Research Statement

I am a computing education researcher, and I draw inspiration from domains such as software engineering, learning sciences and computing education for my research. I have expertise in designing technology-enhanced learning environments for topics in computer science, such as software design and computer networks, and in other domains like geometry and biology. I have conducted several studies which explore the effectiveness of these learning environments, using both quantitative and qualitative methods.

I am broadly interested in understanding how learners reason with software artifacts, and in designing effective pedagogies to improve their reasoning. In 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 comprehend class and sequence diagrams using the design tracing pedagogy.

My future research agenda involves understanding the role metacognition plays in students' being able to better reason with programs. Specifically, I am interested in exploring how metacognitive strategies are useful in developing students' program comprehension and debugging skills. I intend to investigate how metacognitive strategies like diagramming, and providing visualizations of a program trace affect students' program comprehension and debugging, and how metacognitive scaffolds can be incorporated into programming environments.

Developing Software Design Evaluation Skills in Computing Undergraduates using a Technology-enhanced Learning Environment

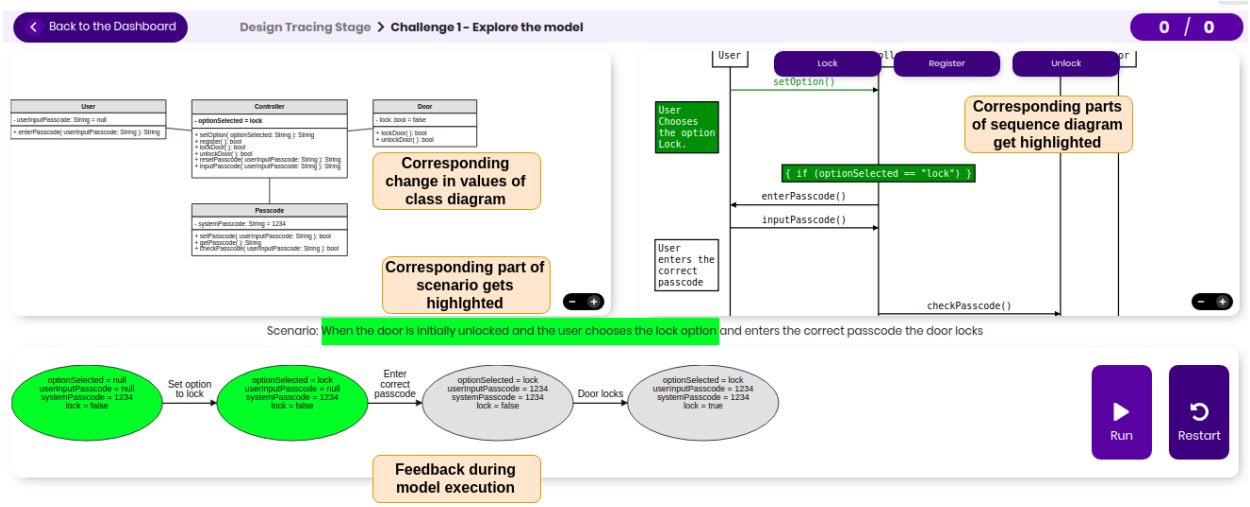
Evaluating software designs is an important practice required of software engineering students. When students graduate and enter the software industry, they usually work on large existing systems, and are required to design additional features based on new requirements. This requires them to comprehend 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, students find it difficult to develop an integrated understanding of different structural and behavioural diagrams in the design [1].

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" [2], a framework used in program comprehension, and extended it for UML design diagrams. I used qualitative research methods such as interpretive 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 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 [3].

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.

Activities and features in VeriSIM are informed by theories in learning sciences and science education such as model-based learning [4] and model-order progression [5]. 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 as a full paper in ICSE 2020 [6], and as a poster in Koli Calling 2020 [7].

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 by extending the theory of program comprehension, and characterization of expert and novices processes in software design evaluation.

Future Research Agenda:

Developing metacognitive strategies for 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 investigate how metacognitive strategies like diagramming and providing visualizations can help in improving students' comprehension and debugging skills. I am also interested in exploring how these strategies can be incorporated into existing programming environments and IDEs. I describe these future research directions below.

Diagramming as a metacognitive scaffold for comprehension and debugging:

Studies in computing as well as other STEM subjects like physics, have shown that diagramming can help learners' understand abstract concepts, and can improve problem solving. In computing education, studies have also shown that students who sketched a diagram were more successful on code reading problems involving loops, arrays, and conditionals [8]. Although such studies show the benefits of diagramming, there haven't been specific diagramming strategies or scaffolds introduced to help students reflect on their thinking about a program's execution, and while debugging a program. My research aims to investigate whether provide explicit scaffolds to create diagrams while comprehending and debugging can lead to better performance in these activities.

Scaffolding students with visualizations of program trace:

Program tracing has shown to be an effective strategy in helping students' comprehend a given program. However, how tracing can be taught has not been given sufficient emphasis. In my thesis, I adapted the concept of model-order progression from science education and have shown its effectiveness in the case of modelling scenarios in design diagrams. I believe that the model-based learning paradigm and model order progression can serve as teaching strategies for tracing as well. For example, an instructor can scaffold students to first explore a trace, and then correct a trace, followed by completing an incorrect trace of a program. Such progression activities can serve as metacognitive scaffolds, and can help students reflect on the process of tracing, which in turn can enhance their program comprehension skills.

Providing metacognitive scaffolds in programming environments:

Metacognitive strategies in programming have primarily been paper-based strategies, and have also been incorporated in automated assessment tutors (AAT), code editors, and as visualizations tools. Metacognitive scaffolds can also be embedded in programming environments and IDEs. For example, prompts can be shown in a debugger to help students reflect on their debugging process. Another example can be to provide prompts in an IDE at specific areas in order to aid comprehension. I believe such features will further enhance the design of existing programming environments, and can help environments move towards providing better metacognitive scaffolds for novice programmers.

Other Projects:

Gesture Based Learning of Geometry:

This project was a part of a research collaboration with T.G. Lakshmi, Soumya Narayana , Dr. Sahana Murthy from IIT Bombay, and Dr. Sanjay Chandrasekharan from HBCSE, Mumbai. We developed **Geometry via Gestures (GvG)**, a web-based application which enables learners to interact with 3D objects using their gestures. GvG interfaces with the Leap Motion Controller, in order to provide gesture support to students. We conducted an exploratory study to examine students' perception of learning while using GvG. The findings from this study show that students learn about structure and property of 3D geometrical objects after using GvG. These findings have been published as full papers in T4E 2016 [9] and ICCE 2016 [10].

Developing Game-Based Virtual Environments:

In 2016, I collaborated in two projects with Dr. Maiga Chang, from Athabasca University. We developed two game-based learning environments to teach concepts in biology and computer networks. We built these learning environments in Open Wonderland, an open source software for creating 3D virtual worlds. The design of these virtual environments have been published in ICALT 2016 [11][12].

References:

[1] Burgueño, L., Vallecillo, A., & Gogolla, M. (2018). Teaching UML and OCL models and their validation to software engineering students: an experience report. *Computer Science Education*, *28*(1), 23-41. https://doi.org/10.1080/08993408.2018.1462000

[2] Schulte, C. (2008, September). Block Model: an educational model of program comprehension as a tool for a scholarly approach to teaching. In *Proceedings of the Fourth international Workshop on Computing Education Research* (pp. 149-160). https://doi.org/10.1145/1404520.1404535

[3] **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

[4] Buckley, B. C., Gobert, J. D., Horwitz, P., & O'Dwyer, L. M. (2010). Looking inside the black box: assessing model-based learning and inquiry in BioLogica™. *International Journal of Learning Technology*, *5*(2), 166-190.

[5] Mulder, Y. G., Lazonder, A. W., & de Jong, T. (2011). Comparing two types of model progression in an inquiry learning environment with modelling facilities. *Learning and Instruction*, *21*(5), 614-624.

[6] **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

[7] **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

[8] Cunningham, K., Blanchard, S., Ericson, B., & Guzdial, M. (2017, August). 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* (pp. 164-172). https://doi.org/10.1145/3105726.3106190

[9] Narayana, S., **Prasad, P**., Lakshmi, T. G., & Murthy, S. (2016, December). Geometry via Gestures: Learning 3D geometry using gestures. In *2016 IEEE Eighth International Conference on Technology for Education (T4E)* (pp. 26-33). IEEE. https://doi.org/10.1109/T4E.2016.014

[10] Lakshmi, T. G., Narayana, S., **Prasad, P.**, Murthy, S., & Chandrasekharan, S. (2016). Geometry-via-Gestures: Design of a gesture based application to teach 3D Geometry. In Proceedings of the 24th international conference on computers in education (pp. 180-189). Mumbai, India: Asia-Pacific Society for Computers in Education.

[11] Deep, A., **Prasad, P.**, Narayana, S., Chang, M., & Murthy, S. (2016, July). Game Based Learning of Blood Clotting Concepts. In 2016 IEEE 16th International Conference on Advanced Learning Technologies (ICALT) (pp. 526-530). IEEE https://doi.org/10.1109/ICALT.2016.70

[12] Alse, K., Ganesh, L., **Prasad, P.**, Chang, M., & Iyer, S. (2016, July). Assessing Students' Conceptual Knowledge of Computer Networks in Open Wonderland. In 2016 IEEE 16th International Conference on Advanced Learning Technologies (ICALT) (pp. 513-517). IEEE https://doi.org/10.1109/ICALT.2016.22