MULBERRY TECHNOLOGIES LLC

Engineering Discipline in Motion: Rusty-BOS Case Study

How SDD + TDD + DevSecOps Practices Enable Production-Ready, Memory-Safe Infrastructure

PROJECT OVERVIEW

Rusty-BOS is a proof-of-concept railroad Positive Train Control (PTC) Back Office Server (BOS) & Computer Aided Dispatch (CAD) platform designed to demonstrate how Specification-Driven Development (SDD) + Test-Driven Development (TDD) + Development Security Operations (DevSecOps) engineering discipline with AI-accelerated development can produce production-ready, memory-safe systems. The project spans 5 integrated Rust crates (~150K+ lines of code) and handles secure protocol conformance, real-time message routing, track topology management, crew authority verification, and operator dashboards.

This project was started on 17-October-2025 with no outside funding or direction. This project is based on the subject matter expertise and experience of the founder of Mulberry Technologies and is used to show that an AI-accelerated Software Development Lifecycle (SDLC) can produce production grade software. The Rusty-BOS project is an excellent proving ground for new AI-accelerated processes and procedures and was used as a learning platform for the founder to gain experience and knowledge regarding how best to effectively and efficiently employ AI as a development accelerator.

Current status: High-readiness core infrastructure (foundation library, protocol engine, administrative CLI, AI diagnostics) with a clear architectural roadmap toward production deployment. The platform demonstrates compile-time memory safety (CISA/ONCD alignment) while maintaining performance, operational correctness, and full DevSecOps automation from commit to deployment.

COMPANY DATA & CODES

Entity: Mulberry Technologies LLC

Socioeconomic: SDVOSB (VetCert Pending)

UEI: FLEZB9N4SKR7

CAGE: 23X34

Primary NAICS: 541511 (Custom Programming)

Secondary NAICS: 513210 (Software Publishers)

PSC Tower Codes: DA01 (IT Labor), DD01 (Delivery)

THE CHALLENGE

Mission-critical infrastructure demands zero memory vulnerabilities, deterministic state transitions, and strict protocol conformance under real-world load. Traditional C/C++ approaches introduce buffer overflows, race conditions, and use-after-free bugs; these require expensive post-deployment patches in high-consequence environments. Development velocity often conflicts with safety: manual testing bottlenecks, late-stage protocol failures, and regressions slow iteration and increase risk.

Hypothesis: *A disciplined AI-accelerated SDD/TDD/DevSecOps pipeline in Rust can deliver both safety guarantees and rapid, confident iteration.*

RUSTY-BOS ARCHITECTURE: EVOLUTION & LESSONS LEARNED

Five-Crate Modular Architecture

- **Shared Foundation (rusty-bos-common):** Shared infrastructure including RBAC, TLS/mTLS, protocol parsing, WebSockets, PostgreSQL abstraction, alert and audit services.
- **Protocol Engine (rusty-bos-ptc):** Core S-9361 (Positive Train Control messaging) protocol layer, message routing, 28+ message type handlers, operations dashboard, and protocol validation simulator.
- **Track Data & Topology System (rusty-bos-cad):** Computer Aided Dispatch (CAD) crate handling track management, topology validation, spatial calculations, and persistent track data structures.
- **Administrative CLI (rusty-bos-admin):** Operational command-line interface for managing locomotives, territories, sessions, and system health.
- **AI Assistant (rusty-bos-ai):** LLM-based diagnostics with semantic search capabilities, integration with the protocol engine for real-time system analysis.

Lesson Learned: Modular separation of concerns isolates blast radiuses, enables parallel feature development, and allows independent testing without requiring the entire system to be running.

Client-Side Rendering → Server-Side Rendering (SSR) Evolution

Early Iteration: Initial client-side rendering (CSR) for operator dashboard created tight coupling between front and back end.

Problem: JavaScript complexity, poor offline behavior, slow first paint on constrained networks. The development process with a CSR dashboard resulted in an almost perpetual re-design upon any update on the server side.

Solution: Migrated to Axum + Server-Side Rendering, generating HTML on the server and sending complete pages to clients. Increased response time and reduced any lag to near imperceptible levels.

Benefit: Deterministic rendering, reduced client-side logic, faster perceived load time, graceful degradation when network connectivity is poor, better accessibility.

Status: Live in rusty-bos-ptc dashboard; operator usability improved measurably.

SDD Alone → SDD + TDD: Test-First Quality

Early Approach: Pure Specification-Driven Development (SDD): write requirements and design, then implement. Used spec-kit to establish the overall project constitution with eight-teen core principals that guided the project from the beginning. Broke down the larger system of a PTC BOS into individual elements that needed to be built for a

successful server. Every component for the PTC BOS and CAD were individual specifications that had the triad of documents, requirements, design, and task list, before implementation was allowed to proceed.

Problem: Implementations diverged from protocol specifications due to: late-stage conformance failures; regression bugs during architectural changes; tech-stack adjustments to better technologies for the project implementation, complexities in the system design.

Solution: Embedded Test-Driven Development (TDD) as a co-equal phase: write conformance test suites before implementation begins. Tests become the executable specification. A two-tier TDD system implements the rigor of Red → Green → Refactor (tier 1) when the code design is the test surface or a regression would break a safety-adjacent contract. The second tier is for observational rather than contract-defining testing or where end-to-end or integration testing already exists upstream. Specifications are allowed to be a mix of tier 1 and tier 2. This ensures TDD is followed in all aspects of development.

Evidence:

- Golden Regression Corpus: 40+ protocol test vectors covering all primary message types and edge cases.
- Zero-Panic Enforcement: All critical paths validated for unwrap-free code.
- Multi-Tier Testing: Unit tests for protocol parsers, property-based tests for edge cases, end-to-end test harness with real TLS encryption and protocol wrapping.

Outcome: 100% protocol conformance pass rate across all test scenarios; zero post-deployment protocol regressions.

ENGINEERING DISCIPLINE: SDD + TDD + DEVSECOPS-FIRST

This three-pillar approach ensures that safety, quality, and security are woven into every phase of development, not bolted on at the end.

4.1 Specification-Driven Development (SDD)

- Requirements, design, and implementation tasks are documented and tracked from inception.
- Every feature is traced back to a formal requirement; every code change links to a design decision.
- Specifications are stored as code-adjacent documents, versioned alongside the codebase, enabling audit trails and change history.

Outcome: Complete traceability from mission need to deployed code.

4.2 Test-Driven Development (TDD)

- Tests written before or alongside implementation, not after.
- Multi-tier validation strategy: unit tests for protocol parsers and business logic, golden regression suites ensuring byte-accurate conformance, property-based testing for edge cases, zero-panic enforcement in critical paths.

Outcome: Regressions caught during development, not after deployment. Confidence to refactor and evolve architecture without breaking deployed behavior.

4.3 DevSecOps-First (Designed & In Development)

Architecture & Roadmap: Designed CI/CD pipeline includes automated security and quality gates at every commit:

- Static analysis (memory safety, unsafe code detection).
- Supply chain validation (dependency vulnerability scanning, SBOM generation).
- Protocol conformance validation via automated simulator.
- Cryptographic integrity checks.

Current State: Testing discipline and zero-clippy enforcement are live; full CI/CD pipeline automation is documented and planned for implementation as the system scales toward production.

Outcome: When complete, continuous compliance with federal memory-safety mandates (CISA/ONCD), not just point-in-time certification.

KEY OUTCOMES & PROOF POINTS

Metric	Evidence
Memory Safety	100% compile-time elimination of buffer overflows, use-after-free, and data races. Zero memory vulnerabilities in critical paths.
Protocol Conformance	40+ golden regression tests; 100% pass rate across all message types and scenarios tested.
Code Quality	Zero compiler warnings on master branch across all 5 crates; 150K+ lines of code compiling and testing clean.
Regression Prevention	Golden test corpus provides safeguard against regressions during refactoring and architectural evolution.
DevSecOps Automation	Testing discipline and static analysis enforcement in place; full CI/CD pipeline automation documented and in development.
Architectural Readiness	High-readiness core subsystems ready for continued development and production scaling. Clear architectural roadmap for evolution.

PATH TO PRODUCTION

Rusty-BOS demonstrates that a disciplined SDD/TDD/DevSecOps AI-accelerated approach can build production-ready systems in a faster than normal timeframe. The platform has reached a stage where core subsystems are stable and well-tested, with a validated roadmap for continued engineering toward full production deployment.

Additional engineering work remains to scale the system to handle higher message volumes and to integrate advanced features, but the foundation is solid, the testing discipline is proven, and the development velocity is sustainable.

TRANSFERABLE PATTERNS FOR MULBERRY TECH CLIENTS

1. Modular Rust Architecture as a proven scaling pattern (multi-crate separation of concerns).

2. SDD + TDD + DevSecOps as a unified quality & compliance framework for safety-critical systems.

3. Automated protocol validation using simulator-based test harnesses (reusable across multiple domains).

4. Server-side rendering for operator dashboards (performance and reliability under constrained networks).

5. Traceability from requirement through deployment enabling audit, compliance discussions, and ATO preparation.

CONTACT INFORMATION

Managing Member: Celso Moreira De Souza Jr.

Phone: +1 (520) 396-8009

Email: contact@mulberrytechsystems.com

Web: https://mulberrytechsystems.com