Inferring Students' Tracing Behaviors from Interaction Logs of a Learning Environment for Software Design Comprehension

Prajish Prasad
Interdisciplinary Programme in Educational Technology
Indian Institute of Technology Bombay
Mumbai, India
prajish.prasad@iitb.ac.in

Sridhar Iyer
Interdisciplinary Programme in Educational Technology
Indian Institute of Technology Bombay
Mumbai, India
sri@iitb.ac.in

ABSTRACT

VeriSIM is a learning environment which helps software engineering students comprehend a given design, by enabling them to trace different scenarios in the design. Learners trace different scenarios by constructing a state diagram. In VeriSIM, learners go through four challenges which help them progressively trace scenarios in the design. VeriSIM provides learners affordances to add, edit, delete states, as well as "run" a given state diagram. All such learner interactions are logged in the system. In this paper, we analyze the interaction logs of 12 students, in order to infer certain behavior patterns. The preliminary analysis of these patterns shows that the order of the challenges help learners effectively trace scenarios.

CCS CONCEPTS

• **Social and professional topics → Software engineering education**.

KEYWORDS

UML modelling, design tracing, interaction log analysis

ACM Reference Format:
Prajish Prasad and Sridhar Iyer. 2020. Inferring Students' Tracing Behaviors from Interaction Logs of a Learning Environment for Software Design Comprehension. In *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research (Koli Calling'20), November 19–22, 2020, Koli, Finland*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3428029.3428564

1 INTRODUCTION

Modelling software designs is an essential skill required of computing students. However, studies have documented that students have difficulties in modeling software designs [2]. Students have difficulties in dealing with models having multiple views, such as the structural view (e.g. class diagrams), and the behavioral view (e.g. sequence diagrams), and how the overall system specifications actually work based on these views [1]. In order to help students develop an integrated understanding of design diagrams, we have developed the design tracing pedagogy, and have incorporated it into the VeriSIM learning environment.

Koli Calling '20, November 19–22, 2020, Koli, Finland

ACM ISBN 978-1-4503-8921-1/20/11.
https://doi.org/10.1145/3428029.3428564

2 THE VERISIM LEARNING ENVIRONMENT

The main objective of the VeriSIM learning environment is to help students trace scenarios in the design using the design tracing pedagogy. In design tracing, students trace the control flow and data flow across different diagrams for a given scenario of system execution. They construct a model of the scenario execution, which is similar to a state diagram. The values in the states correspond to the values of relevant variables, and the transitions correspond to different parts of the scenario. Based on the execution of parts of the scenario, variables in the state change as well.

In VeriSIM, students are provided with a class diagram and sequence diagrams of an automated door locking system. Students go through four challenges which progressively help them trace a given scenario. The scenario presented to students is different in each challenge. In every challenge, VeriSIM provides students opportunities to visualize the execution of the state diagram by clicking on a "Run" button. Students can proceed to the next challenge only if the given state diagram is correct, which can be checked by clicking on "Run".

The order of these challenges is based on model order progression [3], which has shown to improve modelling in learners. In Challenge 1, learners explore an already constructed state diagram. During the run of the state diagram, learners observe corresponding changes in the class diagram and sequence diagram. In Challenge 2, students are required to correct an incorrect state diagram. When learners click on the "Run" button for the given state diagram, the states which are incorrect are highlighted in red, as shown in Figure 1. Learners are supposed to edit the states by changing the values of the variables in the state. In Challenge 3, the states are empty, and relevant data variables and their values have to be added. Finally, in Challenge 4, learners have to construct the state diagram from scratch. Learners can add, edit, and delete data, events, and states by clicking on the "Edit" button (shown in Figure 1). Each of these actions are logged by the system in the interaction logs.

3 METHOD

The main research question which we investigate in this paper is - **"What are different strategies which students use to trace scenarios in a given design?"**

In order to answer this research question, we conducted a study with 12 second year CS/IT undergraduates (m=9, f=3). Students interacted with VeriSIM at their own pace. Interactions of all users were stored in a single CSV file, which was ordered by timestamp. From this file, we extracted the actions performed by each user in Challenge 2, 3 and 4. We excluded Challenge 1, since students did

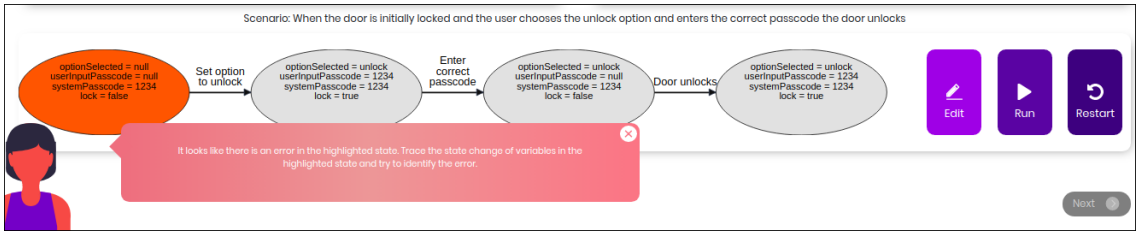


Figure 1: In Challenge 2, learners correct an incorrect model

not modify the state diagram in this challenge. For each challenge, user actions were then plotted onto a graph, with the x-axis corresponding to the time, and the y-axis corresponding to the actions in that challenge (examples in Figure 2 and 3). We manually examined the 36 graphs (12 users x 3 challenges) in order to identify different strategies which students used.

4 RESULTS AND DISCUSSION

Manual examination of the graphs led to identification of three distinct strategies across the three challenges. A summary of the number of students using each strategy is shown in Table 1.

Strategy 1: All modifications followed by single run at the end: In this strategy, students worked on the state diagram by modifying (adding/editing/deleting) data variables, and modifying (adding/editing/deleting) states, and clicked on 'run' only at the end to verify their correctly constructed state diagram (an incorrect state diagram would have led to further edits and runs). An example graph of Strategy 1 is shown in Figure 2.

Strategy 2: Two cycles of modification and run: In this strategy, students modified data variables and states, clicked on 'run', and modified the data/states once again.

Strategy 3: Multiple cycles of modification and run: In this strategy, students performed multiple cycles of modifying data variables and states, and clicking on run, as shown in Figure 3.

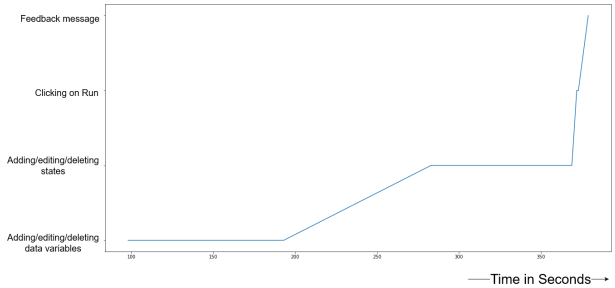
The analysis of the logs show that most students (8 out of 12) attempt Challenge 2 by employing multiple cycles of modification and run (Strategy 3). But by the time they reach Challenge 4, most students (8 out of 12) shift to using the 'run' only at the end (Strategy 1). These results gives us indicators that previous challenges helped students internalise the execution of the control flow and data flow of the given scenario. We can also infer that they did not frequently use the 'run' scaffold to periodically validate their model in latter challenges.

Table 1: Number of students using a strategy in each challenge based on their interaction logs

	Challenge 2	Challenge 3	Challenge 4
Strategy 1	2	2	8
Strategy 2	2	4	2
Strategy 3	8	6	2

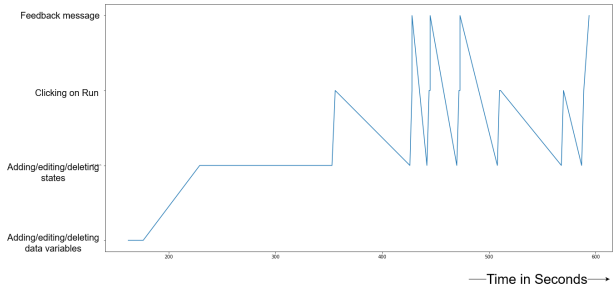
5 CONCLUSION AND FUTURE WORK

In this paper, we described the preliminary analysis of the interaction logs of 12 students, in order to infer their tracing behaviors.



1.png

Figure 2: The graph of logs of a challenge classified as Strategy 1



3.png

Figure 3: The graph of logs of a challenge classified as Strategy 3

We intend to conduct further studies with students in order to validate the inferred strategies. We also intend to apply machine learning techniques (such as pattern mining) in future studies in order to automate the process of inferring patterns from logs, and also determine finer patterns from interaction logs.

REFERENCES

[1] Loli Burgueño, Antonio Vallecillo, and Martin Gogolla. 2018. Teaching UML and OCL models and their validation to software engineering students: an experience report. *Computer Science Education* 28, 1 (2018), 23–41.

[2] Anna Eckerdal, Robert McCartney, Jan Erik Moström, Mark Ratcliffe, and Carol Zander. 2006. Can graduating students design software systems? *ACM SIGCSE Bulletin* 38, 1 (2006), 403–407.

[3] Yvonne G Mulder, Ard W Lazonder, and Ton de Jong. 2011. Comparing two types of model progression in an inquiry learning environment with modelling facilities. *Learning and Instruction* 21, 5 (2011), 614–624.