VeriSIM: A Learning Environment for Comprehending Class and Sequence Diagrams using Design Tracing

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

In this paper, we describe the design tracing strategy, which enables students to comprehend class and sequence diagrams by tracing different scenarios. In design tracing, for a given scenario, students identify relevant variables from the class diagram, relevant events from the sequence diagram and trace the flow of these data variables and events by constructing a state diagram. We have developed a web-based learning environment - VeriSIM, which trains students to apply the design tracing strategy. We conducted a study where 86 final-year undergraduates interacted with VeriSIM. Findings from the pre-test and post-test show that students are able to trace a given scenario by identifying relevant variables and events and are able to simulate change of state for these variables. A focus-group interview was also conducted with 13 participants in order to understand their perception of the usefulness of design tracing. A thematic analysis of the focus-group interview showed that students perceived design tracing helped them understand the relationship between different diagrams and identify different scenarios in the design. Interaction with VeriSIM also helped students understand the usefulness of creating class and sequence diagrams. These results show that design tracing can be a useful pedagogy to help learners form an integrated and correct understanding of class and sequence design diagrams.

CCS CONCEPTS

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

KEYWORDS

design tracing, class diagram, sequence diagram, integrated understanding, learning environment

ACM Reference Format:
Prajish Prasad and Sridhar Iyer. 2020. VeriSIM: A Learning Environment for Comprehending Class and Sequence Diagrams using Design Tracing. In *Software Engineering Education and Training (ICSE-SEET'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3377814.3381705

1 INTRODUCTION

When graduating students enter the software industry, they join large software development teams as the most inexperienced member and spend most of their time resolving bugs and writing additional features based on new requirements [1, 8]. When new requirements are provided, they need to have an integrated understanding of the design in order to add features into the design. This requires students to comprehend an already existing design, incorporate the required feature in the design and evaluate if the design satisfies the intended goals. Hence, a necessary skill which graduating students require is comprehending software designs. However, novice software designers find it difficult to comprehend large programs and make sense of the code [25].

Current software design courses do not focus on teaching students how to comprehend design diagrams and many students cannot evaluate an existing work and refactor it to be better [3, 26]. Although there has been extensive research done on models of program comprehension [4, 22, 23, 26], sufficient focus has not been given on comprehension of design diagrams, and teaching-learning strategies for software design diagram comprehension.

In this paper, we describe the design tracing pedagogy which helps students comprehend a subset of design diagrams i.e. the class and sequence diagrams by tracing scenarios on the given design. In design tracing, students identify relevant variables from the class diagram, relevant events from the sequence diagram and trace the flow of these data variables and events for a given scenario of system execution. The tracing is done by explicitly constructing a state diagram. We have chosen class and sequence diagrams as they correspond to the structural and behavioral categories of design diagrams respectively. They are commonly used, as well as the first diagrams which students learn about in their course. We have developed a web-based learning environment VeriSIM, which trains students to apply the design tracing pedagogy for various scenarios in a given design. We hypothesize that as students identify and trace different scenarios, they will be able to comprehend the class and sequence diagrams better. We conducted a study with 86 final-year computer engineering and information technology engineering students. We investigate whether students are able to perform design tracing after interacting with VeriSIM. We also investigate their perceptions on the usefulness of design tracing and how their perceptions of class and sequence diagram changed after interacting with VeriSIM.

2 RELATED WORK

2.1 Student difficulties in design

Teaching design often means teaching object-oriented (OO) design, and typically entails subjects such as the Unified Modeling Language (UML), design patterns, OO design principles, and use-case analysis [14]. However, studies have shown that a majority of graduate students are not competent in designing software systems [9, 19]. Students do not take efforts to describe the behavior of the system [19], have difficulties in understanding the purpose and relationships between the various diagrams [27, 30], and are unable to abstract real world problems [27]. Their software designs lack consistency and have missing information like class operations or dependencies [31]. Students produce class diagrams that are incomplete, with many concepts at inconsistent abstraction levels, and sequence diagrams with missing responsibilities and objects [27].

2.2 Strategies for comprehending design diagrams

Comprehension of design diagrams has been explored in the context of inspecting design diagrams in order to identify defects. One strategy to inspect design diagrams is to use checklists [11, 17, 24]. Although checklists are simple to use and cost-effective [6], they are not comprehensive. They cannot cover all verification and validation aspects of a software design and need to be accompanied by additional techniques [10]. Inspecting design diagrams has also been explored from a process perspective. A perspective-based reading scenario (PBR) supports the checking of a document from a particular stakeholder's perspective [16]. The perspective-based techniques for requirements reading aids analysis of the document by asking readers to generate a secondary artefact that contains the same information but using a different syntax. The authors claim that this technique helps in uncovering semantic deficiencies. Travassos et al. [32] have formulated a set of reading techniques, termed as traceability based reading, which help students integrate information across diagrams (horizontal based reading) and also between diagrams and textual requirements (vertical reading) as shown in Figure 1. Horizontal reading is used to trace between design documents to ensure consistency. Vertical reading is used to trace between design and requirements in order to ensure correctness and completeness. Using these reading techniques, students were asked to report defects in the given design diagrams. Results from their study show that the techniques did lead to defects being detected.

Although these reading techniques have been used to inspect design diagrams, their use as possible teaching-learning strategies have not been explored.

2.3 Tracing as a strategy for program comprehension

Program tracing is the process of emulating how a computer executes a program [12]. It involves writing the state change of variables after each line of code. Previous research has shown that students who use tracing are able to better comprehend a given program [7, 18, 34]. Cunningham et al. explored the relation between what students sketched and their ability to comprehend simple

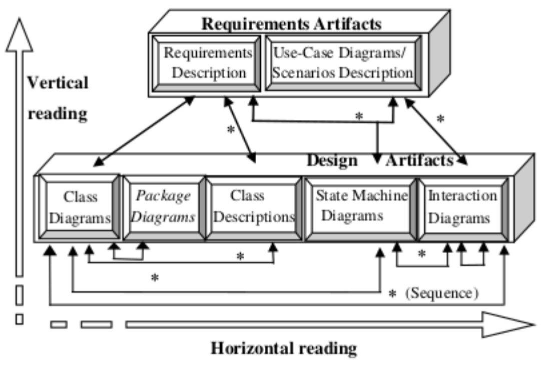


Figure 1: Summary of reading techniques taken from Travassos et al. [32]

python programs. They found that students who used tracing performed better than students who did not trace at all. Xie et al. [34] explicitly taught this tracing strategy to learners and found that program comprehension improved.

Tracing a program has several advantages. The state change of variables are written down in the form of memory tables or some other external representation. The external representation helps students manage the cognitive load of remembering values of variables during execution, especially if the program is large or complex [34]. The external representation also serves as an artefact which exhibits students' understanding of the program. As students trace, they take an active role in describing the 'run' of the program, as opposed to reading through the code [7]. Tracing also forces students to think what happens next in the program.

We have adapted the program tracing technique for design diagrams. We investigate how this technique, which is similar to program tracing can help in tracing scenarios and thereby aid in an integrated understanding of class and sequence diagrams.

3 DESCRIPTION OF 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. Students are provided with the description of the system requirements and a set of class and sequence diagrams. The key steps in design tracing are as follows:

(1) For a given scenario, identify relevant data variables from the class diagram. A class diagram contains multiple classes and their associations. However, in order to trace a given scenario, a subset of relevant classes and variables are chosen.
(2) For a given scenario, identify relevant events/messages from the relevant sequence diagram. A design contains multiple sequence diagrams which describe different behaviors of the system. In order to trace the given scenario, a particular sequence diagram (or set of sequence diagrams) are chosen and relevant messages are identified.
(3) Using the identified data variables and events, construct a model similar to a state diagram. The process of state diagram construction is as follows:

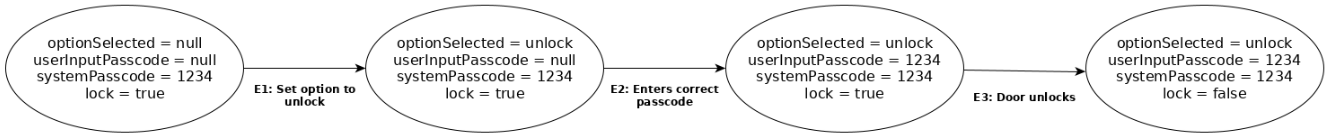


Figure 2: Design tracing example for a scenario in an automated door locking system design

(a) Start with an initial state, whereby initial values of relevant variables are chosen.
(b) Based on the initial part of the scenario, identify the relevant event which causes the transition to the next state.
(c) Based on the next part of the scenario, construct the next state by filling values of the relevant data variables.
(d) Continue (b), (c) till the final state is reached. This indicates that the entire scenario has been traced using the state diagram.

Consider an example which highlights this pedagogy. An automated door locking system design is provided, which contains a class diagram and 3 sequence diagrams - for lock, unlock and register. The following scenario has been provided - "When the door is initially locked and the user selects the unlock option and enters the correct passcode, the door unlocks". We apply design tracing to this scenario in the following manner:

(1) Relevant data variables are identified from the class diagram. For example, based on the design diagrams, optionSelected, userInputPasscode, systemPasscode and lock are relevant variables.
(2) Relevant events/messages are identified from the Unlock Sequence Diagram. For example: "E1: Set option to lock", "E2: Enter correct passcode" and "E3: Door unlocks" are relevant events for this scenario.
(3) Based on the identified data variables and events, the state diagram is constructed as shown in Figure 2.

3.1 Hypothesized advantages of design tracing

In design tracing, the learner is explicitly constructing a state diagram. The state diagram serves as a model of the execution of the scenario. The model integrates both the structure (data variables) and behavior (messages/functions in the sequence diagram) of a given scenario. The values in each state can be determined by analyzing the class diagram and the transitions help in analyzing the behavior. We hypothesize that as learners construct, revise and evaluate their model, they are able to develop an integrated understanding of the class and sequence diagram. Similar to program tracing, the state diagram can serve as a representation of students' understanding of the scenarios in the design.

Design tracing supports both horizontal and vertical reading techniques. The construction of the state diagram enforces students to look at the class diagram for relevant variables and their corresponding values in the state and relevant events in the sequence diagram which serve as transition to the next state. They perform horizontal reading by tracing between design diagrams. They are also performing vertical reading since they are modeling the given scenario using the state diagram.

As learners identify and trace different scenarios, we believe they will be able to comprehend class and sequence diagrams better. Firstly, by identifying different scenarios, learners are able to broadly explore the design solution space. Secondly, by tracing each of these scenarios, learners are forced to think deeply about the flow of data and events for each scenario. Hence, both the broad exploration of the design by identifying scenarios, and simulating the data and event flow for each scenario can help in an improved understanding of the class and sequence diagrams.

4 DESCRIPTION OF THE LEARNING ENVIRONMENT

4.1 Overview of VeriSIM

VeriSIM(**Veri**fying designs by **SIM**ulating scenarios) is a web-based learning environment which trains learners to apply the design tracing pedagogy. VeriSIM is available online on *https://verisim.tech*

In VeriSIM, learners are first introduced to the requirements and design diagrams for an "Automated Door Locking System". They then trace scenarios in the design using the design tracing pedagogy. Finally, they reflect on their overall learning and how it will be useful for them in the future. In VeriSIM, these activities are presented as various challenges to learners. VeriSIM takes learners through different stages which contain challenges. The three stages are - Problem Understanding Stage, Design Tracing Stage and Reflection Stage. As learners attempt these challenges, they gain points and acquire skills.

4.2 Stages and Challenges in VeriSIM

An overview of the stages and challenges in VeriSIM is shown in Figure 3. We describe each stage and challenge in detail below.

4.2.1 Introduction to VeriSIM. After learners log in to the system, they are presented with the learning objectives and an introductory video. In the video, learners are presented with a scenario of them having graduated and entered a software startup as a software developer. They are introduced to their team and project manager. The project manager then explains the importance of comprehending design diagrams. After watching the video, learners are directed to the VeriSIM dashboard (Figure 4). The dashboard contains the three stages and corresponding challenges in each stage. The objective of this introduction is to set the context and situate the learner in a software team setting. This is intended to ensure that they understand the need for design comprehension in the software industry and to motivate them to solve the upcoming challenges.

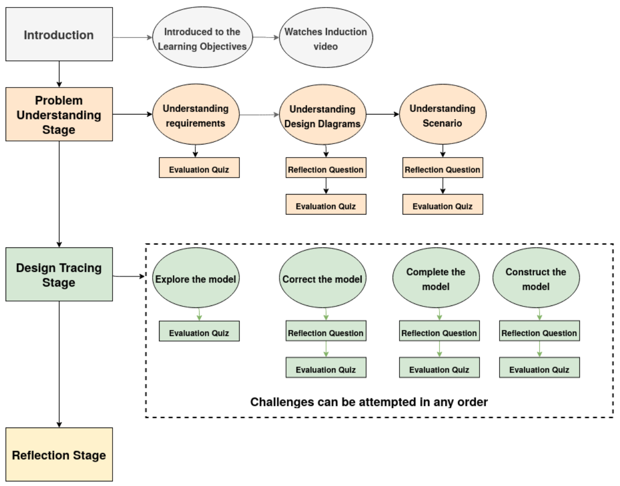
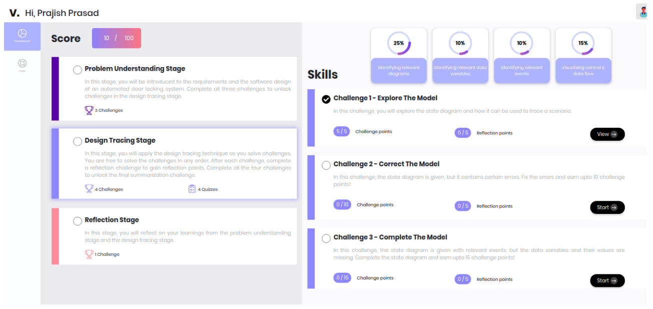


Figure 3: Overview of learning activities in VeriSIM


V. Hi, Prajish Prasad

Score 10 / 100

Problem Understanding Stage
In this stage, you will be introduced to the requirements and the software design of an automated door locking system. Complete all three challenges to unlock challenges in the design tracing stage.
3 Challenges

Design Tracing Stage
In this stage, you will apply the design tracing technique as you solve challenges. You are free to solve the challenges in any order. After each challenge, complete a reflection challenge to gain reflection points. Complete all the four challenges to unlock the final summarization challenge.
4 Challenges 4 Quizzes

Reflection Stage
In this stage, you will reflect on your learnings from the problem understanding stage and the design tracing stage.
1 Challenge

Skills
25% Identifying relevant diagrams
10% Identifying relevant data variables
10% Identifying relevant events
15% Visualizing control & data flow

Challenge 1 - Explore The Model
In this challenge, you will explore the state diagram and how it can be used to trace a scenario.
5 / 5 Challenge points 0 / 5 Reflection points View

Challenge 2 - Correct The Model
In this challenge, the state diagram is given, but it contains certain errors. Fix the errors and earn upto 10 challenge points!
0 / 10 Challenge points 0 / 5 Reflection points Start

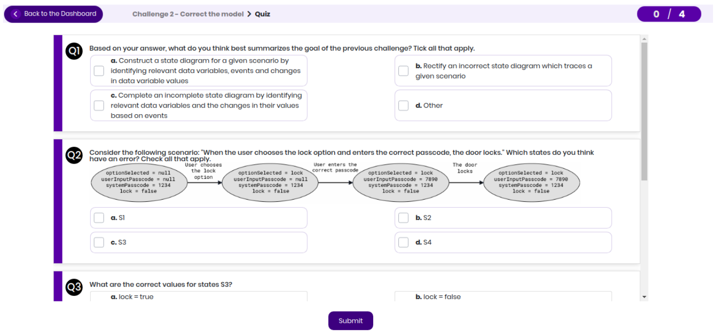
Challenge 3 - Complete The Model
In this challenge, the state diagram is given with relevant events, but the data variables and their values are missing. Complete the state diagram and earn upto 15 challenge points!
0 / 15 Challenge points 0 / 5 Reflection points Start

Figure 4: The VeriSIM dashboard

4.2.2 *Problem Understanding Stage.*

In the Problem Understanding Stage, learners are introduced to the requirements and the software design of an "Automated Door Locking System". This stage comprises three challenges.

(1) **Challenge 1 - Understand the client and requirements** - The learner is presented with the requirements of the automated door locking system. They then solve multiple choice questions based on the given requirements. The objective of this formative evaluation is to make sure that learners have understood the requirements.
(2) **Challenge 2 - Understand the software design diagrams** - The learner is presented with the design diagrams of the door locking system. A pedagogical agent gives an overview of the class diagram and sequence diagrams. After the learner goes through these diagram, the learner is asked to identify defects (if any) in the diagrams and answer some multiple choice questions based on the diagrams. These reflection and formative evaluation questions enable learners to reflect on and closely analyze the given diagrams.
(3) **Challenge 3 - Understand scenarios** - In this challenge, the system explains what is meant by a scenario and displays a scenario based on the design. The learner is then asked to identify other scenarios in the design and answer multiple choice questions based on the scenarios.


Back to the Dashboard Challenge 2 - Correct the model > Quiz 0 / 4

Q1 Based on your answer, what do you think best summarizes the goal of the previous challenge? Tick all that apply.
a. Construct a state diagram for a given scenario by identifying relevant data variables, events and changes in data variable values
b. Rectify an incorrect state diagram which traces a given scenario
c. Complete an incomplete state diagram by identifying relevant data variables and the changes in their values based on events
d. Other

Q2 Consider the following scenario: "When the user chooses the lock option and enters the correct passcode, the door locks." Which states do you think have an error? Check all that apply.

```
optionSelected = null        optionSelected = lock        optionSelected = lock        optionSelected = lock
userInputPasscode = null     userInputPasscode = null     userInputPasscode = 7890     userInputPasscode = 7890
systemPasscode = 1234        systemPasscode = 1234        systemPasscode = 1234        systemPasscode = 1234
lock = false                 lock = false                 lock = false                 lock = false
```

User chooses the lock option
User enters the correct passcode
The door locks

a. S1
b. S2
c. S3
d. S4

Q3 What are the correct values for states S3?
a. lock = true
b. lock = false

Submit

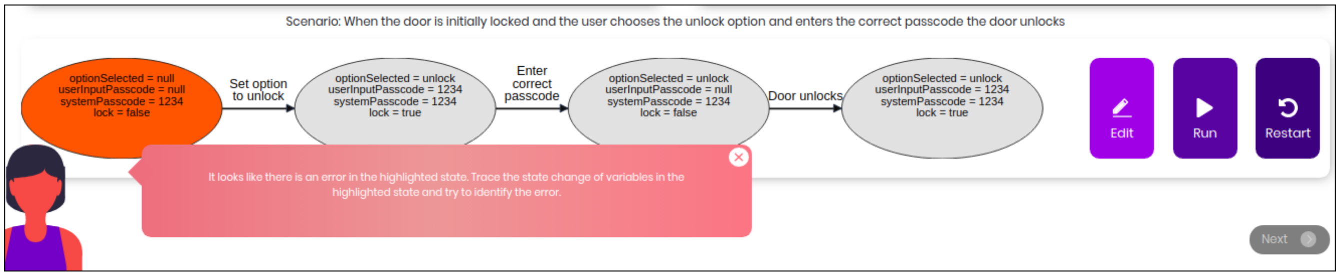
Figure 5: Reflection and evaluation questions after a challenge in VeriSIM

The pedagogical basis of the challenges in the Problem Understanding Stage is to help learners build an adequate understanding of the requirements and the design diagrams. This understanding is a pre-requisite to trace scenarios in the next stage. Literature also emphasises that development of an adequate problem representation is important and essential for high performance in software design [28]. These challenges ensure that learners have a detailed understanding of the problem (requirements) and solution (design diagram) space. The challenges in this stage ensure that learners have analyzed requirements and scenarios of the design.

4.2.3 *Design Tracing Stage.*

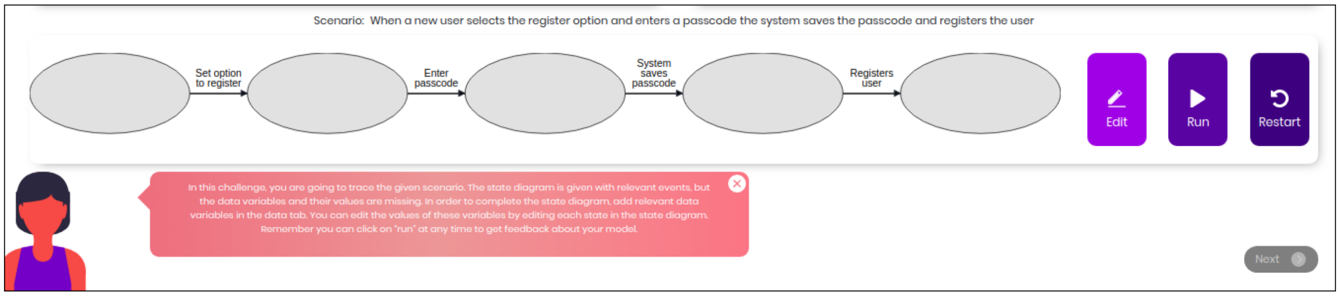
In the Design Tracing Stage, learners are introduced to the design tracing strategy. This stage comprises four challenges in increasing order of complexity. In each challenge, a different scenario of the same design is provided. Learners are free to attempt the challenges in any order. Each challenge is followed by reflective and evaluative questions [13]. An example is shown in Figure 5. The reflection question makes students reflect on what they have done in the challenge. The evaluation questions test them on what they learnt in the challenge. This stage along with the Reflection Stage is initially locked i.e. it can be attempted only after the learner completes the Problem Understanding Stage. The challenges in the Design Tracing Stage are as follows:

(1) **Explore the model** - In this challenge, learners are introduced to the state diagram for a scenario. The agent explains the different parts of the state diagram and how the scenario is being traced. The learner clicks on the "Run" button which helps them visualize the execution of the state diagram. During the run of the state diagram, learners observe corresponding changes in the class diagram and sequence diagram.
(2) **Correct the model** - The objective of this challenge is to fix certain errors in the state diagram and thereby correctly trace the given scenario. 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 6a. Learners are supposed to fix the error by changing the values of the variables in each state.


Scenario: When the door is initially locked and the user chooses the unlock option and enters the correct passcode the door unlocks

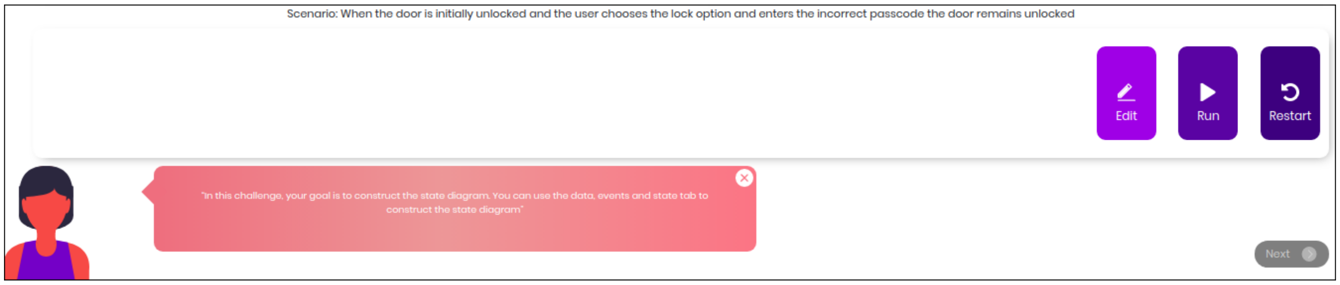
It looks like there is an error in the highlighted state. Trace the state change of variables in the highlighted state and try to identify the error.

(a) In "Correct the model" challenge, learners correct the incorrect state diagram provided


Scenario: When a new user selects the register option and enters a passcode the system saves the passcode and registers the user

In this challenge, you are going to trace the given scenario. The state diagram is given with relevant events, but the data variables and their values are missing. In order to complete the state diagram, add relevant data variables in the data tab. You can edit the values of these variables by editing each state in the state diagram. Remember you can click on "run" at any time to get feedback about your model.

(b) In "Complete the model" challenge, learners identify relevant data variables and their state change


Scenario: When the door is initially unlocked and the user chooses the lock option and enters the incorrect passcode the door remains unlocked

"In this challenge, your goal is to construct the state diagram. You can use the data, events and state tab to construct the state diagram"

(c) In "Construct the model" challenge, learners construct the entire state diagram

Figure 6: Challenges in VeriSIM which progressively enable learners to construct the state diagram

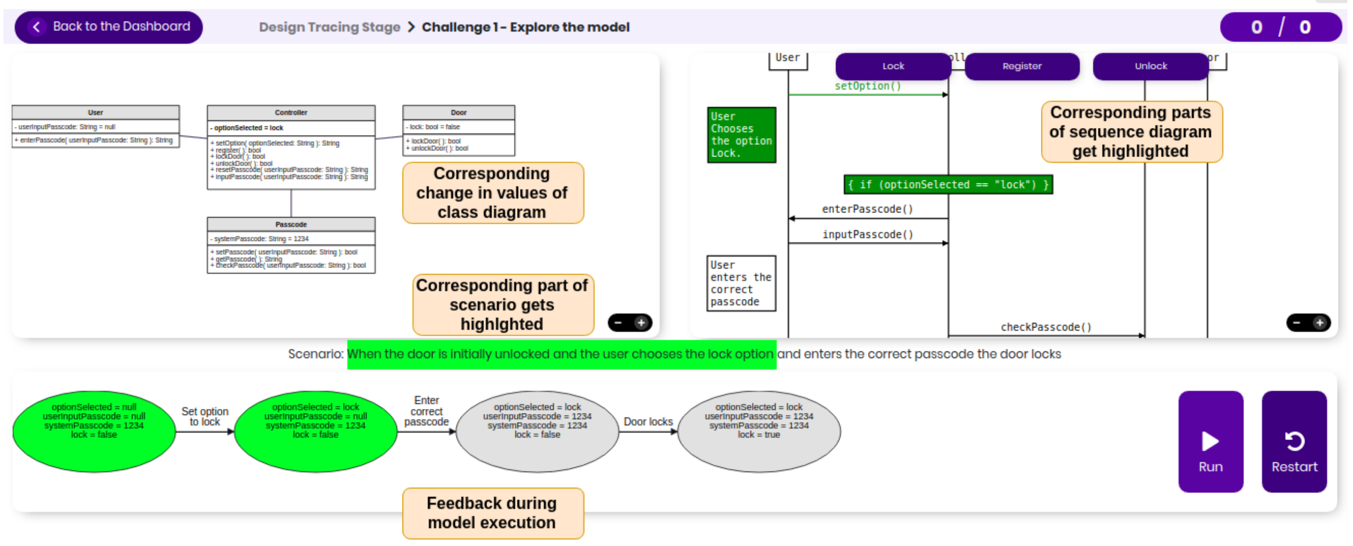
(3) **Complete the model** - In this challenge, the state diagram is given with relevant events, but the data variables and their values are missing, as shown in Figure 6b. In order to complete the state diagram, learners have to add relevant data variables from the data tab. Learners can edit the values of these variables by editing each state in the state diagram.
(4) **Construct the model** - The objective of this challenge is to construct the state diagram from scratch, as shown in Figure 6c. Learners can use the data, events and state tab to construct the state diagram.

In all the challenges in the design tracing stage, the objective of the learner is to trace the scenario using the state diagram. The state diagram has to match the expert model for the learner to successfully complete the challenge. Each state in the state diagram is compared with the corresponding state in the expert model and appropriate feedback is provided by the system.

We have designed the pedagogy for the design tracing stage by drawing on literature from model based learning and program visualization.

Model-based learning: In the Design tracing stage, learners progressively learn to construct the state diagram. The order of challenges is based on different strategies employed in modelling literature. Learners start by prior exploration of the model, and then interact with erroneous and partially worked out models, and finally construct the entire model. In the first challenge, they observe the run of the state diagram i.e. the execution trace of the scenario. In the second challenge, there is an error in the data flow, which the learner has to correct. In the third challenge, the entire data flow has to be traced by the learner. In the fourth challenge, control flow and data flow has to be traced. Each of these modelling activities, such as prior exploration [15], learning from partial [20] and erroneous models [33], have been shown to be beneficial to students. This model order progression [21] can scaffold learners to effectively construct a model for a given scenario.

Program Visualization: A key feature of VeriSIM is allowing learners to execute the state diagram. This feature is based on existing work in program visualization literature. Program visualization systems (PV) are used to display program executions automatically


Back to the Dashboard
Design Tracing Stage > Challenge 1 - Explore the model
0 / 0
User
- userInputPasscode: String = null
+ enterPasscode(userInputPasscode: String): String
Controller
- optionSelected = lock
+ setOption(optionSelected: String): String
+ register( ): bool
+ lockDoor( ): bool
+ unlockDoor( ): bool
+ resetPasscode(userInputPasscode: String): String
+ inputPasscode(userInputPasscode: String): String
Door
- lock: bool = false
+ lockDoor( ): bool
+ unlockDoor( ): bool
Passcode
- systemPasscode: String = 1234
+ setPasscode(userInputPasscode: String): bool
+ getPasscode( ): String
+ checkPasscode(userInputPasscode: String): bool
Corresponding change in values of class diagram
Corresponding part of scenario gets highlghted
User
Lock
Register
Unlock
setOption()
User Chooses the option Lock.
{ if (optionSelected == "lock") }
enterPasscode()
inputPasscode()
User enters the correct passcode
checkPasscode()
Corresponding parts of sequence diagram get highlighted
optionSelected = null userInputPasscode = null systemPasscode = 1234 lock = false
Set option to lock
optionSelected = lock userInputPasscode = null systemPasscode = 1234 lock = false
Enter correct passcode
optionSelected = lock userInputPasscode = 1234 systemPasscode = 1234 lock = false
Door locks
optionSelected = lock userInputPasscode = 1234 systemPasscode = 1234 lock = true
Feedback during model execution
Run
Restart

Scenario: When the door is initially unlocked and the user chooses the lock option and enters the correct passcode the door locks

Figure 7: Affordance of the run feature in VeriSIM

or semi-automatically. These systems allow teachers to demonstrate and students to explore the runtime behavior of programs [29]. In our case, we have extended the program visualization idea to software design diagrams. In VeriSIM, the execution is entirely simulated, and not automatic. However, we believe that the simulation can help in students' understanding of the execution of the scenario and corresponding changes in different design diagrams.

VeriSIM provides the affordance to visualise the state-wise execution of the model, as shown in Figure 7. Based on each click of the "Run" button, appropriate highlights are made in the class and sequence diagram. Hence learners are able to visualise the relevant changes in the class and sequence diagrams in a particular state. For example - A transition to a new state triggers a set of events in the sequence diagram. The visualization shows the corresponding changes in the class diagram (change of values of variables) for each event/message in the sequence diagram. The learner can replay this execution step, review previous execution steps and move forward. As the learner observes these visualizations and corresponding changes in the design diagrams, they are able to visualize the state change of different variables at a given state and also understand the relationship between the class and sequence diagram.

4.2.4 Reflection Stage. In the Reflection Stage, the system provides learners with reflection questions which help them summarise what they learnt in the previous two stages and their perceptions of the usefulness of the design tracing pedagogy.

5 RESEARCH DESIGN

5.1 Research Questions

The research questions for this study are as follows:

- RQ 1: Does VeriSIM enable learners to trace a given scenario?
- RQ 2: How does VeriSIM change learners' perception of the usefulness of class and sequence diagrams?
- RQ 3: What are learner perceptions of the usefulness of the design tracing?

5.2 Study Procedure

The study was conducted with 86 final year (fourth year) computer engineering and information technology engineering students (48 male and 38 female), in their own institution. The medium of instruction in their institution is English, hence all participants were comfortable in reading, writing and speaking in English. Their average age was 21 years. The engineering institution is located in a metropolitan city in our country. Participation in the study was completely voluntary, and participants could withdraw from the study at any point during the study. A few days prior to the study, participants had to fill a registration form with their basic information like name, branch, overall percentage in the last semester, and rate their confidence in understanding of object-oriented design, class and sequence diagrams. All participants had undergone a Software Engineering course in the previous semester, and hence were familiar with class diagrams and sequence diagrams. All participants were comfortable working on a computer. Due to space constraints, all the participants could not be accommodated in a single room at once. Hence, the study was divided into 2 sessions - one in the morning and the other in the afternoon. 60 students (30 male and 30 female) participated in the morning session. 26 students (18 male and 8 female) participated in the afternoon session.

The study took place in a computer lab. We initially explained the main goals of the workshop and the activities participants will perform during the workshop. After this introduction, participants solved a pre-test. After solving the pre-test, they started interacting with VeriSIM. Participants worked individually with VeriSIM at their own pace. We did not interfere while they interacted with VeriSIM, but were available to answer any doubts which arose during their interaction. After participants finished interacting with VeriSIM, they solved a post-test. They then filled a feedback form,

Table 1: Rubric for evaluating tracing questions in the pre-test and post-test

	Missing (0)	**Almost (1)**	**Target (2)**
Identifying data variables	Missing all relevant data variables from the class diagram	Identifies some relevant variables Adds irrelevant data variables	Identifies all relevant data variables No irrelevant data variables added
Identifying relevant events	Missing all relevant events Separation of events is not seen	Identifies some relevant events Identifies some irrelevant events Separation of events is unclear	Identifies all relevant events No irrelevant events included Separation of events is clear
Simulating state change	No mention of state change of variables	State change of some variables are mentioned with variable-value pairs	State change of all variables are clearly mentioned with correct variable-value pairs

which contained usability and other feedback questions. After the study, a semi-structured focus group interview was conducted in each session (8 participants (5 male and 3 female) in the morning session and 5 participants (2 male and 3 female) in the afternoon session). Based on the academic details filled by participants in the registration form, students who had varied prior academic performance were chosen as the participants of our focus group interview. On an average, participants took 1.5 hours to interact with VeriSIM. Each study session (pre-test, VeriSIM, post-test, feedback and focus-group interview) took around 3 hours.

5.3 Data Sources

The data sources for the study are as follows:

(1) Pre-test - In the pre-test, participants were provided with the requirements and design of an ATM system (1 class diagram and 3 sequence diagrams). They were provided with the following incomplete scenario - *"The user has a balance of Rs.5000 in his account, enters the correct PIN and withdraws Rs.500"*. The participant had to explain the sequence of steps and changes that occur in the system from the beginning to the end on execution of this scenario.
(2) Post-test - In the post-test, participants were provided with the requirements and design of an online library system (1 class diagram and 3 sequence diagrams). They had to trace the following scenario using the design tracing technique - *"When a new user selects the register option, and enters the username and password, the user registers successfully."*
Both the pre-test and the post-test question tested students' ability to identify relevant data variables, events and changes in state of these variables on execution of the scenario.
(3) Feedback form - The feedback form asked participants to rate their confidence in class and sequence diagram understanding after interacting with VeriSIM. There were also open-ended questions asking them what they learnt in the workshop and also the features of VeriSIM which they found useful.
(4) Semi-structured focus group interview - A semi-structured focus-group interview was conducted with students in order to gather their perceptions about design tracing and their learning with VeriSIM. The list of questions asked are provided in Table 2.

Table 2: Focus Group Interview Questions

Focus Group Interview Questions
1. What are the main things you learnt from the workshop?
2. What according to you is design tracing?
3. What is the usefulness of constructing the state diagram? Is there value to it?
4. How do you think what you learnt in this workshop is useful for you in the future?

5.4 Data Analysis

5.4.1 Grading design tracing questions. In order to answer RQ 1 (Does VeriSIM enable learners to trace a given scenario?), we analyzed the tracing question in the pre-test and post-test. Out of 86 participants, 11 did not stay till the end of the study and did not attempt the post-test. Hence we analyzed only the remaining 75 responses in the pre-test and post-test. We designed a rubric in order to evaluate student responses in the pre-test and post-test (Table 1). The rubric contains three criteria which are needed to trace a given scenario: (1) identifying relevant data variables, (2) identifying events and (3) simulating change of state. Three levels (Missing, Almost and Target) have been chosen based on how many relevant data variables, events and state changes students are able to identify for a given scenario. For data variables, students are required to identify all relevant and correct variables from the class diagram. For events, students are required to identify all relevant events from the scenario and sequence diagrams, with explicit distinction between events specified. For states, students are required to simulate change of state with relevant variable value pairs.

In order to establish content validity of the rubric, the first author discussed the criteria and levels of the rubric with another researcher (non-author) who was not involved in the research study. Three answer sheets were used to refine wordings of certain criteria. In order to establish reliability of the grading using the rubric, another rater (non-author) graded a subset of the questions. Both the first author and the rater independently graded 2-3 questions. They then discussed how their scoring was done, and differences in scoring was resolved. They then independently graded 10 questions. Inter-rater reliability of the rubric was established by calculating Cohen's Kappa [5] for each criteria. The first author then graded the

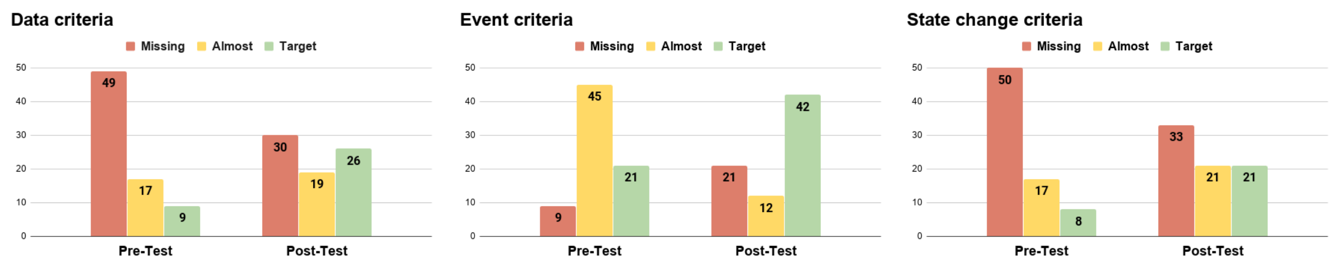


Figure 8: Bar graph showing pre-test and post-test scores for data, event and state change criteria

remaining students' questions based on the rubric. This procedure was done for both the pre-test and the post-test.

The Cohen's kappa for each criteria in the pre-test is as follows: $Kappa_{Data} = 1, Kappa_{Event} = 0.83, Kappa_{State} = 0.81$. The Cohen's kappa for each criteria in the post-test is as follows: $Kappa_{Data} = 0.85, Kappa_{Event} = 0.85, Kappa_{State} = 0.85$.

5.4.2 Thematic analysis of interviews and feedback form. In order to answer RQ 2 (How does VeriSIM change learners' perception of the usefulness of class and sequence diagrams?) and RQ 3 (What are learner perceptions of the usefulness of the design tracing?), we used two data sources: the focus group interview transcripts and the feedback form questions. We analyzed the focus group interview and the questions in the feedback form using a thematic analysis approach. Thematic analysis is the process of identifying patterns or themes within qualitative data [2]. The first author transcribed the focus group interviews. The first author and another researcher independently coded the transcript, and came up with initial codes. They then individually checked for emerging patterns and categorised the codes into themes. After independently coming up with themes, both researchers discussed between themselves, reviewed and defined the themes and came to an agreement on the final themes.

The question asked in the feedback form was - *"What are the main things you learned from the workshop?"*. We received 75 participant responses from the feedback form. A similar thematic analysis procedure was followed for these responses. The themes which emerged from the focus group interview and feedback form responses correspond to learner perceptions about their understanding of class and sequence diagrams and the usefulness of design tracing.

6 RESULTS

6.1 RQ 1: Students tracing scenarios

We answer our first research question by analyzing the pre-test and post-test answers. In the pre-test, the mean scores for the data, event and state criteria are $0.47(SD = 0.7), 1.16(SD = 0.62)$ *and* $0.44(SD = 0.68)$ respectively. In the post-test, the mean scores for the data, event and state criteria are $0.95(SD = 0.87), 1.28(SD = 0.88)$ *and* $0.84(SD = 0.84)$ respectively. These results show that participants were able to identify data variables and changes in the state, better in the post-test. However, there was only a marginal difference in their ability to identify events. These results also show that

Table 3: Participants' category-wise scores for pre-test and post-test

Criteria for tracing scenarios	Pre-test Mean(SD)	Post-test Mean(SD)
Identifying relevant data variables	0.47(0.70)	0.95(0.87)
Identifying relevant events	1.16(0.62)	1.28(0.88)
Simulating state change	0.44(0.68)	0.84(0.84)
Total Score	2.07(1.70)	3.07(2.09)

in both pre-test and post-test, participants were able to identify events of a scenario better than identifying data and states. We hypothesize that participants were able to identify events better, because identifying events can be done by observing the scenario itself as well as a cursory glance at the relevant sequence diagram. Identifying data variables and constructing the state diagram requires a more detailed analysis of the class diagram as well as the sequence diagram.

Figure 8 shows the bar graph of participants score in the data, event and state change criteria for pre-test and post-test. In the post-test, around 60% of the participants were able to identify almost/all the relevant variables (45/75) and state changes for the given scenario (42/75), as opposed to only around 35% in the pre-test (data - 26/75, state change - 25/75). However, more students were able to identify almost/all events in the pre-test (66/75), as opposed to the post-test (54/75).

6.2 RQ 2: Change in learners' perception of usefulness of class and sequence diagrams

We answer our second research question by examining the broad themes which emerged from the thematic analysis of the focus group interview transcripts and feedback responses. The major themes are shown in Table 4. Primarily, VeriSIM helped learners change their attitude towards class and sequence diagrams. Even though students were taught how to create class and sequence diagrams in their software design course, they did not use it explicitly while designing systems for their projects. Students also realised the importance of design diagrams to plan a software project. Interaction with VeriSIM helped them realise the benefits of creating class and sequence diagrams at the start. Rather than directly jumping into coding, creating class and sequence diagrams at the start can help them catch errors early on. This helps them manage their

Table 4: Participants' perception of usefulness of design diagrams

Theme	Sub-Themes	Example
Change in attitude towards design diagrams		*" 1. I believe it changed my attitude towards class and sequence diagram, because when we practice class and sequence diagrams, I tend to not give it a lot of importance, because when we make projects, we simply channel certain requirements and based on that, you know we will be making."*
Project planning	1. Importance of creating class diagrams at the start	1. "For this I got to know actually, I got to know the advantages of designing a class diagram before actually starting with the project." *2. "Because of the mistakes I made in my second year, that I used to directly start coding and not make class diagrams, this would have saved a lot of time for me."*
	2. Helps to save time	*1. "Rather than implementing on the project and then later getting stuck and then identify what it is, so here you have an idea of what is going on, and then you can improve on your project. It saves time."* *2. "So this is useful, as it charts out, how to even manage our time, for even smaller teams"*
	3. Facilitate knowledge transfer among changing design teams	*1. "But I get why it's so necessary in bigger teams, like he mentioned, if one person leaves and another person needs to come, he needs to be in touch with what's happening, so show him these diagrams, he will figure it out."*

Table 5: Participants' perception of usefulness of design tracing

Theme	Example
Understanding the logical flow of the design	*1. "It helped us think as in, how to construct a proper state diagram, how the flow goes from one state to another."* *2. "Given a scenario, like what if a a person, it's basically a set of steps, I do certain things, after that I proceed to this thing, after that I proceed to this thing. Now given a user requirement, and the particular step is not there, the check is not there, the condition that the user has mentioned, such as in this case, the showing of books, it's not there, then definitely the process flow has broken, which the design tracing did help."*
Thinking of different scenarios in the design	*1. "Thinking of all scenarios that you are planning in your project."* *2. "Yes, how do we validate the machine? Suppose we have to enter an age, and we have to identify, suppose the person enters like minus, negative numbers. So that .. those cases are not there only. So identify various scenarios."*
Identifying missing features and errors in the design	*1. "What design tracing I believe helped me was, uh. gave a scenario. If I can model the scenarios, different different scenarios, based on the sequence diagram I can identify whether the given sequence diagram is correct or wrong, you know what error is there"* *2. "Identifying missing features, and if its not there then you can add more features, you can manipulate as per your requirements and all, because you have an idea of how your overall project is going to look like."*
Understanding variables and their change in values	*1. "How which values get affected at what point of time etc, can be understood through the design tracing."* *2. "then if he says he wants to unlock, then again it will check, then it will say enter passcode all of that, so in the state diagram, you had to kind of fill those variables."* *3. "Looking into what all variables are there, what all variables have to be changed, what all conditions have to be."*
Integrated understanding of different design diagrams	*1. "I learnt how to read attributes from class diagram and events from the sequence diagram and implement it to understand the project in a better way."* *2. "and how I need to traverse through that state diagram, I need to know the sequence diagram very well, or the class diagram very well. So it was useful to learn how it goes step by step."* *"3. It was kind of hard, but we would have to first understand the sequence diagram system, to put forth values to put if its false, or if it is true, and there was one where the default values were given"*
Interpret requirements and design better	*1. "The workshop helped me in analyzing the requirements and coming up with different ways to deal with situations" "* *2. "How to understand the problem in a better way step by step."* *3. "How to map requirements to design and design tracing"*

project better and hence save time. Design diagrams are also useful to convey information to other members of the design team.

6.3 RQ 3: Perceptions of usefulness of design tracing

The broad themes which emerged from the thematic analysis done on the focus group interview transcripts and feedback responses are shown in Table 5. Participants reported that design tracing helped them understand the logical flow of the design. The state diagram helped them understand how the sequence of execution goes logically from one state to another. Design tracing also helped them think of different cases and scenarios in the design and imagine how users will use the system. Modeling different scenarios using the state diagram helped them detect errors in the sequence diagram and identify user requirements not considered in the sequence diagram. Design tracing also helped them understand different variables and their change in values at different points in time. By knowing which value changed at a certain point in time, errors can be uncovered.

Design tracing helped them understand the relationship between different design diagrams. Participants stated that they had to understand the class and sequence diagram in order to complete the state diagram. This gave them an integrated understanding of the class and sequence diagrams. Design tracing also helped them gain a better understanding of the requirements and design. It helped them analyze the requirements systematically and map them to the given design diagrams.

6.4 Limitations

The study was conducted a few days before participants' midterms, and hence we were aware of participants' time constraints. Since the study happened on a single day, participants could have been fatigued towards the end, i.e. while solving the post-test. Due to the absence of a control group, we cannot conclusively claim that VeriSIM alone changed their perceptions about their understanding of class and sequence diagrams. During the focus-group interview, answers from students could have biased other students, but counter-questions were posed in order to mitigate this bias.

7 DISCUSSION

Our results show that explicitly teaching students to trace scenarios in the design can help students understand class and sequence diagrams better. By explicitly constructing the state diagram, students were able to form an integrated understanding of the class and sequence diagram (*"and how I need to traverse through the state diagram, I need to know the sequence diagram very well, or the class diagram very well"*). Students also reported a change in their attitude towards class and sequence diagrams (*"I believe it changed my attitude towards class and sequence diagram,.. because I tend to not give it a lot of importance"*). We believe students' interaction with VeriSIM can help them overcome common reported difficulties of understanding the purpose and relationship of different diagrams as discussed in Section 2.1.

Students also reported that design tracing helped them identify missing features and errors in the design. By modeling different scenarios, students were able to perform both horizontal reading (between the class diagram and sequence diagram) and vertical reading (between design diagrams and scenarios). Hence, the design tracing pedagogy is an effective application of how these reading techniques can be used as a teaching strategy for inspecting and understanding class and sequence diagrams.

In the focus group interview, students reported how design tracing helped them understand variables and their change in values. In the feedback form, participants were asked to rate their confidence on a scale from 1 to 5 (5 being absolutely confident) regarding their knowledge of identifying variables, events and simulating state change for tracing a scenario. The mean score for data was 4.07, for events was 4.12 and for state change was 4.1. These high self-perception scores also correspond to what students reported in the focus group interview. However, their performance in the post-test did not correspond to their self-perception. Although students performed better in the post-test on average, many students could still not identify relevant variables and simulate state changes for the given scenarios. However, in both the pre-test and the post-test, they were able to identify events better than identifying variables and state changes.

This finding is consistent with results from program comprehension literature. A distinction between "surface knowledge" and "deep knowledge" is well established in program comprehension [23]. Program knowledge concerning operations and control structures reflect surface knowledge, i.e. knowledge that is readily available by looking at a program. In contrast, knowledge concerning data flow and function of the program reflect deep knowledge which is an indication of a better understanding of the code. Students had sufficient surface knowledge about the structure of the sequence diagrams to help them identify events, but they could not identify relevant variables and simulate the data flow of variables for a given scenario. These results suggest that more training is required to be given to students in helping them identify variables and simulating their change of values for a given scenario.

8 CONCLUSION AND FUTURE WORK

In this paper, we describe the design tracing pedagogy and the learning environment which uses this pedagogy to train students in understanding class and sequence diagrams. The results of the study show that learners perceive design tracing to be useful to understand the relationship between different diagrams, and in identifying scenarios and missing features in the design. However, more training is required for students to identify relevant data variables and simulate their flow for different scenarios. This can help students acquire "deep knowledge" about the given design and thereby comprehend class and sequence diagrams better.

In the current version, we have restricted our focus to helping learners identify the interconnection between two types of diagrams - the class diagram and the sequence diagram. The design diagrams chosen in VeriSIM do not include advanced concepts like inheritance, association etc. Design tracing can be used as a precursor strategy and can be applied to comprehend simple as well as complex design diagrams. We would like to add more examples and design diagram scenarios which will enable learners to apply the design tracing strategy to different design examples. These examples can contain different design diagrams, and can be introduced for advanced concepts as well.

9 ACKNOWLEDGEMENTS

We thank Kinnari Gatare for the user-interface design of VeriSIM, Bhupender Singh Maan for the development of VeriSIM, T.G. Lakshmi and Herold P.C. for the initial design and planning of activities in VeriSIM.

REFERENCES

[1] Andrew Begel and Beth Simon. 2008. Struggles of new college graduates in their first software development job. In *ACM SIGCSE Bulletin*, Vol. 40. ACM, 226–230.

[2] Virginia Braun and Victoria Clarke. 2012. Thematic analysis. (2012).

[3] Eric Brechner. 2003. Things they would not teach me of in college: what Microsoft developers learn later. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM, 134–136.

[4] Jean-Marie Burkhardt, Françoise Détienne, and Susan Wiedenbeck. 2002. Object-oriented program comprehension: Effect of expertise, task and phase. *Empirical Software Engineering* 7, 2 (2002), 115–156.

[5] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[6] Kendra ML Cooper, Sheila Liddle, and Sergiu Dascalu. 2005. Experiences Using Defect Checklists in Software Engineering Education.. In *CAINE*. 402–409.

[7] Kathryn Cunningham, Sarah Blanchard, Barbara Ericson, and Mark Guzdial. 2017. Using tracing and sketching to solve programming problems: Replicating and extending an analysis of what students draw. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*. ACM, 164–172.

[8] Barthélémy Dagenais, Harold Ossher, Rachel KE Bellamy, Martin P Robillard, and Jacqueline P De Vries. 2010. Moving into a new software project landscape. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 275–284.

[9] 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.

[10] Albert Endres and H Dieter Rombach. 2003. *A handbook of software and systems engineering: Empirical observations, laws, and theories*. Pearson Education.

[11] Michael E Fagan. 1999. Design and code inspections to reduce errors in program development. *IBM Systems Journal* 38, 2.3 (1999), 258–287.

[12] Sue Fitzgerald, Beth Simon, and Lynda Thomas. 2005. Strategies that students use to trace code: an analysis based in grounded theory. In *Proceedings of the first international workshop on Computing education research*. ACM, 69–80.

[13] X Ge and SM Land. 2004. A conceptual framework for scaffolding ill-structured problem-solving processes using question prompts and peer interactions. *Educational Research Technology and Development* 52, 2 (2004), 1042–1629.

[14] Chenglie Hu. 2013. The nature of software design and its teaching: an exposition. *ACM Inroads* 4, 2 (2013), 62–72.

[15] Birgit Kopainsky, Stephen M Alessi, Matteo Pedercini, and Pål I Davidsen. 2015. Effect of prior exploration as an instructional strategy for system dynamics. *Simulation & Gaming* 46, 3-4 (2015), 293–321.

[16] Oliver Laitenberger, Colin Atkinson, Maud Schlich, and Khaled El Emam. 2000. An experimental comparison of reading techniques for defect detection in UML design documents. *Journal of Systems and Software* 53, 2 (2000), 183–204.

[17] Soren Lauesen. 2002. *Software requirements: styles and techniques*. Pearson Education.

[18] Raymond Lister, Elizabeth S Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, et al. 2004. A multi-national study of reading and tracing skills in novice programmers. In *ACM SIGCSE Bulletin*, Vol. 36. ACM, 119–150.

[19] Chris Loftus, Lynda Thomas, and Carol Zander. 2011. Can graduating students design: revisited. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, 105–110.

[20] Yvonne G Mulder, Lars Bollen, Ton de Jong, and Ard W Lazonder. 2016. Scaffolding learning by modelling: The effects of partially worked-out models. *Journal of research in science teaching* 53, 3 (2016), 502–523.

[21] 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.

[22] Michael P O'brien. 2003. Software comprehension–a review & research direction. *Department of Computer Science & Information Systems University of Limerick, Ireland, Technical Report* (2003).

[23] Nancy Pennington. 1987. Comprehension strategies in programming. In *Empirical studies of programmers: second workshop*. Ablex Publishing Corp., 100–113.

[24] Shari Lawrence Pfleeger, Les Hatton, Charles C Howell, and Chuck Howell. 2002. *Solid software*. Prentice Hall.

[25] Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. 2012. How do professional developers comprehend software?. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 255–265.

[26] Carsten Schulte, Tony Clear, Ahmad Taherkhani, Teresa Busjahn, and James H Paterson. 2010. An introduction to program comprehension for computer science educators. In *Proceedings of the 2010 ITiCSE working group reports*. ACM, 65–86.

[27] Ven Yu Sien. 2011. An investigation of difficulties experienced by students developing unified modelling language (UML) class and sequence diagrams. *Computer Science Education* 21, 4 (2011), 317–342.

[28] Sabine Sonnentag. 1998. Expertise in professional software design: A process study. *Journal of applied psychology* 83, 5 (1998), 703.

[29] Juha Sorva, Jan Lönnberg, and Lauri Malmi. 2013. Students' ways of experiencing visual program simulation. *Computer Science Education* 23, 3 (2013), 207–238.

[30] Dave R Stikkolorum and Michel RV Chaudron. 2016. A Workshop for Integrating UML Modelling and Agile Development in the Classroom. In *Proceedings of the Computer Science Education Research Conference 2016*. ACM, 4–11.

[31] Benjy Thomasson, Mark Ratcliffe, and Lynda Thomas. 2006. Identifying novice difficulties in object oriented design. *ACM SIGCSE Bulletin* 38, 3 (2006), 28–32.

[32] Guilherme Travassos, Forrest Shull, Michael Fredericks, and Victor R Basili. 1999. Detecting defects in object-oriented designs: using reading techniques to increase software quality. In *ACM Sigplan Notices*, Vol. 34. ACM, 47–56.

[33] Frances M Wijnen, Yvonne G Mulder, Stephen M Alessi, and Lars Bollen. 2015. The potential of learning from erroneous models: comparing three types of model instruction. *System dynamics review* 31, 4 (2015), 250–270.

[34] Benjamin Xie, Greg L Nelson, and Andrew J Ko. 2018. An explicit strategy to scaffold novice program tracing. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 344–349.