

reqtool User Guide & Technical Reference Manual

Git-Backed Requirements Management System & OpenAPI 3.1 REST API

Document Version:	0.4.2
Publication Date:	October 2026
Compliance Standards:	21 CFR Part 11, ISO 26262, OMG ReqIF 1.2, OpenAPI 3.1
Target Audience:	Systems Engineers, Lead Developers, QA & Compliance Managers

Executive Overview

reqtool is an enterprise-grade requirements management system built on top of standard Git workflows and YAML 1.2 files. It provides full cryptographic traceability, audit versioning, AI duplicate detection, automated event-driven workflow rules, and compliance exporters for ISO 26262 and 21 CFR Part 11 regulatory audits.

1. Overview & Quickstart

reqtool

Git-backed YAML requirements management tool for embedded and IoT product teams.

Requirements, principles, and TBDs live as individual YAML files in a git repository. The tool handles content hashing, semantic versioning, full validation, module reuse, and export. A FastAPI backend serves a React UI and a fully-typed REST API documented as OpenAPI 3.1.

Quick start

```
# Install (Python 3.11+)
pip install reqtool # from PyPI (when published)
# or from source:
pip install -e . # pre-built UI is included -- no npm step required

# Initialise a new repository -- choose a flavour:
mkdir my-reqs && cd my-reqs
git init

req init # blank repo (prompts for ID and title)
req init agile # agile types: theme/initiative/epic/feature/story/task/bug/spike
req init systems # systems types, formal/safety-formal workflow
req init defaults # agile + systems types, formal workflows, and pre-populated principles
req init demo # full IoT sensor demo dataset

# Start the server (opens http://localhost:8765 with full UI)
req serve
```

Note: The React UI (`src/reqtool/ui/dist/`) is pre-built and committed to

the repository. `pip install` is the only step needed. Rebuild the UI only if

you modify files under `src/reqtool/ui/src/`:

```
```bash
```

```
cd src/reqtool/ui && npm install && npm run build
```

```
```
```

Repository layout

After `req init` your requirements repository will look like this:

```

my-reqs/
  ■■■■ .reqtool/
  ■ ■■■■ config.yaml # Repo config (git settings, export, validation)
  ■ ■■■■ enums.yaml # Project-specific enum extensions and NFR key registry
  ■ ■■■■ hierarchy.yaml # Type system, workflow definitions, DoR/DoD templates
  ■ ■■■■ help.yaml # Custom help content (optional; falls back to defaults)
  ■ ■■■■ templates.yaml # Requirement creation templates
  ■ ■■■■ custom_fields.yaml # Custom field schema
  ■ ■■■■ filter_presets.yaml # Saved sidebar filter combinations
  ■■■■ requirements/
  ■ ■■■■ <uid>.yaml # One file per requirement
  ■■■■ principles/
  ■ ■■■■ <uid>.yaml
  ■■■■ tbds/
  ■ ■■■■ <uid>.yaml
  ■■■■ modules/
  ■ ■■■■ <module-id>/
  ■ ■■■■ _module.yaml # Module manifest
  ■ ■■■■ <uid>.yaml # Module requirements
  ■■■■ products/
  ■ ■■■■ <product-id>/
  ■ ■■■■ _product.yaml # Product manifest (module refs, variants, root ordering)
  ■ ■■■■ _product.lock # Lock file written by `req import`
  ■■■■ comments/ # Threaded comments (outside content hash)
  ■■■■ attachments/ # File attachments
  ■■■■ exports/ # Generated output (gitignore-able)

```

Item types

reqtool supports two complementary vocabularies that can coexist in the same repository.

Agile planning hierarchy

| Type | Prefix | Workflow | Description |
|------------|--------|-------------|--|
| theme | TH | lightweight | Highest strategic grouping; multi-year outcome |
| initiative | IN | lightweight | Large body of work within a theme |
| epic | EP | sprint | Significant user-facing capability |
| feature | FT | sprint | Discrete piece of functionality |
| story | US | sprint | Unit of user value |
| task | TK | sprint | Concrete unit of technical work |
| bug | BG | sprint | A defect |
| spike | SP | sprint | Time-boxed investigation |

Hierarchy: theme → initiative → epic → feature → story / task / bug / spike

Systems engineering types

| Type | Prefix | Workflow | Description |
|------------------|--------|---------------|---|
| stakeholder_need | SN | lightweight | Top-level systems intent |
| functional | FR | formal | Capability the system shall provide |
| performance | PR | formal | Quantified speed/throughput/capacity constraint |
| security | SR | formal | Protection against threats |
| safety | SAF | safety_formal | Harm prevention constraint |
| interface | IR | formal | System boundary definition |
| compliance | CR | formal | Legal/standard/contractual obligation |
| constraint | CON | formal | Design or implementation restriction |
| operational | OR | formal | Operation/maintenance constraint |
| physical | PHR | formal | Physical/environmental constraint |

Cross-layer: agile items link to systems requirements via explicit relationship types (satisfies, implements, traces_to).

Workflows

Four workflow types are defined in `defaults/hierarchy.yaml`. Override any per-project in `.reqtool/hierarchy.yaml`.

| Workflow | Used by | States |
|---------------|--|---|
| lightweight | theme, initiative, stakeholder_need | draft → active → done / cancelled |
| sprint | epic, feature, story, task, bug, spike | backlog → ready → in_progress → in_review → done |
| formal | functional, performance, interface, etc. | draft → in_review → reviewed → approved → deprecated |
| safety_formal | safety | Same as formal, with a transition guard on approved requiring ACs and safety_classification |

Workflow guards

Guards are configurable pre-conditions enforced at transition time. If a guard fails, the transition returns 409 `workflow_guard`. They are defined per target state in `hierarchy.yaml`:

```
workflows:
  safety_formal:
    guards:
      approved:
        requires_fields: [content.description, content.rationale, safety_classification]
```

```
requires_ac: true
message: "Safety requirements must have description, rationale, ACs, and
safety_classification."
```

Use POST `/requirements/{uid}/transition` for explicit transitions, or PUT `/requirements/{uid}` (which also enforces guards when status changes).

Fields

Core fields

| Field | Type | Description |
|---------------|--------|---|
| uid | string | Immutable UUID7. Used in relationships and API calls. |
| id | string | Human-readable identifier (e.g. US-042). Auto-assigned from type prefix if omitted. |
| title | string | One-line summary. |
| status | string | Current workflow state. Initial value set from workflow initial. |
| priority | string | critical / high / medium / low |
| req_type | string | Item type (see table above). |
| owner | string | Accountable person (from <code>enums.yaml</code> owner list). |
| assignee | string | Person doing the work. Distinct from owner. |
| iteration | string | Named iteration reference (sprint name, PI, milestone). |
| estimate | float | Numeric effort estimate. Unit configured by <code>estimate_unit</code> in <code>enums.yaml</code> . |
| dor_checklist | list | Definition of Ready; <code>{item, checked}</code> pairs. Populated from <code>hierarchy.yaml</code> defaults. |
| dod_checklist | list | Definition of Done; same structure. |

All long-form text fields (`content.description`, `content.rationale`, `content.extended_description`, AC text, DoR/DoD labels, comment text) support Markdown, rendered server-side via POST `/render/markdown`.

Estimate rollout

GET `/requirements/{uid}/rollup` returns the sum of all direct children's estimate values. The result is read-only and computed on demand. Unit is the repo-wide `estimate_unit` from `enums.yaml`.

Relationships

The primary structural relationship is parent/child via `parentId`. Additional relationships are explicit typed links in the `relationships` array.

Built-in relationship types

| Type | Direction | Description |
|----------------|-----------------|---|
| relates_to | non-directional | Generic association |
| depends_on | → | This item cannot proceed until the target is complete |
| depended_by | ← | Computed inverse of <code>depends_on</code> ; read-only |
| satisfies | → | Agile item satisfies a systems requirement |
| implements | → | Agile item implements a requirement |
| traces_to | → | General traceability link |
| derived_from | → | This item was derived from the target |
| conflicts_with | non-directional | This item is in tension with the target |
| supersedes | → | This item replaces the target |

Add or remove types in `enums.yaml` under `relationship_types`.

Cross-layer rules in `hierarchy.yaml` define which relationship types are valid between which source and target types. Violations produce a validation **warning**, not a hard error.

Change impact analysis

`GET /requirements/{uid}/impact` returns the transitive downstream closure: all items linked via `derived_from`, `depends_on`, `implements`, or `satisfies`, grouped by type, plus the direct upstream chain.

Definition of Ready / Definition of Done

DoR and DoD checklists are `{item, checked}` lists stored on each requirement. Templates are defined per type in `hierarchy.yaml` and applied automatically on creation.

```
dor:
story:
- item: Acceptance criteria defined
- item: Estimate provided
- item: Dependencies identified
```

Check items off via `PUT /requirements/{uid}` with the updated `dor_checklist` / `dod_checklist`.

Kanban board

GET /kanban returns a column-per-state board using the workflow states for the filtered type. Cards include: uid, id, title, assignee, estimate, priority, req_type.

Query parameters: req_type, iteration, owner, assignee.

Real-time updates

Connect to GET /ws (WebSocket) for a live event stream. The server broadcasts:

- created -- new artefact written
- updated -- artefact changed (includes changed_fields)
- deleted -- artefact soft-deleted
- status_changed -- status field changed
- comment_added, attachment_added

Event payload: {event, uid, id, type, payload}.

Outbound webhooks (configured in .reqtool/config.yaml) remain for external integrations.

Filter presets

Save named filter combinations for the sidebar:

```
PUT /filter-presets/{id} {"label": "Sprint 3", "filters": {"iteration": "Sprint-3", "req_type": "story"}}
GET /filter-presets list all presets
DELETE /filter-presets/{id} delete a preset
```

Presets are stored in .reqtool/filter_presets.yaml.

Help system

GET /help returns contextual help content from defaults/help.yaml (or .reqtool/help.yaml if present). Use ?topic= for a specific section: concepts, fields, workflow, relationships, estimates, dor_dod, validation, git_baselines, export.

CI badge

Add a validation badge to your README:

```
![reqtool validated](http://localhost:8765/badge/validated)
```

GET /badge/validated returns a Shields.io-compatible SVG. Green = passing, yellow = warnings, red = errors. Add ?nocache=1 to force re-evaluation.

CLI reference

```
req init Blank repo (prompts for ID and title)
req init agile Agile types, sprint workflow, DoR/DoD defaults
req init systems Systems types, formal/safety-formal workflow
req init defaults Blank repo with engineering principles pre-populated
req init demo Full IoT sensor demo dataset

req serve Start API + UI at http://localhost:8765
req validate Full validation suite; exit 1 on errors
req verify Verify all content hashes; exit 1 on mismatch

req new requirement Create a requirement interactively
req new principle Create a principle interactively
req new tbd Create a TBD interactively

req new-agile story Create a story (auto-increments US-NNN)
req new-agile task Create a task (TK-NNN)
req new-agile bug Create a bug (BG-NNN)
req new-agile spike Create a spike (SP-NNN)
req new-agile epic Create an epic (EP-NNN)
req new-agile feature Create a feature (FT-NNN)
req new-agile theme Create a theme (TH-NNN)
req new-agile initiative Create an initiative (IN-NNN)

req export jsx Export JSX viewer component
req export markdown Export Markdown files
req export csv Export CSV

req import <module-id> Import a module into a product
req baseline create <name> Create a baseline tag
req baseline list List all baseline tags
req diff [--since <ref>] Show changes since a ref
req stats Print metrics summary
req show <id-or-uid> Print an artefact to stdout
req history <id-or-uid> Print git log for an artefact
req openapi [-o openapi.yaml] Export OpenAPI specification
```

API overview

All endpoints are documented at <http://localhost:8765/docs> (Swagger UI) or <http://localhost:8765/redoc>.

Requirements

```
GET /requirements List (filter by domain, status, type, owner, tag)
POST /requirements Create
GET /requirements/{uid} Get
PUT /requirements/{uid} Update (enforces workflow guards on status change)
DELETE /requirements/{uid} Soft-delete
POST /requirements/{uid}/transition Explicit workflow transition with guard enforcement
GET /requirements/{uid}/impact Transitive downstream impact closure
GET /requirements/{uid}/rollup Computed estimate rollup for children
GET /requirements/{uid}/history In-file + git history
GET /requirements/{uid}/relationships Outgoing + incoming relationship links
POST /requirements/bulk Bulk field update across multiple UIDs
POST /requirements/{uid}/clone Clone a requirement
POST /requirements/{uid}/approve Approve and set approval block
POST /requirements/{uid}/commit Stage + git commit a single requirement
POST /requirements/{uid}/ac Add acceptance criterion
PUT /requirements/{uid}/ac/{ac_uid} Update acceptance criterion
DELETE /requirements/{uid}/ac/{ac_uid} Delete acceptance criterion
```

Hierarchy

```
GET /hierarchy Full hierarchy config
GET /hierarchy/types Configured item types
GET /hierarchy/children/{req_type} Allowed child types
```

Kanban

```
GET /kanban Column-per-state board (filter: req_type, iteration, owner, assignee)
```

Help

```
GET /help Full help content
GET /help?topic=<key> Specific help section
```

Filter presets

```
GET /filter-presets List saved presets
PUT /filter-presets/{id} Save or update a preset
DELETE /filter-presets/{id} Delete a preset
```

Other endpoints

```
GET /requirements/tree Nested tree for a product
GET /products/{id}/tree Product tree
GET /kanban Kanban board
```

```

GET /ws WebSocket event stream
GET /badge/validated Shields.io SVG badge
GET /validate Full validation result
GET /validate/{uid} Single artefact validation
GET /metrics Repository health metrics
GET /workflow Workflow config
GET /hierarchy Full hierarchy config
GET /templates Requirement templates
GET /enums Merged enums
GET /git/status Git working tree status
POST /git/commit Batch commit
GET /baselines List baselines
POST /baselines Create baseline tag
GET /baselines/{name}/diff Diff against baseline
GET /diff Requirement diff since ref
POST /render/markdown Render Markdown to HTML
GET /health Health check
GET /version Tool + schema version

```

Modules

Requirements modules are self-contained sets of requirements stored outside product definitions in a top-level `modules/` directory. Multiple products can reference the same module.

```

modules/
■■■ comms-baseline/
■■■ _module.yaml # Manifest: title, version, overrideable fields
■■■ <uid>.yaml # Module requirements

```

Reference a module from a product manifest:

```

modules:
- id: comms-baseline
version: 1.2.0
overrides:
<uid>:
priority: critical

```

Lock a module with `req import` (writes `_product.lock`). Verify integrity with `GET /modules/{id}/verify`.

Validation

Run offline with `req validate` (exits 1 on errors) or via `GET /validate`.

Common warnings and how to resolve them:

| Code | Resolution |

|-----|-----|

```
missing_owner	Set owner to a person from enums.yaml
missing_ac	Add at least one acceptance criterion
missing_rationale	Fill in content.rationale
hash_mismatch	File edited outside reqtool; run req verify
unknown_relationship_target	Target UID does not exist
cross_layer_violation	Relationship type not permitted between these types
dor_incomplete	DoR items not checked before transition to In Progress
dod_incomplete	DoD items not checked before transition to Done
```

Configuration

`.reqtool/config.yaml` controls git settings, export paths, validation flags, and the fallback workflow config (used when no `hierarchy.yaml` is present).

`.reqtool/enums.yaml` extends the core enums and configures:

- `estimate_unit` -- repo-wide unit label for estimates (points, days, hours)
- `relationship_types` -- additional relationship type IDs beyond the built-in set
- `id_domain` -- maps ID prefix → type for suggestion

Development

```
pip install -e ".[test]"
pytest tests/test_store.py tests/test_validation.py tests/test_cli.py # ~20s
pytest tests/test_features_v03.py # ~30s
pytest tests/test_api.py -k "not watchdog" # ~60s
pytest tests/test_regression.py # ~30s
```

The watchdog-based incremental reload tests are slow (they wait for filesystem events). Exclude with `-k "not watchdog"` during development.

Packaging & Release Pipeline

Installation Options

- **PyPI / Wheel Installation:**

```
`bash
```

```
pip install reqtool
```

- **Local Wheel Distribution:**

```
`bash
pip install dist/reqtool-0.3.20-py3-none-any.whl
```

- **Development Installation:**

```
`bash
git clone https://github.com/markbac/reqtool.git
cd reqtool
pip install -e ".[test]"
```

Automated Release Pipeline (`.github/workflows/release.yml`)

The repository includes an automated GitHub Actions release pipeline triggered on git tag creation (`v*.*.*`):

1. Builds Python source distribution (`.tar.gz`) and wheel (`.whl`).
2. Verifies `reqtool` CLI execution.
3. Automatically creates GitHub Releases with release notes.

Automated GitHub Copilot Code Review

PR code reviews are configured using repository-level guidelines in `[.github/copilot-instructions.md](.github/copilot-instructions.md)` and an automated PR Quality Gate workflow in `[.github/workflows/copilot-code-review.yml](.github/workflows/copilot-code-review.yml)`:

- **Architectural Rules:** Enforces FastAPI APIRouter scoping under `src/reqtool/routers/` and specialized stores under `src/reqtool/stores/`.
- **Prose Linter Checks:** Evaluates requirement descriptions against INCOSE quality standards (passive voice, vague words, escape clauses).
- **Safety & Schema Verification:** Validates ISO 26262 ASIL and DO-178C DAL classifications.

Documentation Index & Sitemap

For detailed guides, literature-backed requirements engineering philosophy, and architecture specifications:

- **■ [User Guide & Operations Manual](docs/USER_GUIDE.md):** Complete operational guide covering CLI commands, REST endpoints, React UI workstation, workflow transitions, and module overlays.

- ■ [Requirements Philosophy & Best Practices](docs/REQUIREMENTS_PHILOSOPHY.md): Grounding in Robertson & Robertson (Volere), INCOSE standards, ISO 29148, and how reqtool enforces good requirements management.
- ■■ [Architecture & Technical Specification](ARCHITECTURE.md): Comprehensive component design, data models, store locking, API Routers, WebSocket CRDTs, and export engines.
- ■ [Contributing Guidelines](CONTRIBUTING.md): Developer environment setup, testing guidelines, and submission workflow.

2. Comprehensive User Guide

`reqtool` Complete User Guide & Operations Manual

This guide provides end-to-end instructions for installing, configuring, and operating reqtool via the Command Line Interface (CLI), REST API, and React Web Workstation.

1. Installation & Environment Setup

1.1 Requirements

- Python **3.11** or higher.
- Git **2.30+**.
- Optional: `vale` CLI (for external prose style linting).

1.2 Installation Commands

Standard Package Installation:

```
pip install reqtool
```

Local Wheel Installation:

```
pip install dist/reqtool-0.3.20-py3-none-any.whl
```

Editable Development Installation:

```
git clone https://github.com/markbac/reqtool.git
cd reqtool
pip install -e ".[test]"
```

Verify the installation:

```
reqtool --help
```

2. Command Line Interface (CLI) Reference

reqtool (or alias req) provides comprehensive sub-commands:

`reqtool init` — Initialise a Repository

Initialises a `.reqtool/` directory and creates configuration files:

```
reqtool init # Blank repository
reqtool init agile # Agile planning hierarchy (Theme/Initiative/Epic/Feature/Story)
reqtool init systems # Systems engineering hierarchy (Stakeholder Need/Functional/Performance)
reqtool init defaults # Combined Agile + Systems types with pre-populated principles
reqtool init demo # Full IoT sensor demo dataset
```

`reqtool serve` — Start API & Web Workstation

Launches the FastAPI backend and serves the embedded React UI workstation:

```
reqtool serve # Starts on http://localhost:8765
reqtool serve --port 8000 # Custom port
reqtool serve --host 0.0.0.0 # Bind all network interfaces
```

`reqtool validate` — Run Validation & Prose Linter

Runs full schema, enum, hierarchy, safety, and INCOSE prose quality validation:

```
reqtool validate # Pretty console output (exits 1 on errors)
reqtool validate --json # Machine-readable JSON output
reqtool validate --strict # Treat warnings as blocking errors
```

`reqtool export` — Export Artefacts

Exports requirements to various standard formats:

```
reqtool export --format csv # Export to CSV
reqtool export --format reqif # Export to standard ReqIF XML
reqtool export --format markdown # Export to Markdown document
reqtool export --format jsx # Export to React JSX component
```

`reqtool diff` — Inspect Git Changes

Shows requirement modifications since a specified Git commit or tag:

```
reqtool diff HEAD~1 # Changes in last commit
reqtool diff v1.0.0 # Changes since tag v1.0.0
```

`reqtool new-agile` — Create Agile Items

Interactively creates new agile planning items:

```
reqtool new-agile story "User Login"
reqtool new-agile epic "Authentication Subsystem"
```

— — —

3. React Web Workstation User Guide

Launching `reqtool serve` opens the interactive web workstation in your browser (<http://localhost:8765>).

[illegible]

3.1 Multi-Level Tree Explorer View

- Located in the left sidebar.
- Displays nested requirement hierarchies (Themes → Initiatives → Epics → Features → Stories).
- **Drag-and-Drop Re-parenting:** Click and drag any node to re-parent it under a new parent requirement.

3.2 High-Density Bulk Grid View (Spreadsheet Mode)

- Access via **Tools** → **Bulk Editing Grid**.
- Enables rapid inline editing of titles, statuses, priorities, owners, and disciplines across hundreds of requirements simultaneously.
- Batch save changes with one click.

3.3 Interactive D3/Cytoscape Network Graph

- Access via **Traceability** → **Graph View**.
- Renders an interactive node graph visualizing bidirectional relationships between requirements, principles, and verification items.
- Zoom, pan, and click nodes to open detailed inspectors.

3.4 Side-by-Side Version Diff Inspector & Command Palette (Ctrl+K)

- **Command Palette:** Press Ctrl+K (or Cmd+K) anywhere in the application to instantly search requirements, trigger action shortcuts, or switch products.
- **Diff Inspector:** Compare historical git versions side-by-side with color-coded diff highlighting.

3.5 Release Roadmap & Kanban Board

- Access via **Kanban / Roadmap**.
- View requirements organized by workflow states (Draft, In Review, Approved, Done) or target release milestones.
- Drag-and-drop cards between columns to transition workflow states.

4. Workflows, State Transitions & Checklists

Requirements follow configurable workflow state transitions:

```
stateDiagram-v2
[*] --> draft
draft --> in_review: Submit for Review (DoR Check)
in_review --> approved: Approve (AC & Rationale Required)
in_review --> draft: Request Changes
approved --> in_progress: Start Work
in_progress --> done: Complete Work (DoD Check)
done --> [*]
```

- **Definition of Ready (DoR):** Enforces mandatory checklist items (e.g. rationale present, acceptance criteria written) before transitioning to `in_review`.
- **Definition of Done (DoD):** Enforces verification completion checklist items before transitioning to `done`.

5. Modules, Variant Overlays & Parameter Substitution

5.1 Reusable Modules

Store shared requirements in `modules//_module.yaml`. Import external git repository modules using:

```
reqtool import <module-id>
```

5.2 Parameter Substitution & Overlays

Product manifests (`products//_product.yaml`) can parameterize shared module requirements using template variables:

```
id: iot-sensor-v2
title: "IoT Sensor Variant 2"
variables:
  max_voltage: "12V"
  sampling_rate: "100Hz"
modules:
```

```
- id: power-management-module
overrides:
REQ-010:
priority: critical
```

Requirements containing `{{max_voltage}}` will automatically substitute "12V" during tree building and API export.

3. Product Datasheet & Technical Specifications

`reqtool` -- Product Overview & Technical Specification

Version 0.4.0 | Enterprise Systems Engineering & Agile Requirements Management Engine

1. Executive Summary

`reqtool` is a high-performance, git-backed requirements engineering workstation and multi-user server designed for complex engineering programmes (aerospace, automotive, medical devices, defense, robotics, and software).

Combining formal systems engineering rigor (INCOSE quality linter, ISO 26262 / DO-178C compliance workflows, ReqIF 1.2 exchange) with modern agile planning (Kanban boards, epics, stories, tasks, bugs), `reqtool` stores every requirement, principle, and TBD as an individual, human-readable YAML file inside a Git repository.

2. Exhaustive Feature Matrix & Technical Capabilities

■ 2.1 Git-Backed Data & Storage Engine

- **File-per-Item Store:** Every requirement (`requirements/*.yaml`), principle (`principles/*.yaml`), TBD (`tbd/*.yaml`), module (`modules/`), and product (`products/`) is stored as an individual human-readable YAML 1.2 file.
- **SHA-256 Hashing & Semantic Versioning:** Automatic content hashing and semantic versioning (`1.0.0` → `1.1.0` on content modification; `2.0.0` on major structural changes).
- **Incremental File System Watchdog:** Background Watchdog file system observer reloads modified files in `<10ms` without restarting the server.
- **Native Git Branching & Diffs:** Execute `req diff HEAD~5` or `GET /diff?since=ref` to inspect field-level delta histories across commits, branches, and baseline tags.
- **In-Tool Repository Switcher & Initializer:** Switch active working repositories or initialize new repositories directly inside the UI without restarting.

■ 2.2 Formal Systems Engineering & Agile Planning

- **Unified Artefact Hierarchy:** Supports both formal systems engineering types (*stakeholder_need*, *system_requirement*, *functional*, *performance*, *interface*, *safety*, *compliance*) and agile items (*initiative*, *epic*, *feature*, *user_story*, *task*, *bug*).
- **Safety Criticality Tagging:** ASIL (A/B/C/D) and DAL (A/B/C/D/E) safety classification tagging.
- **Verification Method Allocation:** Allocate verification methods (*Test*, *Demonstration*, *Analysis*, *Inspection*, *Simulation*, *Similarity*).
- **Kanban Board Workstation:** Visual drag-and-drop workflow status transitions with customizable approval guards.
- **Traceability & Dependency Matrix:** Bi-directional parent-child hierarchy, verification links, and impact analysis graph (GET /traceability/matrix).

■ 2.3 INCOSE Linter & AI Intelligence

- **INCOSE Prose Quality Linter:** Real-time checking against INCOSE rules (flags passive voice, ambiguous terms like "suitable", "flexible", missing imperatives, unquantified NFR values).
- **AI Semantic Duplicate & Overlap Detector:** GET /analysis/detect-duplicates scans all requirements and calculates word token similarity scores to flag overlapping requirements.
- **Spec-to-Test Case Auto Generator:** POST /analysis/generate-tests/{uid} automatically generates positive test procedures and acceptance criteria test steps.
- **Acceptance Criteria Generator:** Gherkin Given-When-Then scenario generator (POST /analysis/generate-ac).

■ 2.4 Multi-User Server, Authentication & RBAC

- **Multi-Client Server Host** (reqtool-server.exe): Multi-client repository hosting listening on 0.0.0.0:8765.
- **Remote Client Desktop App** (reqtool-client.exe): Desktop application workstation equipped with ■ Server Login & Connection UI and dynamic server URL switching (http://remote-server:8765).
- **User Authentication:** Password hashing (PBKDF2-HMAC-SHA256) and signed JWT access tokens (/auth/login, /auth/me, /auth/users).
- **Default Admin Seeding:** Auto-seeds default admin credentials (admin / admin123) in .reqtool/users.yaml.
- **Role-Based Access Control (RBAC):** Permission enforcement across user roles (*admin*, *editor*, *reviewer*, *viewer*).

■ 2.5 Cryptographic Signatures & Audit Package Exporter

- **Cryptographic Electronic Signatures:** HMAC-SHA256 / Ed25519 digital signature generation on requirement approvals (SIG-v1-xxxxxxxx...).
- **21 CFR Part 11 / ISO 26262 Compliance Exporter:** GET /export/audit-package packages audit manifests, cryptographic signature ledgers, requirement dumps, and verification logs into a compressed .zip dossier.

■ 2.6 Visual Architecture & Diagram Canvas

- **Dual-Engine Diagram Canvas:** Client-side Mermaid.js and Server-side PlantUML (POST /render/plantuml via plantuml.jar).
- **Pre-Loaded Templates:** System Hierarchy, Sequence Flow, Component Diagram, Traceability Graph.
- **One-Click Snippet Copy:** Export formatted Markdown code blocks directly into requirement documentation.

■ 2.7 Flexible Iteration, Sprint & Gate Management

- **Methodology Modes:** Flexible support for **Scrum** (Sprint Backlog & Velocity), **Waterfall** (Phase & Gate Review), **Wagile** (Hybrid Sprint + Gate), and **Kanban** (Continuous Flow).
- **Milestone Gate Tracking:** Track planned, active, completed, and closed status, story point capacity, start/end dates, and milestone CDR gate readiness.
- **Iteration Metrics** (GET /iterations/{id}/metrics): Burndown charts and completion percentage.

■ 2.8 Ecosystem & Webhook Integrations

- **GitHub Webhook Ingestion** (POST /webhooks/github): Automatically parses commit messages ("Fixes REQ-SYS-001") and links commits/PRs to requirement verification logs.
- **Outbound Webhooks:** Configurable HTTP webhooks with HMAC-SHA256 signatures triggered on requirement changes and approvals.
- **ReqIF 1.2 XML Exchange:** OMG ReqIF 1.2 XML import and export (GET /export/reqif) for IBM DOORS, Jama, and Polarion.
- **OpenAPI 3.1 Spec:** Auto-generated REST API specification (/openapi.json).

3. Standalone Executable Matrix (5 Binary Targets)

| Executable | Output Binary | Port / Binding | Primary Usage |

| :--- | :--- | :--- | :--- |

| **Multi-User Server** | reqtool-server.exe | 0.0.0.0:8765 | Central server hosting repository for team networks. |

| **Remote Client App** | reqtool-client.exe | Client GUI | Desktop client with remote server login & connection modal. |

| **Web Workstation** | reqtool-web.exe | 127.0.0.1:8765 | Workstation auto-launching web browser interface. |

| **Desktop Workstation** | reqtool-app.exe | Client GUI | Workstation auto-launching native desktop app window. |

| **CLI Tool** | reqtool.exe | CLI | Command-line interface for scripts and CI/CD pipelines. |

4. Technical Specifications

| Parameter | Specification |

| :--- | :--- |

| **Storage Architecture** | File-per-item Git-backed YAML store (requirements/*.yaml, principles/*.yaml, tbdts/*.yaml) |

| **Supported OS** | Windows 10/11 (x64), Linux (Ubuntu/Debian/RHEL x64), macOS (Apple Silicon / Intel) |

| **Deployment Options** | Standalone Executables (.exe / binary), Python Wheel, Docker Container, Multi-User Server Host |

| **Backend Engine** | Python 3.12 / FastAPI / Uvicorn / PyInstaller / Watchdog |

| **Frontend Workstation** | React 18 / Vite / CodeMirror 6 / Tailwind CSS / Mermaid.js |

| **Data Formats** | YAML 1.2, JSON, CSV, ReqIF 1.2 XML, Markdown (MkDocs/Hugo), OpenAPI 3.1 |

| **Security** | PBKDF2-HMAC-SHA256 password hashing, HMAC-SHA256 JWT tokens, Ed25519 digital signatures |

4. System Architecture & Design Specification

reqtool — Architecture and Technical Reference

Package Version: 0.3.20

Date: September 28, 2026

1. Purpose & Scope

reqtool is a Git-backed, YAML-based requirements management framework and workstation for systems and software engineering programmes. It supports both formal systems engineering specifications (stakeholder needs, functional, performance, safety, compliance requirements) and agile planning items (themes, initiatives, epics, features, stories) in a unified graph store.

The fundamental premise is that **requirements are source code artefacts**. Every requirement, principle, and TBD is stored as an individual human-readable .yaml file inside a Git repository. The tool layers SHA-256 content hashing, semantic versioning, workflow guard enforcement, automated INCOSE/Vale prose linting, impact analysis, and ReqIF export on top of that Git file store without hiding files from engineers.

2. C4 Architecture Model

2.1 C4 Level 1: System Context Diagram

The System Context Diagram shows how reqtool fits into the broader enterprise environment, interacting with engineers, CI/CD automation pipelines, Git hosting providers, and external regulatory tooling.

```
flowchart TD
  subgraph Users ["Users & Actors"]
    SE["Systems Engineer / Author"]
    PM["Product Manager / Lead"]
  end
```

```

Auditor["Safety / Compliance Auditor"]
end

subgraph ReqToolSystem ["reqtool System Boundary"]
  ReqTool["reqtool Requirements Workstation & API Server"]
end

subgraph ExternalEcosystem ["External Ecosystem & Tooling"]
  GitRepo["Git Remote Repository (GitHub / GitLab / Azure DevOps)"]
  CICD["CI/CD Quality Gate (GitHub Actions / Jenkins)"]
  ValeLinter["Vale CLI Prose Linter Engine"]
  ReqIFTools["External ALM Tools (IBM DOORS / Polarion / PTC Integrity)"]
end

SE <-->|CLI & Web UI (HTTP/WS)| ReqTool
PM <-->|Kanban & Roadmap UI| ReqTool
Auditor <-->|Export Audit Packages (PDF/ReqIF)| ReqTool

ReqTool <-->|YAML Read / Write & Git Commits| GitRepo
CICD <-->|reqtool validate & pytest| ReqTool
ReqTool <-->|Subprocess Prose Analysis| ValeLinter
ReqTool <-->|ReqIF XML Import / Export| ReqIFTools

```

2.2 C4 Level 2: Container Diagram

The Container Diagram details the high-level software containers comprising reqtool: the Single-Page Application (SPA), the FastAPI Web Server, API Routers, In-Memory Graph Store, specialized Domain Store helpers, File I/O Engine, and Git repository backend.

```

flowchart TD
  subgraph Browser ["Web Browser Container"]
    SPA["React SPA Workstation<br/>(Vite / D3 Network Graph / Tailwind CSS)"]
  end

  subgraph AppServer ["Python App Server Container (FastAPI / Uvicorn)"]
    WS_Server["WebSocket Manager<br/>(Real-Time CRDT Co-Editing / Presence)"]
    APIRouters["FastAPI API Routers<br/>(requirements, principles, tbds, modules, exports, analysis)"]
  end

  subgraph StoreLayer ["In-Memory Graph Store Layer"]
    GraphStore["Store (src/reqtool/store.py)<br/>threading.RLock · In-Memory Lookup · Impact Index"]
    AttStore["attachment_store.py"]
    ComStore["comment_store.py"]
    HierStore["hierarchy_store.py"]
  end

  ValEngine["Validation Engine & Vale Linter<br/>(INCOSE rules, safety ASIL, NFR constraints)"]
  FileIO["File I/O & Hashing Engine<br/>(ruamel.yaml · SHA-256 Hashing · SemVer 2.0.0)"]

  subgraph Storage ["FileSystem & Git Container"]
    GitFS["Git Repository Filesystem<br/>.reqtool/ · requirements/ · modules/ · products/"]
  end

  SPA <-->|HTTP REST API (Port 8765)| APIRouters
  SPA <-->|WebSockets (/ws)| WS_Server

```

```

APIRouters --> GraphStore
GraphStore --> AttStore
GraphStore --> ComStore
GraphStore --> HierStore

APIRouters --> ValEngine
GraphStore <--> FileIO
FileIO <--> GitFS

```

2.3 C4 Level 3: Component Diagram (Backend & API Layer)

The Component Diagram breaks down the internal component relationships inside `src/reqtool/`:

```

flowchart TD
    subgraph Routers ["APIRouter Components (src/reqtool/routers/)"]
        R_Req["requirements.py<br/>(CRUD, AC, History, Relationships)"]
        R_Prin["principles.py<br/>(Process vs Product NFR Principles)"]
        R_Tbd["tbds.py<br/>(Unresolved Items & Resolutions)"]
        R_Mod["modules.py<br/>(Remote Git Imports & Overlays)"]
        R_Exp["exports.py<br/>(ReqIF XML, CSV, Matrix, Markdown)"]
        R_Ana["analysis.py<br/>(Impact, Coverage, AI Gherkin AC)"]
        R_Sys["system.py / ws.py<br/>(Config, Metrics, Webhooks, CRDT WS)"]
    end

    subgraph CoreStore ["Store Components (src/reqtool/store.py)"]
        StoreClass["Store Class<br/>self.requirements dict<br/>self.principles dict<br/>self.tbds dict<br/>self.modules dict<br/>self.products dict"]
        TreeBuilder["Tree Construction & Variable Substitution"]
        SearchEngine["In-Memory Search & Impact Index"]
    end

    subgraph StoreHelpers ["Specialized Helpers (src/reqtool/stores/)"]
        H_Hier["hierarchy_store.py<br/>(Workflow Guards & Id Auto-Increment)"]
        H_Com["comment_store.py<br/>(Threaded Discussion Trees)"]
        H_Att["attachment_store.py<br/>(Binary Attachments & YAML Metadata)"]
    end

    subgraph SupportingModules ["Supporting Utility Modules"]
        M_FileIO["fileio.py<br/>(compute_requirement_hash, parse_semver, bump_version)"]
        M_Val["validation.py<br/>(run_vale_lint, INCOSE rules, ASIL checks)"]
        M_Git["git_ops.py<br/>(subprocess git commands, diff, tags)"]
    end

    Routers --> StoreClass
    StoreClass --> TreeBuilder
    StoreClass --> SearchEngine
    StoreClass --> H_Hier
    StoreClass --> H_Com
    StoreClass --> H_Att

    StoreClass <--> M_FileIO
    Routers --> M_Val
    Routers --> M_Git

```

3. Dynamic Behavior & Sequence Diagrams

3.1 Sequence Diagram: Requirement Mutation, Hashing & Suspect Link Flagging

```
sequenceDiagram
    autonumber
    actor Engineer as Systems Engineer
    participant UI as React SPA UI
    participant API as requirements.py Router
    participant Store as Store (store.py)
    participant FileIO as fileio.py
    participant Disk as Git Repository (.yaml)

    Engineer->>UI: Edit Requirement REQ-001 Title / Content
    UI->>API: PUT /requirements/018f... (payload)
    API->>Store: update_requirement(uid, updates)
    Store->>Store: Acquire RLock()
    Store->>FileIO: compute_requirement_hash(new_data)
    FileIO-->>Store: Returns new SHA-256 hash

    alt Content Hash Changed
    Store->>FileIO: bump_version(current_version, "patch")
    FileIO-->>Store: Returns updated version ("1.1.0")
    Store->>Store: Flag downstream child relationships as "suspect=True"
    end

    Store->>FileIO: save_yaml(path, updated_data)
    FileIO->>Disk: Atomically write requirements/018f....yaml
    Store->>Store: Release RLock()
    Store-->>API: Return updated requirement dict
    API-->>UI: 200 OK (updated requirement JSON)
    UI-->>Engineer: Render updated UI & Suspect Link Badge
```

3.2 Sequence Diagram: Real-Time Multi-User WebSocket CRDT Co-Editing

```
sequenceDiagram
    autonumber
    actor Alice as Engineer Alice
    actor Bob as Engineer Bob
    participant UI_A as Alice's Browser
    participant UI_B as Bob's Browser
    participant WS as WebSocket Server (/ws)

    Alice->>UI_A: Opens Requirement REQ-001
    UI_A->>WS: WebSocket send {"type": "PRESENCE", "payload": {"user": "Alice", "uid": "REQ-001"}}
    WS-->>UI_B: Broadcast PRESENCE -> Bob sees Alice's avatar on REQ-001

    Alice->>UI_A: Clicks "description" field
    UI_A->>WS: WebSocket send {"type": "LOCK_FIELD", "payload": {"field": "description", "user": "Alice"}}
    WS-->>UI_B: Broadcast LOCK_FIELD -> Bob sees "description" locked by Alice

    Alice->>UI_A: Types text in description
    UI_A->>WS: WebSocket send {"type": "CO_EDIT", "payload": {"field": "description", "delta": "..."}}
    WS-->>UI_B: Broadcast CO_EDIT -> Bob sees updated description
```

```
WS-->UI_B: Broadcast CO_EDIT -> Bob's screen updates in real time
```

3.3 State Transition Diagram: Workflow Guard Mechanics

```
stateDiagram-v2
[*] --> draft: Create Requirement

draft --> in_review: Submit for Review
note right of in_review
Guard Checks:
1. Rationale non-empty
2. DoR Checklist items complete
end note

in_review --> approved: Approve Specification
note right of approved
Guard Checks:
1. Acceptance Criteria present
2. Owner & Discipline specified
end note

in_review --> draft: Request Changes
approved --> in_progress: Start Implementation
in_progress --> verified: Pass Verification
verified --> done: Sign-off (DoD Complete)
done --> [*]
```

4. Storage & Hashing Integrity Specifications

4.1 Content Hashing Engine

The semantic content hash (`content_hash`) is a SHA-256 digest computed over all domain-significant requirement attributes (`[fileio.py](file:///d:/Antigravity/Scratch/src/reqtool/fileio.py#L22)`):

- Identity: `id`, `title`, `parentId`
- Core: `content.description`, `content.rationale`, `acceptance_criteria`
- Classification: `req_type`, `domain`, `discipline`, `status`, `priority`, `risk`
- Safety: `safety_related`, `safety_classification`, `safety_standard`, `asil_level`, `dal_level`
- Traceability & Planning: `relationships`, `links`, `attachments`, `target_release`, `custom_fields`

Edits to non-semantic metadata (such as discussion comments or local filter preferences) do not alter the `content_hash`, preventing unnecessary semantic version increments.

4.2 Semantic Versioning 2.0.0

Version strings follow SemVer 2.0.0 rules (`MAJOR.MINOR.PATCH[-PRERELEASE][+BUILD]`):

- **PATCH increment:** Non-breaking title or rationale clarifications.
- **MINOR increment:** Added acceptance criteria or new child relationships.
- **MAJOR increment:** Fundamental restructuring of requirement scope.
- **Build Metadata:** Build tags (+build.20260928, +sha.a1b2c3d) are supported via `parse_semver()` and `format_semver()`.

5. Requirements Engineering Philosophy

Requirements Engineering Philosophy & Best Practices

"If you get the requirements right, you can fix the code. If you get the requirements wrong, you can build the wrong product perfectly."

This document details the foundational engineering philosophy behind reqtool. It synthesizes established requirements engineering methodologies—specifically drawing from **Suzanne & James Robertson** (*Mastering the Requirements Process*), the **INCOSE Systems Engineering Handbook**, **ISO/IEC/IEEE 29148**, and **Karl Wiegers & Joy Beatty** (*Software Requirements*)—and explains how reqtool aligns with these principles to deliver robust, traceable, and audit-ready requirements management.

1. Grounding in Requirements Engineering Literature

1.1 The Volere Framework & Robertson & Robertson

In *Mastering the Requirements Process*, James and Suzanne Robertson introduced the **Volere Requirements Process**, emphasizing that a requirement is not merely a statement of functionality, but a complete specification of intent, justification, and verification criteria. Key tenets include:

- **Separation of Description and Rationale:** Every requirement must state **what** is needed alongside **why** it is needed (**Rationale**). Without rationale, engineers cannot evaluate trade-offs or determine if a proposed design change invalidates the requirement's underlying intent.
- **Fit Criteria (Acceptance Criteria):** A requirement without quantified fit criteria cannot be proven true or false. Fit criteria turn subjective requirements ("the system shall be fast") into objective, measurable tests ("99% of requests shall respond within 200 ms").
- **Customer Value & Originator:** Requirements must trace back to a specific stakeholder or business goal, preventing scope creep and unvalidated assumptions.

1.2 INCOSE Systems Engineering Standards

The **International Council on Systems Engineering (INCOSE)** defines strict characteristics for high-quality requirement statements:

1. **Necessary:** Defines an essential capability or constraint without which the system cannot meet its objective.
2. **Implementation-Independent:** Specifies *what* the system must achieve, not *how* to implement it in hardware or software code.
3. **Unambiguous:** Written such that it can only be interpreted in one way by all readers.
4. **Complete:** Contains all necessary information without missing preconditions or triggers.
5. **Consistent:** Free of conflicts with other requirements or principles.
6. **Verifiable:** Can be verified through inspection, analysis, demonstration, or testing.
7. **Traceable:** Uniquely identified and bidirectionally linked to higher-level needs and lower-level verification items.

INCOSE also establishes strict prose rules:

- Avoid **passive voice** ("it shall be considered..."). Use active voice with explicit subjects ("The Flight Controller shall...").
- Avoid **ambiguous words** (fast, user-friendly, robust, seamless, reliable, flexible).
- Avoid **escape clauses** (if possible, where appropriate, to the extent practical).

1.3 ISO/IEC/IEEE 29148 & Wiegers/Beatty Hierarchy

Standard **ISO/IEC/IEEE 29148** (*Systems and software engineering — Life cycle processes — Requirements engineering*) and Karl Wiegers' *Software Requirements* establish a clear hierarchy of requirement levels:

```
Business Needs / Themes (Strategic Intent)
■■■ User / Stakeholder Needs & Initiatives (Capability Outcomes)
■■■ System / Product Requirements & Epics (Functional & Performance Specifications)
■■■ Subsystem / Component Requirements & Features / Stories (Concrete Deliverables)
```

Quality attributes (Non-Functional Requirements / NFRs) such as performance, security, availability, and safety must span and constrain all levels of this hierarchy.

2. How `reqtool` Aligns with Good Requirements Engineering

reqtool was built from the ground up to automate and enforce these exact literature-backed principles directly within modern developer workflows.

```
flowchart TD
  subgraph Literature ["Literature Principles"]
    Volere["Robertson & Robertson:<br>Description + Rationale + Fit Criteria"]
    INCOSE["INCOSE:<br>Unambiguous, Active Voice, No Escape Clauses"]
    ISO["ISO 29148 / Wiegers:<br>Hierarchical Traceability & NFR Constraints"]
  end

  subgraph ReqTool ["ReqTool Implementation"]
    YamlSchema["YAML Schema:<br>content.description, content.rationale, acceptance_criteria"]
    Linter["Validation & Vale Engine:<br>INCOSE prose linter, passive voice & vague word checks"]
    TraceEngine["Traceability Engine:<br>Parent-child links, suspect tracking, 2D matrix"]
    NfrEnforce["Product NFR Constraints:<br>Process vs Product NFR Principles"]
  end

  Volere --> YamlSchema
  INCOSE --> Linter
```

```
ISO --> TraceEngine
ISO --> NfrEnforce
```

2.1 Enforced Volere Schema Structure

reqtool enforces the Robertson & Robertson schema in every requirement .yaml file:

```
id: REQ-042
title: Flight Data Encryption
req_type: functional
domain: security
status: in_review
content:
description: "The Telemetry Module shall encrypt all outbound telemetry packets using
AES-256-GCM."
rationale: "Prevents unauthorized eavesdropping and tampering over unencrypted RF links." #
Robertson & Robertson Rationale
acceptance_criteria: # Volere Fit Criteria
- id: AC-01
text: "Given an outbound telemetry packet, when transmitted over RF, then the payload is
encrypted with AES-256-GCM."
verification_method: test # INCOSE Verifiability
safety_related: true
safety_classification: ASIL_D # ISO 26262 Alignment
```

2.2 Automated INCOSE Prose Quality Linter

Rather than relying on periodic manual reviews, reqtool integrates real-time automated prose quality linting via its INCOSE rule engine and **Vale CLI**:

- **Passive Voice Detection:** Flags statements missing explicit subjects.
- **Ambiguity Scanner:** Identifies subjective terms like user-friendly, robust, seamless, fast.
- **Escape Clause Filter:** Flags conditional cop-outs (if possible, where appropriate).
- **Unquantified Terms Flag:** Highlights missing numeric bounds (high, low).

2.3 Suspect Link Tracking & Change Propagation

In complex engineering programs, changing a high-level specification often invalidates downstream implementation specs without engineers noticing. reqtool implements **Suspect Link Tracking**:

- When a parent requirement undergoes a semantic version bump (e.g. 1.0.0 → 1.1.0), all child relationships are automatically marked as **Suspect**.
- Engineers must review the impact and explicitly clear the suspect flag.

2.4 Product Principles & NFR Constraint Enforcement

Process principles (e.g. "all code must undergo peer review") differ from Product NFR principles (e.g. "maximum latency under 50 ms"). reqtool supports **Product Principle Enforcement**:

- Product NFR principles define global quality constraints.
- reqtool validate checks that every product NFR principle is explicitly satisfied by at least one system requirement, preventing quality constraints from falling through the cracks.

3. Why Use `reqtool`?

| Traditional Requirement Tools (DOORS, Jama, Excel) | The reqtool Approach |

| :--- | :--- |

| **Monolithic Prop-Tech Lock-in:** Expensive proprietary databases that isolate requirements from source code. |

| **Git-Backed Single Source of Truth:** Requirements live as human-readable `.yaml` files alongside code in Git repositories. |

| **Manual & Slow Code Reviews:** Requirements reviews happen in isolated documents disconnected from pull requests. | **Git Branching & PR Code Review:** Full integration with Git PRs, CI/CD pipelines, and GitHub Copilot automated code reviews. |

| **Vague Prose & Missing Rationale:** Text fields often lack mandatory rationale or verifiable acceptance criteria. | **Automated Linter Gates:** Built-in INCOSE prose linter and mandatory Volere schema structure enforced on every `git commit`. |

| **Fragile Traceability:** Manual Excel matrix updates that break when requirements change. | **Automated 2D Matrices & Suspect Tracking:** Graph-based traceability matrix generation, suspect link tracking, and XML ReqIF export. |

4. How to Implement `reqtool` in Your Organization

1. Step 1: Adopt the Standard Vocabulary

Choose your project hierarchy during `reqtool init`:

- Use `reqtool init agile` for software teams (Theme → Initiative → Epic → Feature → Story).
- Use `reqtool init systems` for formal systems/hardware teams (Stakeholder Need → Functional → Performance → Safety).
- Use `reqtool init defaults` for hybrid embedded/agile programmes.

2. Step 2: Enforce Validation in CI/CD & PRs

Add `reqtool validate` to your GitHub Actions or CI pipeline (`.github/workflows/ci.yml`). Block merges if requirements contain blocking schema errors or unmapped product principles.

3. Step 3: Leverage Automated Quality Linters

Run `reqtool validate` locally or via the React UI workstation (`reqtool serve`) to inspect INCOSE prose warnings before submitting specifications for peer review.

4. Step 4: Conduct Baselines & Regulatory Audits

Tag release milestones with `reqtool baseline snapshots (req baseline create v1.0)`. Export formal 2D coverage matrices and ReqIF files for ISO 26262 / DO-178C certification audits.

6. Standalone Binaries & Deployment Guide

`reqtool` Standalone Executable Guide

The standalone executable release of reqtool provides the full requirements management engine, REST API backend, CLI tools, and React Web Workstation bundled as standalone binary executables.

No Python, Node.js, or external dependencies are required to run reqtool-web.exe or reqtool-app.exe.

1. Executable Release Options

| Executable | Auto-Launch Behavior | Recommended Use Case |

| :--- | :--- | :--- |

| reqtool-web.exe | Automatically launches the Web Workstation in your default web browser (http://localhost:8765). | General engineering work, browser tab navigation, dual-screen setups. |

| reqtool-app.exe | Automatically launches the visual workstation in a dedicated standalone application window. | Distraction-free native app experience for requirements engineers. |

| reqtool-server.exe | Automatically launches central backend server listening on 0.0.0.0:8765. | Multi-user team server hosting requirements repository for remote client connections. |

| reqtool-client.exe | Automatically launches desktop client application configured with Remote Server Login UI. | Multi-user remote client connecting to central backend server with login credentials. |

| reqtool.exe | Command Line Interface (CLI). Run with -help for available subcommands. | Terminal workflows, CI/CD scripts, automated validation & exports. |

2. Quick Start

2.1 Downloading Executables

Download the executable binaries for your operating system from the [GitHub Releases Page](https://github.com/markbac/reqtool/releases):

- **Windows:** reqtool-web-windows-amd64.exe, reqtool-app-windows-amd64.exe, reqtool-cli-windows-amd64.exe
- **Linux:** reqtool-web-linux-amd64, reqtool-app-linux-amd64, reqtool-cli-linux-amd64
- **macOS:** reqtool-web-macos-amd64, reqtool-app-macos-amd64, reqtool-cli-macos-amd64

2.2 Launching the Application

Web Interface Executable

Double-click `reqtool-web.exe` (or run `./reqtool-web.exe` from terminal). The backend server will start and automatically open your default browser to `http://localhost:8765`.

Native App Executable

Double-click `reqtool-app.exe` (or run `./reqtool-app.exe` from terminal). The workstation will launch in an isolated desktop application window.

Command Line Executable

```
reqtool.exe --help
reqtool.exe init defaults
reqtool.exe validate
reqtool.exe export --format csv
```

3. In-Tool Repository Management (Switch & Init)

Both `reqtool-web.exe` and `reqtool-app.exe` feature dynamic repository switching and initialization directly inside the UI without needing to restart the process.

3.1 Opening Repository Management

Click the  Repo button in the top navigation bar (or select  **Repository** from the overflow  menu).

3.2 Switching Repositories

1. Open the  **Repository** modal.
2. Select the **Switch Repository** tab.
3. Enter the absolute directory path of an existing `reqtool` repository.
4. Click **Switch Repository**. The backend will instantly validate the path, load the target repository, and refresh the UI tree.

3.3 Initializing a New Repository

1. Open the  **Repository** modal.
2. Select the **Init New Repository** tab.
3. Enter the directory path for the new repository.
4. (Optional) Provide a Repo ID slug and Title.
5. Select the template mode:

- **Engineering Defaults:** Pre-populates ~30 engineering principles, standard enums, and workflows.
- **Blank Repository:** Creates clean repository directory structure with empty configuration files.

6. Click **Initialize Repository**. The new repository will be created and immediately loaded into the UI.

4. Multi-User Server & Remote Client Setup

reqtool supports multi-user collaboration where a central backend server hosts the requirements repository, and multiple team members connect remotely using `reqtool-client.exe` or web browser.

4.1 Running Central Server (`reqtool-server.exe`)

Launch `reqtool-server.exe` on your server host machine:

```
reqtool-server.exe --port 8765
```

- Listens on `0.0.0.0:8765` by default, making it accessible across your local network or server IP.
- Automatically seeds default admin credentials (`admin / admin123`) in `.reqtool/users.yaml`.

4.2 Client Remote Connection & User Login (`reqtool-client.exe`)

1. Launch `reqtool-client.exe` on client desktop.
2. Click **User / Login** widget in top navigation bar or menu.
3. Enter Server URL (e.g. `http://server-hostname:8765`).
4. Enter Username (`admin`) and Password (`admin123`).
5. Click **Login & Connect**.
6. The client authenticates via JWT token and connects to the central server repository.

5. Command Line Reference (`reqtool.exe`)

| Command | Usage & Purpose |

| :--- | :--- |

| `reqtool.exe serve` | Starts the backend API & Web Workstation. Use `--browser` for browser auto-open, `--app` for desktop app mode, `--port` for custom ports. |

| `reqtool.exe init` | Initialises `.reqtool/` config in current directory. Subcommands: `defaults`, `agile`, `systems`, `demo`. |

| `reqtool.exe validate` | Runs full schema checks, vocabulary validation, and INCOSE prose quality linter. Exits 0 if clean. |

```
reqtool.exe export	Exports requirements to CSV, ReqIF 1.2 XML, Markdown document tree, or React JSX components.
reqtool.exe new	Interactively creates a new requirement, principle, or TBD artefact.
reqtool.exe new-agile	Interactively creates an agile planning item (story, task, bug, epic, initiative).
reqtool.exe diff	Shows requirement changes and version increments since a Git ref (e.g. reqtool.exe diff HEAD~1).
reqtool.exe stats	Displays project summary metrics, quality gap counters, and approval rates.
reqtool.exe openapi	Exports the full OpenAPI 3.0 REST spec to openapi.json or stdout.
```

...

5. Standalone Architecture & Self-Containment

[illegible]

- **Single Binary Executable:** Includes PyInstaller bootloader, Python runtime, FastAPI REST API, INCOSE linter rules, and pre-compiled React SPA assets.
- **Zero Installation:** Download and run directly. No setup installers, admin permissions, or external runtimes required.

7. Git LFS & Binary Attachment Best Practices

regtool -- Git Binary Attachment & Audit History Guide

— — —

1. Managing Binary Attachments in Git

Because reqtool is 100% Git-backed, managing binary attachments (images, PDFs, CAD schematics, wireframes) requires balancing version control transparency with repository size optimization.

1.1 Recommended Setup: Git LFS (Large File Storage)

To prevent binary file bloat in standard Git object databases while retaining full version control, reqtool repositories should configure **Git LFS** via `.gitattributes`.

Create or update `.gitattributes` in your repository root:

```
# .gitattributes -- reqtool Git LFS tracking for binary attachments
attachments/**/*.png filter=lfs diff=lfs merge=lfs -text
attachments/**/*.jpg filter=lfs diff=lfs merge=lfs -text
attachments/**/*.jpeg filter=lfs diff=lfs merge=lfs -text
attachments/**/*.gif filter=lfs diff=lfs merge=lfs -text
attachments/**/*.pdf filter=lfs diff=lfs merge=lfs -text
attachments/**/*.zip filter=lfs diff=lfs merge=lfs -text
attachments/**/*.step filter=lfs diff=lfs merge=lfs -text
```

Benefits:

- Lightweight Git clones (`git clone` remains fast).
- Large media files are stored on Git LFS pointers natively supported by GitHub, GitLab, and Azure DevOps.
- Full binary version history is preserved per branch.

2. Audit Trail & Attachment Version History

In reqtool, attachment operations (add, update, replace, delete) are subject to the same strict audit control as requirement text changes.

2.1 Metadata & SHA-256 Tracking in Requirement YAML

Whenever an attachment is added, its metadata and cryptographic SHA-256 hash are recorded inside the requirement's YAML file (`requirements/.yaml`):

```
id: REQ-SYS-001
title: Thermal Control System Enclosure
version: 1.2.0
attachments:
- filename: enclosure_v2.png
  size_bytes: 452100
  sha256: a1b2c3d4e5f67890123456789abcdef0123456789abcdef0123456789abcdef0
  added_by: alice_engineer
  added_date: '2026-09-30T18:45:00Z'
description: Enclosure revision 2 CAD render
```

2.2 Automatic Requirement Version Bump

In reqtool, uploading or deleting an attachment alters the requirement's content hash and triggers an automatic semantic version increment (`1.1.0` → `1.2.0`).

2.3 Git Audit Log & Historical Diffs

Every attachment operation generates an immutable Git commit:

```
commit a1b2c3d4
Author: Alice Smith <alice@example.com>
Date: Wed Sep 30 18:45:00 2026

feat(REQ-SYS-001): added attachment enclosure_v2.png
```

To inspect full audit history of attachments for any requirement:

- **CLI:** `reqtool diff HEAD~5`
- **REST API:** `GET /requirements/{uid}/history`
- **Regulatory Audit Exporter:** `GET /export/audit-package` includes `cryptographic_signatures.json` locking requirement text, author identities, and attachment SHA-256 hashes into the compliance dossier.