Research Statement

I am a computing education researcher, and I draw inspiration from diverse fields such as software engineering, computing education and educational technology for my research. I have expertise in designing technology-enhanced learning environments for topics in computer science, such as software design and computer networks. I have conducted several empirical studies with students to explore the effectiveness of these learning environments, using both quantitative and qualitative methods, and have published findings of my research in reputable conferences such as ICSE and ICER.

I am broadly interested in **understanding how learners reason with software artifacts** – such as code and software designs, and in designing effective technologies to improve their reasoning. I believe this is a significant research goal, as "reasoning" is at the heart of what software professionals do as they create, modify, reuse and test software. Towards this goal, as part of my doctoral research, I investigated how learners evaluate a given software design (UML diagrams), and the difficulties they faced. This led me to develop a technology-enhanced learning environment VeriSIM, which enables learners to better understand and reason with a software design.

My future research agenda involves **studying how students reason with programs, specifically as they comprehend and debug code**. I intend to develop tools and strategies to help students effectively plan, and reflect on their programming process. This ability to think about and reflect upon one's thought process is referred to as metacognition. Specifically, I am interested in developing metacognitive scaffolds like hints, programming strategies etc. and incorporate these into programming environments. I then intend to empirically evaluate how these strategies and programming environments are useful in developing students' program comprehension and debugging skills.

Another research project I intend to pursue is **to understand how live-coding videos are contributing towards student learning and develop effective strategies for such live coding sessions.** Live coding is the process of writing code live on a computer. Developers and even instructors use video platforms such as YouTube and Twitch to perform live coding. I intend to analyse such videos to understand in what ways they help students learn programming concepts, and develop tools and strategies which will promote effective teaching and learning of programming using live coding.

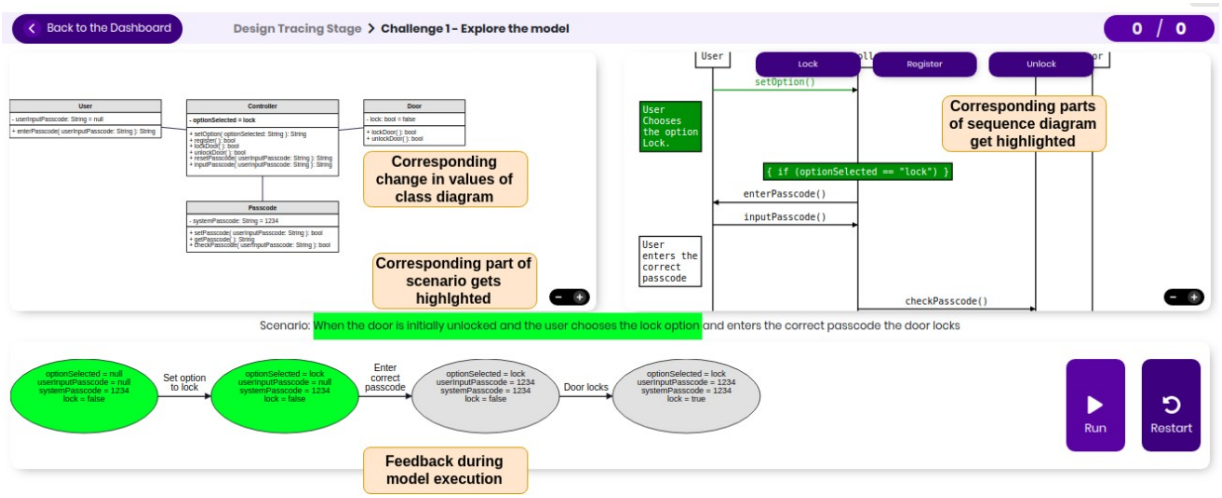
Thesis Topic: Fostering Software Design Evaluation Skills in Computing Undergraduates using a Technology-enhanced Learning Environment

Evaluating software designs is an important practice required of software engineering students. When students graduate and enter the software industry, they usually work on large existing systems, and are required to design additional features based on new requirements. This requires them to comprehend an already existing design, incorporate the required feature in the design, and evaluate if the design satisfies the intended goals. However, students face various difficulties in understanding and evaluating a design. For example, studies have shown that students find it difficult to develop an integrated understanding of different structural and behavioural diagrams in the design.

I analysed existing literature and conducted studies with students in order to understand how they evaluate a software design, and the difficulties they face. In studies with students, I investigated what mental models students use while evaluating a given software design. I adapted the "Block Model", a framework used in program comprehension, and extended it for UML design diagrams. I used qualitative research methods such as content analysis and video data analysis to analyse data. The key insight which I gained from existing work and novice studies is that effective evaluation depends on the mental model which students create based on the requirements and the design. Experts create a rich mental model of the system and simulate various scenarios in the design. However, novices have difficulties in simulating such scenarios, and their models focussed only on superficial aspects of the design. They also have difficulties in simulating the dynamic behaviour represented in the design. Findings from these studies have been published as a full paper in ICER 2020 [1].

These insights form the basis of the proposed learning environment: **VeriSIM** (Verifying designs by SIMulating scenarios). VeriSIM trains students to identify and model scenarios in the design, using the design tracing strategy. 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, 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.

In VeriSIM, students go through four challenges, in increasing order of difficulty, which progressively help them trace a given scenario. In the first challenge, learners explore the given state diagram, which corresponds to the trace of a scenario. They then correct an incorrect state diagram, complete an incomplete state diagram, and create a state diagram in the final challenge. VeriSIM also provides learners tools to construct and refine their model. In every challenge, VeriSIM provides students opportunities to visualize the execution of the state diagram by clicking on a "Run" button. VeriSIM includes features such as evaluation questions, scaffolds for articulation and reflection, and a pedagogical agent to guide students as they model and trace different scenarios in the design.



Affordance of the run feature in VeriSIM

I conducted classroom studies with students to evaluate the effectiveness of VeriSIM and its features. Findings from these studies show that VeriSIM improved students' ability to trace scenarios and evaluate the design against the given requirements. Student perceived that VeriSIM helped them develop an integrated understanding of the design, and also in identifying missing features and defects in the design. I also analyzed interaction logs of students in order to understand how they interacted with VeriSIM. The key finding is that the order of activities in VeriSIM facilitate student understanding of modelling a scenario. Findings from these classroom studies have been published in ICSE 2020 [2] and ICALT 2021 [3], and as a poster in Koli Calling 2020 [4].

The major contributions of my doctoral thesis are the VeriSIM pedagogy, and the features in the VeriSIM learning environment. The thesis also contributes towards computing education research and software engineering by extending the theory of program comprehension, and characterization of expert and novices processes in software design evaluation.

Future Research Agenda:

1. Developing metacognitive scaffolds for effective comprehension and debugging:

Metacognition is the ability to think about and reflect upon one's thought process. It involves knowledge of one's own planning strategies for learning and problem-solving. Metacognitive strategies have been studied in fields such as psychology and education, and there is strong evidence that improving metacognitive skills in students leads to better learning outcomes. However, their use in computing disciplines is only beginning to be explored.

Metacognitive strategies have been developed for programming - such as decomposing a problem, solving a test case after reading a problem prompt, etc. The focus of these strategies has primarily been on creating programs. There are equally important skills required for programming, such as comprehension and debugging. Comprehension and debugging also require high levels of metacognition, because these skills require understanding the "thinking" of the behavior and purpose of an already existing program, and involves mapping thoughts and intentions one had while writing a program, to the output of that program. There is a lack of emphasis on developing specific metacognitive strategies for program comprehension and debugging, as well as how these strategies can fit into existing learner activities and programming environments.

As part of my future research agenda, I intend to develop and embed metacognitive scaffolds in programming environments and IDEs, specifically to aid comprehension and debugging in students. For example, prompts can be shown in a debugger to help students reflect on their debugging process. Error messages can be made more understandable for learners. Prompts can be provided in an IDE at specific areas in order to aid comprehension. I believe this will lead to better tools for novice programmers, as well as further enhance the design of existing programming environments.

I intend to investigate the effectiveness of these tools by conducting quantitative as well as qualitative studies with students. I will examine whether students who use the tool show improvements in program comprehension and debugging. Interviews and think-aloud studies with students will help determine how and why these tools are useful. I also intend to examine students' interaction patterns of using the tool.

Interactions such as use of metacognitive scaffolds, mouse clicks, keystrokes, navigation across the tool and tool elements, can be logged in the tool. Various AI techniques such as pattern mining and clustering can be used to examine differences in students' problem solving processes. Findings from the think-aloud studies and the analysis of behaviour patterns in the tool together can contribute towards how students are using metacognitive strategies during program comprehension and debugging.

Potential undergraduate student projects can be the development of metacognitive programming tools and IDEs, and use of AI/ML techniques to analyse interaction data. Potential PhD topics can be studying the effectiveness of these strategies in students, and how these strategies can be adopted in classrooms by instructors. I envision that the findings of this research will lead to widespread adoption of these strategies in classrooms and programming environments, leading to better metacognitive skills for students and improved learning outcomes.

2. Developing Effective Tools and Strategies for Live-coding

Live coding is the process of writing code live on a computer. In recent times, several developers as well as instructors have been doing live coding. Research has shown that developers are motivated to do live coding for various reasons, ranging from knowledge sharing, socialising with others, and building an online identity. However, such live coding sessions have their own limitations, such as maintaining engagement with learners, as well as inability for learners to explore the code along with the developer.

As part of this research, I intend to focus on instructors doing live coding and develop strategies which will help students learn better from sessions. There are several live-coding videos available on platforms like YouTube, Twitch etc. MOOCs such as Coursera, EdX and NPTEL have programming courses where instructors do live sessions. Analysis of videos on these platforms can be done to identify what strategies instructors use while live coding. For example, these videos can be analysed based on whether instructors are providing sufficient time for reflecting on their own programming process. Logs of such videos contain important information such as video engagement, likes, comments etc. Data mining of such logs can be done to gain additional insights. Such analysis can provide guidelines for instructors to help them plan effective live coding sessions.

Another research thread which I intend to pursue is to extract useful information from live coding videos. Live coding videos are typically quite long. However, they contain important information which can be beneficial to learners. For example, a developer might explain an important programming concept in between a long debugging session. I intend to create tools which use AI techniques to automatically infer important episodes from long live coding sessions. These episodes can be made available to learners which can help them gain important programming insights without watching the entire livestream.

I believe these tools and strategies can help instructors plan better live coding sessions, as well as help students use live coding resources as an effective way to gain awareness and appreciation of what experts do while they perform a programming task.

Relevant Publications:

[1] **Prasad, P.,** & Iyer, S. (2020, August). How do Graduating Students Evaluate Software Design Diagrams?. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (pp. 282-290). https://doi.org/10.1145/3372782.3406271

[2] **Prasad, P.**, & Iyer, S. (2020, June). VeriSIM: a learning environment for comprehending class and sequence diagrams using design tracing. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training* (pp. 23-33). https://doi.org/10.1145/3377814.3381705

[3] Satavlekar, S., Nath, D., Priyadarshini, R., **Prasad, P.**, Singh, D. K., & Rajendran, R. (2021, July). Unraveling Learner Interaction Strategies in VeriSIM for Software Design Diagrams. In *2021 International Conference on Advanced Learning Technologies (ICALT)* (pp. 308-310). IEEE.

[4] **Prasad, P.**, & Iyer, S. (2020, November). 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 (pp. 1-2). https://doi.org/10.1145/3428029.3428564