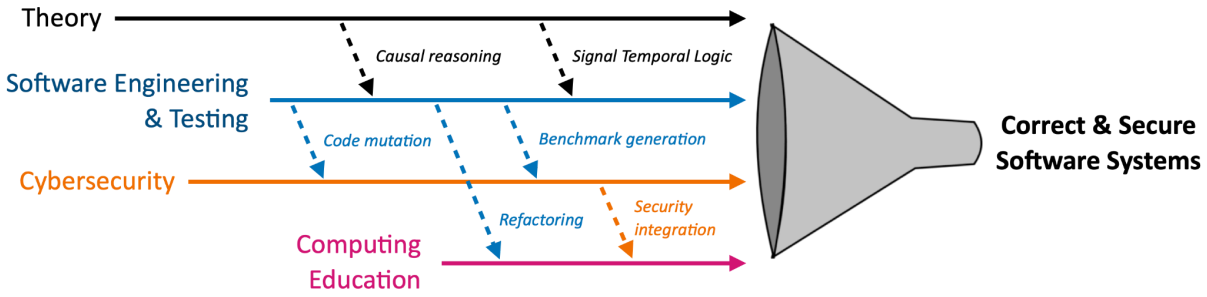
Research Statement

Chris Poskitt
School of Computing and Information Systems
Singapore Management University
Tel: (65) 6828-1376; Email: cposkitt@smu.edu.sg
16th December 2025

Research Overview

My research focuses on **engineering correct and secure software systems**, aiming to make them reliable and trustworthy, especially in contexts where dependability is crucial. I am particularly interested in adapting traditional software quality techniques to non-traditional and complex systems such as autonomous vehicles, critical infrastructure, and (more recently) AI agents. I work at the **intersection of theory and practice**, applying rigorous software theory to develop practical solutions for emerging domains. For instance, using signal temporal logic to formally define traffic laws, then fuzzing to find ways an autonomous vehicle could inadvertently violate them. Additionally, I explore how insights from my research can be translated to the classroom and help prepare the next generation of software engineers.



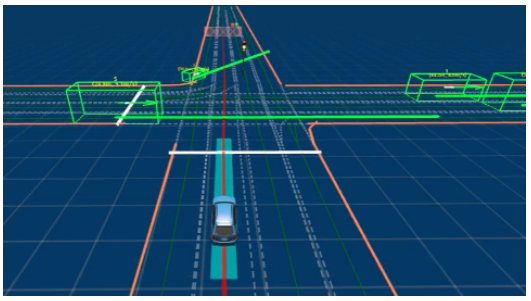
Some of the key overarching questions my research aims to address include:

- How can traditional approaches for software quality be generalised to complex cyber-physical systems?
- How do we build defence mechanisms for critical infrastructure that effectively detect/prevent attacks?
- How do we intelligently fuzz cyber-physical systems to uncover rare (but feasible) scenarios that violate safety properties?
- How can undergraduate curricula better prepare students to engineer correct, secure, and maintainable code?

Recent Research Highlights

In the following, I summarise some recent results from projects I have collaborated on.

Fuzzing, Enforcing, and Repairing Autonomous Vehicles



Our research on autonomous vehicles adapts rigorous software engineering techniques to the problem of driving safety, spanning specification, testing, enforcement, explanation, and repair. We began by formalising traffic laws using signal temporal logic and developing LawBreaker [1], a fuzzing framework for systematically uncovering law violations by autonomous driving systems (ADS). We then moved from detection to prevention with REDriver [2], a runtime enforcement framework that monitors planned trajectories and applies minimal, gradient-based corrections when violations are predicted. To explain why violations and accidents occur, we proposed ACAV [3], which performs automated causal analysis on AV accident records to identify the events responsible for safety failures. Most recently, in FixDrive [4], we closed the loop by introducing an automated repair framework that analyses near-misses/violations and generates general, interpretable driving-strategy repairs. FixDrive leverages multimodal large language models to translate critical moments from driving records into high-level rules that generalise beyond individual incidents, integrate with existing ADS architectures, and can be computed efficiently offline, enabling practical and iterative improvement of autonomous driving behaviour.

1. *LawBreaker: An Approach for Specifying Traffic Laws and Fuzzing Autonomous Vehicles*, Y. Sun, C.M. Poskitt, J. Sun, Y. Chen, Z. Yang. **ASE'22**
2. *REDriver: Runtime Enforcement for Autonomous Vehicles*, Y. Sun, C.M. Poskitt, Z. Zhang, and J. Sun. **ICSE'24**
3. *ACAV: A Framework for Automatic Causality Analysis in Autonomous Vehicle Accident Recordings*, H. Sun, C.M. Poskitt, Y. Sun, J. Sun, and Y. Chen. **ICSE'24**
4. *FixDrive: Automatically Repairing Autonomous Vehicle Driving Behaviour for $0.08 per Violation*, Y. Sun, C.M. Poskitt, K. Wang, and J. Sun. **ICSE'25**

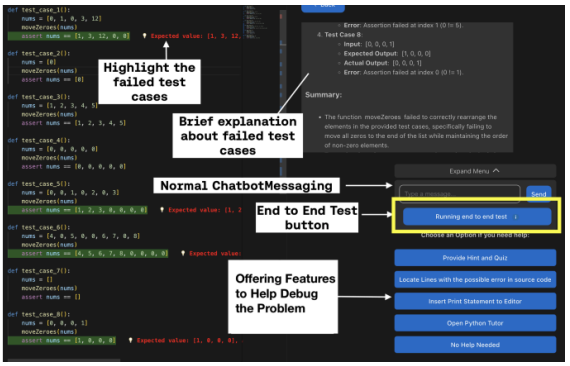
Engineering Robust Cyber-Physical Systems

Our research on cyber-physical systems (CPS), particularly in the context of critical infrastructure, focuses on strengthening system dependability through principled testing, semantic foundations, and adaptive defence. We first addressed the problem of inadequate test diversity in industrial control systems by developing a causality-guided fuzzing approach [5] that identifies the minimal causal events responsible for unsafe physical states and actively steers the search toward causally distinct tests, substantially improving coverage on a real-world water treatment system. To underpin rigorous analysis and testing of CPS software, we introduced K-ST [6], a formal executable semantics for the Structured Text language used in PLCs, enabling systematic validation of compiler correctness and uncovering previously unknown defects. More recently, we have shifted from testing in isolation to strengthening CPS defences themselves: in Evo-Defender [7],

we proposed an evolutionary attacker–defender framework in which guided fuzzing and incremental learning are coupled to iteratively harden anomaly detectors against increasingly subtle and diverse attacks, achieving significantly improved detection on realistic CPS testbeds. Complementing these technical contributions, we conducted a systematic literature review of CPS security modelling [8], synthesising the fragmented landscape of threat and attack modelling approaches and identifying fundamental gaps in how existing methods capture the dynamic, multi-stage, and cross-domain nature of real CPS attacks.

5. *Finding Causally Different Tests for an Industrial Control System*, C.M. Poskitt, Y. Chen, J. Sun, and Y. Jiang. **ICSE'23**
6. *K-ST: A Formal Executable Semantics of the Structured Text Language for PLCs*, K. Wang, J. Wang, C.M. Poskitt, X. Chen, J. Sun, and P. Cheng. **TSE'23**
7. *Developing a Strong CPS Defender: An Evolutionary Approach*, Q. Hu, C.M. Poskitt, J. Sun, and Y. Chen. **RAID'25**
8. *Security Modelling for Cyber-Physical Systems: A Systematic Literature Review*, S. Huang, C.M. Poskitt, and L.K. Shar. **TCPS'25**

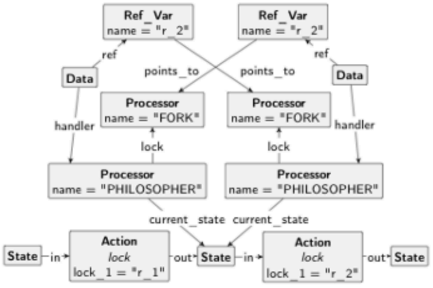
Software Engineering Education



Our research in software engineering education focuses on designing learning interventions and tools that help students actively develop core engineering skills—such as refactoring, debugging, and independent learning—rather than passively consuming solutions. In one line of work [9], we addressed the challenge of teaching code refactoring to novices through a *mistake-based* approach in which students first construct functionally correct but intentionally "smelly" code before refactoring it, improving familiarity, confidence, and transfer to their own practice. In a complementary strand, we investigated how students learn to debug programs, proposing *Simulated Interactive Debugging* [10], an AI-assisted approach that guides students step-by-step through the debugging process without revealing solutions. Building on this, we developed *CodeHinter* [11], an interactive debugging assistant that combines fault localisation with LLM-generated hints and quizzes to support novice debuggers while mitigating over-reliance on AI-generated fixes. Finally, to support independent learning beyond coding tasks, we introduced *Slide++* [12], a platform that automatically augments lecture slides with credible and relevant supplementary resources. Together, these works reflect a cohesive agenda centred on scaffolding learning through principled use of automation and AI, while preserving student agency and skill development.

9. *Fixing Your Own Smells: Adding a Mistake-Based Familiarisation Step When Teaching Code Refactoring*, I. Tan and C.M. Poskitt. **SIGCSE'24**
10. *Simulated Interactive Debugging*, Y. Noller, E. Chandra, S. Chandrashekar, K. Choo, C. Jegourel, O. Kurniawan, and C.M. Poskitt. **ASE'25 NIER**
11. *Designing for Novice Debuggers: A Pilot Study on an AI-Assisted Debugging Tool*, O. Kurniawan, E. Chandra, C.M. Poskitt, Y. Noller, K.T.W. Choo, and C. Jegourel. **Koli Calling'25**
12. *Towards Automated Slide Augmentation to Discover Credible and Relevant Links*, D. Dinushka, C.M. Poskitt, K.C. Koh, H.N. Mok, and H.W. Lauw. **AIED'24 LBR**

Reasoning about Graph Manipulations



As part of my PhD, I investigated formal techniques for reasoning about programs whose behaviour is naturally modelled using graphs and algebraic graph transformations. Although this is no longer my primary research focus, I continue to contribute to this area. In particular, in recent work [13,14], we generalised Peter O'Hearn's incorrectness logic to graph programs. Unlike traditional program logics, which establish properties that must hold for all executions, incorrectness logic is designed to prove the existence of specific executions, making it especially well suited to graph programs, where high nondeterminism is intrinsic to the computation model. This work provides a principled foundation for reasoning about bug existence and may serve as a basis for sound symbolic execution and bug-finding techniques for graph-based programming languages.

13. *Monadic Second-Order Incorrectness Logic for GP 2*, C.M. Poskitt and D. Plump. **JLAMP'23**
14. *Incorrectness Logic for Graph Programs*, C.M. Poskitt. **ICGT'21**