Software Requirements Specification (SRS)

AutoDoc — CI/CD Code Analyzer & Confluence Updater

Course: CS4398

Section: 001

Group: SM Group 6

Prepared by: Logan Lay, Noah Masoud, and Ryan Mitchell

Date: 09/23/2025

Table of Contents

1. Introduction

1.1 Purpose

This SRS specifies requirements for AutoDoc, a tool that runs in a CI/CD pipeline to analyze code changes and automatically update associated Confluence documentation. It serves as the formal communication between the development team and course staff prior to implementation.

1.2 Product Scope

- Python backend integrates into CI/CD to ingest diffs and metadata from runs.
- Static analysis of Python and TypeScript code to detect public/API changes and likely breaking changes.
- Rule-based mapping of code areas to Confluence pages.
- Template-driven generation of content patches.
- Angular/TypeScript UI for configuration, preview/approval, and audit/rollback.

1.3 Intended Audience and Reading Suggestions

- Developers: Sections 3–5, 7–9.
- Testers: Sections 4–6, 10.
- Tech Writers/Reviewers: Sections 3–5, 7, 11B.
- Instructor/TAs: Sections 1–2 (overview), 4–6 (traceability), 10–11 (plan & artifacts).

1.4 Document Conventions

- Dotted numbering for hierarchical structure (e.g., 4.2.3).
- “Shall” indicates a binding requirement; “Should” indicates a recommendation.
- Priorities: H (High), M (Medium), L (Low).

1.5 Definitions, Acronyms, and Abbreviations

- CI/CD — Continuous Integration / Continuous Delivery.
- MR/PR — Merge/Pull Request.
- AutoDoc Rule — Mapping from code selector(s) to Confluence target and template.
- Patch — Proposed Confluence content delta for a page update.
- Storage Format — Confluence’s XML-based Storage representation.

1.6 References

- IEEE/ISO SRS guidance (course materials).
- Atlassian Confluence REST API documentation.
- GitHub Actions / GitLab CI documentation.
- Python 3.11+, Angular 17+ documentation.

2. Overall Description

2.1 Product Perspective

AutoDoc is invoked by CI as a job/step. The Python service reads the workspace and diffs, produces a structured change report (JSON), and—based on rules—creates Confluence page patches. The Angular/TS UI surfaces runs, patches, approvals, and rollback.

2.2 Product Functions

1. Retrieve and parse changed files/diffs and PR metadata.
2. Analyze Python/TypeScript ASTs for added/modified/removed public symbols and signature changes.
3. Generate update patches from templates and rule mappings.
4. Gate updates behind an approval workflow (configurable).
5. Apply updates to Confluence with versioning and audit logging.
6. Provide run history, error details, and rollback.

2.3 User Classes and Characteristics

- Developer (Primary): Sets rules, reviews patches, approves/rolls back.
- Tech Writer (Secondary): Authors templates and reviews changes.
- Instructor/TA (Stakeholder): Evaluates functionality and test artifacts.
- Admin (Secondary): Manages tokens and space configuration.

2.4 Operating Environment

- Backend: Python 3.11+, Linux container/runner (CI).
- Frontend: TypeScript.
- CI Providers: GitHub Actions and GitLab CI (minimum).
- Confluence Cloud (primary; DC optional).

2.5 Design and Implementation Constraints

- CI job time budget (target ≤5 minutes typical).
- Confluence API rate limits and content format constraints.
- Secrets stored via CI secret manager; never committed.

2.6 Assumptions and Dependencies

- Valid Confluence API token with page edit scope.
- Network access to Confluence from runners.
- Majority of code in Python and TypeScript initially.

3. External Interface Requirements

3.1 User Interfaces (Angular/TS)

- Landing/Dashboard: Last N runs with status, filters.
- Connections: Confluence base URL, space key, API token (masked), test connection.
- Rules: CRUD for mapping rules (selector, target page, template, auto-approve).
- Templates: CRUD with preview; variables per template.
- Run Details: Diff summary, analyzer warnings, generated patches.
- Patch Preview: Side-by-side old/new; Approve/Reject with comment.
- Audit: History of applied patches; rollback actions.

3.2 Software Interfaces

- CI Adapters: Shell entrypoint and Docker image for Actions/GitLab.
- SCM (optional): Post PR/MR comment summary if token provided.
- Confluence REST: Search, get/update/create page, get history/version.

3.3 Communications Interfaces

HTTPS to Confluence and optional SCM; RESTful JSON between UI and backend.

4. System Features

4.1 CI Job & Change Report

Description & Priority: AutoDoc runs as a CI job, reads diffs, and emits a structured JSON change report. Priority: H

Preconditions: CI job has workspace with changed files and commit SHA.

Postconditions: Change report artifact stored; run recorded.

Stimulus/Response:

7. CI invokes entrypoint with --commit, --repo, optional PR ID.
8. Backend gathers changed files and unified diffs.
9. Backend saves change_report.json as an artifact and exposes run ID.

- FR-1 (H): The system shall accept commit, branch, repo, and PR/MR ID as inputs.
- FR-2 (H): The system shall parse changed files and diffs.
- FR-3 (H): The system shall emit a machine-readable change report artifact (JSON).
- FR-4 (M): The system should optionally post a PR/MR summary comment.

4.2 Static Analysis & Change Detection

Description & Priority: Analyze Python/TS to detect public API changes and potential breaking changes. Priority: H

Preconditions: Change report produced.

Postconditions: Analyzer findings appended to run.

Stimulus/Response:

10. Analyzer builds ASTs for changed Python/TS files.
11. Public symbols and signatures are extracted before/after.
12. Breaking changes flagged (e.g., parameter removed).

- FR-5 (H): Support Python/TS symbol extraction (functions, classes, exports).
- FR-6 (H): Detect added/removed/modified symbols; flag signature changes.
- FR-7 (M): Extract docstrings/JSDoc and @doc annotations.
- FR-8 (M): Detect common HTTP endpoints (FastAPI/Flask; Express/Nest).

4.3 Confluence Mapping & Patch Generation

Description & Priority: Map code to pages using rules; generate patches via templates. Priority: H

Preconditions: Analyzer findings available; rules exist.

Postconditions: Patch objects created per target page.

Stimulus/Response:

13. Mapper matches file paths/modules to rule targets.
14. Template engine renders content with placeholders.
15. Patch objects created (before/after; diff preview).

- FR-9 (H): Map code selectors to Confluence page IDs/locators.
- FR-10 (H): Generate patches using templates (Markdown or Storage Format).
- FR-11 (H): Keep an audit record per patch.

4.4 Review & Approval Workflow

Description & Priority: Review patches in UI; approve/reject; optional auto-approve. Priority: H

Preconditions: Patches created; user authenticated.

Postconditions: Approved patches applied or queued; rejections recorded.

Stimulus/Response:

16. Reviewer opens Patch Preview.
17. Reviewer approves or rejects with comment.
18. Approved patch triggers Confluence update.

- FR-15 (H): UI shall provide diff preview (before/after).
- FR-16 (H): UI shall allow Approve/Reject with comment.
- FR-18 (M): Auto-approve should be supported per rule.

4.5 Page Update, Versioning & Rollback

Description & Priority: Apply updates with versioning; allow rollback using Confluence history. Priority: M

Preconditions: Patch approved; token valid.

Postconditions: Page version incremented or rollback performed.

Stimulus/Response:

19. Backend fetches latest page version.
20. Backend applies patch; handles edit conflicts.
21. On rollback, backend restores a previous version.

- FR-11 (H): Apply updates via Confluence REST preserving versioning.
- FR-25 (M): Provide rollback to a selected prior version.

4.6 Configuration, Templates & Test Mode

- FR-19 (H): CRUD for rules with: name, selector (glob/regex), space key, page ID, template ID, auto-approve flag.
- FR-20 (H): CRUD for templates with documented placeholders.
- FR-14 (M): Dry-run mode should produce patches without updating Confluence.
- FR-22 (M): Test a rule/template against a specific commit.

4.7 Observability & Notifications

- FR-23 (H): Structured logs with run correlation IDs.
- FR-24 (H): UI displays run/patch status and error details.
- FR-4 (M): Optional PR/MR comment on success/failure.

4.8 Security & Access Control

- FR-26 (H): Secrets stored only in CI secret manager/env; never in artifacts.
- FR-27 (H): UI requires local auth for demo.
- FR-28 (M): Tokens masked in logs/UI.
- FR-29 (M): Read-only viewer role for history.

5. Use Cases

UC-1: CI Run Generates Proposed Updates

Field	Value
Brief Description	CI invokes AutoDoc; patches are generated for corresponding pages.
Preconditions	Valid token; rules configured; code changes present.
Postconditions	Patches attached to run and ready for review.
Events Flow	1) CI calls AutoDoc with --commit and repo metadata. 2) Analyzer produces change report and findings. 3) Mapper creates patches per rules. 4) Run recorded with status 'Awaiting Review'.
Alternate Flows/Exceptions	A1: Confluence unreachable → run marked Failed; logs include retry details. A2: No matching rules → run marked Completed (no patches).

UC-2: Reviewer Approves a Patch

Field	Value
Preconditions	Patches exist; reviewer authenticated.
Postconditions	Page updated; version incremented; audit entry recorded.
Events Flow	1) Open Patch Preview and inspect diff. 2) Click Approve; backend applies patch. 3) UI updates run status to 'Success'.
Alternate	Edit conflict → system fetches latest and re-bases; if still conflicting, mark 'Manual Action Required'.

UC-3: Configure a New Rule

Field	Value
Preconditions	Confluence connection configured.
Postconditions	Rule saved; test preview passes.
Events Flow	1) Create rule with selector src/api/** and template api_ref_v1. 2) Use Test Mode for commit abc123. 3) Save rule; enable auto-approve if desired.

UC-4: Dry Run from CI

Field	Value
Preconditions	--dry-run flag passed.
Postconditions	No Confluence updates; patches stored as artifacts.

UC-5: Rollback a Page

Field	Value
Preconditions	Page updated by AutoDoc; prior versions exist.
Postconditions	Page content restored to selected version; audit logged.

UC-6: Token Misconfiguration

Field	Value
Preconditions	Invalid/expired token.
Postconditions	Run fails with actionable error; secrets never logged.
Alternate Flow	Retry after secret update → rerun succeeds.

6. Non-Functional Requirements

6.1 Performance

- NFR-1: Analyzer runtime shall be ≤ 5 minutes for a change set ≤ 500 LOC on a repo ≤ 10k LOC.
- NFR-2: Confluence update round-trip should be ≤ 10 seconds per page (network permitting).

6.2 Reliability & Robustness

- NFR-3: Network/API errors shall be retried with exponential backoff (3 attempts).
- NFR-4: Runs shall fail fast when tokens are invalid and provide remediation hints.

6.3 Usability & Accessibility

- NFR-5: All critical actions (approve/reject/rollback) shall be ≤ 3 clicks from Run Details.
- NFR-6: UI should meet basic keyboard navigation and focus management.

6.4 Portability & Compatibility

- NFR-7: Provide Docker image for CI; POSIX shell entrypoint.
- NFR-8: Work with GitHub Actions and GitLab CI (minimum).

6.5 Security & Privacy

- NFR-9: Secrets shall be masked in logs/UI and never persisted to disk unencrypted.

- NFR-10: Output shall be sanitized to prevent script injection in Confluence content.

6.6 Maintainability & Supportability

- NFR-11: ≥ 70% unit test coverage for analyzers/connectors; linting (ruff/eslint) and type checking (mypy/tsc).
- NFR-12: Configuration documented with examples; one-command local dev setup.

7. Data Requirements

7.1 Data Entities

- Run: {id, repo, branch, commit_sha, started_at, completed_at, status}
- Change: {run_id, file_path, symbol, change_type, signature_before, signature_after}
- Rule: {id, name, selector, space_key, page_id, template_id, auto_approve}
- Template: {id, name, format, body, variables[]}
- Patch: {id, run_id, page_id, diff_before, diff_after, approved_by, applied_at}
- Credential (meta only): {id, label, created_at, last_used_at} (token value not stored)

7.2 Data Retention & Archival

Keep last 100 runs in local DB; export JSON artifacts per run; rely on Confluence version history for content restore.

8. Design/Implementation Constraints & Standards

- Dockerized backend; SQLite acceptable for capstone.
- Follow IEEE-style SRS organization as in course exemplars.

9. Hardware/Software Requirements

Developer: Linux/macOS/Windows; Python 3.11+; Node 20+; 8GB RAM; Docker.

User (Reviewer): Modern browser (Chrome/Firefox/Edge).

CI Runner: Linux x86_64 with Docker.