WASP Software Engineering Assignment

Yansong Huang, CHALMERS
yansong.huang@chalmers.se

August 21, 2025

1 Introduction / Abstract of my research area

I am a PhD student at the Division of Vehicle Engineering and Autonomous Systems. I work specifically in the Vehicle Dynamics group. In this section, I will introduce vehicle engineering briefly. Then I will explain the details of my PhD project.

We see that, among other things, electrification and fierce global competition place ever higher demands on faster and more efficient development of new vehicle concepts, even within a "classic" area such as wheel suspension design. The wheel suspension is one of the most architecture-heavy systems in the car and much of the car's overall characteristics are determined by how well we succeed with the development of the wheel suspension. A clear bottleneck in the development of a new wheel suspension today are the complex performance requirements and which today require time-consuming calculations to evaluate for each iteration of the design. Volvo Cars has already done some development of methods where we can directly go from requirements to a reduced set of possible design parameters by utilizing certain linear relationships between some requirements and the characteristics of a wheel suspension. This has proven to be an effective method for reducing lead time, but today the method only covers a small amount of requirements. Standard methods in optimization have proved difficult to use in the design of a wheel suspensions because it is easily caught in local optima that do not work in practice. The experience is that we therefore often devote more time to creating, constraining and controlling the optimization than it typically takes to manually iterate to a solution. This research project focus on developing improved methods, tools and knowledge combined with AI to speed up solution delivery time for chassis systems.

2 Two principles from Robert's lectures and how they relate to my research

In this section, two principles from Robert's lectures that relate to my research will be discussed. The principles are Black Box testing, and version control.

2.1 Principle 1: Black Box testing

Black box testing is a software testing methodology where the internal structure, implementation details, and code of the system under test are not known to the tester. The tester focuses solely on the inputs and expected outputs, treating the software as a "black box." In automotive software development, black box testing is particularly crucial due to safety-critical requirements and regulatory compliance. For example, when testing Electronic Control Units (ECUs) for functions like Anti-lock Braking Systems (ABS) or Engine Control Modules (ECM), testers validate that the

system responds correctly to various input scenarios (brake pedal pressure, wheel speed sensors, throttle position) without needing to understand the internal algorithms. This approach is essential in automotive development because it mirrors real-world usage conditions, ensures compliance with automotive standards like ISO 26262, and allows for independent verification by third-party testing organizations. In my research on vehicle dynamics and chassis systems, black box testing principles could be applied when validating the performance of wheel suspension optimization algorithms, where we focus on whether the system meets performance requirements (ride comfort, handling characteristics) regardless of the specific optimization method implemented internally.

2.2 Principle 2: Version Control

Version control systems like Git are essential tools for managing code changes and enabling effective collaboration in software development. Git allows multiple developers to work on the same codebase simultaneously by tracking all changes, maintaining a complete history of modifications, and providing mechanisms to merge different contributions safely. The benefits for collaboration are substantial: team members can work on different features in parallel through branching, changes can be reviewed before integration through pull requests, and any problematic modifications can be easily reverted. In my vehicle dynamics research, version control is crucial when developing optimization algorithms for wheel suspension design, as it allows me to experiment with different approaches while maintaining stable versions, collaborate with colleagues on algorithm improvements, and ensure that all computational results can be reproduced by tracking the exact code version used. Additionally, Git's distributed nature means that each team member has a complete backup of the project history, providing resilience against data loss and enabling offline work.

3 Two principles from SAAB's guest lecture and how they relate to my research

In this section, I will talk about two concepts from SAAB's guest lecture. The two concepts are Software development lifecycle and Autonomous decision making.

3.1 Software development lifecycle

The Software Development Lifecycle (SDLC) is a systematic approach to developing software that is particularly crucial in automotive applications. In automotive contexts, SDLC must address safety-critical requirements, real-time constraints, and regulatory compliance standards like ISO 26262. The key phases include Requirements Engineering (defining safety and performance criteria), Architecture (designing system structure and ECU integration), Implementation (coding in C/C++ for real-time systems), Verification (extensive testing including Hardware-in-the-Loop), and Deployment (manufacturing installation and OTA updates).

In my vehicle dynamics research, SDLC principles are highly relevant when developing optimization algorithms for wheel suspension design. The requirements phase helps define performance criteria for ride comfort and handling characteristics. The architecture phase structures the optimization framework and integration with existing vehicle systems. Implementation involves coding algorithms in Python or MATLAB. Verification ensures the optimization results meet real-world performance requirements through simulation and testing. Finally, deployment involves integrating validated algorithms into Volvo's development tools and processes. This systematic approach is essential in my research because wheel suspension systems are safety-critical components that directly

affect vehicle stability and passenger safety. Following SDLC principles ensures that optimization tools I develop are reliable, maintainable, and meet automotive industry standards.

3.2 Autonomous application

Autonomous application in software systems involves algorithms that can make real-time decisions without human intervention, which is particularly critical in safety-critical applications like automotive systems. In modern vehicles, autonomous decision-making systems are implemented in various Electronic Control Units (ECUs) that must respond to sensor inputs within strict timing constraints. For example, Electronic Stability Control (ESC) systems autonomously decide when to apply individual wheel brakes based on vehicle dynamics measurements, while Adaptive Cruise Control systems make speed and following distance decisions based on radar and camera inputs.

In my vehicle dynamics research, autonomous application principles are highly relevant for developing intelligent wheel suspension systems. The optimization algorithms I develop could potentially be integrated into adaptive suspension systems that autonomously adjust damping characteristics based on real-time road conditions, vehicle speed, and driving style. Such systems would need to make rapid decisions about suspension settings to optimize ride comfort and handling performance without driver intervention. Furthermore, as vehicles become increasingly autonomous, the wheel suspension optimization algorithms could interface with higher-level autonomous driving systems to anticipate and prepare for different driving scenarios, such as adjusting suspension stiffness before cornering or optimizing damping for highway versus city driving conditions. This application of autonomous application ensures that vehicle dynamics are continuously optimized for safety and performance.

4 Data Scientists versus Software Engineers

This section explores the key differences between data scientists and software engineers based on the insights from chapters "1. Introduction" and "2. From Models to Systems" of the CMU "Machine Learning in Production" book (`https://mlip-cmu.github.io/book/01-introduction.html`).

4.1 Question 1

The authors do accurately describe observable differences that stem from educational backgrounds and professional contexts. Data science programs typically emphasize statistics, algorithms, and optimizing model accuracy on clean datasets, while software engineering curricula focus on requirements gathering, system architecture, testing, and building complete products for end users. These different educational paths genuinely create different mental models, with data scientists often optimizing for accuracy metrics on test sets while software engineers optimize for user satisfaction, maintainability, and business goals within real-world constraints. The workflow differences are also real - the exploratory, notebook-based approach common in data science does create friction with the more structured, specification-driven workflows typical in traditional software engineering. However, I believe many of these differences are circumstantial rather than essential characteristics of the people themselves. These distinctions largely reflect how the fields evolved historically - data science emerged from statistics and academia with their emphasis on research and experimentation, while software engineering grew from industry needs to build reliable systems at scale. The "unicorn" framing for people with both skillsets seems to perpetuate the idea that these skills are naturally incompatible, but I've observed many developers successfully learn across both domains when organizational circumstances require it. The rarity might be more about career

incentive structures and specialization pressure than inherent difficulty in acquiring cross-domain skills. What concerns me most is the risk of overgeneralization, even though the authors acknowledge their characterizations are "oversimplified." If we tell data scientists they inherently "don't enjoy infrastructure work" or suggest that software engineers approach machine learning "naively," we risk creating self-fulfilling prophecies where people internalize these identity boundaries. In reality, what we're often seeing is specialization pressure in large organizations where people focus deeply on one area because there's simply too much to learn, career advancement rewards deep expertise over breadth, and team structures reinforce these professional boundaries.

4.2 Question 2

I believe we're heading toward greater integration of skills across traditional boundaries, with the dominant trend being convergence rather than further specialization, driven by several key technological and organizational forces. The most compelling evidence for convergence comes from AI-assisted development tools like GitHub Copilot and ChatGPT, which are dramatically lowering barriers to cross-domain skill acquisition. Data scientists can now generate production-quality code without years of software engineering training, while software engineers can implement ML pipelines without deep statistical knowledge. These tools are democratizing technical skills that previously required extensive specialization, making the "unicorn" problem less about rare individuals and more about having the right tooling. Modern ML infrastructure is also forcing convergence. Platforms like Hugging Face and cloud ML services abstract away much of the traditional complexity that created the data science/software engineering divide. When model deployment becomes as simple as pushing to Git or calling an API, the boundary between "model building" and "system building" blurs. The rise of foundation models and prompt engineering is shifting ML work from deep algorithmic expertise toward engineering-focused concerns like prompt optimization and multi-model orchestration. However, complete role merger won't happen. Instead, I expect new specialization patterns that cut across traditional boundaries. We're already seeing "ML Platform Engineers," "AI Product Managers," and "ML Systems Architects" - roles requiring deep understanding of both domains while specializing in integration challenges. Business pressures strongly favor this integration, as the authors note that "87 percent of machine-learning projects fail" largely due to coordination problems between separate teams. Educational institutions are responding with "ML Engineering" programs that explicitly combine both curricula, suggesting the next generation will enter with more integrated skill sets. I predict that within five to ten years, we'll see a dominant "ML Systems Engineer" role combining core competencies from both domains - ML fundamentals, systems thinking, software engineering practices, and product intuition. Traditional "pure" specialists will still exist in research or large platform companies, but for most organizations building ML-enabled products, integration skills will become more valuable than narrow specialization. This mirrors web development's evolution, where "full-stack" developers became the norm despite initial front-end/back-end specialization. As tooling matured and abstractions improved, the business value of integration skills eventually outweighed narrow specialization for most use cases.

5 Paper analysis

In this section, I will analyze two papers from the CAIN (Conference on AI Engineering) conference series to understand current research trends in AI engineering and their relevance to my vehicle dynamics research.

5.1 Paper 1: Dataflow graphs as complete causal graphs https://arxiv.org/pdf/2303.09552

5.1.1 Core Ideas and Their Software Engineering Importance

The paper by Paleyes et al. presents a groundbreaking connection between flow-based programming (FBP) and causal inference that has profound implications for AI system engineering. The core insight revolves around the observation that dataflow graphs, which are naturally produced when designing software systems using FBP paradigms, can directly serve as complete structural causal models without requiring additional causal discovery processes. In traditional software architectures like object-oriented or microservice-based systems, understanding the complete data dependency graph requires sophisticated tooling and often remains incomplete due to hidden connections and encapsulated data flows. However, FBP systems inherently expose all data dependencies through their explicit dataflow graphs, where each component represents a causal conditional distribution and connections represent causal relationships.

This breakthrough is particularly crucial for AI systems engineering because modern AI applications suffer from what the authors term "intellectual debt" - the growing difficulty in understanding complex system behaviors as they scale. AI systems typically involve intricate pipelines where data flows through multiple preprocessing steps, feature engineering components, model inference engines, and post-processing modules. When these systems fail or behave unexpectedly, engineers currently rely on manual investigation, expensive monitoring tools, or incomplete dependency analysis. The paper's approach offers a systematic way to trace causality through these complex AI pipelines, enabling automated fault localization, precise business impact analysis, and cost-effective experimentation without running expensive A/B tests on production systems.

5.1.2 Relation to Current Research Focus

For researchers working on distributed AI systems or MLOps practices, the fault localization framework offers a principled approach to understanding failures in complex, multi-component AI architectures. Traditional monitoring approaches focus on individual service metrics, but this causal reasoning framework enables understanding of how failures propagate through the entire system. Additionally, if your research touches on AI system reliability, interpretability, or trustworthiness, the explicit causal modeling provides a foundation for building more transparent and explainable AI systems where the reasoning chain from input data to final decisions can be systematically analyzed and validated.

5.1.3 Integration into a Larger AI-Intensive Project

Consider a comprehensive smart city traffic management system that represents the kind of large-scale AI-intensive project where these ideas would prove invaluable. This fictional system integrates multiple AI components: real-time traffic flow prediction models processing data from thousands of sensors citywide, computer vision systems analyzing traffic camera feeds for incident detection, reinforcement learning algorithms optimizing traffic light timing, route recommendation engines for citizens, and predictive maintenance systems for infrastructure. Each component generates and consumes multiple data streams, creating a complex web of dependencies where understanding causality becomes critical for system operation and maintenance.

In this smart city project, the paper's causal reasoning framework would address several critical challenges. When traffic flow predictions suddenly become inaccurate, Algorithm 1 could systematically trace whether the root cause lies in sensor malfunctions, environmental changes affecting

the data distribution, model drift in the prediction algorithms, or upstream issues in data preprocessing pipelines. For business analysis, city planners could use causal attribution to understand why certain traffic optimization strategies succeed or fail, identifying whether improvements stem from better sensor coverage, algorithmic enhancements, or external factors like citizen behavior changes. The experimentation capabilities would be particularly valuable for testing new traffic management algorithms - rather than deploying changes city-wide and risking traffic disruption, causal inference could predict the system-wide effects of algorithmic modifications.

My WASP research could integrate into this project by developing specialized tools and methodologies for AI-intensive urban systems. This might involve creating domain-specific causal attribution metrics that account for the temporal and spatial dependencies inherent in city-scale systems, developing real-time monitoring frameworks that can detect and explain anomalies as they propagate through the AI pipeline, or establishing safety-critical protocols for causal reasoning in systems that directly impact public welfare. The intersection of causal reasoning, AI systems engineering, and urban computing presents rich opportunities for advancing both theoretical understanding and practical capabilities.

5.1.4 Adaptation of Research Direction

To better support the paper's vision of causally-aware AI engineering, research adaptations should focus on extending the theoretical foundations while addressing the practical challenges of implementing causal reasoning in production AI systems. The current framework, while promising, primarily demonstrates feasibility on relatively simple examples and would require significant development to handle the complexity, scale, and real-time requirements of modern AI systems. Research could focus on developing more sophisticated causal attribution algorithms that account for the probabilistic and often non-deterministic nature of machine learning models, where the same input might produce different outputs due to inherent randomness in neural networks or sampling-based algorithms.

A particularly important research direction involves scaling causal reasoning to large, dynamic AI systems where the dataflow graph itself evolves as models are retrained, new components are deployed, or system architecture changes. Current causal inference methods often assume static graph structures, but AI systems in production undergo continuous evolution. Research could explore how to maintain causal reasoning capabilities in the face of such changes, potentially through automated graph updating mechanisms or hierarchical causal modeling approaches that can adapt to architectural modifications.

Furthermore, the integration of causal reasoning with existing AI engineering practices requires developing new software engineering methodologies. This includes creating design patterns for building AI systems using flow-based programming principles, establishing testing frameworks that leverage causal graphs for more comprehensive validation, and developing monitoring and observability tools that provide causal insights rather than just performance metrics. The research could also explore how causal reasoning capabilities affect AI system design decisions, potentially leading to new architectural patterns that optimize for both performance and causal transparency.

Long-term research adaptation should also consider the intersection with AI ethics and regulatory compliance. As AI systems become subject to increasing regulatory scrutiny, the ability to provide causal explanations for system behavior becomes not just technically valuable but legally necessary. Research could explore how causal graphs can support algorithmic auditing, bias detection, and compliance reporting, potentially developing frameworks that automatically generate causal explanations suitable for regulatory review or help identify potential sources of unfair bias in AI decision-making processes.

5.2 Paper 2: A Data Source Dependency Analysis Framework for Large Scale Data Science Projects https://arxiv.org/pdf/2212.07951

5.2.1 Core Ideas and Their Software Engineering Importance

The paper introduces "data source dependency hell" - a ML-specific problem where data changes cause model failures that can't be traced through code analysis alone. The core solution is an automated static analysis framework that maps all data dependencies in ML pipelines, enabling proactive monitoring instead of reactive debugging. This is crucial for AI systems because data changes happen frequently and silently across organizational boundaries, making traditional dependency tracking insufficient for maintaining reliable ML applications at scale.

5.2.2 Relation to My Research

This framework would complement research in MLOps reliability and AI system testing by providing the foundational dependency graphs needed for targeted testing strategies. If your work involves AI system monitoring or quality assurance, the static analysis approach could support automated test generation and more efficient incident response based on understanding which components are affected by specific data changes.

5.2.3 Integration into a Larger AI-Intensive Project

Consider a financial fraud detection system processing millions of daily transactions across multiple data sources (customer databases, sanctions lists, credit scores, market feeds). The dependency mapping framework would automatically identify all affected fraud models when any data source changes, enabling proactive alerts before detection capabilities are compromised. Your WASP research could extend this with compliance-aware dependency tracking and real-time monitoring for regulatory requirements in financial AI systems.

5.2.4 Adaptation of My Research Direction

To better support the vision of comprehensive data dependency management in AI systems, research adaptations should focus on extending the static analysis framework to handle the increasing complexity and dynamic nature of modern ML systems. The current approach, while effective for traditional ML pipelines, would benefit from enhancements that address the probabilistic and evolving nature of AI systems. Research could explore developing dependency analysis techniques that understand semantic relationships between data sources, not just syntactic dependencies, enabling more intelligent impact assessment when data sources change in subtle but significant ways.

A particularly important research direction involves scaling dependency analysis to handle real-time, streaming AI systems where data dependencies form and dissolve dynamically. Current static analysis approaches assume relatively stable code structures, but modern AI systems increasingly incorporate online learning, real-time feature stores, and adaptive model architectures that modify their data dependencies based on runtime conditions. Research could investigate hybrid static-dynamic analysis approaches that maintain dependency graphs for streaming systems, potentially incorporating techniques from program synthesis and runtime verification to ensure dependency tracking remains accurate as systems evolve.

Furthermore, the integration of dependency analysis with AI-specific quality assurance practices requires developing new methodologies that understand the unique failure modes of ML systems. Unlike traditional software where functionality is deterministic, AI systems can degrade gradually due to data drift, concept shift, or feature correlation changes that may not be immediately apparent

through traditional dependency tracking. Research could explore extending the dependency framework with causal reasoning capabilities, potentially connecting with the earlier discussed work on dataflow graphs as causal graphs, to provide more sophisticated understanding of how data changes propagate through AI systems and affect model behavior.

Long-term research adaptation should also consider the intersection with emerging AI governance and responsible AI practices. As AI systems become subject to increasing regulatory scrutiny and ethical oversight, dependency tracking becomes not just a reliability tool but a governance necessity. Research could explore how dependency graphs can support AI auditing processes, bias detection across data pipelines, and compliance reporting for complex AI systems. This might involve developing privacy-preserving dependency analysis techniques that can track data lineage without exposing sensitive information, or creating automated compliance checking systems that verify data usage against established governance policies. The ultimate goal would be establishing dependency-aware AI engineering practices that ensure both technical reliability and ethical compliance throughout the AI system lifecycle.

5.3 Research Ethics & Synthesis Reflection

5.3.1 Search and Screening Process

For this assignment, I employed a systematic approach to identify relevant papers from the CAIN conference series. I began by accessing the CAIN conference proceedings through the ACM Digital Library and IEEE Xplore, focusing on the most recent conferences (2021-2024) to ensure contemporary relevance. My initial search strategy used broad keywords related to AI engineering: "machine learning systems," "AI engineering," "MLOps," and "production AI." I then refined the search using more specific terms like "dataflow," "dependency analysis," "causal inference in ML," and "AI system reliability."

The screening process involved a three-stage approach: first, I filtered papers based on title relevance to software engineering aspects of AI systems; second, I reviewed abstracts to identify papers that addressed practical engineering challenges rather than pure algorithmic contributions; finally, I read the introductions and conclusions to assess the depth of software engineering insights and potential relevance to automotive AI applications. I prioritized papers that demonstrated clear connections between theoretical AI concepts and practical system engineering challenges, as these would provide the most valuable insights for understanding how AI engineering principles apply to real-world applications like my vehicle dynamics research.

5.3.2 Pitfalls and Mitigations

During the search process, I encountered several papers with misleading titles that appeared highly relevant but proved less useful upon closer examination. For instance, several papers titled with "AI Engineering" or "Machine Learning Systems" focused primarily on algorithmic improvements rather than engineering methodologies. Papers about "intelligent systems" often described domain-specific AI applications without addressing the underlying software engineering challenges that make such systems maintainable and reliable.

To mitigate these issues, I adjusted my screening strategy to place greater emphasis on the methodology sections and related work discussions, which typically revealed whether a paper addressed engineering concerns versus purely algorithmic ones. I also began cross-referencing papers with their citation patterns - papers that cited software engineering literature alongside AI research were more likely to address the interdisciplinary challenges relevant to this assignment. Additionally, I learned to look for specific keywords in abstracts such as "production," "deployment,"

"maintenance," "debugging," and "system reliability" as stronger indicators of engineering-focused content than general AI terminology.

5.3.3 Ethical Considerations

To ensure originality in my analysis, I took several important steps throughout the writing process. First, I maintained detailed notes about my own interpretations and insights while reading each paper, clearly distinguishing between the authors' stated contributions and my own analytical conclusions. When discussing how papers relate to my research or to broader AI engineering challenges, I focused on developing novel connections and applications rather than simply summarizing existing content.

For the integration scenarios and research direction adaptations, I deliberately created original examples and frameworks that, while inspired by the papers' concepts, represent new applications and extensions of their ideas. The smart city traffic management system example, for instance, combines concepts from both papers with domain knowledge from urban computing and my own research background to create a novel integration scenario. Throughout the writing process, I ensured that all direct quotes or specific technical details were properly attributed while clearly marking my own analytical contributions and interpretations as distinct from the original authors' work.