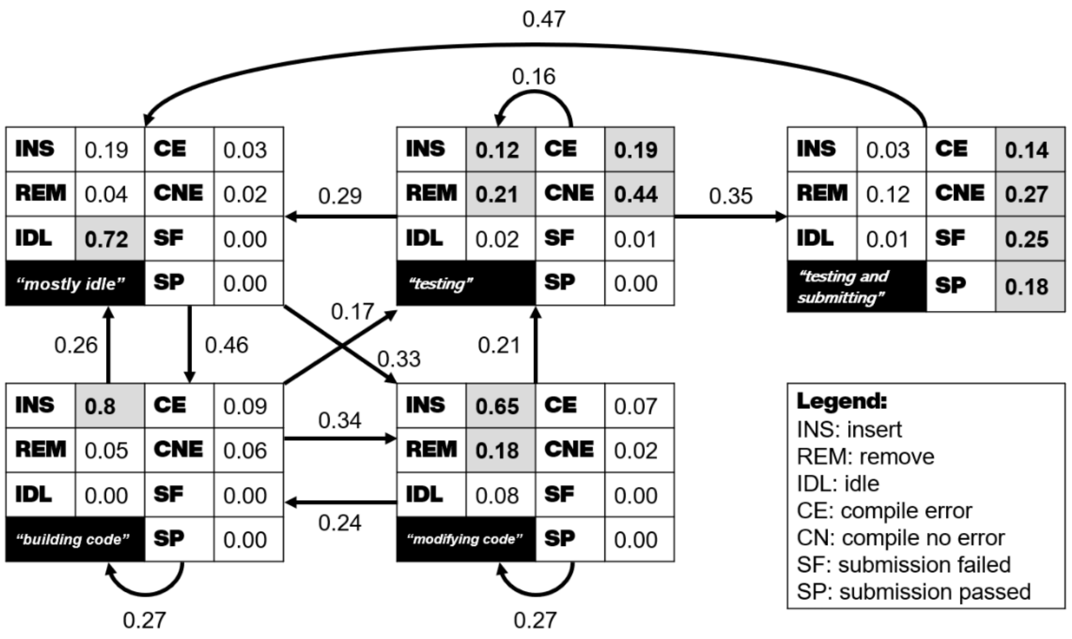


*Figure 5.* Hidden Markov model for online system natural environment setup showing emission probabilities and transition probabilities at least 15%. All values are rounded off to two decimal places.

## 4.2 *HMM for Online System in the Wild*

Interestingly, the HMM trained from the action sequences extracted from the online system dataset yielded similar states to the HMM trained from the laboratory setup, despite being both trained from random initial model parameters. One state appears to be a “mostly idle” state, with a high emission probability for the idle action (85%). There are two similar states, which we both labelled as “building code” states, in which high emission probabilities for character insertion could be observed (88% and 81%). There appears to be not as much removal actions in this model compared to the previous one, which suggests that modifications of the code did not occur as strongly as it did in the laboratory setup. Similar to the previous model, however, there was also a state that likely represented “testing” of the code, with emissions of compilations and typing actions, as well as a state the likely represented “testing and submission” of the code, with emissions for compilation and submission actions.

Similar to the laboratory setup, the initial probability distribution of this model has a high probability (71%) of starting on the “mostly idle” state. However, there was a fairly high chance (23%)

of starting on the "building code (2)" state as well. This suggests that unlike the laboratory setup, students don't take as much time reading the problem before starting to execute some typing actions.

While the states' emission probabilities suggest similar hidden events for both of the environments, one noticeable difference is on the transition probabilities of the model. While both "building code" states are similarly persistent to their counterparts in the laboratory setup model, there is a noticeable lack of strong transitions from these states to the "testing" state. This reveals that in the online system, there was weaker evidence found that students moved towards the testing and submission phase.

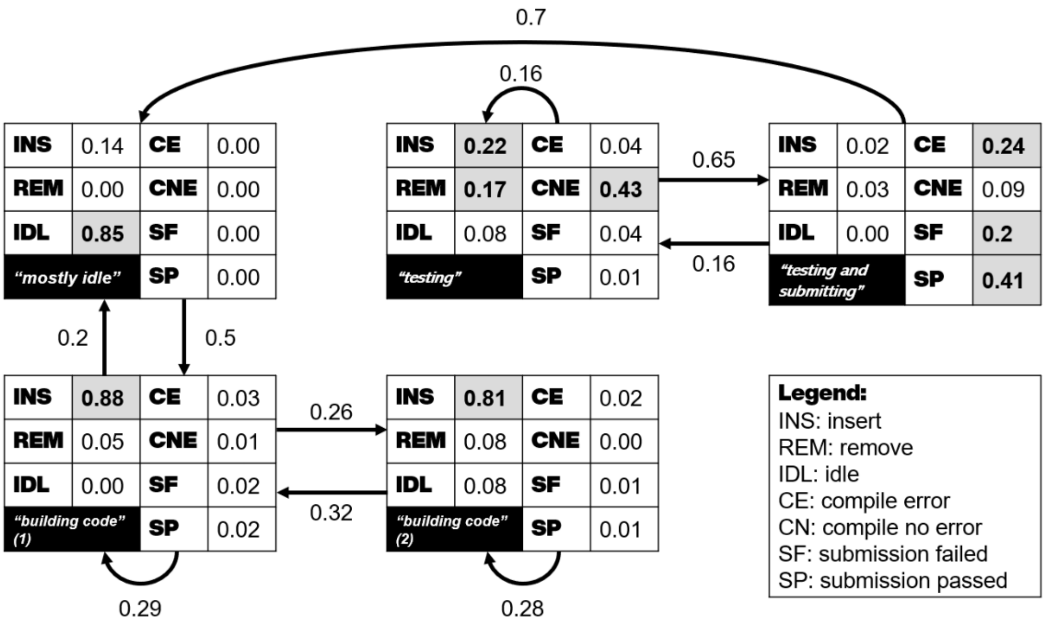


*Figure 6.* Hidden Markov model for online system natural environment setup showing emission probabilities and transition probabilities at least 15%. All values are rounded off to two decimal places.

## *4.3 Usage Statistics*

The online system was used in substantially more sessions than the laboratory setup (note that the number of participants in the latter is fixed). In the laboratory setup, we recruited a total of 73 students which yielded a total of 468 sessions. In this case, a session refers to a single problem attempted by a student. On the other hand, the online system was used by 96 unique users with a total of 2,259 sessions.

Although the HMMs capture sequential information of the coding sessions, it does not provide information on the length of the sequences. In order to further investigate the differences between the controlled laboratory and online system setups, we analyzed the number of actions in each sequence for each of the two environments. Figure 7 shows a side by side boxplot comparing the sequence lengths. For the controlled laboratory setup, the mean is 39.09 actions and the median is 19 actions, while for the online system the mean is 11.83 actions and the median is 7 actions.

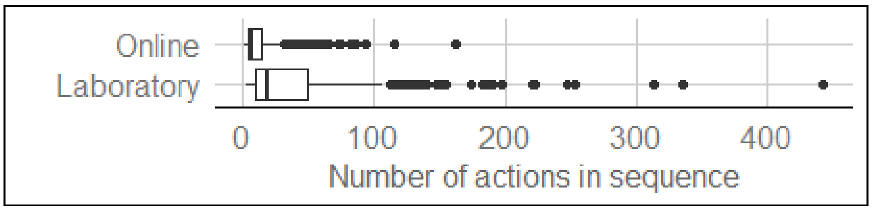


*Figure 7.* Number of actions in the sequences collected from the two environments.

Despite being used in more sessions, the number of actions performed in a single session was noticeably less in general in the online system compared to the controlled laboratory setup. This is also likely related to why there was a weaker transition to the testing states in the HMM for the online system, as students tend to give up and quit earlier resulting into generally shorter sessions.

Another interesting data to look at is the distribution of the number of sessions for each unique student that used the online system. This is shown in Figure 8. Majority of the users only used the system for one time or only a few times. On the other hand, there are fewer students used the system constantly and did a lot of sessions. This reveals differences on students' motivations to use a system in settings where they are not enforced to do so.

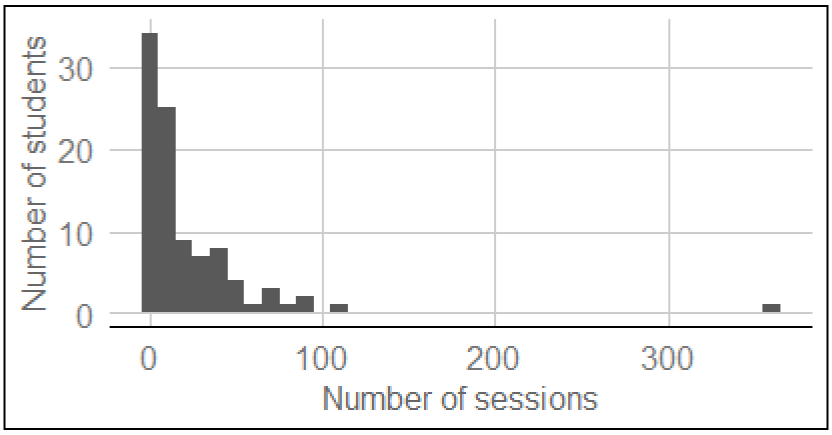


*Figure 8.* Distribution showing the number of sessions done by each unique user in the system.

## 5. Discussion

In this section, we discuss the main findings in this study and its implications in education-related research.

We have used HMMs to model freshmen students' coding behaviors while solving introductory programming exercises. These models can be used to gain an understanding of the general trajectories of actions undertaken by students when solving exercises. In the HMM models, the problem solving process were broken into logical states such as building the code, testing the code, and being idle. From the models produced, we could understand that a problem solving session usually starts with an idle state, followed by a moment of building the code, followed by a moment of testing or debugging, which may either result in a correct submission or move back to the idle state.

Such models can also have applications in artificial intelligence in education. For example, a series of actions could be mapped to sequence of states using an algorithm like the viterbi algorithm. The viterbi algorithm takes in an observation sequence as input and estimates the most probable sequence of hidden states that generated that sequence. Using this algorithm, contextual information could be estimated from raw action sequences, allowing for a better a student model. This contextual information could then be used to improve how intelligent programming tutors provide personalized feedback while the student is solving programming exercises. Currently, we only focused on action sequences and solving behaviors, but there is a possibility of using other modalities such as facial information to model emotions while doing programming exercises, thus allowing for the development of affective programming tutors that can respond emotionally to students without special and often invasive equipment.

In this study, we have also performed a comparison of students' coding behaviors between a controlled laboratory setup and an online system that could be used freely. Although there were many similarities between the two models, there were some transitions that were weaker in the online setup. In particular, students appeared to be less persistent in solving the problems and thus did not make as much effort to modify, test, and submit their code. This finding has implications on research being done in the field.

Researchers should always consider that students' behavior in controlled laboratory setups may not always be exactly the same when studies are reproduced in an uncontrolled natural environment. In this particular study, there are certain conditions that are present in the laboratory setup which were not present in the online system environment. The students were conscious that their actions were being recorded in the laboratory setup. Furthermore, students who participated in the laboratory setup voluntarily committed to participate for a set duration of time, as opposed to those who used the online system who could use the system freely as they wished. Students were also allowed to complete the exercises in the order they wished in the online system, while exercises had to be solved sequentially in the controlled laboratory setup. These differences are often overlooked when drawing conclusions from studies that are performed in controlled environments. Thus, when student models are trained heavily on data from controlled laboratory setups, researchers must consider those models are acceptably representative of students in natural environments as well.

## 6. Conclusion and Future Work

In this paper, we presented an analysis of student behaviors in programming exercises using HMMs, and compared models trained from data in a controlled laboratory setup with that trained from data in a natural environment. Our findings show that while the models are similar, there are evidences of less persistence in solving the problems in the online setting. Such differences highlight the need for researchers to consider the reproducibility of findings from controlled setups to more natural environments. Currently, we have only used simple units of actions in this study to analyze the behavior of students in programming activities in this study. More in-depth features such as types of errors and the classification of different phases and events during the programming sessions could be done as next steps for those who are interested in a more detailed analysis of the session data.

## Acknowledgements

The authors would like to thank Mr. Fritz Kevin Flores, Mr. Manuel Carl Toleran, and Mr. Kayle Anjelo Tiu for assisting in the facilitation of the data collection process in the Philippines.

## References

Bloom, B. S. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. Educational researcher, 13(6), 4-16.

Drummond, C. (2009). Replicability is not reproducibility: nor is it good science.

Kumar, A. N. (2019, June). Providing the Option to Skip Feedback–A Reproducibility Study. In International Conference on Intelligent Tutoring Systems (pp. 180-185). Springer, Cham.

Martin, J. H., & Jurafsky, D. (2009). Appendix A. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. Upper Saddle River: Pearson/Prentice Hall. Retrieved from https://web.stanford.edu/~jurafsky/slp3/A.pdf

Tiam-Lee, T. J., & Sumi, K. (2018, June). Adaptive feedback based on student emotion in a system for programming practice. In International Conference on Intelligent Tutoring Systems (pp. 243-255). Springer, Cham.

Tiam-Lee, T. J., & Sumi, K. (2019, June). Analysis and Prediction of Student Emotions While Doing Programming Exercises. In International Conference on Intelligent Tutoring Systems (pp. 24-33). Springer, Cham.