# Beyond Code: Engineering Trustworthy Software Systems with AI at Scale

Workshop Report

**August 2026**

CCC

Computing Community Consortium

Visioning

## Authors

Randal Burns, Johns Hopkins University
Sebastian Elbaum, University of Virginia
Gabrielle Allen, University of Wyoming
Nils Aschenbruck, Osnabrück University
Terry Benzel, University of Southern California
William Gropp, University of Illinois Urbana-Champaign
Rick Kazman, University of Hawaii
Manish Parashar, University of Utah

## Suggested Citation

Burns, R., Elbaum, S., Allen, G., Aschenbruck, N., Benzel, T., Gropp, W., Kazman, R., Parashar, M. (2026). *Beyond Code: Engineering Trustworthy Software Systems with AI at Scale.* Washington, D.C.: Computing Research Association (CRA). http://cra.org/ccc/wp-content/uploads/sites/2/2026/08/Beyond-Code-Engineering-Trustworthy-Software-Systems-with-AI-at-Scale.pdf

## About the Computing Community Consortium (CCC)

A programmatic committee of the Computing Research Association (CRA), the Computing Community Consortium's (CCC) mission is to lead visioning activities that bring together the computing research community — from across academia, industry, and government — to engage with future research challenges. CCC convenes leaders across disciplines, builds consensus, and strategically communicates those long-term visions to shape the field and address emerging national and global challenges.

*The Computing Community Consortium (CCC) is supported by the U.S. National Science Foundation under Grant No. 2300842. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the U.S. National Science Foundation.*

# TABLE OF CONTENTS

# Executive Summary

With the rapid evolution and proliferation of AI-assisted software development, the paradigm for creating and maintaining software is fundamentally shifting. Developers are transitioning from primary code authors to orchestrators of complex AI systems, workflows are evolving to incorporate autonomous agents and maximize production, and organizations are aggressively prioritizing AI-centered budgets, all while navigating a landscape defined by high speed and significant uncertainty. Steering these transformations requires the computing community to rethink how we capture user intent, ensure system quality at AI generative scales, and train the next generation of developers.

The workshop **Beyond Code: Engineering Trustworthy Software Systems with AI at Scale** convened experts in artificial intelligence, software engineering, programming languages, cybersecurity, and systems engineering to address the evolving role of AI in developing complex and large-scale software systems. The goal of the workshop was to envision in this rapidly evolving area how AI-powered development techniques and tools, and the organizations that depend on them must transform to move beyond code to cost-effective, verifiable, secure, and maintainable systems and deployments. Below are the summarized key findings and recommendations from the workshop on which we elaborate in following sections of the report.

## Key Findings

1. **(F.1.1.):** Current code models treat natural language as a primary interface, but its ambiguity and lack of formal semantics create a lossy translation gap.
2. **(F.1.2.):** Generative tools amplify productivity differences, acting as a creative multiplier for expert engineers while creating intellectual dependency, deskilling, and a crutch for less experienced practitioners.
3. **(F.1.3.):** First-generation models are constrained by human-centric programming syntax, forcing inefficient trial-and-error generation loops.
4. **(F.1.4.):** While models produce functional code logic, they frequently fail on critical quality atributes such as performance, security, and numerical accuracy.
5. **(F.2.1.):** The developer's role is elevating to multi-agent orchestrator and shifting to early-stage requirements and specifications from code generation.
6. **(F.2.2.):** As industry commands frontier models and compute infrastructure, academia risks being relegated to post-hoc evaluation of commercial artifacts.
7. **(F.3.1.):** Operating teams of agents introduce coordination friction, composability failures across siloed agent outputs, and integration liabilities at deployment.

8. **(F.3.2.):** As developers distance themselves from code generation, the underlying rationale behind design choices is lost, leaving future engineering teams with opaque codebases and diminished capacity to maintain legacy systems.
9. **(F.3.3.):** The velocity and volume of automated code generation outpaces human auditing capacity and renders traditional quality metrics obsolete.

## Recommendations

1. **(R.1.) Bridging the Intent Gap:** Conduct research into new models and workflows capable of interactive requirement elicitation, proactively querying users to disambiguate intent and surface implicit assumptions.
2. **(R.2.) Assurance Through Neuro-Symbolic Integration:** Develop new neuro-symbolic models that generate code with corresponding formal proofs, coupling neural generation with symbolic reasoning.
3. **(R.3.) Cross-Layer Orchestration:** Develop orchestration mechanisms to reduce friction between human-human interactions mediated with agents, agents and humans, and agents with agents.
4. **(R.4.) Goal-Driven Security Interfaces and Secure-by-Construction:** Develop new AI models that make security an intrinsic property of the generated artifacts.
5. **(R.5.) Training Data Transparency and IP Governance:** Create frameworks for incorporating data transparency and intellectual property governance for AI-generated code and deployments, including tools for membership inference and provenance tracking.
6. **(R.6.) Robustness Through Transpilation:** Advance AI transpilation to serve as a pathway to safe, secure, and verifiable systems, enabling the modernization of legacy codebases and offering a diversity of implementations that can be cross-validated.
7. **(R.7.) AI-Assisted Deployments:** Advance AI-assisted deployment techniques that reason about dependencies, monitor environments continuously, and automatically detect and remediate anomalies without manual intervention.
8. **(R.8.) Industrial-Academic Alliances:** Establish meaningful industrial-academic alliances that secures sufficient computing for academia to target the forefront of AI capabilities.
9. **(R.9.) Reskilling the Developer:** Evolve curricula and training to emphasize requirements, specifications, and system design, creating managers of AI agents who can orchestrate and evaluate automated workflows.

## Mappings

| Finding | Recommendations |
|---|---|
| **From User Intent to Code:** Lossy translation gap between high-level ambiguous intent and concrete, rigid code. | **Bridging the Intent Gap:** Conduct research into new models/workflows for interactive requirement elicitation, capturing quality attributes, formalizing specifications, and maintaining continuous traceability/verification.<br><br>**Assurance Through Neuro-Symbolic Integration:** Pair generative neural models with symbolic reasoning/proofs to verify intent and code correctness. |
| **Skills and Creativity:** Generative AI acts as a creative multiplier for experts but causes deskilling and is a crutch for less experienced practitioners. | **Reskilling the Developer:** Evolve education and professional training toward requirements engineering, specifications, system design, and AI agent orchestration. |
| **Machine Interface:** Traditional human-centric programming languages cause inefficient trial-and-error loops for current LLMs. | **Cross-Layer Orchestration:** Develop novel higher-level, platform-agnostic languages and interfaces for clear human-agent and inter-agent communication.<br><br>**Assurance Through Neuro-Symbolic Integration:** Utilize structured intermediate representations that support automated verification and formal methods during generation. |
| **Quality Attribute Expertise Shortfall:** Models produce functionally correct code but fail on non-functional dimensions like security, performance, reliability, and numerical accuracy. | **Goal-Driven Security Interfaces and Secure-by-Construction:** Allow users to express security intent at high levels while embedding secure-by-default patterns into models and tools.<br><br>**Robustness Through Transpilation:** Use transpilation to force output into safe, secure, and verifiable representations.<br><br>**Bridging the Intent Gap:** Explicitly capture non-functional requirements and quality attributes in formalized specifications. |
| **Developer's Profile:** Developers are shifting vertically to orchestrators and horizontally to early requirements/specification. | **Reskilling the Developer:** Adapt curricula and industry training to prepare engineers as managers of multi-agent workflows and high-level architects. |

| | |
|---|---|
| **Academic Research Identity:** Industry dominance in compute, models, and data risks relegating academia to post-hoc analysis of commercial artifacts. | **Industrial-Academic Alliances:** Secure access to frontier compute infrastructure for academic researchers via national initiatives and industry partnerships, aligning research on foundational next-gen models and problems. |
| **Orchestration and Deployment:** Multi-agent development causes coordination friction, composability failures, and deployment integration/liability issues. | **Cross-Layer Orchestration:** Implement frameworks to resolve human-human, human-agent, and agent-agent friction via formal inter-agent interfaces and telemetry dashboards.<br><br>**AI-Assisted Deployments:** Develop models capable of reasoning about dependencies, continuous monitoring, and automated anomaly remediation.<br><br>**Training Data Transparency and IP Governance:** Establish membership inference and provenance tracking to manage copyright, licensing, and code provenance liabilities. |
| **Cognitive Overload, Debt, and the Maintenance Deficit:** Distance from code creates loss of design rationale, leading to opaque codebases and an inability to maintain legacy systems. | **Bridging the Intent Gap:** Establish traceability mechanisms that record system lineage and rationale behind design choices to ease long-term maintenance.<br><br>**Robustness Through Transpilation:** Modernize legacy codebases and convert implementations into verifiable representations.<br><br>**Assurance Through Neuro-Symbolic Integration:** Retain precise specification states and code rationale through formal intermediate representations.<br><br>**Goal-Driven Security Interfaces and Secure-by-Construction:** Automatically update, continuously monitor, and remediate codebases against new threats. |
| **Quality Assurance at Generative AI Scale:** Rapid code generation outpaces human auditing capacities, and traditional coverage metrics become obsolete. | **Assurance Through Neuro-Symbolic Integration:** Automate formal verification and theorem proving as scalable alternatives to manual quality assurance.<br><br>**Training Data Transparency and IP Governance:** Establish gatekeeping and auditing mechanisms for inbound training and generated data. |

| | |
|---|---|
| | **AI-Assisted Deployments:** Develop model execution layers that monitor environments continuously and automatically audit/remediate runtime execution. |

## Context

Rapid advances in the quality and accessibility of AI-generated code are transforming the practice of software engineering. In 2025, almost all software developers use AI agents as part of their development process, reports indicate 83.5% (Stack Overflow, 2025) to 90% (DeBellis et al., 2025) rates of adoption. These advances are causing disruptions at many levels. They are leading to the development of paradigms like "vibe coding," where software is provisioned entirely through high-level natural language intent rather than traditional programming language construction. These advances are upending computer science education and software engineering techniques, threatening the relevance of much coursework and many long-taught best practices. These advances are also compelling industry leaders to prioritize AI-centric budgets under the assumption that this will lower engineering costs and lead to better software, eventually replacing traditional developers, and covering the full software lifecycle. Banking on such a future, industry is already restructuring its workforce, and starting to evaluate and hire engineers primarily based on macro-level system design skills and their capacity to direct, verify, and scaffold AI-generated output (King, 2026).

However, an operational reality check challenges this vision, exposing a significant gap between enterprise ambition and actual AI system capabilities. AI usage remains mostly confined to localized code generation under human supervision. Accuracy concerns and a lack of trust in AI's ability to handle complex tasks limits its deployment in high-responsibility, safety-critical domains. Furthermore, only a small minority of engineers rely on AI for planning and deployment. At the same time, the influx of automated code is overwhelming traditional quality assurance paradigms, outpacing the throughput of human reviews and analysis tools. These limitations are most acute in brownfield development environments burdened by legacy systems and aged constraints, where successful adoption has lagged (DeBellis et al., 2025).

As we seek to extend AI beyond code generation into full system development, these disconnects between potential and reality are bound to get larger. The workshop set out to identify and characterize these gaps, highlight emerging opportunities, and propose actionable recommendations.

# Major Findings

Findings from this workshop are organized into three themes. Prevalent Gaps identifies deficiencies in current AI-assisted development both in technologies and developer skills. Rapid Evolution addresses the fast-moving landscape reshaping software engineering, including the transformation of the developer role, volatile compute economics, and academia's shifting position relative to industry. High Operational Burden examines the mounting challenges of developing, deploying and maintaining AI-driven systems at scale.

## F.1. Prevalent Gaps

### *F.1.1. From User Intent to Code*

Existing LLMs and the toolchains built on them fail to bridge the distance between a stakeholder's initial, often ambiguous desire and the level of rigor required to guide automated system development. This difficulty stems from a fundamental translation gap: natural language remains a lossy medium for specifying intent, causing critical nuances and unstated constraints to be lost without clear indicators or traceability as human desires are converted into natural language and subsequently into code. Without mechanisms to quantify and mitigate this lossiness throughout the pipeline, generated systems inevitably remain misaligned with true user intent.

A primary driver of this gap is that current toolchains rely on first-generation code models that operate primarily as passive text generators. Because these initial model architectures lack native mechanisms to maintain state, track specification lineage, or surface implicit assumptions, they inherently exacerbate the distance between high-level human intent and executable software logic.

These limitations directly impact the developer experience. As practitioners shift away from code, often building and deploying systems without fully understanding the underlying code, current interfaces fail to bridge the gap between intent and execution. Traditional IDEs remain optimized for human-readable code manipulation, leaving developers without interfaces capable of inspecting, steering, or aligning first-generation models across high-level intent, architectural patterns, and agentic workflows.

### *F.1.2. Skills and Creativity*

AI may amplify differences among developers rather than uniformly benefitting everyone. Highly-skilled software engineers become much more productive, whereas less capable

practitioners do not fully understand the generated code, leading to intellectual dependency, increased rework, and, in the long term, deskilling.

This effect is also seen in education. While better students use AI effectively to augment their knowledge, poorer students may use it as a crutch, actually undermining learning. This creates questions for the future of computer science education: How do we best use AI to help students acquire the rigorous thinking required to understand the design, creation, and evolution of complex systems? Can we achieve this goal if AI is performing much of the intellectual heavy lifting?

AI can act as a creative amplifier: enabling brainstorming, rapidly generating ideas or prototypes, and accelerating experimentation. However, there are limits. Creativity arises from unexpected connections of ideas that surprise and inform us. Creative insights break established patterns. Because AI systems are trained on historical data, they tend to generate outputs that reflect the center of their training distribution, making them highly effective at synthesizing existing knowledge but less likely to produce truly novel, outlier conceptual breakthroughs. Thus, AI may best be understood as a collaborator that accelerates exploration, but not a replacement for human creativity.

### *F.1.3. Machine Interface*

Current programming languages are human-centric by design, optimized for the linear cognition, memory constraints, and readability requirements of software developers. This creates a significant impedance mismatch for current LLMs, which treat software primarily as sequential text streams. Programming languages force current models to navigate syntactic ambiguity, fragile dependencies, and low-level implementation details, leading to costly trial-and-error loops that consume excessive tokens and obscure intent.

We lack high-level, platform-agnostic languages designed specifically for generative agents, as well as a new generation of models capable of natively processing, operating, and even reasoning over them. Without such intermediate representation, it will not be possible to scale quality assurance and integrate formal methods at the generation stage needed to ensure system correctness. This extends to the interactions among multiple agents, which also require specific languages, formal semantics, and structured interfaces that preserve intent with precision.

### *F.1.4. Quality Attribute Expertise Shortfall*

AI-assisted code generation tools exhibit an expertise deficiency when tasked with meeting critical quality attributes. While LLMs excel at generating functionally correct code for many

tasks, they consistently falter on performance, validity, security, and reliability. In distributed systems, for example, generated code frequently lacks the proper concurrency controls required for parallel efficiency and fault tolerance, just as it struggles to preserve numerical accuracy and reproducibility in scientific domains. Similarly, when using compromised training data, LLMs may learn unsafe patterns that increase vulnerabilities.

These limitations are compounded in specialized low-resource application areas where scarce training data deprives models of the examples necessary to learn these complex constraints. Again, this deficit is very evident in software security. LLMs enable domain experts with no traditional coding background to build complex software, however, these users lack the adversarial mindset and specialized vocabulary required to identify the vulnerabilities the AI inadvertently introduces. As a result, a critical disconnect remains between the ease of generating functional logic and the rigorous domain expertise required to ensure system integrity.

## F.2. Rapid Evolution

### *F.2.1. Developer's Profile*

Generative AI is driving two simultaneous shifts in software engineering: a vertical shift upward in the level of cognitive abstraction, and a horizontal shift leftward in the software development lifecycle. Together, these shifts are fundamentally redefining both the day-to-day role of the developer and the future bottleneck for system development.

Vertically, the engineer's role is elevating from code generation to agent orchestration. In this new paradigm, computational thinking is no longer about managing syntax or individual lines of code; instead, it is about managing abstraction, anticipating model behaviors, and ensuring the integrity of entire system-scale engines. Engineers must now multiplex across tasks, acting simultaneously as domain experts, system architects, and rigorous reviewers who direct networks of generative agents.

Horizontally, as AI increasingly automates implementation, the bottleneck in the software lifecycle is shifting to envisioning and specifying, a phenomenon termed "shift left" in system development. Because the cost of generating functional logic is approaching zero, the engineering burden re-centers entirely on these earliest phases of creation. Consequently, engineers must develop stronger skills in eliciting requirements, validating user intent, and precisely specifying system constraints. In this way, the foundational objective of software development transitions from asking "Did we build the system right?" to "Did we build the right system?", the core question driving verification and validation.

Simultaneously, the shift in abstraction also empowers non-programmer domain experts to construct software. This phenomenon presents significant opportunities such as accelerating domain-specific innovation by lowering barriers to entry. However, with the current infrastructure, it can also introduce vulnerabilities, technical debt, and silent failures in deployed systems.

### *F.2.2. Academic Research Identity*

The identity of academic research is increasingly challenged as industry commands the development of leading frontier models and their derivative software development tools. Driven by massive capital investment, industry controls the compute infrastructure, proprietary datasets, and engineering talent required to push the state of the art. Even with some access to near-state-of-the-art models, academia's capacity to drive research innovation in this space has shifted. Industry operates at a release velocity where base model improvements and native toolchains rapidly subsume capabilities that previously required bespoke academic prototyping. Furthermore, because industry holds the surrounding infrastructure (e.g., full telemetry, orchestration frameworks, deployment environments) academic groups lack the production context required to build and test next-generation models and tooling at scale.

Consequently, academic researchers find themselves playing catch-up, focusing on evaluating or reverse engineering industrial outputs, auditing black box systems for biases, and attempting to reproduce industry findings at a fraction of the scale, or, increasingly, migrating to industry altogether. This resource asymmetry alters the traditional research lifecycle; instead of inventing the foundational systems of tomorrow, academia is shifting toward a reactive stance of post-hoc artifact analysis. While university labs maintain a lead in safety related areas, from studying the human-centric sociotechnical effects and ethical implications of AI to developing safer software with AI models, academia is losing its relevance in other technical areas.

## F.3. High Operational Burden

### *F.3.1. Orchestration and Deployment*

When shifting from AI code generation to employing an army of AI agents to build an entire system, friction arises in managing their coordination during development, and in validating the complex, unvetted software assembly they deliver at deployment time. Agents can overwrite each other's changes, fail to communicate context across sub-tasks, and struggle to properly interface with the development environment (e.g., legacy APIs, compilers, repositories, sensors). Without robust orchestration frameworks, the overhead of auditing and debugging

these multi-agent interactions threatens to negate the productivity gains offered by the underlying automation.

This coordination friction becomes apparent during system integration. Because individual agents operate with non-deterministic outputs and siloed context, combining their code introduces composability challenges. Even if every agent successfully completes its isolated programming task, the aggregated whole rarely preserves critical non-functional properties like overall performance, numerical accuracy, and structural safety. This is magnified when AI agents are non-deterministic, fail unpredictably, or operate with conflicting goals. This development-time challenge culminates in a heavy validation burden at the deployment stage.

Because an automated workforce can generate massive volumes of infrastructure-as-code and deep dependency trees at an unreviewable pace, it frequently introduces integration faults and expansive new attack surfaces. The resulting deployment is often saddled with software of muddy provenance, fraught with configuration mismatches, unvetted third-party libraries, and systemic liabilities regarding copyright and intellectual property leakage.

### *F.3.2. Cognitive Overload, Debt, and the Maintenance Deficit*

As developers distance themselves from source code, a compounding cognitive debt and traceability debt accumulate, leading to a maintenance deficit for the next generation of legacy software. This crisis emerges because the connection between high-level intent and low-level implementation becomes opaque. When a system is generated rather than authored, the underlying rationale behind both high-level design choices and low-level implementation decisions is entirely lost. Consequently, developers lose the capacity for manual intervention and deep system maintenance, creating a fragile future where software remains functional only as long as the original generation context is preserved.

This vulnerability is exacerbated by the sheer velocity of AI-driven production. LLMs enable developers to generate more code faster and this code must be understood, reviewed, and maintained. As the volume of generated code outstrips the capacity for meaningful oversight, poor quality and redundant code may pollute codebases. As developers increasingly rely on AI to produce code, they lose the deep understanding of their own codebases that historically enabled effective maintenance and evolution. Ultimately, this transition of the human role from active creator to overwhelmed auditor leaves future software teams ill-equipped to evolve legacy AI systems.

### *F.3.3. Quality Assurance at Generative AI Scale*

The transition to AI-driven development has created an imbalance between the velocity of code generation and the capacity for sufficient oversight. While LLMs can accelerate the translation of requirements into specifications and then code, the resulting explosion in code volume renders traditional human-centric auditing an impossibility. This creates a critical assurance gap where the sheer scale of automated output outpaces validation and verification infrastructure. This gap worsens as human reviewers face severe cognitive fatigue and automation bias when auditing these massive AI-generated pull requests, causing them to miss more subtle faults. Such faults are already resulting in security breaches.

Furthermore, traditional quality metrics fail in this paradigm. For example, AI agents can autonomously generate far reaching yet flawed test suites that artificially inflate coverage metrics while validating incorrect behaviors. Without a fundamental pivot toward equally scalable, automated quality-checking mechanisms, such as automated property-based testing and formal semantic verification, the industry faces a significant risk of deploying opaque systems whose safety and dependability cannot be guaranteed at the speed of their creation.

## Recommendations

### R.1. Bridging the Intent Gap

Current generative models and their associated workflows suffer from an intent gap between what a user envisions and what the LLM ultimately generates. To close this gap, this recommendation advocates for elevating requirements and specifications to active, first-class elements of the lifecycle, alongside developing a new generation of models architected natively to maintain state, query users, and reason over specifications. This requires establishing an iterative process rooted in shared mental models, making the underlying assumptions of both the user and the generative models transparent to support continuous refinement.

Next-generation models and their surrounding workflows must look beyond functional logic to explicitly capture non-functional requirements and quality attributes (e.g., performance, availability, and security), ensuring the system is robust under real-world constraints. To serve as a dependable foundation for this process, these specifications must be sufficiently formalized, moving beyond ambiguous natural language to structures that permit rigorous validation and verification while remaining adaptable during interaction.

Furthermore, we must establish traceability mechanisms that record the historical lineage of the system, tracking exactly how initial user intent evolved into these concrete specs, how those specs were refined, and the rationale behind why those choices were made to ease long-term

maintenance. Finally, this alignment must be continuously verified through bidirectional triangulation mechanisms that contrast the formalized specifications against other generated artifacts, such as code and test suites, and map them back again to instantly catch structural or behavioral deviations against the specifications.

## R.2. Assurance Through Neuro-Symbolic Integration

While LLMs fluidly translate ambiguous human intent into code, they can guarantee neither semantic alignment with the user's true intent nor the absolute correctness of the implementation, resulting in subtle vulnerabilities, logical flaws, and hallucinatory behaviors that existing reviewing and testing processes fail to catch. Symbolic methods offer the necessary verification but are constrained by limited scalability and steep manual formalization costs. A neuro-symbolic paradigm has the potential to overcome these limitations by pairing the generative fluency of neural models with the deterministic verification gates of symbolic reasoning. Realizing this integration requires developing new models built from the ground up for neuro-symbolic execution and substantial techniques and tooling to embed automated specification formalization and verification infrastructure directly into current development pipelines. In such an architecture, generative AI simultaneously produces both code and formalized specifications encoded in a structured intermediate representation.

Under this vision, a backend then checks the LLM-generated code against the encoded specifications using enriched automated solvers and theorem provers, and leveraging existing knowledge about software components already proven right by others. Crucially, when gaps or verification failures are identified, precise diagnostics and proof-state metadata are translated into feedback to improve code and specification generation. This defines an automated, iterative refinement process of both the code and the specifications until formal verification succeeds, ensuring that systems are formalized to the exact degree required for the task while providing a scalable alternative to manual code review.

## R.3. Cross-Layer Orchestration

We need new engineering paradigms to orchestrate the multi-layered friction in agent-driven system development, which manifests across human-human, human-agent, and agent-agent boundaries. To mitigate human-human friction where stakeholders must mediate through automated intermediaries, development environments must inherently capture the provenance of every generative decision to enable clear conflict resolution protocols.

To address human-agent friction regarding the synchronization of shared mental models, we must evolve traditional text-centric IDEs into orchestration dashboards providing real-time

telemetry. Simultaneously, we must design new generations of models natively trained on novel, higher-level languages that allow humans to state intent at a higher level of abstraction than legacy code syntax, providing clear semantics that AI systems can interpret without ambiguity.

Finally, to resolve agent-agent friction involving resource conflicts and state drift, the field must move away from ambiguous natural language toward formal, verifiable interfaces for inter-agent communication, drawing from formal specification languages or interactive theorem provers, to support automated contract compliance. Integrating these targeted mechanisms across the toolchain will ultimately allow us to reason about complex agent ensembles formally, ensuring systems are predictably steered toward dependability and correctness rather than discovered through failure.

## R.4. Goal-Driven Security Interfaces and Secure-by-Construction

Research should pursue interfaces that allow users to express and refine security intent without requiring expertise in security implementation. Rather than expecting developers to specify cryptographic protocols, access control mechanisms, or input validation routines, future AI-assisted environments should allow users to precisely state what they want to protect — "this data is confidential," "only authorized users can perform this action," or "this input comes from untrusted sources" — and automatically translate these goals into appropriate security mechanisms.

This capability must be paired with new secure-by-default AI models and development tools that embed security into the generation process itself: selecting safe libraries, applying secure coding patterns, and avoiding common vulnerability classes without user intervention. This also includes safeguarding models against active threat vectors, such as indirect prompt injection where adversarial inputs corrupt agent intent, and data-level risks where models learn compromised coding patterns. Secure-by-construction next generation models and tooling must actively detect and isolate these manipulation attempts across both the training pipeline and operational execution. The long-term vision is AI-assisted development in which security is a human accessible, goal-driven conversation between user intent and intelligent tooling.

AI capabilities should be redirected toward strengthening software security through novel tooling approaches. Cross-AI development and validation offers a promising direction: generating code with one model while using different models to generate adversaries and test cases that probe for vulnerabilities. This approach addresses the risk of model monoculture, where reliance on a single dominant model creates systemic blind spots. We also need research toward evolving auditing, rewriting, and deployment practices. AI tooling must continuously update implementations, configurations, and deployments in response to newly

emerging threats. This demands systems that monitor for new vulnerability classes, automatically assess whether existing codebases are affected, and propose or apply remediation without requiring manual intervention for each threat.

## R.5. Training Data Transparency and IP Governance

Research must establish frameworks for training data transparency and intellectual property governance that address the unique challenges of AI-generated code. Current code generation systems operate as black boxes with respect to their training data. Users cannot determine whether generated code incorporates copyrighted material, violates patents, or leaks proprietary information from other sources. IP concerns flow in both directions. Generated code may inadvertently reproduce protected content from training data, while user interactions with AI systems may expose proprietary code that subsequently influences outputs for other users.

The emergence of IP indemnification programs from major providers suggests the market is developing practical risk allocation mechanisms faster than technical solutions. The community needs better tools for membership inference and provenance tracking that links generated outputs to sources. Addressing these challenges demands developing new models architected for verifiable attribution, alongside gatekeeping and auditing mechanisms for inbound training data that establish clear licensing and attribution chains, enabling organizations to assess legal and ethical risks before deployment.

## R.6. Robustness Through Transpilation

Research must elevate transpilation beyond traditional code transformation to serve as a mitigation mechanism against the quality, performance, security risks created by AI-generated software. Because generative models frequently output opaque, extensive, monoculture, and hard-to-audit code, transpilation has the potential to become a pathway to force generated output into safe, secure, and verifiable representations, such as modernizing legacy codebases by converting C++ to Rust to guarantee memory safety or converting serial implementations to parallel for performance gains.

Transpilation should also encompass providing equivalence guarantees across platforms, ensuring for example that CPU, GPU, TPU, and FPGA implementations can be used interchangeably. Achieving this requires developing new models trained natively to reason over cross-platform semantics and confidence measures, automated equivalence testing, and verification approaches to ensure transpiled code maintains original functionality and quality attributes. Positioned this way, transpilation can become a containment strategy for enforcing

desirable properties by construction (e.g., security through safe languages, verifiability through formal representations, and maintainability through modernized codebases) to directly counter AI-driven code fragility and monoculture risks.

## R.7. AI-Assisted Deployments

We must address the reality that configuration and deployment errors can cause significant harm, yet these aspects of software systems receive less attention from the AI community. The field should develop new model capabilities and agents that are cognizant of package contents, deployment requirements, and can continuously monitor federated environments where conditions change beneath running systems.

This requires opening the black box of current deployment practices: moving beyond opaque containerized packages to representations where agents can reason about what is in a package, what dependencies it requires, and what context is needed for safe deployment. Research must bridge the significant trust gap between current AI capabilities, which can flag anomalies and suggest fixes, and the level of reliability required before organizations will allow AI agents to autonomously rewrite and redeploy production code without human approval.

## R.8. Industrial-Academic Alliances

Current industrial-academic alliances in the areas of software development with LLMs primarily leverage academia as a creator of evaluation benchmarks. For industry, the funding cost is offset by gaining access to independent, rigorous validation protocols that internal product teams may face market pressure to bypass. However, this remains a superficial touchpoint, limiting the broader impact of academic research and bottlenecking the development of a robust talent pipeline to serve the industry. Rectifying this imbalance is difficult, as it requires navigating complex competing incentives, intellectual property boundaries, and an accelerated corporate research cycle that often outpaces traditional academic timelines.

Yet, one critical lever that can be adjusted quickly (and with immediate outsized effect) is securing academic access to frontier compute infrastructure through a combination of industry funding and national compute initiatives. Providing these resources would allow university researchers to move beyond toy examples and directly target the forefront of AI capabilities which occurs at scale. In return, industry stands to gain an expert research partner focused on solving some of the real-world problems that reside behind corporate walls.

For its part, academia must also rebalance its own approach to collaboration. The challenge lies in identifying a new strategic sweet spot: academic research must align more closely with real-world industry challenges, yet fully leverage its insulation from market pressures to pursue

the larger, foundational opportunities that corporate labs routinely bypass. This sweet spot may lie, for example, in moving beyond evaluating and complementing first-generation models to actively designing next-generation models built from the ground up to prioritize formal verification, security-by-construction, and long-term maintainability.

### R.9. Reskilling the Developer

Historically, computer science education prioritized training students to become expert coders by focusing heavily on algorithms, data structures, and programming language mastery, operating under the assumption that a developer's primary value lay in manual code generation. Today, because AI predominantly executes these implementation tasks, that traditional emphasis is obsolete, and is demanding a "shift left" in both educational and professional focus. To adapt, curricula must move away from isolated programming exercises and instead center on requirements engineering, precise technical specifications, and system architecture. Thus, the definition of computational thinking must evolve to incorporate the cognitive ability to manage much higher-level abstractions while actively anticipating, testing for, and mitigating the stochastic, unpredictable nature of agentic systems.

Because students will no longer build mental models through programming, we also need entirely new pedagogical approaches. Education must explicitly cultivate system-level intuition and complexity management, training students to empirically measure and validate these systems. Ultimately, we must prepare students to become experts in their target application domains so that they can cost-effectively guide, scope, and oversee AI agents. This mandate extends far beyond the university classroom and must be injected into the tech industry to reskill the millions of current developers whose traditional coding roles are actively evaporating.

## Concluding Remarks

AI-assisting development is fundamentally transforming how software systems are conceived, built, and maintained. Envisioning the future of this domain is inherently challenging. AI-driven software development is a fast-moving field characterized by rapid technological shifts, shifting economic baselines, and constant operational uncertainty.  This report acknowledges and confronts these challenges. By offering contextualized findings and actionable recommendations, it provides a roadmap of research opportunities and directions to guide the community beyond code generation and toward AI-powered models, techniques, tools, and practices for cost-effectively building and deploying trustworthy systems.

# Appendices

## Workshop Participants

| First Name | Surname | Affiliation |
|---|---|---|
| Gabrielle | Allen | University of Wyoming |
| Ilkay | Altintas | University of California, San Diego Supercomputer Center |
| Nils | Aschenbruck | Osnabrück University |
| Clark | Barrett | Stanford University |
| Jared | Bauer | Atlassian |
| Andrew | Begel | Carnegie Mellon University |
| Terry | Benzel | University of Southern California Information Sciences Institute |
| Yuriy | Brun | University of Massachusetts Amherst |
| Randal | Burns | Johns Hopkins University |
| Franck | Cappello | Argonne National Laboratory |
| Alvaro | Cardenas | University of California Santa Barbara |
| Ewa | Deelman | University of Southern California Information Sciences Institute |
| Prem | Devanbu | University of California, Davis |
| Isil | Dilig | University of Texas |
| Fred | Douglis | Peraton Labs |
| Matt | Dwyer | University of Virginia |
| Sebastian | Elbaum | University of Virginia |
| Rayid | Ghani | Carnegie Mellon University |
| Rachel | Greenstadt | New York University |
| William | Gropp | University of Illinois Urbana-Champaign |
| Sumit | Gulwani | Microsoft |
| Thomas | Hauser | National Center for Atmospheric Research |
| Mrinal | Karvir | Intel |
| Daniel S. | Katz | University of Illinois Urbana-Champaign |
| Rick | Kazman | University of Hawaii |
| Brian | Kirk | IEEE Computer Science |

| | | |
|---|---|---|
| Andrian | Marcus | National Science Foundation |
| Erik | Meijer | Leibniz Labs |
| Daniela | Oliveira | National Science Foundation Security, Privacy, and Trust in Cyberspace |
| Manish | Parashar | University of Utah |
| Blaine | Reeder | University of Missouri |
| Neha | Rungta | Amazon |
| Vandana | Rungta | Amazon |
| Sanjit A. | Seshia | University of California, Berkeley |
| Bjorn | Stansvik | Byggr |
| Kevin | Storer | Storer Enterprise Strategy |
| Ladan | Tahvildari | University of Waterloo |
| Benjamin | Tan | University of Calgary |
| Michela | Taufer | University of Tennessee Knoxville |
| Dan | Wallach | Defense Advanced Research Projects Agency |
| Lingming | Zhang | University of Illinois Urbana-Champaign |

## Workshop Agenda

**February 25-26, 2026**

The Westin San Francisco Airport
1 Old Bayshore Hwy, Millbrae, CA 94030

**February 25, 2026**

| Time | Session Description | Room |
|---|---|---|
| 8:30 a.m. | Breakfast | Aspen |
| 8:30 a.m. | Registration | Bayshore Foyer |
| 9:30 a.m. | Welcome and Workshop Goals | Bayshore |
| 9:45 a.m. | Panel Discussion with Visionaries in the Field<br>**Neha Rungta**, Amazon<br>**Clark Barrett**, Stanford University<br>**Kevin Storer**, Storer Enterprise Strategy | Bayshore |
| 10:45 a.m. | Lightning Introductions | Bayshore |
| 11:25 a.m. | Introduce Breakout Groups | Bayshore |
| 11:30 a.m. | Breakouts Round 1: Ideation | Poplar, Laurel, Oak, Bayshore |
| 12:30 p.m. | Lunch | Aspen |
| 1:45 p.m. | Breakouts Round 1: Ideation Continued | Poplar, Laurel, Oak, Bayshore |
| 2:15 p.m. | Breakouts Round 1 Full Group Report Out | Bayshore |
| 3:00 p.m. | Break | Bayshore Foyer |
| 3:30 p.m. | Speaker: **Erik Meijer**, Leibniz Labs<br>*In Code They Think, In Proof We Trust* | Bayshore |
| 4:00 p.m. | Breakouts Round 2: Crosscutting Ideation | Poplar, Laurel, Oak, Bayshore |
| 5:15 p.m. | Breakouts Round 2 Full Group Report Out | Bayshore |

| | | |
|---|---|---|
| 6:00 p.m. | Dinner | Aspen |

**February 26, 2026**

| Time | Session Description | Room |
|---|---|---|
| 8:30 a.m. | Breakfast | Aspen |
| 9:30 a.m. | Speaker: **Sanjit Seshia**, University of California, Berkeley<br>*Full-Stack AI-Enabled Formal Methods: Past, Present, and Future* | Bayshore |
| 10:00 a.m. | Breakouts Round 3: Research Questions, Impact | Poplar, Laurel, Oak, Bayshore |
| 11:00 a.m. | Break | Bayshore Foyer |
| 11:30 a.m. | Breakouts Round 3: Full Group Report Out | Bayshore |
| 12:00 p.m. | Lunch | Aspen |
| 1:00 p.m. | Breakouts Round 4: Pulling It All Together | Poplar, Laurel, Oak, Bayshore |
| 2:30 p.m. | Wrap Up and Closing Remarks | Bayshore |
| 3:00 p.m. | Adjourn | Bayshore |

## Acknowledgements


### *Contributions*

The authors acknowledge comments, contributions, and edits from the following people:

**Fred Douglis**, Peraton Labs
**Alina Gerall**, Computing Research Association
**Daniel Katz**, University of Illinois Urbana-Champaign
**Quinn Spadola**, Computing Research Association
**Benjamin Tan**, University of Calgary

### *Reviewers*

CCC and the authors acknowledge the reviewers whose thoughtful comments improved the report:

**David Jensen**, University of Massachusetts Amhurst
**Edith Elkind**, Northwestern University

### *Sponsors*


**IEEE Computer Society**
This workshop was generously supported by IEEE Computer Society.


**U.S. National Science Foundation**
This workshop was supported by the Computing Community Consortium through the U.S. National Science Foundation under Grant No. 2300842. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the U.S. National Science Foundation.

# References

DeBellis, D., Storer, K., Harvey, N., Beane, M., Edwards, R., Fraser, E., Good, B., Kalliamvakou, E., Kim, G., Maxwell, E., D'Angelo, S., Inman, S., Murillo, A., and Villalba, D. (2025). *DORA 2025 State of AI-assisted Software Development Report*. DORA, Google. https://dora.dev/research/2025/dora-report/

King, E. (2026, February 17). *Meta's AI-enabled coding interview: How to prepare*. Hello Interview. https://www.hellointerview.com/blog/meta-ai-enabled-coding

Stack Overflow. (2025). *2025 Stack Overflow Developer Survey: AI*. Stack Overflow. https://survey.stackoverflow.co/2025/ai/