AI-Assisted Development for Government Compliance: Using Claude Code to Meet Federal Information Security Requirements

Bruce Dombrowski
Independent Researcher
GitHub: brucedombrowski

February 9, 2026

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 examines the application of Claude Code, Anthropic's AI-powered command-line development tool, to government compliance software projects. Drawing on three real-world case studies—a CUI email encryption tool (Send-CUIEmail), a formal decision documentation system, and a Security Verification Toolkit implementing automated NIST SP 800-53 control verification—we demonstrate how AI-assisted development can accelerate compliance artifact generation while maintaining the precision required by federal auditors. We present a five-phase methodology for structuring AI agent workflows around government documentation standards—from requirements capture through version-controlled interaction traceability using git and GitHub issues—evaluate the quality of AI-generated compliance artifacts against manual baselines, and discuss the implications for federal software development practices. Our findings suggest that AI-assisted tooling can reduce compliance documentation effort by shifting the engineer's role from author to reviewer, while the interactive agent model provides the human-in-the-loop oversight that government frameworks require.

Keywords: AI-assisted development, government compliance, NIST, FIPS, CUI, controlled unclassified information, Claude Code, large language models, software engineering, federal information security

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.

The emergence of AI-powered development

tools offers a potential path forward. Large language models (LLMs) trained on technical and regulatory corpora can draft compliance documents, suggest standard references, and generate structured artifacts. However, government work demands accuracy: an incorrect citation to a NIST Special Publication or a mischaracterized FIPS requirement could undermine an entire compliance package.

Claude Code, Anthropic's command-line interface for the Claude family of models, provides an interactive development environment where the AI agent operates directly within the developer's file system and terminal. Unlike web-based chat interfaces, Claude Code can read source files, execute build commands, search codebases, and write artifacts—all under explicit developer approval. This architecture maps naturally to the human-in-the-loop oversight model that government compliance frameworks expect.

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: Send-CUIEmail (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 discussion of the `-agents` mode workflow for multi-agent collaboration on compliance projects.

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]. Code generation tools such as GitHub Copilot, Amazon CodeWhisperer, and Anthropic's Claude have demonstrated capability in producing syntactically correct code across multiple

languages. However, the application of LLMs to *compliance-oriented* development—where correctness encompasses not just functional behavior but regulatory adherence—remains underexplored.

Prior work on AI-assisted documentation generation has focused primarily on API documentation [8] 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.

2.3 Claude Code Architecture

Claude Code operates as a command-line agent with access to the developer's local environment. Key architectural properties relevant to 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. **CLAUDE.md conventions**: Projects can include instruction files that persist agent context across sessions, encoding project-specific compliance requirements.

5. **Multi-agent mode**: The `-agents` flag enables orchestrated workflows where specialized agents handle distinct aspects of a project.

6. **Session continuity**: The `-resume` and `-continue` flags allow sessions to persist across interruptions, preserving the accumulated compliance context that would otherwise need to be reconstructed.

Table 1 documents the CLI switches most relevant to compliance workflows. These are recorded in the project's `CLAUDE.md` to ensure reproducible invocation across sessions and team members.

Table 1: Recommended Claude Code CLI switches for compliance projects

Switch	Purpose
`-agents`	Load multi-agent config (JSON)
`-model`	Select model (opus for review, sonnet for implementation)
`-allowedTools`	Restrict tools per agent role
`-continue`	Resume most recent session
`-verbose`	Log tool calls for audit trail

3 Methodology

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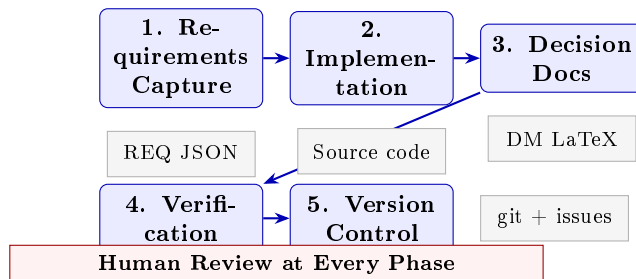

Figure 1: Five-phase methodology for AI-assisted compliance development. Human review occurs at every phase transition.

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 Claude Code 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 Send-CUIEmail requirements document, generated with Claude Code assistance and reviewed by the developer.

```
1  {
2    "id": "REQ -1.1",
3    "standard": "FIPS 197",
4    "section": "Section 1",
5    "text": "The tool SHALL use the
          Advanced
6      Encryption Standard (AES) algorithm
7      for all file encryption operations
          .",
8    "priority": "mandatory",
9    "verification": "inspection"
10 }
```

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

Requirement text uses RFC 2119 [9] 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 Claude Code agent operates within the project's `CLAUDE.md` context, which encodes the compliance standards and architectural constraints. The `AGENTS.md` file (used in the SendCUIEmail project) provides persistent instructions that survive across sessions, as shown in Listing 2:

```
1  ## Compliance Standards
2
3  - **FIPS 140-2**: AES -256-CBC encryption
4  - **NIST SP 800-132**: PBKDF2 -HMAC-
      SHA256
      key derivation (100 ,000 iterations)
5  - **NIST SP 800-171**: CUI handling
6  - **32 CFR Part 2002**: CUI marking
7
```

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:

```
1  \newcommand{\UniqueID}{DM -2026-002}
2  \newcommand{\DocumentDate}{January 19,
      2026}
3  \newcommand{\AuthorName}{PDF Tools
      Working Group}
4  \newcommand{\SubjectField}{Font
      Selection for
      CUI Header Text}
5  \newcommand{\dmContent}{...}
6
7  \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 final phase produces verification 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 Send-CUIEmail 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 is a PowerShell-based tool for encrypting files before email transmission, 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** [10]: Block cipher modes of operation

5. **NIST SP 800-90A** [11]: Random number generation

6. **32 CFR Part 2002** [2]: CUI marking and handling

4.2 AI-Assisted Artifacts

Over the course of development, Claude Code 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 [12])

- **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:

```
1  $f=Read-Host "File"
2  $p=Read-Host "Password"
3  $d=[IO.File]::ReadAllBytes($f)
4  $k=[Rfc2898DeriveBytes]::new(
5      $p,$d[0..15],100000,"SHA256")
6  $a=[Aes]::Create()
7  $a.Key=$k.GetBytes(32)
```

```
8  $a.IV=$d[16..31]
9  $c=$a.CreateDecryptor()
10     .TransformFinalBlock($d,32,$d.Length
          -32)
11 [IO.File]::WriteAllBytes(
12     ($f-replace'\.Locked$',''),$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 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 [4], a pure-Bash security scanning and compliance documentation system that automates the verification of federal security controls. 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]: RA-5 (Vulnerability Scanning), CM-6 (Configuration Settings), CM-8 (Component Inventory), SA-11 (Developer Testing), SI-2/3/4/5/12 (Information Integrity), SC-8 (Transmission Protection), MP-6 (Media Sanitization), CA-2 (Assessment)

- **NIST SP 800-171** [3]: 11 corresponding CUI protection requirements

- **NIST SP 800-88** [13]: Media sanitization (secure deletion)

- **BOD 22-01** [14]: CISA Known Exploited Vulnerabilities cross-referencing

- **FIPS 199** [15]: Security categorization of federal information

6.2 Requirements Traceability

The toolkit maintains a complete traceability chain in JSON format:

Requirement → NIST Control → Script → Test → Evidence

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 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 2.7 major versions and over 140 GitHub issues.

7 Multi-Agent Workflow

Claude Code's `-agents` mode enables orchestrated workflows where multiple specialized agents collaborate on a project. For government compliance work, we propose the role-based agent architecture shown in Figure 2.
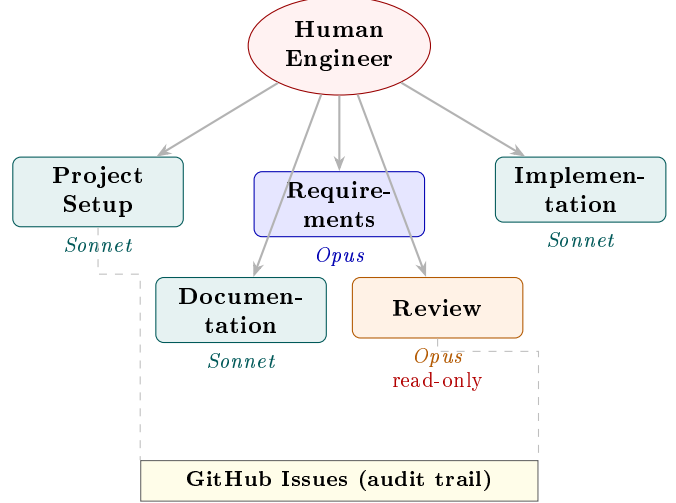

Figure 2: Multi-agent architecture for compliance projects. Teal agents use Sonnet; blue/orange agents use Opus. The review agent has read-only access (NIST SP 800-53 AC-5). All agents log interactions to GitHub issues.

7.1 Agent Roles

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

9

summarizes the five-agent configuration developed for this paper.

Model selection reflects the cognitive demands of each role: the `requirements` and `review` agents use Opus for its stronger reasoning over regulatory interpretation and cross-reference validation, while `implementation` and `documentation` use Sonnet for its favorable speed-to-quality ratio on structured, template-following tasks. 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.

8 Discussion

8.1 Quality of AI-Generated Compliance Artifacts

Our experience indicates that Claude Code produces 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 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.

8.3 Human-in-the-Loop Compliance

Government frameworks increasingly require evidence of human oversight in automated processes. Claude Code's permission model—where each file write, command execution, and

Table 3: Agent configuration summary (agents.json)

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

code edit requires explicit developer approval—provides 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.

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.

8.4 Standards-Based Review Process

The review agent itself operates according to established QA standards, making the review process auditable and reproducible. Table 4 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.

8.5 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 cloud-connected modes are unsuitable for classified work. The methodology presented here applies only to unclassified and CUI environments.

Table 4: QA standards applied to the AI-assisted review process

Standard	Control	Application
IEEE 1028 [16]	Software Reviews	Review structure: severity classification (CRITICAL/MINOR), findings format, disposition
IEEE 29148 [17]	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 [18]	Software Quality	Documentation quality: completeness, accuracy, consistency checks
MIL-STD-498 [19]	A.5.19: Traceability	Cross-reference integrity between REQ, VER, DM, and source code

8.6 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 14 GitHub issues spanning 6 development sessions, with each issue labeled according to the scheme described in Section 3.5. The git history contains 9 semantically versioned commits (v0.1.0 through v0.5.0, with ongoing work toward v0.6.0), each corresponding to a distinct compliance-relevant action. 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).

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 authorization packages and CMMC assessments.

5. **Comparative studies**: Quantitative comparison of AI-assisted vs. manual compliance documentation effort across multiple projects and team sizes.

10   Conclusion

This paper has demonstrated a five-phase methodology for applying AI-assisted development tools—specifically Claude Code—to 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 Claude Code 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.

As government agencies and contractors face increasing pressure to demonstrate compliance across expanding regulatory frameworks, AI-assisted tooling offers a practical path to maintaining documentation quality without proportional increases in engineering effort. The methodology, agent configurations, and process documentation presented here provide a foundation for teams seeking to adopt this approach.

Acknowledgments

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] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

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

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

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

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

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