Table of Contents

1 AI-Augmented Development Methodology

1.1 Introduction: State of the Art Meets Production Reality

Modern software development has reached an inflection point. AI coding agents—particularly Anthropic's Claude Code CLI and similar tools—have transitioned from experimental curiosities to production-grade development accelerators. However, the gap between casual experimentation and effective production usage is measured in extensive, sustained practice, not blog posts written after weekend trials.

This document outlines my approach to AI-augmented development workflows, the economic rationale behind systematic tooling adoption, and the professional requirements necessary for these methodologies to succeed in team environments.

1.2 Depth of Practice

1.2.1 Investment Profile

1000+ hours of hands-on practice across multiple AI coding platforms:

- **Claude Code CLI**: Extensive Agent Skills development, custom workflow design, and adoption advocacy
  - **Usage attestation**: 60,374+ sessions since July 2024 (https://share.cleanshot.com/xN1DCR9X)
  - 528 million tokens processed, $2,953 API costs demonstrating systematic production usage
- **Production deployment**: Not prototyping or experimentation—these tools run critical business workflows daily
- **Cross-platform experience**: Claude Code, Codex, Cursor, and other AI-augmented IDEs
- **Production systems built**: Multiple business-critical systems developed entirely using AI-augmented workflows

What extended practice reveals: 1000+ hours of production usage uncovers capabilities and limitations that aren't apparent in initial exploration. Patterns emerge about what works consistently, what fails unpredictably, and how to structure workflows for reliable results.

1.2.2 Claude Code CLI Specialization

Agent Skills Development: Active contribution to Claude Code ecosystem through:

- Custom Agent Skills creation for specialized workflows
- Internal popularization at Eon Labs (team adoption and training)
- Deep understanding of Claude Code's tool use patterns, context window management, and autonomous agent capabilities
- Experience with both successful and failed automation attempts—understanding each AI model's capability boundaries through iterative audit and constraint refinement

What "mastery" actually means:

- Understanding prompt engineering patterns that produce consistent results

- Structuring prompts to delegate all implementation decisions to AI agents while maintaining human oversight on business requirements
- Designing context-aware workflows that leverage large context windows effectively
  - When Claude Code auto-compacts sessions that exceed context limits, maintaining continuity requires explicit state management: single-source-of-truth plan files (YAML/JSON), SLO validation gates tracked in TodoWrite, and success criteria documented in-repository
  - Multi-step workflows spanning sessions succeed when each phase has clear completion signals, not implicit state—allowing the next session to resume from explicit checkpoints rather than reconstructed context
- Building reliable systems on top of non-deterministic AI outputs
- Managing API costs through efficient prompt design and caching strategies
  - Two key techniques for cost-efficient prompt design: Agent Skills (composable, reusable prompt modules that load on-demand) and Progressive Disclosures (revealing information incrementally rather than front-loading entire context)
  - These patterns substantially reduce token usage compared to monolithic prompts while maintaining or improving output quality

1.2.3 My Practice: AI-First Development Methodology

Through extensive production usage, my workflow has evolved to maximize AI agent capabilities while focusing human effort where it delivers unique value.

Human Role—Bridging Business Reality to AI Implementation:

- **Business requirements interpretation**: Understanding real-world needs and translating them into clear specifications
- **Architecture selection**: Choosing appropriate architectural patterns based on business constraints, scalability needs, and team capabilities
- **Prompt engineering**: Crafting precise, thorough prompts that communicate requirements and context to AI agents
- **Output review and iteration**: Auditing AI-generated implementations, identifying issues, refining prompts based on results

AI Role—All Discretionary Implementation Decisions:

In my practice, AI agents handle:

- Complete implementation (code generation, testing, documentation)
- Architecture execution and detailed design decisions
- Requirements analysis and technical specification
- Code auditing and quality assessment
- Refactoring and optimization decisions

Key Principle: Humans make business-driven decisions (what to build, why it matters, which architecture fits business needs). AI makes all discretionary technical decisions (how to implement, which libraries, specific patterns, code structure).

Why This Works: The interface between business reality and AI capability is human prompting. My focus is making that bridge as accurate and thorough as possible—not second-guessing technical implementation details that AI agents handle more consistently at scale.

Important Clarification: This describes my evolved methodology after extensive practice. It's not a universal prescription. Teams adopting AI-augmented development typically progress gradually, and different practitioners find different balances based on their experience level and comfort with agent autonomy.

1.3 Core Principle: Let AI Agents Use Standard Patterns

Business advantage comes from proprietary knowledge (NSW regulatory expertise, operational workflows). **Software robustness comes from standard patterns** (widely-used libraries, community-validated architectures).

Through extensive production usage, I've learned to let AI agents handle all discretionary technical decisions using idiomatic, modern patterns—while reserving human judgment for business requirements and domain-specific NSW regulatory knowledge.

Practical approach:

- Start with minimal constraints, audit outputs systematically
- Let agents discover current best practices rather than prescribing outdated approaches
- Add constraints only where audit reveals actual problems, not pre-emptively
- Trust agents for routine implementation, reserve human judgment for architecture and NSW compliance

For Netstrata's $12-14M software investment: Competitive advantage comes from the decade of NSW strata management expertise embedded in the system, not from custom technical patterns. Using standard, well-supported tools maximizes maintainability and protects that investment.

1.4 Knowledge Transfer Approach

Training methods: Onboarding workshops (Claude Code CLI setup, prompt engineering basics), pair programming sessions (hands-on problem solving on actual codebase), custom workflow design (building Agent Skills for recurring tasks), ongoing assessment and iteration.

Realistic expectations: AI agents don't replace engineering judgment—they reallocate it from implementation details to business requirements and architecture. Learning curve is weeks to proficiency, months to mastery. Some developers embrace these tools quickly; others need more time. Effective adoption requires collective buy-in, not forced adoption.

1.5 Demonstration-First Assessment

My approach: Work on real Phase 1 completion tasks using AI-first workflows, document measurable outcomes, let the team assess value based on evidence from their actual codebase—not theoretical arguments.

30-60 Day Trial Period:

- Pair programming sessions on actual Netstrata codebase challenges
- Document time savings, code quality improvements, velocity gains
- Results-based decision: if methodology accelerates Phase 1 completion, explore broader adoption; if fit isn't there, we have clear evidence

Why timing matters: Early adoption provides 12-24 months of velocity advantage before these tools become industry standard. By the time broader adoption occurs, Netstrata will have integrated AI-first development into end-2026 completion timeline.

Mutual assessment process:

- *Netstrata assesses*: Do AI-augmented workflows deliver measurable value? Does velocity improvement justify learning investment?
- *I assess*: Is team open to modern development practices? Will leadership support capability building?
- Better to identify alignment or misalignment through demonstration and data

Team adoption approach: Gradual and organic—start with demonstrations, let interested team members experiment, build capability progressively. Not forcing adoption, but creating environment where exploration is encouraged with leadership support. Proficiency takes weeks/months of practice.

1.6 Conclusion: Methodology Informs Contribution

This AI-first methodology directly shapes how I'd contribute to Netstrata's three phases:

Phase 1 (Software Completion): Apply AI-first workflows to Phase 1 completion tasks, document measurable velocity gains, transfer knowledge to interested team members, accelerate progress toward end-2026 deadline through systematic agent orchestration.

Phase 2 (WA Migration): AI agents excel at migration infrastructure—documentation generation, validation scripts, data transformation pipelines. Apply AI-first development to WA customer rollout preparation.

Phase 3 (External Rollout): If Phase 1-2 demonstrations prove value, scale AI-first practices across broader external rollout efforts. Let results from earlier phases inform adoption decisions.

The core insight: Early adoption of evolved methodologies provides competitive advantage. For a $12-14M software investment approaching end-2026 completion, AI-first development offers measurable velocity gains at the most critical moment.

Early adoption provides 12-24 months of practical advantage before these tools become industry standard. Let production results on Netstrata's actual codebase demonstrate value.