Git and AI Coding Agents for Government Compliance: A Human-in-the-Loop Methodology for Federal Information Security Requirements

Bruce Dombrowski
Independent Researcher
GitHub: brucedombrowski

February 10, 2026
Version: `f6f06b3`

Abstract

Government software development demands rigorous compliance with federal standards including NIST Special Publications, FIPS cryptographic requirements, and CUI handling regulations under 32 CFR Part 2002. These requirements impose significant documentation overhead—formal requirements traceability, decision memoranda, verification matrices, and regulatory cross-references—that traditionally consumes substantial engineering effort. This paper presents a methodology for combining *git version control* and *AI coding agents* to address this burden. Git provides the tamper-evident audit trail, branching workflow, and configuration management that government frameworks require; AI coding agents draft compliance artifacts, generate structured requirements, and produce traceability documentation under human review. Together, they shift the engineer's role from author to reviewer while maintaining the human-in-the-loop oversight that compliance frameworks demand. We present a five-phase methodology—from requirements capture through version-controlled interaction traceability—and evaluate it through three case studies: a CUI email encryption tool, a formal decision documentation system, and a Security Verification Toolkit implementing automated NIST SP 800-53 control verification. Using Claude Code (Anthropic) as the AI agent implementation, a single engineer produced 1,087 commits, 227,000+ lines of code, and 149 release tags across 17 repositories in 67 calendar days—including 7 decision memoranda, 29 formally traced requirements, and automated verification of 14 NIST SP 800-53 controls. The methodology is agent-platform agnostic; the principles apply to any agentic AI tool with file system access and human approval workflows.

Keywords: AI coding agents, git, version control, government compliance, NIST, FIPS, CUI, controlled unclassified information, large language models, software engineering, federal information security, configuration management

1 Introduction

Regulated software development imposes a dual burden on developers: the software must correctly implement domain-specific standards, and the *process* of building that software must be formally documented. This burden is not unique to any single domain—it applies equally to safety-critical systems (DO-178C, IEC 61508), information-critical systems (HIPAA, SOX), and the federal information security context examined in this paper. A tool that encrypts files per FIPS 197 [1] is insufficient if the development team cannot produce a requirements traceability matrix linking each implementation decision to the governing standard. This documentation overhead—decision memoranda, verification documents, requirements specifications—is where many small teams and independent developers struggle to meet government expectations.

Two technologies converge to address this gap. First, *git version control* provides the tamper-evident, cryptographically hashed change history that government configuration management standards (NIST SP 800-53 CM-3) require. Every change is attributed, timestamped, and linked to its parent state; the commit log serves as a permanent audit record that cannot be silently altered. Second, *AI coding agents*—large language models that operate directly within the developer's file system and terminal, reading source files, generating artifacts, and executing commands under human approval—can draft compliance documents, suggest standard references, and produce structured artifacts at a pace that manual authoring cannot match. However, government work demands accuracy: an incorrect citation to a NIST Special Publication or a mischaracterized FIPS requirement could undermine an entire compliance package.

The combination of git and AI coding agents creates a workflow where the AI drafts and the human reviews, with every interaction captured in a version-controlled, auditable record. This paper demonstrates this approach using Claude Code (Anthropic) as the AI agent implementation, though the methodology applies to any agentic AI tool with file system access and a human-in-the-loop approval model.

This paper makes the following contributions:

1. A five-phase methodology for using AI agents in government compliance software development, from requirements capture through version-controlled interaction traceability.

2. Three case studies demonstrating AI-assisted development of compliance artifacts: SendCUIEmail (a CUI encryption tool), a LaTeX-based decision memoranda system, and a Security Verification Toolkit implementing automated NIST SP 800-53 control verification.

3. An audit traceability framework using git (configuration management) and GitHub issues (interaction logging) to provide bidirectional provenance between human directives and AI-generated artifacts.

4. A standards-based review process mapped to IEEE 1028, NIST SP 800-53, and ISO/IEC 25010, with enforced separation of duties between authoring and auditing agents.

5. A multi-agent architecture for compliance projects, with role-based separation of duties and quantitative output analysis.

6. A practical adoption pathway from commercial AI tools through open-source validation to enterprise deployment within FedRAMP-authorized environments.

2 Background and Related Work

2.1 Government Compliance Landscape

Federal information security is governed by a layered framework of executive orders, regulations, and technical standards. Executive Order 13556 established the Controlled Unclassified Information (CUI) program, implemented through 32 CFR Part 2002 [2]. The National Institute of Standards and Technology (NIST) provides the technical backbone through publications including:

- **NIST SP 800-171** [3]: Protecting CUI in Nonfederal Information Systems
- **NIST SP 800-53** [4]: Security and Privacy Controls for Information Systems
- **NIST SP 800-132** [5]: Recommendation for Password-Based Key Derivation
- **FIPS 197** [1]: Advanced Encryption Standard (AES)
- **FIPS 140-2** [6]: Security Requirements for Cryptographic Modules

Compliance requires not only that software implementations adhere to these standards, but that organizations maintain documentation demonstrating adherence—what auditors term "evidence of compliance." This evidence typically includes requirements specifications, design decisions, test plans, and verification matrices that trace each requirement to its implementation and test.

2.2 AI-Assisted Software Development

The application of large language models to software engineering has been studied extensively [7], with recent surveys covering both documentation automation [8] and the emerging role of autonomous coding agents [9]. Code generation tools such as GitHub Copilot, Amazon CodeWhisperer, and Anthropic's Claude have demonstrated capability in producing syntactically correct code across multiple languages. In the public sector, Ng et al. [10] report 21–28% productivity improvements from AI coding assistants at GovTech Singapore, along with governance recommendations for using cloud-based tools with open-source code and self-hosted alternatives for confidential government projects—a distinction that directly parallels the adoption pathway discussed in Section 8.

However, the application of LLMs to *compliance-oriented* development—where correctness encompasses not just functional behavior but regulatory adherence—remains underexplored. Marino et al. [11] benchmark LLMs' ability to assess regulatory compliance (EU AI Act), demonstrating that frontier models can approximate expert judgment, but their work focuses on compliance *assessment* rather than artifact *generation*. Prior work on AI-assisted documentation generation has focused primarily on API documentation [12] and code comments. The generation of *regulatory* documentation—where the AI must reason about the relationship between code implementations and published standards—presents distinct challenges including citation accuracy, regulatory interpretation, and the need for conservative (rather than creative) text generation.

The current generation of AI coding tools spans a spectrum of integration depth. *Inline completion* tools (GitHub Copilot, Amazon CodeWhisperer, Tabnine) operate within the editor, suggesting code as the developer types. These tools excel at reducing keystroke-level effort but lack the broader project context needed for compliance work—they cannot read a NIST standard reference and produce a corresponding requirements document. *Chat-based* tools (ChatGPT, Gemini) provide conversational interfaces but operate in isolation from the developer's file system, requiring manual copy-paste of code and artifacts. *Agentic* tools (Claude Code, Cursor, Windsurf, Aider) operate directly within the developer's environment, reading and writing files, executing commands, and maintaining session context. This agentic architecture is essential for compliance work, where the AI must simultaneously reason about source code, published standards, and the traceability relationships between them.

The critical property for compliance applications is an *explicit approval model*: every file write, command execution, and code edit requires human confirmation. Huang et al. [9] confirm that professional developers maintain deliberate oversight of AI agents' design decisions rather than delegating control—a pattern that aligns with the human-in-the-loop oversight government compliance frameworks (NIST SP 800-53 AC-5, SA-11) require. Tool permission systems also enable the separation-of-duties pattern described in Section 7, where review agents are denied write access at the tool level rather than by convention. Claude Code implements this model; other agentic tools vary in the granularity of their approval workflows.

2.3 Agentic AI Tool Architecture

AI coding agents operate as command-line tools with access to the developer's local environment. The architectural properties essential for compliance work include:

1. **File system access**: The agent reads and writes files directly, enabling it to analyze source code and produce artifacts in-place.

2. **Tool use with approval**: Each action (file read, edit, command execution) requires developer approval, providing the human oversight that compliance frameworks demand.

3. **Context persistence**: The agent maintains conversation context across a session, allowing iterative refinement of compliance artifacts.

4. **Instruction files**: Projects include persistent instruction files (e.g., `CLAUDE.md` in Claude Code [13], `.cursorrules` in Cursor) that encode project-specific compliance requirements across sessions.

5. **Multi-agent orchestration**: Configurations that enable multiple specialized agents—each with defined roles, model selection, and permitted tools—to collaborate on a single project.

6. **Session continuity**: Sessions persist across interruptions, preserving the accumulated compliance context that would otherwise need to be reconstructed.

To implement this architecture, the project records its agent invocation configuration in a version-controlled instruction file. Table 1 maps the methodology-level concepts to their implementation in the tool used in this paper's case studies.

Table 1: Methodology concepts and their implementation (Claude Code)

Concept	Implementation
Multi-agent config	`-agents` (JSON file)
Model selection	`-model` (reasoning vs. throughput)
Tool restrictions	`-allowedTools` (per role)
Session resume	`-continue`
Audit logging	`-verbose`

3 Methodology

We developed a methodology for AI-assisted government compliance development organized around five phases, illustrated in Figure 1. Each phase leverages AI coding agent capabilities while maintaining the human-in-the-loop oversight essential to compliance work.

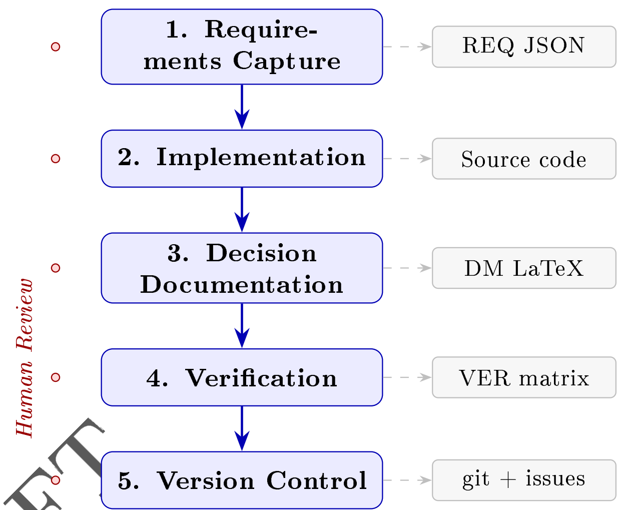

Figure 1: Five-phase methodology for AI-assisted compliance development. Human review occurs at every phase transition.

3.1 Phase 1: Requirements Capture

Government projects begin with requirements derived from applicable standards. In our methodology, the developer identifies the governing standards (e.g., NIST SP 800-132 for key derivation) and instructs the AI agent to generate a structured requirements document.

The agent produces requirements in machine-readable JSON format, enabling downstream tooling to generate formatted documents and traceability matrices. Each requirement includes:

- A unique identifier (e.g., `REQ-1.1`)
- The governing standard and section reference
- The requirement text
- Classification as mandatory or recommended

- Verification method (inspection, test, analysis)

Listing 1 shows an excerpt from the SendCUIEmail requirements document, generated with AI agent assistance and reviewed by the developer.

```json
{
  "id": "REQ-1.1",
  "standard": "FIPS 197",
  "section": "Section 1",
  "text": "The tool SHALL use the Advanced
    Encryption Standard (AES) algorithm
    for all file encryption operations.",
  "priority": "mandatory",
  "verification": "inspection"
}
```

Listing 1: Requirements specification excerpt (REQ-2026-001)

Requirement text uses RFC 2119 [14] keywords (SHALL, SHOULD, MAY) to distinguish mandatory from recommended requirements, following the convention established in IETF and NIST publications. The developer's role shifts from *authoring* requirements to *reviewing* them—verifying that the AI's interpretation of the standard is correct and that no requirements are omitted. This review-centric workflow is faster than drafting from scratch while preserving the technical judgment that compliance demands.

3.2 Phase 2: Implementation with Compliance Awareness

During implementation, the AI agent operates within the project's instruction files, which encode compliance standards and architectural constraints. `CLAUDE.md` provides project-wide instructions (build commands, repository scope, conventions), while `AGENTS.md` defines role-specific compliance context (applicable standards, verification methods, regulatory constraints). Both files persist across sessions, ensuring that every agent invocation begins with the correct compliance posture. Listing 2 shows the compliance context from the SendCUIEmail project:

```
## Compliance Standards

- **FIPS 140-2**: AES-256-CBC encryption
- **NIST SP 800-132**: PBKDF2-HMAC-SHA256
   key derivation (100,000 iterations)
- **NIST SP 800-171**: CUI handling
- **32 CFR Part 2002**: CUI marking
```

Listing 2: AGENTS.md compliance context excerpt

This ensures that every agent session begins with awareness of the applicable standards, reducing the risk of non-compliant suggestions.

3.3 Phase 3: Decision Documentation

Government compliance frequently requires documenting *why* a particular approach was chosen, not merely *what* was implemented. Decision memoranda serve this purpose. In our methodology, when the developer makes a design choice (e.g., selecting Cinzel over Trajan Bold for CUI headers, or choosing TikZ over PDF manipulation for form layout), they instruct the agent to generate a formal decision memo.

The LaTeX/Decisions repository implements a template-wrapper pattern where each decision memo defines metadata variables and content, then includes a shared template. Listing 3 illustrates this separation:

```latex
\newcommand{\UniqueID}{DM-2026-002}
\newcommand{\DocumentDate}{January 19, 2026}
\newcommand{\AuthorName}{PDF Tools Working Group}
\newcommand{\SubjectField}{Font Selection for
 CUI Header Text}
\newcommand{\dmContent}{...}
\input{_template.tex}
```

Listing 3: Decision memo template pattern

This pattern enables the AI agent to produce new decision memos by following the established template, ensuring consistency across the documentation package.

3.4 Phase 4: Verification

The verification phase produces documents that map each requirement to its implementation evidence. The agent reads the source code, locates the relevant implementation for each requirement, and generates a verification matrix with file paths, line numbers, and explanatory text.

Table 2 shows an excerpt from the SendCUIEmail verification document.

Table 2: Verification matrix excerpt (VER-2026-001)

Req.	**Evidence**	**Method**
REQ-1.1	`Encrypt.ps1`: `[Aes]::Create()` call	Inspection
REQ-1.2	`$KEY_SIZE = 32` (256 bits)	Inspection
REQ-2.3	`$ITERATIONS = 100000`	Inspection
REQ-3.1	`RandomNumberGenerator` usage	Inspection

3.5 Phase 5: Version Control and Interaction Traceability

The preceding development phases produce artifacts, but compliance also demands *evidence of process*—a verifiable record of who made which decisions, when changes were introduced, and how human-agent interactions shaped the final deliverables. We use git version control and GitHub issues as complementary traceability mechanisms.

3.5.1 Git as Audit Trail

Every meaningful action—creating a requirements document, fixing a review finding, adding a compliance scan—is captured as an atomic git commit on the project's main branch. Each commit message describes the compliance-relevant change (e.g., "Fix all 13 review findings from issue #1; add QA standards framework"). This produces a linear, tamper-evident history that auditors can inspect with standard tooling (`git log`, `git diff`).

Git's properties align directly with government configuration management requirements. NIST SP 800-53 CM-3 (Configuration Change Control) requires organizations to "document, approve, and track changes to the system" [4]. The git commit log serves as this change record: each commit is cryptographically hashed, timestamped, attributed to an author, and linked to its parent commits. Unlike informal change logs, git history cannot be silently altered without breaking the hash chain.

The project uses Semantic Versioning (SemVer) with a `CHANGELOG.md` following the Keep a Changelog convention. Version numbers encode the significance of changes: major versions for structural reorganization, minor versions for new content, and patch versions for corrections. Each release is tagged (`git tag -a vX.Y.Z`) and the changelog entries reference the GitHub issues that motivated each change. This provides a human-readable change history that complements the machine-level detail in the git log.

The Security Verification Toolkit case study (Section 6) embeds the git commit hash directly into its compliance attestation PDFs, binding each attestation to a specific, reproducible configuration state.

3.5.2 GitHub Issues as Interaction Log

While git captures *what changed*, GitHub issues capture *why it changed* and *who directed the change*. All human-agent interactions in this project are logged as GitHub issues using a structured labeling scheme:

- `human-prompt`: A human directive to the AI agent (e.g., "expand agents.json with additional agent roles")
- `agent-output`: Agent-generated analysis or findings (e.g., "13 review findings per IEEE 1028 inspection")
- `decision`: A design or process decision with

rationale (e.g., "IT security standards are standard review criteria")

This labeling scheme creates a queryable audit record. An auditor can filter by `human-prompt` to see every directive the human issued, by `agent-output` to see every AI-generated analysis, or by `decision` to trace the rationale for each design choice. The combination provides bidirectional traceability between human intent and AI action—a key requirement when demonstrating human-in-the-loop oversight to government auditors.

All five agents in the multi-agent configuration (Section 7) include interaction logging in their system prompts, requiring them to create GitHub issues for every substantive human-agent exchange. This ensures that the audit trail is comprehensive regardless of which agent is active.

4 Case Study: SendCUIEmail

4.1 Project Overview

SendCUIEmail [15] is a PowerShell-based tool for encrypting files before email transmission, under active development (currently v0.17.3, prerelease). It is designed for environments where Public Key Infrastructure (PKI) or S/MIME certificate exchange is impractical. The tool addresses a common gap in federal and contractor environments: the need to transmit CUI securely when the only available channel is unencrypted email.

The project's compliance scope spans six federal standards and regulations:

1. **FIPS 197** [1]: AES algorithm specification
2. **FIPS 140-2** [6]: Cryptographic module validation
3. **NIST SP 800-132** [5]: Password-based key derivation
4. **NIST SP 800-38A** [16]: Block cipher modes of operation
5. **NIST SP 800-90A** [17]: Random number generation
6. **32 CFR Part 2002** [2]: CUI marking and handling

4.2 AI-Assisted Artifacts

Over the course of development, the AI agent assisted in producing the following compliance artifacts:

4.2.1 Requirements Document (REQ-2026-001)

A JSON-formatted requirements specification containing 29 requirements across six categories: encryption algorithm, key derivation, random number generation, password handling, file format, and platform requirements. The JSON source-of-truth enables automated generation of formatted PDF documents via a Python build script.

4.2.2 Decision Memoranda (DM-2026-001 through DM-2026-007)

Seven formal decision memos documenting design choices:

- **DM-001**: Cross-platform support strategy
- **DM-002**: File size limit rationale (10 MB)
- **DM-003**: Password transmission method (out-of-band per NIST SP 800-63B [18])
- **DM-004**: Verification document numbering scheme
- **DM-005**: Multi-category CUI support per 32 CFR 2002.20(a)(3)
- **DM-006**: Beta readiness assessment
- **DM-007**: Recipient instruction format selection (HTML)

4.2.3 Verification Document (VER-2026-001)

A line-by-line code verification mapping all 29 requirements to specific implementation evidence in the source code, including file paths, function names, and configuration values.

4.3 Cryptographic Implementation

The core encryption implementation demonstrates how AI-assisted development can produce compliant code. The encrypted file format is:

$$\text{Output} = \text{Salt}_{128} \| \text{IV}_{128} \| \text{AES-256-CBC}(K, \text{IV}, \text{Plaintext}) \quad (1)$$

where K is derived via PBKDF2-HMAC-SHA256:

$$K = \text{PBKDF2}(\text{password}, \text{Salt}, 100000, 256) \quad (2)$$

The implementation uses exclusively platform-provided cryptographic libraries (`System.Security.Cryptography`), avoiding third-party dependencies that would complicate FIPS validation. When Windows FIPS mode is enabled, the tool leverages CMVP-validated cryptographic modules (e.g., Certificate #4515, Kernel Mode Cryptographic Primitives Library, validated under FIPS 140-2 on Windows 10; specific certificate numbers vary by Windows version).

4.4 Recipient Experience Design

A significant AI-assisted design contribution was the recipient decryption workflow. The tool generates a self-contained HTML instruction document (`Decrypt_Instructions.html`) with an embedded PowerShell one-liner, shown in simplified form in Listing 4:

```
$f=Read-Host "File"
$p=Read-Host "Password"
$d=[IO.File]::ReadAllBytes($f)
$k=[Rfc2898DeriveBytes]::new(
    $p,$d[0..15],100000,"SHA256")
$a=[Aes]::Create()
$a.Key=$k.GetBytes(32)
$a.IV=$d[16..31]
$c=$a.CreateDecryptor()
    .TransformFinalBlock($d,32,$d.Length
        -32)
[IO.File]::WriteAllBytes(
    ($f-replace'\.Locked$',''),$c)
```

Listing 4: Decryption logic (simplified from production code)

This design requires no software installation by recipients—only PowerShell, which is built into every modern Windows installation. The production code uses `SecureString` with `SecureStringToBSTR` conversion and file picker dialogs; the listing above is a functionally correct simplification using plaintext password input for clarity. The AI agent helped iterate on the production one-liner to minimize its length while maintaining compliance with the cryptographic parameter requirements.

5 Case Study: Decision Documentation System

The LaTeX/Decisions repository (v0.4, under active development) demonstrates AI-assisted creation of a reusable documentation system for formal decision memoranda. Government programs frequently require Decision Memoranda (DMs) to document technical and policy choices with traceable rationale.

5.1 Template Architecture

The system uses a template-wrapper pattern where a shared base template (`_template.tex`) defines the document layout—headers with organizational logo, footers with document ID and page numbering, and standardized section formatting—while individual decision documents supply metadata and content through LaTeX command definitions.

This separation of concerns enables AI agents to produce new decision memos by populating the established template structure, ensuring visual and structural consistency without requiring the agent to understand the full LaTeX layout implementation.

5.2 SF901 CUI Coversheet Compliance

Three decision memos (DM-2026-001 through DM-2026-003) document the technical approach to generating Standard Form 901 CUI coversheets:

1. **Implementation approach**: LaTeX template recreation rather than PDF manipulation, chosen for alignment with existing infrastructure and independence from external tools.
2. **Font selection**: Cinzel (open-source, SIL OFL) chosen over Trajan Bold (commercial) for the CUI header, balancing visual fidelity with licensing constraints.
3. **Layout strategy**: TikZ with absolute positioning for pixel-precise form reproduction, justified by the form's stability (unchanged since November 2018 per GSA records).

Each decision memo follows the format required by many government programs: identification of options considered, evaluation criteria, selected approach, and rationale with regulatory references.

6 Case Study: Security Verification Toolkit

The third case study examines the Security Verification Toolkit [19], a pure-Bash security scanning and compliance documentation system under active development (currently v2.7.3) that automates the verification of federal security controls. Like all three case studies in this paper, the toolkit is under active development and is presented as a case study in AI-assisted compliance tooling, not as a finished product. Unlike SendCUIEmail (which implements a single compliance function) or the Decision Documentation System (which manages process artifacts), the toolkit addresses the *continuous compliance verification* challenge: demonstrating ongoing adherence to NIST SP 800-53 and NIST SP 800-171 controls through automated scanning and attestation generation.

6.1 Scope and Standards

The toolkit implements 14 NIST SP 800-53 controls and 11 NIST SP 800-171 controls across eight security control families, with each scan mapped to its governing control in machine-readable JSON. The standards addressed include:

- **NIST SP 800-53** [4]: AU-2/3 (Audit Events and Records), CA-2 (Assessment), CM-6/8 (Configuration), MP-6 (Media Sanitization), RA-5 (Vulnerability Scanning), SA-11 (Developer Testing), SC-8 (Transmission Protection), SI-2/3/4/5/12 (Information Integrity)
- **NIST SP 800-171** [3]: 11 corresponding CUI protection requirements
- **NIST SP 800-88** [20]: Media sanitization (secure deletion)
- **BOD 22-01** [21]: CISA Known Exploited Vulnerabilities cross-referencing
- **FIPS 199** [22]: Security categorization of federal information

6.2 Requirements Traceability

The toolkit maintains a complete traceability chain in JSON format:

$$\begin{aligned}&\text{Requirement} \rightarrow \text{NIST Control} \rightarrow \\ &\qquad\qquad \text{Script} \rightarrow \text{Test} \rightarrow \text{Evidence}\end{aligned}$$

A `mapping.json` file links 14 functional requirements (FR-001 through FR-014) to NIST controls, implementation scripts, and test cases, navigable in three directions: by script, by NIST 800-53 control, and by NIST 800-171 control. This bidirectional traceability enables auditors to verify compliance from any starting point—a requirement that many government programs mandate but few tools automate.

The AI agent assisted in generating this traceability framework, producing the initial JSON mappings from the NIST control catalog and iterating with the developer to ensure completeness. The machine-readable format enables downstream automation: generating formatted traceability reports, validating that no requirements are orphaned, and detecting when code changes break previously-verified controls.

6.3 Automated Attestation Generation

A distinguishing feature of the toolkit is its automated generation of formal compliance attestations as PDFs via LaTeX templates. After each scan run, the system produces timestamped, checksummed attestation documents suitable for inclusion in government compliance packages. Each attestation includes:

- Toolkit version and git commit hash (configuration management)
- Scan timestamp in ISO 8601 UTC
- SHA-256 checksums of all scan outputs
- NIST control mappings for each finding
- CUI markings where applicable

The toolkit's design philosophy—*"You are only as good as your last scan"*—enforces that every scan run overwrites previous results, preventing stale attestations from being presented as current evidence. This maps directly to the continuous monitoring requirements of NIST CA-7.

6.4 Codebase Structure Evolution

Code survival analysis using git-of-theseus reveals how the toolkit's codebase structure evolved over its 490-commit history. Figure 2 shows that the `scripts/` (implementation) and `tests/` (verification) directories grew in lockstep throughout the development period—from approximately 10,000 and 3,000 lines respectively at the start of the analyzed period to 27,000 and 8,000 lines at the time of writing. This parallel growth indicates disciplined test coverage practices maintained throughout AI-assisted development: new scanning capabilities were consistently accompanied by corresponding test cases.

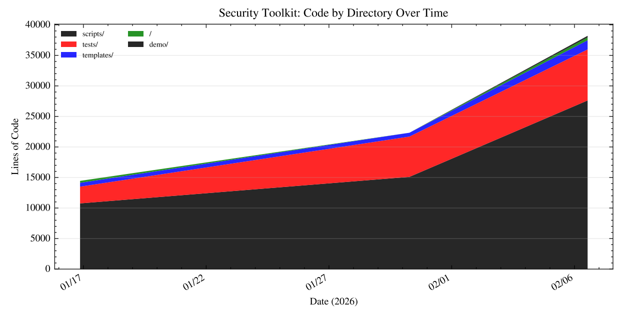

Figure 2: Security Verification Toolkit codebase structure evolution (git-of-theseus). The `scripts/` and `tests/` directories grow in lockstep, indicating consistent test coverage practices throughout AI-assisted development.

6.5 Multi-Agent Development

The toolkit itself was developed using a multi-agent architecture with defined roles: Lead Systems Engineer, QA Engineer, Windows Developer, Documentation Engineer, and Lead Software Developer. Agent coordination uses GitHub issues as the communication channel—the same interaction logging pattern adopted in this paper's methodology. This represents a mature implementation of the multi-agent compliance workflow described in Section 7, validated across 94 version tags and 136 GitHub issues.

7 Multi-Agent Workflow

Multi-agent orchestration enables workflows where multiple specialized AI agents collaborate on a project, each with defined roles, permitted tools, and compliance context. For government compliance work, we propose the role-based agent architecture shown in Figure 3.

7.1 Agent Roles

1. **Project Setup Agent**: Initializes repository structure, creates `CLAUDE.md` with compliance

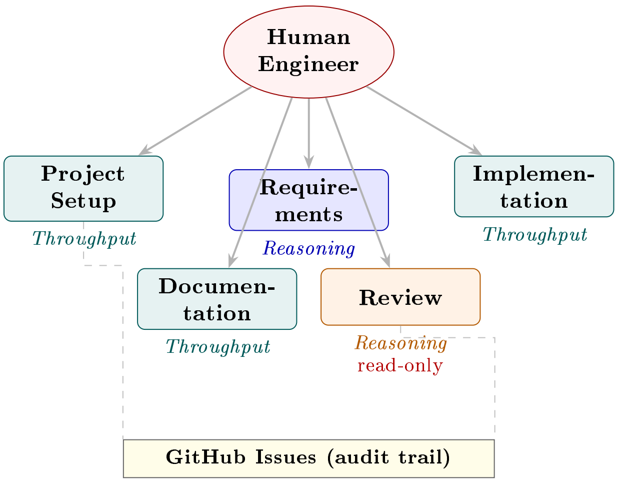

Figure 3: Multi-agent architecture for compliance projects. Teal agents use throughput-optimized models; blue/orange agents use reasoning-optimized models. The review agent has read-only access (NIST SP 800-53 AC-5). All agents log interactions to GitHub issues.

context, establishes documentation templates and directory layout.

2. **Requirements Agent**: Analyzes governing standards and generates structured requirements documents in JSON format.

3. **Implementation Agent**: Writes compliant code within the constraints defined by the requirements and `CLAUDE.md` context.

4. **Documentation Agent**: Produces decision memoranda, verification documents, and traceability matrices.

5. **Review Agent**: Audits artifacts for completeness, citation accuracy, and cross-reference integrity.

7.2 Agent Configuration

Agent definitions are stored in a JSON configuration file that specifies each agent's role, model selection, permitted tools, and a detailed system prompt encoding compliance context. Table 3 summarizes the five-agent configuration developed for this paper.

Model selection reflects the cognitive demands of each role: the `requirements` and `review` agents use a reasoning-optimized model for its stronger performance on regulatory interpretation and cross-reference validation, while `implementation` and `documentation` use a throughput-optimized model for its favorable speed-to-quality ratio on structured, template-following tasks. In our implementation, these correspond to Anthropic's Opus and Sonnet models respectively, but the pattern applies to any model family with tiered capability levels. Notably, the `review` agent is denied write and edit tools, enforcing a separation-of-duties principle where auditors identify problems but do not fix them.

7.3 Workflow Orchestration

The multi-agent workflow proceeds through the five phases described in Section 3, with each agent operating within its defined scope. The key advantage of this architecture is *context isolation*: the requirements agent does not need the full implementation context, and the documentation agent can focus on artifact generation without the overhead of the full codebase in its context window.

This isolation is particularly valuable for government projects where compliance documentation can be extensive—a full NIST SP 800-171 assessment may reference over 100 security requirements, and maintaining all of these in a single agent context is impractical. The separation-of-duties between the `documentation` and `review` agents also mirrors the organizational controls common in government programs, where the author of a compliance artifact should not be the sole reviewer.

7.4 Scrum-Based Agent Orchestration

The pipeline model described above—where agents execute sequentially through defined phases—is effective for linear compliance workflows but does not accommodate the iterative, feedback-driven nature of real-world software de-

Table 3: Agent configuration summary (agents.json)

Agent	Model	Phase	QA Standard	Key Capability
project-setup	Throughput	Setup	—	Repo structure, build config, templates
requirements	Reasoning	Phase 1	IEEE 29148	Standard interpretation, JSON requirements
implementation	Throughput	Phase 2	—	Compliant code within REQ constraints
documentation	Throughput	Phases 3–4	MIL-STD-498	Decision memos, verification docs, LaTeX
review	Reasoning	Cross-cutting	IEEE 1028, NIST 800-53 AC-5	Audit with no write access (read-only)

Note: Phase 5 (Version Control) is performed by all agents via GitHub issue logging. The review agent operates across phases rather than within a single phase.

velopment. An alternative orchestration model maps AI agents to a Scrum team structure, drawing on the Scrum Guide [23] framework that is widely adopted in both commercial and government software programs.

In this model, AI agents assume Scrum roles:

- **Product Owner agent**: Maintains the product backlog (GitHub issues), prioritizes work items based on compliance risk and stakeholder value, and defines acceptance criteria. Uses a reasoning-optimized model for its judgment over regulatory priorities.
- **Scrum Master agent**: Facilitates sprint execution, identifies impediments (blocked issues, failing scans, unresolved review findings), and ensures the team adheres to process standards. Monitors GitHub issues for stalled work and escalates to the human.
- **Developer agents**: One or more implementation and documentation agents that pull work from the sprint backlog and produce deliverables. Use a throughput-optimized model for structured tasks.

This Scrum-based approach offers several advantages for compliance projects:

1. **Iterative refinement**: Rather than producing all requirements before any implementation begins, the team works in sprints where each iteration can incorporate feedback from the previous sprint's review—mirroring the review-centric workflow discussed in Section 8.
2. **Backlog as audit trail**: The GitHub issue backlog serves dual purpose: it is both the Scrum product backlog and the compliance audit trail (Section 3.5). Each sprint produces a traceable increment of compliance artifacts.
3. **Human as stakeholder**: The human engineer acts as the primary stakeholder (and may also serve as the Product Owner), providing direction at sprint boundaries rather than approving every individual action. This scales the human-in-the-loop model to larger projects.
4. **Government familiarity**: Scrum and Agile methodologies are already adopted across federal agencies under guidance such as the GSA 18F Agile Practices Guide and the DoD Agile Software Acquisition Guidebook, reducing the organizational friction of adopting AI-assisted workflows.

A reference implementation of this Scrum-based agent architecture is under development at https://github.com/brucedombrowski/Scrum, applying the Scrum Guide's ceremonies and artifacts to AI agent multi-agent orchestration. This represents a natural evolution from the sequential pipeline model to a more flexible, sprint-based orchestration that better accommodates the iterative nature of compliance development.

7.5 Ecosystem Architecture

The multi-agent workflow, Scrum orchestration, and underlying process framework are maintained as a set of interrelated repositories with clean separation of concerns:

- **systems-engineering** (https://github.com/brucedombrowski/systems-engineering): Defines *how* work is done—the five-phase process, standards framework, traceability model, and artifact conventions.
- **ai-agents** (https://github.com/brucedombrowski/ai-agents): Defines *who* does the work—model-agnostic agent role templates with vendor-specific implementations.
- **Scrum** (https://github.com/brucedombrowski/Scrum): Defines *when* work happens—sprint cadence, backlog management, and Scrum ceremonies.

Individual project repositories (SendCUIEmail, security-toolkit, this white paper) consume these shared definitions while maintaining project-specific instructions. This architecture allows the process, agent templates, and orchestration model to evolve independently and be adopted incrementally by new projects.

8 Discussion

8.1 Quality of AI-Generated Compliance Artifacts

Our experience indicates that AI coding agents produce compliance artifacts that are *structurally sound* but require careful human review for *substantive accuracy*. The AI reliably generates:

- Correct document structure and formatting
- Appropriate standard references (e.g., citing NIST SP 800-132 for PBKDF2)
- Reasonable requirement decomposition
- Accurate code-to-requirement tracing when given source access

Areas requiring human review include:

- *Regulatory interpretation*: Whether a requirement is “mandatory” vs. “recommended” per the governing standard
- *Completeness*: Whether all applicable requirements from a standard have been captured
- *Citation precision*: Verifying specific section numbers within standards
- *Organizational context*: Tailoring requirements to the specific compliance posture of the organization

8.2 Quantitative Output Analysis

Table 4: Aggregate output metrics (67-day period, single engineer)

Metric	Value
Git repositories (measured set)	7
Total commits (measured set)	698
Lines of code (measured set)	113,565
Release tags (measured set)	139
Decision memoranda	7
Requirements (JSON)	29
Verification mappings	29
Security scan scripts	14
Test scripts	14
NIST controls automated	14
GitHub issues (audit trail)	203
Full ecosystem (17 repositories):	
Total commits	1,087
Lines of code	227,000+
Release tags	149
Languages	8

Table 4 summarizes the measurable output of the AI-assisted methodology across the three case studies and supporting infrastructure, produced by a single engineer over 67 calendar days.

The daily commit rate averaged 16.2 commits/day, with the Security Verification Toolkit alone accounting for 490 commits, 94 version tags, and 77,238 lines of code. These figures are not presented as benchmarks—the commit rate reflects a development style characterized by frequent, atomic commits rather than large batch changes—but they indicate the throughput achievable when AI agents handle drafting and the engineer focuses on review and direction.

Figure 4 shows the daily code churn across all repositories. Every period shows net-positive creation (green additions exceeding red deletions), with a sustained deletion rate indicating active refactoring rather than write-once code. The 48,000-line addition spike (late January) corresponds to the Security Verification Toolkit’s test suite expansion—a compliance-critical investment that the AI agent produced concurrently with the scanning scripts.

More significant than the raw volume is the *ratio of compliance artifacts to implementation code*. Traditional compliance workflows produce documentation as a separate, sequential activity after implementation. In the AI-assisted workflow, compliance artifacts (requirements, verification matrices, decision memos, traceability mappings, attestation PDFs) are generated *concurrently* with implementation as a natural byproduct of the agent interaction. The 7 decision memoranda, 29 requirements, and 29 verification mappings in SendCUIEmail were pro-

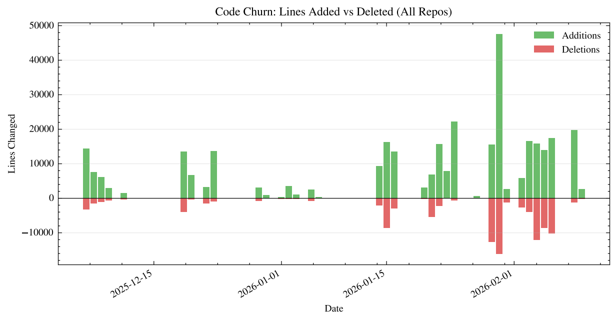

Figure 4: Code churn across all repositories. Green bars show lines added; red bars show lines deleted. Net-positive creation in every period, with sustained deletion indicating active refactoring.

duced in the same sessions that generated the implementation code, not in a separate documentation sprint.

8.3 The Review-Centric Workflow

The most significant shift introduced by AI-assisted compliance development is the transition from an *authoring* model to a *review* model. In traditional compliance work, an engineer reads the governing standard, interprets its requirements, drafts the compliance artifact, and submits it for review. With AI assistance, the engineer specifies the standard and reviews the AI-generated artifact for accuracy.

This shift has two implications. First, it is faster: reviewing a draft is consistently less effort than producing one from scratch. Second, it changes the *skill profile* required: the engineer must be a competent reviewer of compliance documents rather than a competent author. This is a meaningful distinction—many engineers who understand the technical standards struggle with the formal writing conventions of government documentation.

8.4 Concurrent Multi-Project Scalability

The review-centric workflow has a second-order implication: because the human's role shifts from author to reviewer, a single engineer can oversee multiple AI-assisted projects concurrently. During the development of this paper, the author maintained five active projects simultaneously—SendCUIEmail (CUI encryption), a decision documentation system, a Security Verification Toolkit, this white paper, and a Scrum-based agent orchestration system—each with its own AI agent sessions and compliance artifacts. These five projects span seven git repositories tracked in this paper's visualization data, with additional supporting repositories (agent templates, process framework, privacy tooling, cross-platform utilities, web applications, and hardware projects) bringing the total to 17 repositories containing 1,087 commits, 149 release tags, and over 227,000+ lines of code.

Critically, these projects are not merely concurrent; they *cross-pollinate*. Patterns discovered in one project feed into others: the Security Verification Toolkit's scanning infrastructure was applied to the white paper repository (Section 6); the SendCUIEmail project's agent conventions (`AGENTS.md`) informed the multi-agent architecture described in Section 7; the Scrum repo's team structure informed Section 7.4; and this paper documents the methodology used across all projects, creating a feedback loop that improves each project's compliance posture.

Figure 5 illustrates this concurrent development pattern: the cumulative commit timeline shows multiple repositories advancing simultaneously, with the Security Verification Toolkit exhibiting the steepest growth curve while other projects progress in parallel bursts.

This cross-project learning is facilitated by the `CLAUDE.md` convention: insights captured in one project's instructions propagate to others when the engineer applies the same patterns (semantic versioning, interaction logging, QA standards) across repositories. The human engineer serves as the integrator—reviewing agent output across projects, recognizing transferable patterns, and directing agents to apply lessons learned from one domain to another. This is a scalability model that would be impractical without AI assistance: the documentation and compliance overhead of five simultaneous government projects would overwhelm a single engineer working manually.

Moreover, the methodology itself is refined iteratively as the projects progress. The interaction logging requirement (Section 3.5) did not exist at project inception—it was added mid-session when the engineer recognized the need for audit traceability. Similarly, semantic versioning, the `build.sh` script, and the Scrum-based orchestration model were all incorporated as the engineer observed gaps during active development. This organic refinement is itself documented in the GitHub issue trail, creating a meta-level record of how the compliance process evolved. The ability to refine tooling and process *while simultaneously producing compliant artifacts* is a distinctive advantage of the AI-assisted approach: the agent can update its own instructions, rebuild its infrastructure, and continue producing deliverables without the context-switching penalty that a human author would incur.

The scope of this concurrent work extends beyond software compliance. The author's AI-assisted workflow originated with a CAD-based house construction project and a speech-processing application before evolving into the government compliance domain examined here. Across 17 active repositories—spanning systems engineering, hardware interfaces, music production, web development, and federal information security—the same patterns apply: AI agents draft artifacts, the human reviews and directs, and the process is documented through git and issue tracking. The methodology is not specific to government compliance; it is a general-purpose approach to engineering documentation that happens to map well to federal requirements.

A key enabler of cross-project learning is *agent instruction ingestion*: AI agents read instruction files (`CLAUDE.md`, `AGENTS.md`, `agents.json`) from other repositories, absorbing patterns, conventions, and compliance requirements established in sibling projects. This naturally led to the creation of a canonical agent templates repository (https://github.com/brucedombrowski/ai-agents) containing model-agnostic role definitions that any project can inherit. The templates separate the *what* (role responsibilities, standards, interaction protocols) from the *how* (vendor-specific model selection and tool configuration), allowing the same compliance agent patterns to be implemented across different AI platforms. This repository itself emerged organically from the white paper development process—an example of the methodology producing reusable infrastructure as a byproduct of compliance work.

8.5 Stakeholder Accessibility: Bridging the CLI-Browser Gap

The methodology described in this paper is CLI-first: the engineer works in AI coding agents, git, and shell scripts. This creates an adoption barrier when the goal is team-wide participation by non-technical stakeholders—program managers, team leads, auditors—who will not install command-line tools.

The key insight is that while the *engineering* happens in the CLI, the *outputs* are entirely browser-accessible. GitHub and GitLab web interfaces render commit history, file diffs (green lines for additions, red for deletions), issue threads, and merge request discussions without requiring any software installation. The stakeholder's workflow reduces to: open a URL, review the diff, leave a comment, click approve. This is a one-page desk instruction, not a training program.

This pattern was validated in practice: a team lead adopted GitLab for versioning periodic database exports to CSV, adding configuration management (NIST SP 800-53 CM-3) to previously untracked operational data. The team lead did not learn git—they learned to click "History" and read a diff. The CSV format is critical: unlike binary formats (Excel `.xlsx`, PDF), CSV files produce human-readable line-by-line diffs in the browser. For teams working with Excel files, a pre-commit hook that auto-converts `.xlsx` to `.csv` provides the same visibility without changing the user's workflow.

When whole-team review is required, the branch-and-merge-request workflow provides structured approval entirely within the browser.

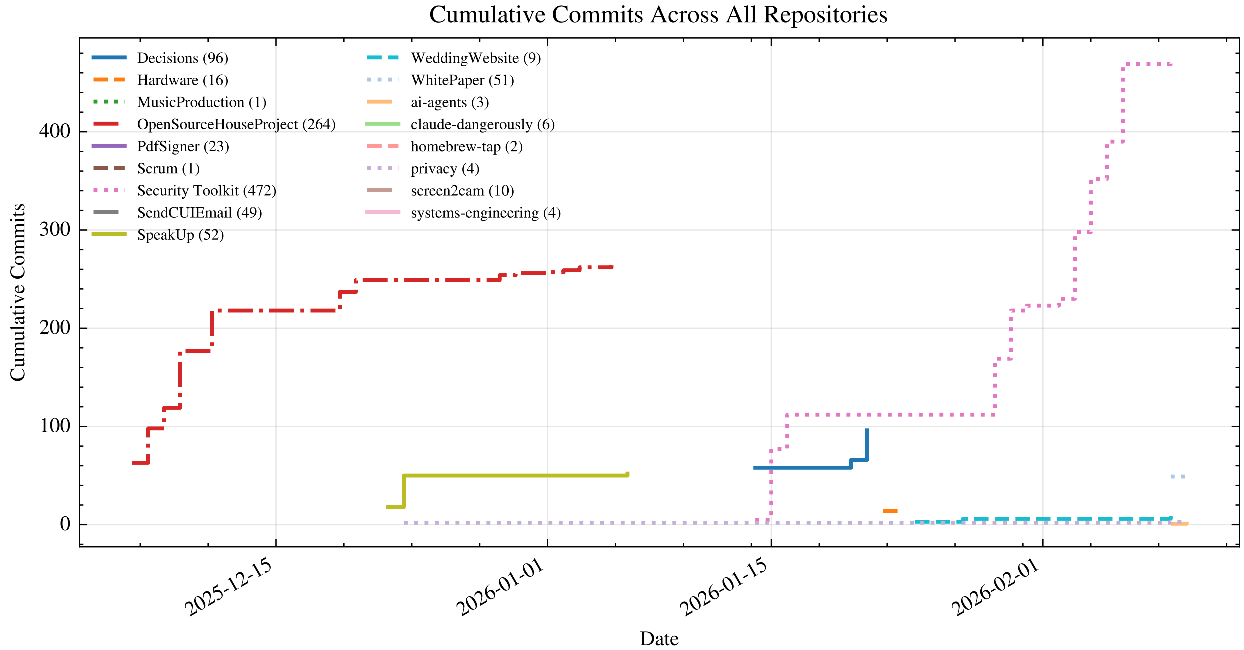

Figure 5: Cumulative commits across all 17 repositories over the 67-day development period. The Security Verification Toolkit dominates with 490 commits and exhibits the steepest growth curve. Multiple repositories advance concurrently, demonstrating the scalability of the AI-assisted review-centric workflow.

Branch protection rules enforce that (1) all changes to the main branch must go through a merge request, (2) required reviewers must approve before merge, and (3) the author cannot approve their own changes. These guardrails map directly to NIST SP 800-53 CM-3 (change control), AC-5 (separation of duties), and AU-3 (audit trail). Once configured, they are enforced automatically—the merge button is physically disabled until all conditions are met.

The generated visualizations (Section 8.6) serve a similar accessibility function. A chart showing over 1,087 commits across 17 repositories communicates project scope more effectively to a non-technical audience than any paragraph. The animated tree visualization (gource) showing file creation and modification over time has proven particularly effective for conveying the scale and structure of development activity to stakeholders unfamiliar with version control concepts.

8.6 Git Data Visualization

To support both the research objectives of this paper and the practical need to communicate project status to non-technical stakeholders, we developed a visualization pipeline that extracts data from git repositories and produces publication-quality charts.

The pipeline follows the sequence: `git log` (structured data extraction) → pandas (aggregation and analysis) → matplotlib with SciencePlots styling (IEEE-formatted charts) → matplot2tikz (PGFPlots export for native LaTeX inclusion). Each chart is generated in three formats: PNG (300 DPI, for presentations), PDF (vector, for print), and TikZ (`.tex`, for direct `\input{}` into LaTeX documents).

The toolkit comprises both custom analysis scripts and established open-source tools:

- **onefetch**: Repository summary cards showing languages, lines of code, commits, and version tags per repo.

- **git-of-theseus**: Code survival analysis using Kaplan-Meier methods—cohort stack plots showing how code ages over time, and extension/directory breakdowns showing how the codebase structure evolves.
- **gource**: Animated tree visualization rendering repository history as a growing organism, with files as nodes and contributors as actors.
- **Custom cross-repo analysis**: Cumulative commit timelines, daily activity by repository, code churn (additions vs. deletions), commit pattern analysis (hour of day, day of week), and ecosystem timeline (Gantt-style active development windows).

Applied to the ecosystem repositories (totaling 1,087 commits and 227,000+ lines of code across 17 repositories), the visualizations revealed several patterns. Figure 6 shows the ecosystem overview: the Security Verification Toolkit dominates with 490 commits, 77,238 lines of code, and 94 version tags. Figure 7 shows the active development windows—multiple repositories developed concurrently by a single engineer with AI agent support. Additional findings include: `scripts/` and `tests/` directories grew in lockstep (indicating disciplined test coverage); development activity concentrated on weekdays with near-zero weekend commits; and the code cohort analysis confirmed that all code is 2026-vintage—consistent with a rapidly growing project where code survival analysis is not yet meaningful but growth trajectories are clearly visible.

Figure 8 shows the temporal distribution of commits: peak activity occurs at 17:00 UTC (noon Eastern) and 03:00–04:00 UTC (late night), with near-zero weekend commits. This pattern—high weekday intensity with no weekend work—is characteristic of a sustainable AI-assisted development cadence where the engineer directs intensive sessions during focused work hours rather than spreading effort thinly across calendar time.

These visualizations serve dual purpose: they are research artifacts that quantify the development activity described in this paper, and they are communication tools that make the same data accessible to non-technical reviewers through browser-viewable charts and an animated video. A training slide deck (`git-workflow-training.pptx`) and a desk instruction (`DI-GIT-001`) were produced as companion artifacts to support organizational adoption.

8.7 Human-in-the-Loop Compliance

Government frameworks increasingly require evidence of human oversight in automated processes. Agentic AI tools with explicit permission models—where each file write, command execution, and code edit requires developer approval—provide natural evidence of human-in-the-loop oversight. Every action taken by the agent is logged and approved, creating an audit trail that maps to the "authorized use" requirements common in government security frameworks.

However, not all human oversight is equal. Two failure modes undermine the value of human-in-the-loop workflows:

1. **Auto-accept mode**: The human approves every agent suggestion without reviewing the content. This satisfies the letter of NIST SP 800-53 AC-5 (a human approved it) but not the spirit (a human exercised judgment).
2. **Rubber-stamp mode**: When no suggestion exists, the human provides only undirected prompts ("go," "continue") without evaluating what the agent should do next.

The distinguishing characteristic of *meaningful* oversight is the *corrective intervention*—a prompt where the human's domain expertise catches a gap that the agent's pattern-matching missed. During this paper's development, approximately 40% of human prompts were corrective interventions that changed outcomes the agent would have shipped incorrectly: identifying a missing audit trail, requiring built artifacts in releases, directing cross-repository template improvements, and reframing the paper's title to reflect its central argument. The remaining 60% were directional ("keep going," "this is top priority")—necessary for maintaining momentum

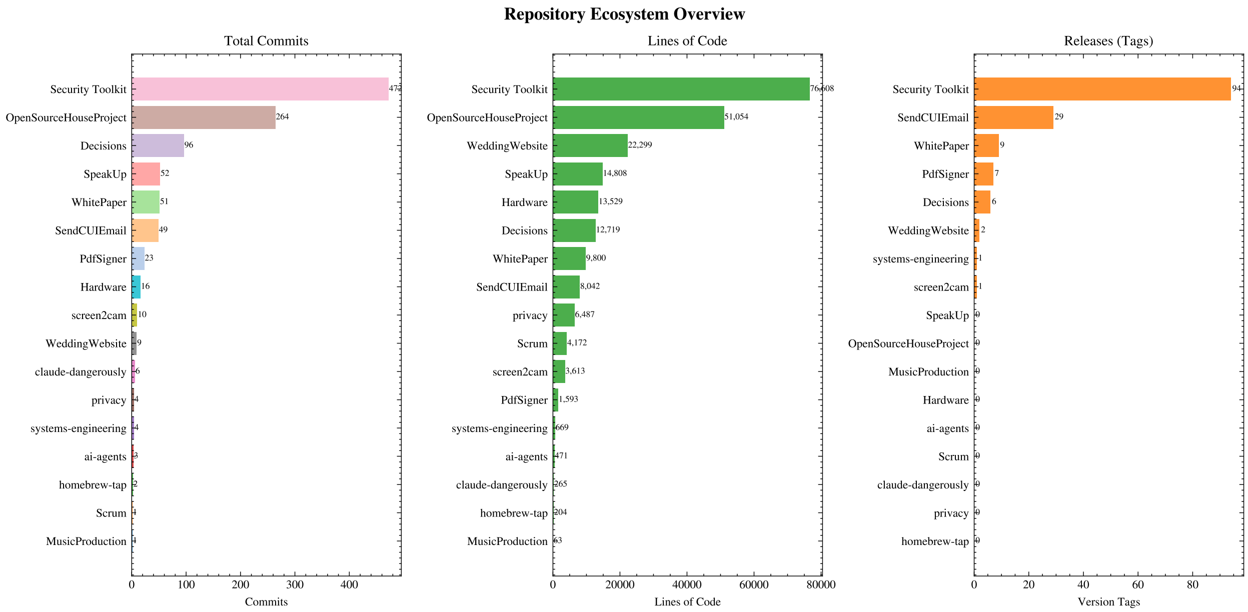

Figure 6: Repository ecosystem overview. Left: total commits per repository. Center: lines of code. Right: version tags (releases). The Security Verification Toolkit dominates all three metrics, reflecting its maturity as the most actively developed case study.

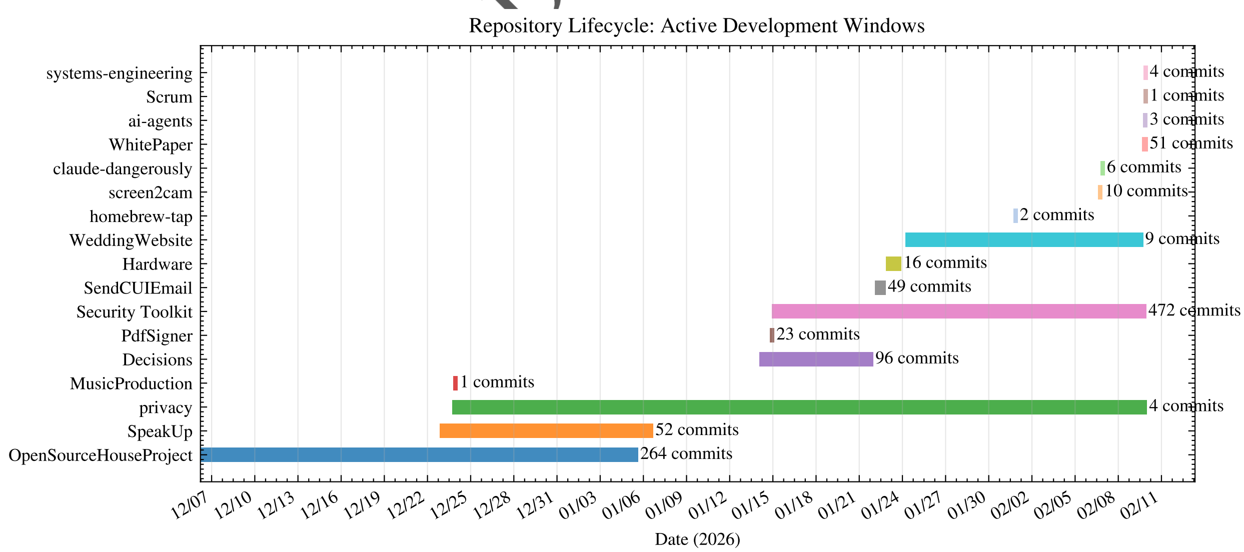

Figure 7: Active development windows for each repository. Bar length indicates the period between first and last commit; labels show total commit counts. Multiple repositories were developed concurrently by a single engineer using AI agent assistance.

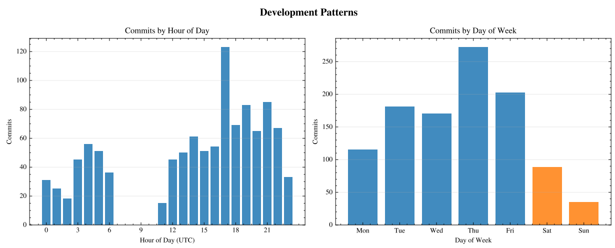

Figure 8: Development patterns across the ecosystem. Left: commits by hour of day (UTC). Right: commits by day of week (blue = weekday, orange = weekend). Peak activity at 17:00 UTC and 03:00–04:00 UTC; near-zero weekend commits.

but not corrective. The methodology's value is not that a human clicks approve; it is that the human's domain expertise catches gaps at a rate sufficient to maintain compliance integrity.

The `CLAUDE.md` convention further supports compliance by encoding organizational and project-specific constraints that persist across sessions. An organization's compliance officer could define `CLAUDE.md` templates that encode mandatory requirements, ensuring that all AI-assisted development within the organization operates within approved boundaries.

8.8 Standards-Based Review Process

The review agent itself operates according to established QA standards, making the review process auditable and reproducible. Table 5 maps each aspect of the review process to its governing standard.

This standards-based approach ensures that the review process itself can withstand audit scrutiny—a critical consideration for government programs where the QA methodology must be as defensible as the artifacts it evaluates. All review findings are documented as GitHub issues with structured severity, recommendation, and standard-violated fields, providing a traceable audit record per IEEE 1028.

8.9 Adoption Pathway: Open Source as Proving Ground

A practical barrier to adopting AI-assisted compliance workflows within government organizations is the combination of procurement delays, risk aversion to unproven tools, and classification constraints. The methodology presented in this paper was developed to address these barriers through a three-stage adoption pathway:

1. **Commercial tools, personal initiative**: An individual engineer uses commercially available AI coding agents—which require no procurement action—to develop methodology and tooling on personal time.

2. **Open-source development**: The engineer publishes the methodology, templates, and tooling as open-source software, validating the approach across multiple projects and producing a public track record of results.

3. **In-house adoption**: The proven methodology and artifacts are brought into the organization, where they can be applied to real projects with organizational data and classification constraints.

Critically, the open-source development stage forces the engineer to solve problems *generically*. Because the repositories are public, no CUI, PII, or proprietary information can appear in the code, templates, or documentation. This constraint—which might seem limiting—is in fact an advantage: the resulting artifacts are inherently safe to share, demonstrate, and review without security concerns. The open-source work serves as a "clean room" implementation that the organization's in-house work can inherit. The same templates that encrypt hypothetical files in SendCUIEmail can encrypt real CUI in production; the same scanning scripts that verify compliance on synthetic test data can verify it on organizational systems.

This pathway directly addresses the classification boundary limitation (Limitation 4 below): rather than requiring AI tools to operate within the classification boundary, the methodology is

Table 5: QA standards applied to the AI-assisted review process

Standard	Control	Application
IEEE 1028 [24]	Software Reviews	Review structure: severity classification (CRITICAL/MINOR), findings format, disposition
IEEE 29148 [25]	Requirements Engineering	Traceability verification: standard → requirement → implementation → test
NIST SP 800-53 [4]	AC-5: Separation of Duties	Review agent denied write/edit tools; auditors cannot modify what they audit
NIST SP 800-53 [4]	SA-11: Developer Testing	Claims verified against source files; assertions checked against implementation
ISO/IEC 25010 [26]	Software Quality	Documentation quality: completeness, accuracy, consistency checks
MIL-STD-498 [27]	A.5.19: Traceability	Cross-reference integrity between REQ, VER, DM, and source code

developed *outside* the boundary with synthetic data, then applied inside it with organizational tooling and data. The AI agent assists in creating the framework; the human applies it to the classified context. The 1,087 commits, 149 release tags, and 227,000+ lines of code across 17 repositories represent the open-source validation stage of this pipeline, a body of evidence an engineer can present to organizational leadership before requesting formal adoption.

8.9.1 Enterprise Deployment Options

The transition from Stage 1 (commercial tools, personal initiative) to Stage 3 (in-house adoption) is enabled by enterprise deployment options that major AI providers now offer. These options address the security and privacy requirements that prevent organizations from sending sensitive data to public cloud endpoints:

- **Government cloud regions**: AI models deployed within FedRAMP [28] High and DoD IL4/5 authorized environments (e.g., Claude in AWS GovCloud via Amazon Bedrock, Claude on Google Cloud Vertex AI). These deployments inherit the cloud provider's existing government authorization boundary.
- **Zero data retention (ZDR)**: Endpoints where no prompts, outputs, or metadata are persisted beyond real-time processing. Customer data is never used for model training.
- **Private networking**: Virtual Private Cloud (VPC) isolation and Private Service Connect routing ensure that AI API traffic never traverses the public internet.
- **Compliance certifications**: SOC 2 Type II audits, FedRAMP authorization, and DFARS 252.204-7012 [29] compliance for contractors handling CUI.

For a government contractor, the deployment path is concrete: the organization procures AI model access through an existing FedRAMP-authorized cloud provider (AWS GovCloud, Google Cloud), deploys within its authorized boundary, and applies the same methodology and templates developed during the open-source stage. The methodology is unchanged; only the data and the deployment environment change.

8.9.2 Inter-Organizational Data Sharing

The same FedRAMP-authorized environments that enable secure AI deployment also enable compliant data sharing between government agencies and their contractors. A defense contractor collaborating with a government program office can operate shared git repositories within a FedRAMP High environment, with both parties subject to the same security controls. DFARS 252.204-7012 [29] requires contractors to implement NIST SP 800-171 [3] controls when handling CUI; FedRAMP-authorized platforms [28] provide the infrastructure to meet these requirements. The git-based methodology described in this paper—where every change is attributed, timestamped, and cryptographically hashed—maps directly to the configuration management (CM-3) and audit logging (AU-3) controls that both parties must demonstrate. Shared repositories with branch protection rules (Section 8) enforce separation of duties across organizational boundaries, providing the same AC-5 compliance for inter-organizational workflows that the multi-agent architecture provides within a single project.

8.10 Limitations

Several limitations should be noted:

1. **Model knowledge currency**: LLM training data has a cutoff date, meaning recent revisions to standards (e.g., updates to NIST SP 800-171 Rev. 3, or the transition from FIPS 140-2 to FIPS 140-3 for new CMVP submissions since 2021) may not be reflected. Developers must verify that AI-cited standards are current.

2. **No formal verification**: AI-generated compliance claims are assertions, not proofs. They do not substitute for formal testing, independent audit, or certification processes such as CMVP validation.

3. **Organizational specificity**: Government compliance is highly context-dependent. The same standard may be interpreted differently across agencies, and AI agents lack organizational knowledge without explicit instruction.

4. **Classification boundaries**: AI tools operating in public cloud endpoints are unsuitable for classified work. FedRAMP High and DoD IL4/5 authorized deployments (e.g., AWS GovCloud, Google Cloud with Private Service Connect) extend the methodology to CUI and controlled environments, but truly classified (Secret/Top Secret) work remains outside scope. The open-source development approach described above mitigates this by developing methodology outside the classification boundary.

5. **Case study maturity**: All three case study projects are under active development and have not reached production release status (SendCUIEmail v0.17.3, Decisions v0.4, Security Toolkit v2.7.3). The compliance artifacts and methodology demonstrated here reflect a development-phase workflow; production deployment would require additional validation, independent testing, and formal authority-to-operate processes.

8.11 Reproducibility and Process Documentation

This paper itself was produced using the methodology it describes. The white paper repository maintains a two-tier documentation structure: `PROCESS.md` provides a human-readable executive summary of each development session, while GitHub issues serve as the authoritative, machine-queryable record of all human-agent interactions.

As of this writing, the repository contains 51 GitHub issues spanning 14 development sessions, with each issue labeled according to the scheme described in Section 3.5. The git history contains 55 semantically versioned commits across 9 release tags (v0.1.0 through v0.9.0), each corresponding to a distinct compliance-relevant action. The repository also contains a visualization toolkit that generates 10 publication-quality charts from git data across the ecosystem, of which 6 are included as figures in this paper (Figures 6–4). Together, these records provide sufficient information for an independent team to reproduce the development process or for an auditor to verify that every artifact has a documented provenance chain.

This dual-track approach—git for configuration management, GitHub issues for interaction traceability—mirrors the separation between configuration management (NIST SP 800-53 CM-3) and audit logging (NIST SP 800-53 AU-3) that government frameworks prescribe. The combination ensures that the process is documented at both the artifact level (what changed) and the decision level (why it changed).

A concrete illustration occurred during this paper's own development: the AI agent completed a full session of work—scanning 16 repositories, updating quantitative metrics, creating release workflows, and feeding improvements back to upstream template repositories—without filing any GitHub issues documenting the interaction. The human reviewer identified the omission in two words: "all this tracked in github issues?" The resulting corrective action produced four issues across two repositories (the audit trail itself), a process finding filed per IEEE 1028 (issue #43), and an upstream template fix ensuring future agents treat issue creation as mandatory rather than deferrable (ai-agents issue #13). This episode demonstrates the value of human-in-the-loop oversight: the agent produced correct artifacts but violated the process that makes those artifacts auditable. The human caught the gap, and the fix was federated across the ecosystem within minutes. The methodology is self-correcting precisely because every deviation becomes a traceable finding.

9 Future Work

Several directions merit further investigation:

1. **Automated compliance testing**: Integrating AI agents with continuous integration pipelines to validate compliance assertions against code changes.
2. **Standard-specific agents**: Training or fine-tuning agents on specific government standards (e.g., a NIST SP 800-171 specialist agent) to improve requirement extraction accuracy.
3. **Cross-reference validation**: Building tools that automatically verify citations between compliance artifacts (requirements ↔ verification ↔ code).
4. **FedRAMP and CMMC application**: Extending the methodology to broader compliance frameworks such as FedRAMP [28] authorization packages and CMMC assessments, leveraging the enterprise deployment options described in Section 8.
5. **Comparative studies**: Quantitative comparison of AI-assisted vs. manual compliance documentation effort across multiple projects and team sizes.
6. **Cross-platform validation**: While the methodology is designed to be platform-agnostic, our case studies use a single AI agent implementation. Replicating the workflow with alternative agentic tools would empirically validate portability and identify which properties are tool-dependent vs. methodology-dependent.

10 Conclusion

This paper has demonstrated a five-phase methodology for combining git version control and AI coding agents to address the challenge of building software that meets government compliance requirements. Through three case studies, we showed that AI agents can produce structurally sound compliance artifacts including requirements specifications, decision memoranda, verification documents, and automated compliance attestations, while the interactive approval model provides the human oversight that government frameworks require.

The key insight is not that AI replaces compliance expertise, but that it *restructures* the compliance workflow. The engineer's role shifts from author to reviewer, the documentation burden decreases without sacrificing rigor, and the multi-agent architecture enables scalable compliance workflows for projects of varying complexity. Critically, the version control and interaction traceability layer—git for configuration management, GitHub issues for human-agent interaction logging—provides the audit evidence that government frameworks demand, mapping directly to NIST SP 800-53 controls CM-3 and AU-3.

This paper itself demonstrates the methodology: it was produced across multiple AI agent sessions, with every human directive and agent action logged as GitHub issues, every change captured in semantically versioned git commits, and the entire process reproducible from the public repository. The fact that an AI agent can assist in producing both the compliance artifacts *and* the auditable process documentation for those artifacts suggests a path toward significantly reducing the overhead of government compliance work.

This work is not a proof of concept. The methodology, agent configurations, and process artifacts presented here are in active use across 17 repositories spanning government compliance, systems engineering, security tooling, and CAD—real projects with real deliverables. The approach produces measurable outcomes: 1,087 commits, 227,000+ lines of code, 149 release tags, and over 130 compliance artifacts across 17 repositories in 67 calendar days—produced by a single engineer with AI agent support. This demonstrates more consistent documentation (fewer gaps, stronger traceability), faster delivery (the review-centric workflow eliminates the authoring bottleneck), and reduced personnel requirements (one engineer with AI agents sustains the documentation overhead that traditionally demands a dedicated compliance team). The engineering experience itself improves when the tedious parts of compliance work are handled by agents and the human focuses on judgment, direction, and review. As government agencies and contractors face increasing pressure to demonstrate compliance across expanding regulatory frameworks, the methodology presented here offers a practical, field-tested foundation for getting real work done.

Acknowledgments

This paper and its supporting artifacts were developed using the methodology it describes, with Claude Code (Anthropic, model: Claude Opus) as the AI agent implementation. All source materials, including the LaTeX source, agent configurations, git history, and process documentation, are available at https://github.com/brucedombrowski/WhitePaper.

References

[1] National Institute of Standards and Technology, "FIPS 197: Advanced Encryption Standard (AES)," tech. rep., NIST, Nov. 2001. Updated May 2023.

[2] Information Security Oversight Office, "32 CFR Part 2002: Controlled Unclassified Information," 2016. Federal Register, Vol. 81, No. 183.

[3] R. Ross, V. Pillitteri, K. Dempsey, M. Riddle, and G. Guissanie, "NIST SP 800-171: Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations," Special Publication 800-171 Rev. 2, National Institute of Standards and Technology, Feb. 2020.

[4] J. T. Force, "NIST SP 800-53: Security and Privacy Controls for Information Systems and Organizations," Special Publication 800-53 Rev. 5, National Institute of Standards and Technology, Sept. 2020.

[5] M. S. Turan, E. Barker, W. Burr, and L. Chen, "NIST SP 800-132: Recommendation for Password-Based Key Derivation," Special Publication 800-132, National Institute of Standards and Technology, Dec. 2010.

[6] National Institute of Standards and Technology, "FIPS 140-2: Security Requirements for Cryptographic Modules," tech. rep., NIST, May 2001.

[7] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey

and open problems," *arXiv preprint arXiv:2310.03533*, 2023.

[8] L. Radosky and I. Polasek, "Large language models in software documentation and modeling: A literature review and findings," in *IEEE 24th World Symposium on Applied Machine Intelligence and Informatics (SAMI 2026)*, 2026. arXiv:2602.04938.

[9] R. Huang, A. Reyna, S. Lerner, H. Xia, and B. Hempel, "Professional software developers don't vibe, they control: AI agent use for coding in 2025," *arXiv preprint arXiv:2512.14012*, 2025.

[10] K. K. Ng, L. Fauzi, L. Leow, and J. Ng, "Harnessing the potential of gen-ai coding assistants in public sector software development," *arXiv preprint arXiv:2409.17434*, 2024.

[11] B. Marino, R. Hunter, Z. Jamali, M. E. Kalpakos, M. Kashyap, I. Hinton, A. Hanson, M. Nazir, C. Schnabl, F. Steffek, H. Wen, and N. D. Lane, "AIReg-Bench: Benchmarking language models that assess AI regulation compliance," *arXiv preprint arXiv:2510.01474*, 2025.

[12] J. Y. Khan and G. Uddin, "Automatic generation of api documentation," in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, pp. 497–501, 2022.

[13] Anthropic, "Claude Code: Anthropic's AI-Powered Command-Line Development Tool." Software, 2025.

[14] S. Bradner, "RFC 2119: Key Words for Use in RFCs to Indicate Requirement Levels." Internet Engineering Task Force, Mar. 1997.

[15] B. Dombrowski, "SendCUIEmail: CUI Email Encryption Tool." GitHub repository, 2026.

[16] M. Dworkin, "NIST SP 800-38A: Recommendation for Block Cipher Modes of Operation," Special Publication 800-38A, National Institute of Standards and Technology, Dec. 2001.

[17] E. Barker and J. Kelsey, "NIST SP 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators," Special Publication 800-90A Rev. 1, National Institute of Standards and Technology, June 2015.

[18] P. A. Grassi, J. L. Fenton, E. M. Newton, R. A. Perlner, A. R. Regenscheid, W. E. Burr, and J. P. Richer, "NIST SP 800-63B: Digital Identity Guidelines — Authentication and Lifecycle Management," Special Publication 800-63B, National Institute of Standards and Technology, June 2017.

[19] B. Dombrowski, "Security Verification Toolkit: Automated NIST SP 800-53 Control Verification." GitHub repository, 2026.

[20] R. Kissel, A. Regenscheid, M. Scholl, and K. Stine, "NIST SP 800-88: Guidelines for Media Sanitization," Special Publication 800-88 Rev. 1, National Institute of Standards and Technology, Dec. 2014.

[21] Cybersecurity and Infrastructure Security Agency, "BOD 22-01: Reducing the Significant Risk of Known Exploited Vulnerabilities." Binding Operational Directive, Nov. 2021.

[22] National Institute of Standards and Technology, "FIPS 199: Standards for Security Categorization of Federal Information and Information Systems," tech. rep., NIST, Feb. 2004.

[23] K. Schwaber and J. Sutherland, "The Scrum Guide." Scrum.org, Nov. 2020. The definitive guide to Scrum: The rules of the game.

[24] IEEE Computer Society, "IEEE 1028: Standard for Software Reviews and Audits," IEEE Standard 1028-2008, Institute of Electrical and Electronics Engineers, 2008.

[25] ISO/IEC/IEEE, "ISO/IEC/IEEE 29148: Systems and Software Engineering — Life Cycle Processes — Requirements Engineering," International Standard 29148:2018, IEEE, 2018.

[26] ISO/IEC, "ISO/IEC 25010: Systems and Software Quality Requirements and Evaluation (SQuaRE)," International Standard 25010:2011, International Organization for Standardization, 2011.

[27] Department of Defense, "MIL-STD-498: Software Development and Documentation," Military Standard MIL-STD-498, U.S. Department of Defense, Dec. 1994.

[28] General Services Administration, "FedRAMP: Federal Risk and Authorization Management Program." U.S. Government Program, 2024.

[29] Department of Defense, "DFARS 252.204-7012: Safeguarding Covered Defense Information and Cyber Incident Reporting." Defense Federal Acquisition Regulation Supplement, 2016.