TWENTY-SEVENTH ANNUAL

TestConX™

Archive

DoubleTree by Hilton
Mesa, Arizona
March 1-4, 2026

CRAFT: Contextual Retrieval & Assembly Framework for Test Programs

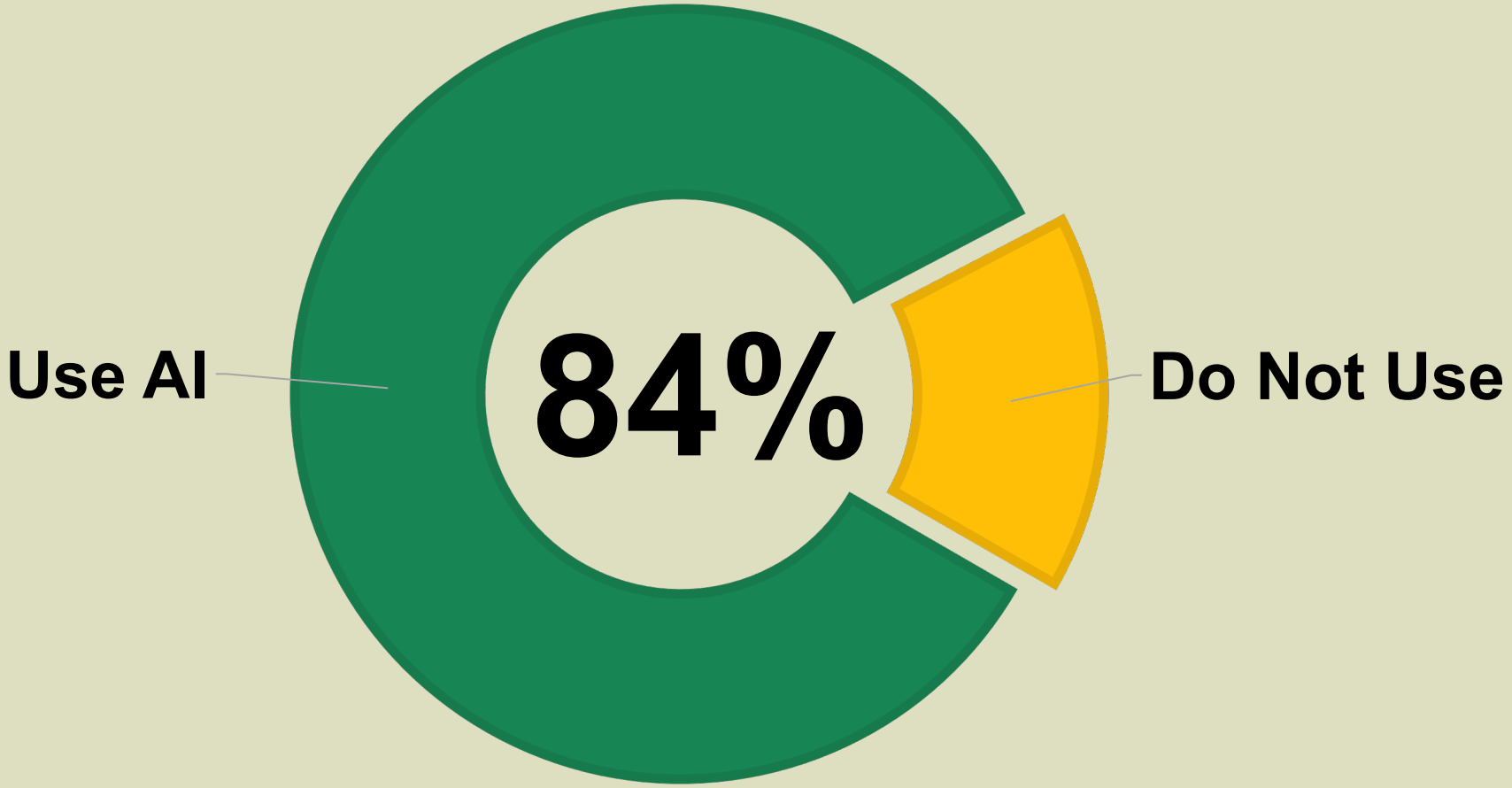
Navnath Raut, Mykola Zakharchuk
Advantest

Mesa, Arizona • March 1–4, 2026

Agenda

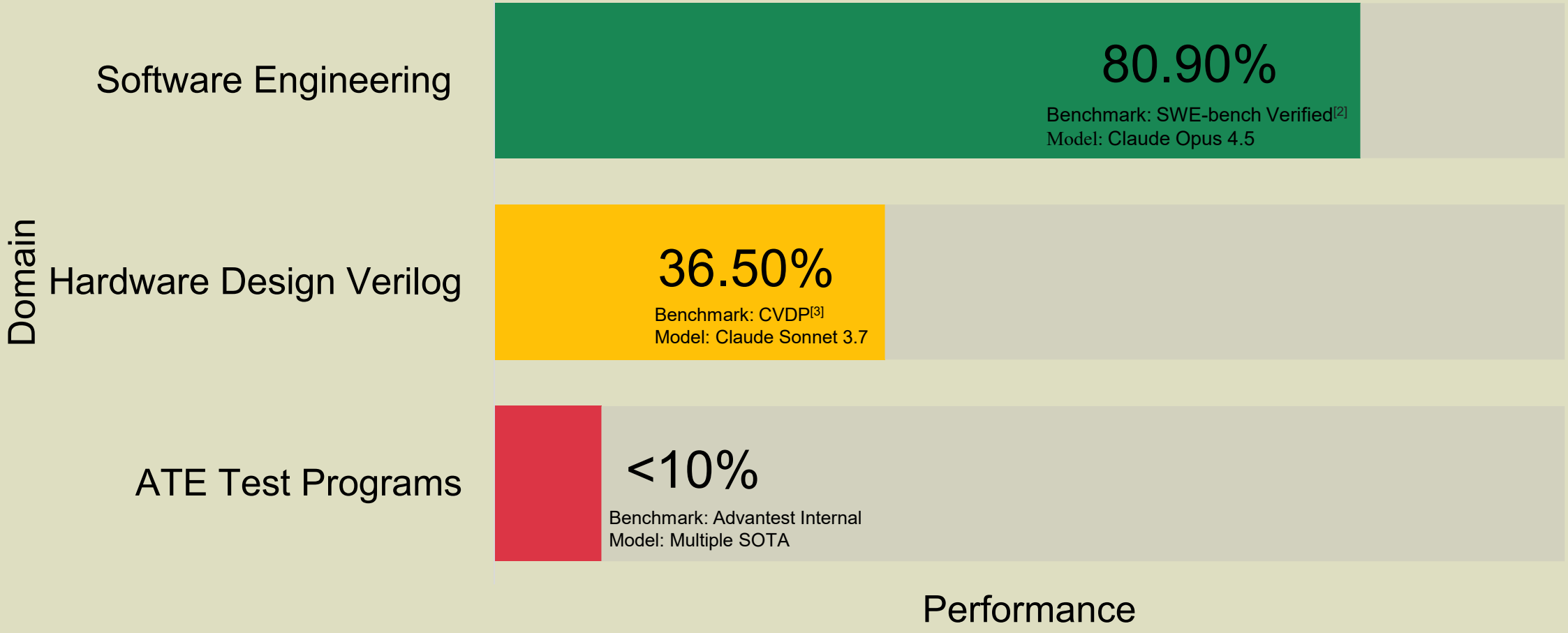
- Motivation
- Problem
- CRAFT Approach
- CRAFT Architecture
- Example
- Conclusion

The Global Shift to AI Coding



Source: https://survey.stackoverflow.co/2025/

The ATE Reality Check



[2]: https://www.anthropic.com/news/claude-opus-4-5
[3]: https://arxiv.org/html/2506.14074v1

The Training Data Void

Public Open Source

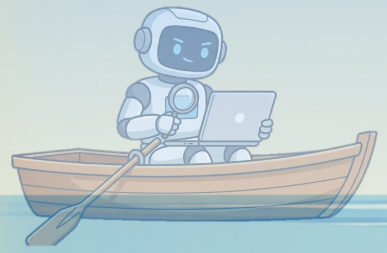
- GitHub, Stack Overflow, Wikipedia.
- Trillions of examples.
- **Result:** AI becomes an **Expert**.

Proprietary ATE Test Programs

- Internal Servers & Secure Labs.
- No public examples
- **Result:** AI has **Zero Understanding**.

"Zero Training Exposure = Zero Domain Intelligence."

The Invisible Context



Generic Coding

- Standard Java, C++ Syntax
- File Structures

Invisible Context

Hardware Context

- Pin Maps
- Voltage Limits
- Timing

Project Context

- Device specification

Domain Semantics

- ATE Platform like SmarTest 8 APIs, etc.

Summary: Why AI Fails in Test Programs

Barrier 1: Domain Gap
(Training Data Void)

- AI systems are not trained on proprietary ATE Test Programs
- They treat Hardware commands as text strings
- Hallucinated Capabilities

Barrier 2: Context Gap
(Invisible Context)

- Test programs have complex dependencies
- AI cannot trace how a small change in one file affects others
- Broken Dependencies & Integration Errors.

Barrier 3: Determinism Gap
(Probabilistic AI)

- AI systems are probabilistic engines
- They settle for “statistically likely” answers rather than “proven correct” ones.
- AI might give you the right code 99 times, and the wrong code once

The Solution: C.R.A.F.T

Contextual Retrieval & Assembly Framework for Test programs

Domain Gap
(Training Data Void)

Knowledge Injection
- Workflows
- API Definitions
- Golden Examples
- Framework Rules
- …

Context Gap
(Invisible Context)

Context Awareness
- Hardware Specs
- Semantic Knowledge Graph
- …

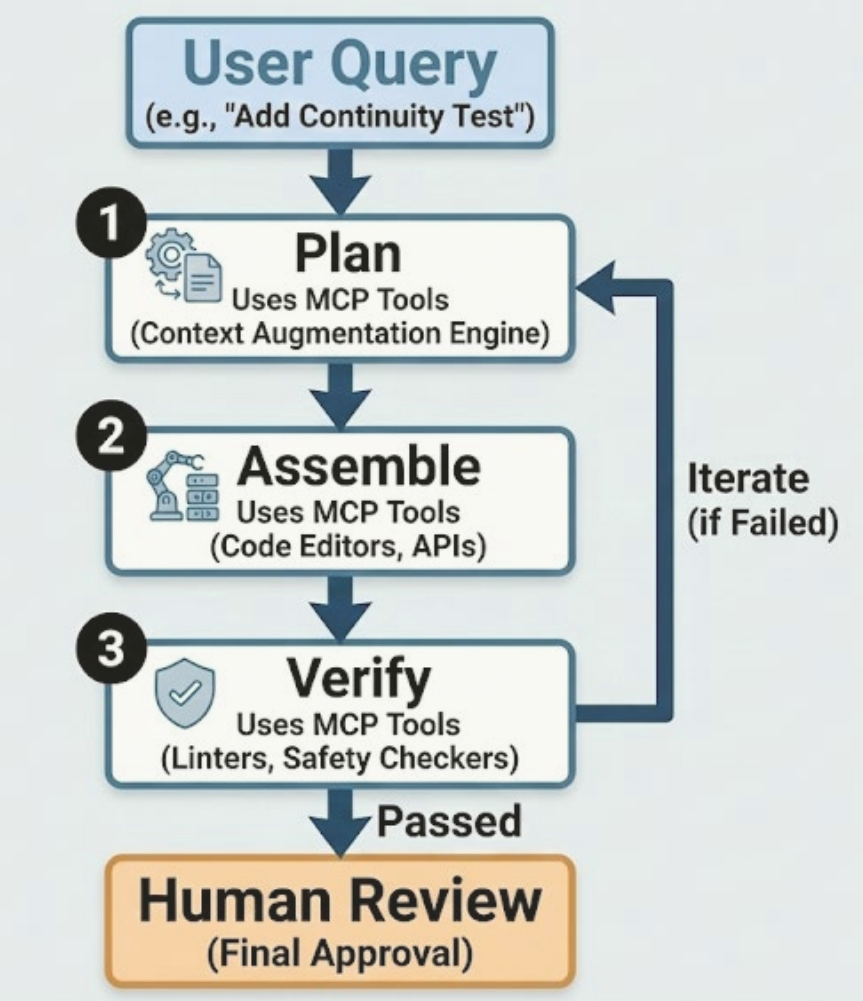
Contextual Retrieval

Determinism Gap
(Probabilistic AI)

Deterministic Validation
- Plan-Assemble-Verify Loop
- Physics-Aware Guardrails
- …

Assembly

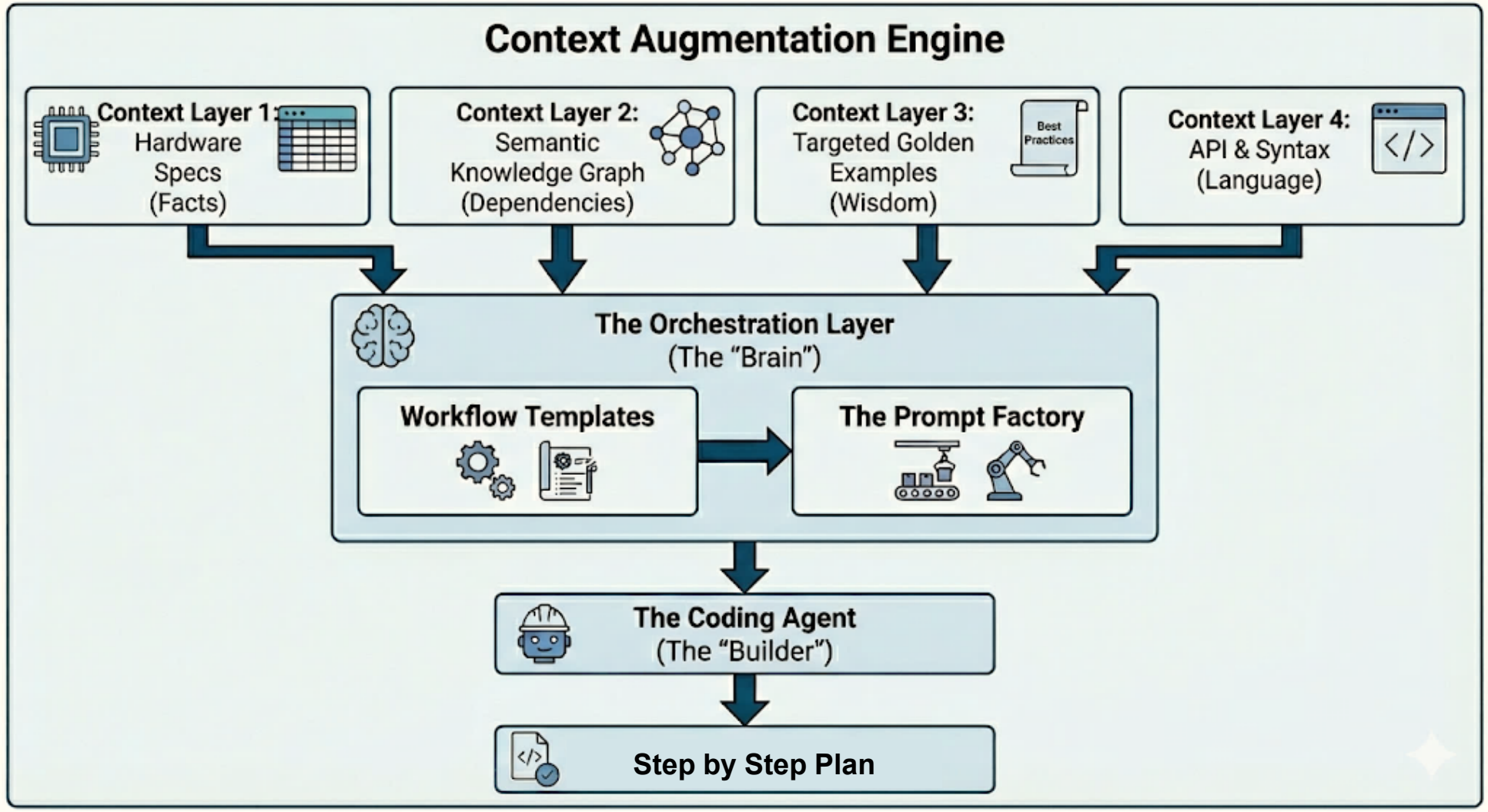
The CRAFT Workflow



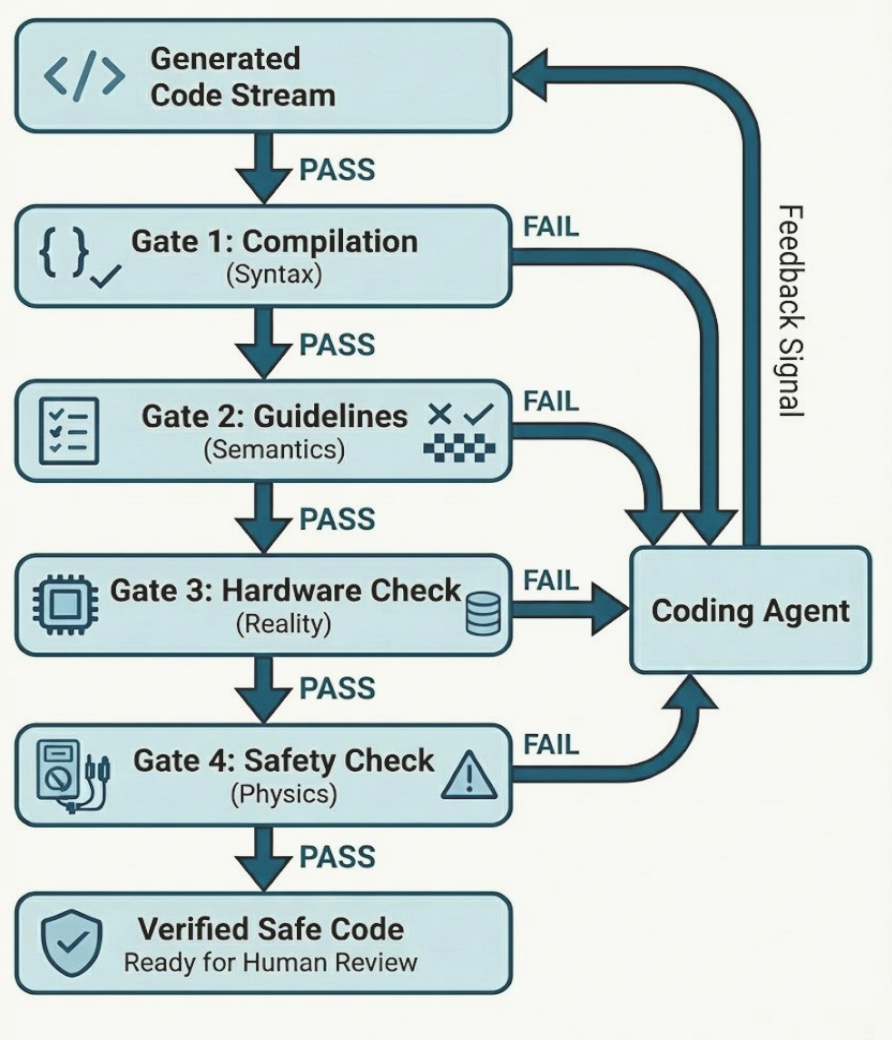
CRAFT resolves the domain knowledge gap and non-deterministic behavior of standard agents by decomposing the task into a three-step lifecycle:

- **Step 1: The Plan (Context Injection)**
  - Injects Domain Knowledge and Project Context to build a rigid engineering plan.
- **Step 2: The Assembly**
  - Translates the plan into code.
- **Step 3: The Verification**
  - Deterministic validation tools block violations. If it fails, the loop retries.

CRAFT Architecture: Context Augmentation

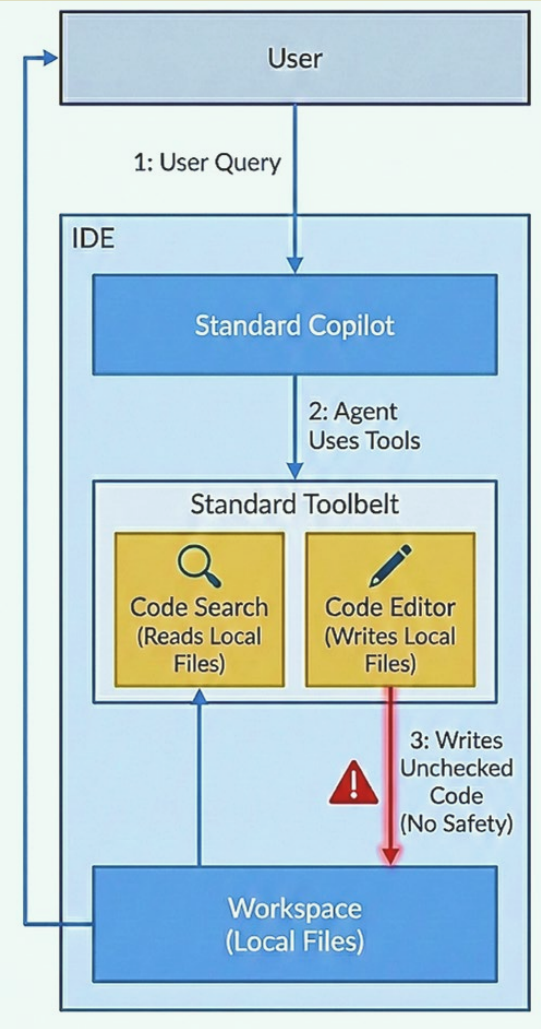


CRAFT Architecture: Deterministic Validation



- **Deterministic Validation**
  - **Compilation:** Feedback from IDE
  - **Guidelines:** Scan for Patterns and Anti-patterns against predefined rules and guidelines
  - **Hardware Check:** Do the pins and instruments exist in hardware definitions?
  - **Specs:** Are voltage/current levels safe against spec limits?
- **Feedback Loop**
  The orchestration layer analyzes the failure type to route the feedback appropriately:
  - Implementation Failures (Compilation & Guidelines)
  - Constraint Failures (Physics/Entities)

Standard Coding Agent Workflow



User Request: “Add TEMP_SENSE pin to the Buck converter test”

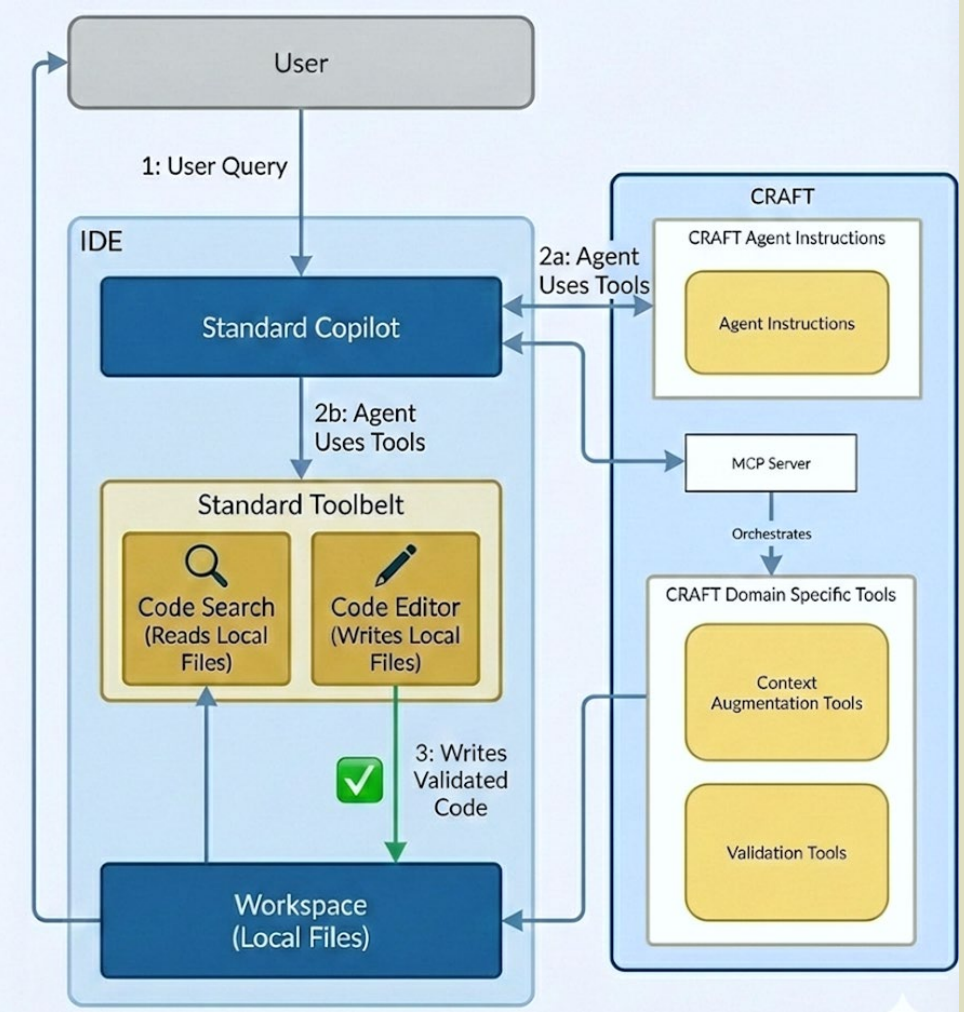
Standard Coding Agent:

- Uses Code Search Tool to find relevant file
- Reads TestMethod file for Buck Converter Test (**Misses other dependencies**)
- Updates the Test Method Code (Possible **Hallucinations)**

Files missed: DUT Board Description, Testflow, Specification File

Hallucinations in Code: APIs, Pin/Sensor spec

Coding Agent with CRAFT Workflow



User Request: "Add TEMP_SENSE pin to the Buck converter test"

Coding Agent with CRAFT Tools

- **Strategic Planning:** Agent selects workflow using Agent Instructions.
- **Dependency Mapping:** Identifies affected files via Knowledge Graph.
- **Context Injection:** Retrieves Golden Examples, APIs, and specific Sensor Specs.
- **Human-in-the-Loop:** Engineer reviews and approves the step-by-step plan.
- **Physics Guardrails:** Validation tools enforce hardware safety limits during implementation

Conclusion

The Problem: Standard AI is Ineffective & Unsafe

- **Ineffective:** low success rate due to missing proprietary context
- **Unsafe:** Generates syntactically correct code that **violates hardware safety limits**

The Solution: CRAFT Architecture

- **Context Injection: Eliminates domain blindness** by retrieving knowledge & dependencies
- **Deterministic Validation: Enforces deterministic Validation** that mitigates probabilistic generation

The Outcome: Engineering Velocity

- **Zero Hardware Hallucinations**
- Engineers spend more time architecting Test programs than coding

COPYRIGHT NOTICE

www.testconx.org