BO·TT·EGA
IT minds

Training program:

Trust Framework for AI-Assisted Development

Info:

Name:	**Trust Framework for AI-Assisted Development**
Code:	**ai-trust**
Category:	AI
Target audience:	architects developers
Duration:	3 days
Format:	20% lecture / 80% workshop hands-on

Stop Being the Bottleneck — Build the Verification Layers That Let You Trust AI Code Without Reading Every Line.

LLMs generate code faster than any human can review. That's the problem — and the opportunity. Human review doesn't scale. If your trust model depends on someone reading every line, you'll never move faster than your slowest reviewer. You're paying for AI speed and getting human-review throughput.

The way out is trust quality gates — verification layers that operate above the code. Does it satisfy the spec? Do behavioral scenarios pass? Does CI stay green on trunk? Does production telemetry confirm expected behavior after deploy? Each gate establishes trust without requiring anyone to read the implementation. Together, they let your team embrace the full pace of LLM code generation.

LLMs aren't that different from human developers. They make mistakes, cut corners, miss edge cases, and drift from intent — just like people do, just faster. The quality gates required to trust LLM output are exactly the same ones engineers have built over the last decades to trust each other's code. Specs, tests, CI, feature flags, monitoring — none of this is new. AI just makes it non-negotiable.

Target Audience:

Senior developers, tech leads, and architects at teams already using AI coding tools (Claude Code, Copilot, Cursor) who know that human review doesn't scale to the pace of AI-generated code. They need trust quality gates that let the team embrace the full speed of LLM code generation instead of bottlenecking on manual review.

Prerequisites::

- 3+ years of professional development experience
- Active use of at least one AI coding tool
- Working knowledge of TypeScript/JavaScript (other languages possible — agreed upon before the training)
- Basic familiarity with testing (unit tests, CI)
- Access to an AI coding tool (Claude Code, Cursor, Copilot, etc. — agreed upon before the training)

BO·TT·EGA
IT minds

What You Will Get:

- Identify and articulate the specific threats in trusting LLM-generated code and artifacts
- Connect proven engineering practices to each threat as concrete countermeasures
- Write specifications that serve as verifiable contracts for AI-generated code
- Use C4 diagrams to communicate architectural intent to AI tools
- Apply TDD with AI — Red-Green-Refactor where AI implements against your tests
- Write pragmatic behavioral scenarios that verify AI output without reading implementation
- Practice trunk-based development with short-lived branches for AI-assisted work
- Implement feature flags as safety nets for AI-generated code in production
- Set up monitoring with anomaly-triggered flag kill switches that protect production automatically
- Combine all practices into a repeatable trust framework for your team
- Implement a real feature using the full framework — without reading a single line of production code

It's all about the content.

- Build trust quality gates that replace manual review with automated verification
- Learn which engineering practices (many decades old) directly counter each LLM threat
- Spec-driven development expanded with C4 architecture diagrams for communicating vision to AI
- AI writes tests and implementation — you supervise the tests, not the code
- Trunk-based development, feature flags, and monitoring as a unified safety net
- Anomaly-triggered flag kill switches that protect production automatically
- Final proof: implement a real feature without reading a single line of AI-generated code

BOTTEGA IT minds

Training program

1. The Threat Landscape — What Can Go Wrong with LLM Output

- 1.1. Taxonomy of LLM code failures: hallucinated APIs, logic drift, silent security vulnerabilities
- 1.2. Threat categories: correctness, security, architecture, testing, drift
- 1.3. Why human review doesn't scale — and what replaces it
- 1.4. Workshop: Generate and examine flawed AI code firsthand
- 1.5. Building a shared threat model for your team

2. Engineering Practices as Countermeasures

- 2.1. Facilitated exploration: which legacy practices counter which threats?
- 2.2. Mapping decades-old engineering discipline to modern LLM failure modes
- 2.3. The revelation: we already have the tools — AI just makes them non-negotiable
- 2.4. Workshop: Connect your threat board to concrete practices

3. The Right Tool for AI-Assisted Development

- 3.1. The AI coding tool landscape: chat, autocomplete, IDE, CLI agent
- 3.2. Why CLI agents integrate best with quality gates (tests, CI, trunk-based flow)
- 3.3. Evaluating your current tool against the trust framework requirements
- 3.4. Headless execution and why it matters

4. Claude Code Quick Start

- 4.1. Getting started with Claude Code (or OpenCode, Aider)
- 4.2. CLAUDE.md: your project's persistent memory
- 4.3. Context management and preventing context poisoning
- 4.4. Workshop: Create your first CLAUDE.md

5. Spec-Driven Development with C4

- 5.1. Specifications as verifiable contracts for AI-generated code

BOTTEGA IT minds

- 5.2. C4 architecture diagrams for communicating vision to AI
- 5.3. How spec + C4 context becomes the contract AI must satisfy
- 5.4. Workshop: Write specs and C4 diagrams that constrain AI execution

6. TDD with AI — Tests as Trust

- 6.1. AI writes both tests and implementation — you supervise the tests
- 6.2. Writing tests that are readable and intention-revealing
- 6.3. Reviewing a test suite as a faster and more reliable verification than reviewing code
- 6.4. Workshop: Supervise AI-generated test suites on a real feature

7. Behavioral Verification — Pragmatic BDD

- 7.1. BDD-style scenarios as executable business rule verification
- 7.2. Practical behavioral contracts — not ceremonial Gherkin
- 7.3. Data-driven scenarios that verify AI output without reading implementation
- 7.4. Workshop: From spec to behavioral scenarios to verified implementation

8. Trunk-Based Development for AI-Assisted Teams

- 8.1. Short-lived branches and continuous integration with AI-generated changes
- 8.2. Small merges that keep AI contributions reviewable and reversible
- 8.3. Branch strategies that match AI development pace
- 8.4. Workshop: Trunk-based workflow with AI-assisted feature development

9. Feature Flags — Deploy Without Fear

- 9.1. Feature flags as safety nets for AI-generated code in production
- 9.2. Gradual rollout, instant rollback
- 9.3. Decoupling deployment from release
- 9.4. Workshop: Implement feature flags with rollout and rollback strategies

10. Monitoring AI-Generated Code in Production

- 10.1. Observability and alerting that closes the trust loop post-deploy

BOTTEGA IT minds

- 10.2. Defining anomaly conditions tied to feature flags
- 10.3. Automatic flag kill switches — production protects itself when AI code misbehaves
- 10.4. Workshop: Set up monitoring with anomaly-triggered flag termination

11. Capstone — Your Team's Trust Framework

- 11.1. Combining all practices into a repeatable framework
- 11.2. Adapting the framework to your team's codebase and workflow
- 11.3. Workshop: Design your team's trust framework document

12. The Proof — Ship a Feature Without Reading the Code

- 12.1. Implement a real change using every layer of the framework
- 12.2. Supervise specs, tests, pipeline, and monitors — not the production code
- 12.3. Without reading a single line of AI-generated implementation
- 12.4. Workshop: End-to-end feature delivery through trust, not review