Chubb Enterprise Ontology Governance

Operating Guide, Change Lifecycle, CI/CD Flowcharts and Junior Developer Playbook

Version 1.0 | August 2026

Purpose: explain exactly how an ontology change moves from a developer laptop to an approved enterprise semantic release.

Scope note: This is an illustrative governance reference for a Chubb insurance context. It should be aligned with Chubb internal SDLC, security, repository, data governance and release policies before production adoption.

Contents

1. Governance objective and mental model

Enterprise ontology governance controls how shared business meaning changes. In an insurance organization, a change to Policy, Claim, Coverage, Party, Product, Underwriter or a relationship between them can alter inference, mappings, APIs, data products and AI context. Therefore ontology changes must be treated as governed semantic releases, not as casual file edits.

Key principle: Keep concept IRIs stable. Track term evolution through Git + ChangeSet + provenance. Version the ontology/module release. Only mint a new concept IRI when the concept identity/meaning is truly replaced.

1.1 Five questions every change must answer

1. What semantic entity changed?
2. Who made the change?
3. Why was it changed and which business requirement/ticket authorized it?
4. What consumers or inferred facts can be affected?
5. Which immutable ontology release contains the approved change?

1.2 Separation of responsibilities

Artifact	Purpose	Example
OWL/RDFS	Meaning, hierarchy and inference	Policy rdfs:subClassOf Contract
SHACL	Data validation	Policy must have policyNumber
SKOS	Controlled values/taxonomies	ClaimStatus: Open, Closed
A-Box sample data	Demonstrate/test usage	policy123 a CyberPolicy
Competency questions	Acceptance/regression behavior	Which policies cover cyber risk?
ChangeSet/PROV	Who/what/why/when audit	CHG-2026-0452

2. Reference architecture

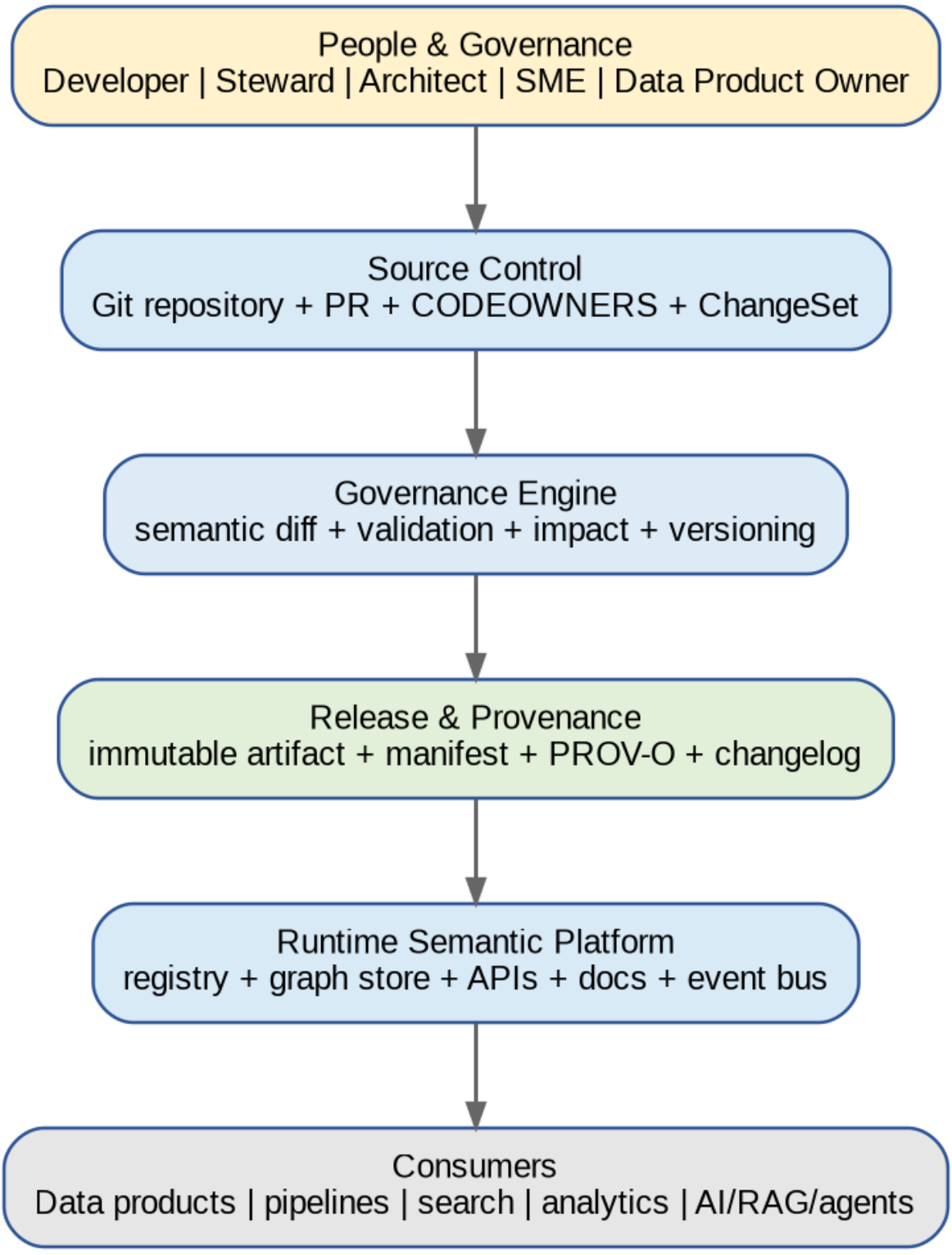


Figure 1 - Governance architecture from human decision to semantic consumers

The governance engine sits between source control and release. This placement is deliberate: local edits can be analyzed automatically, but enterprise semantic state changes only after the approved release is built and published.

2.1 Golden path

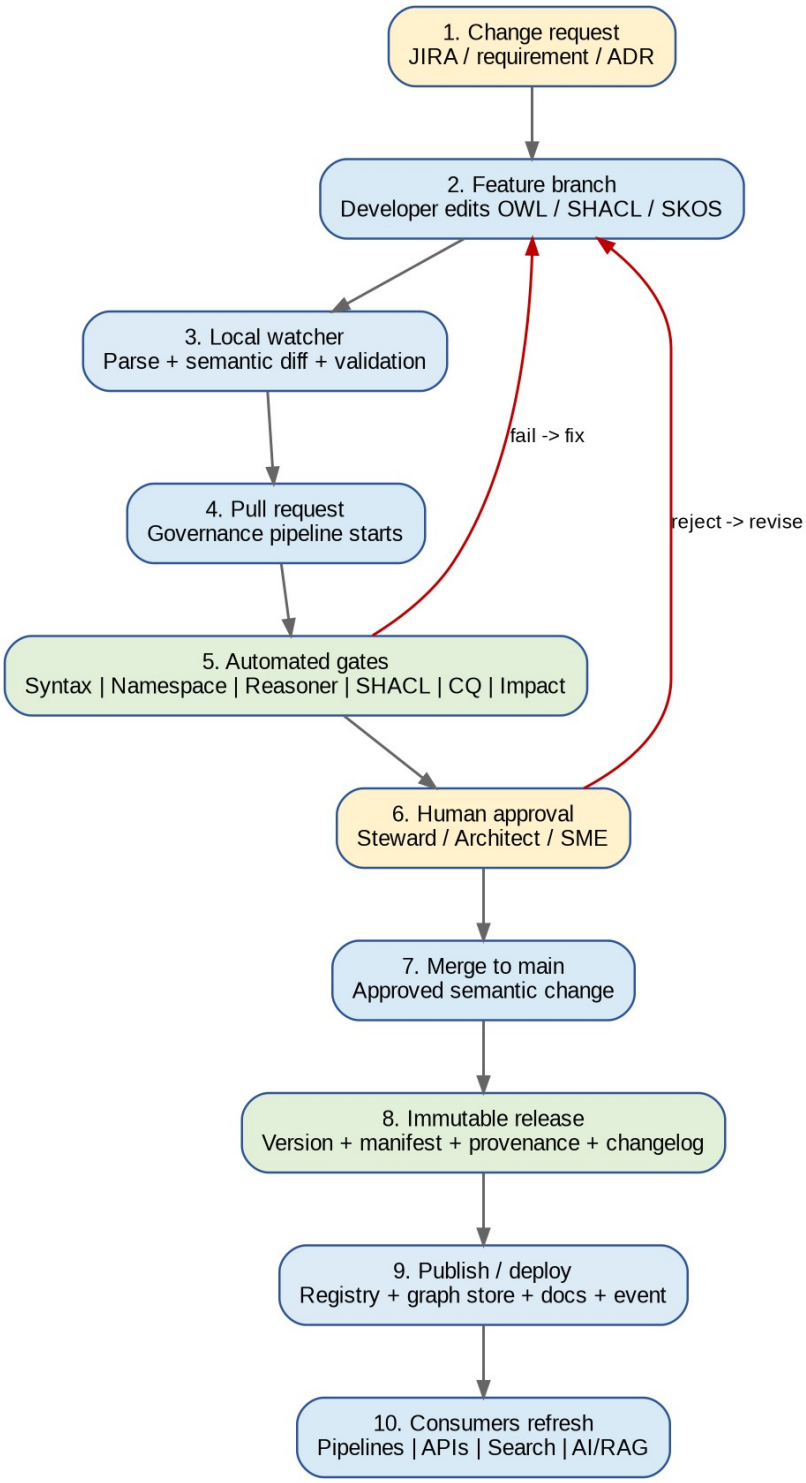


Figure 2 - End-to-end ontology change lifecycle

3. Where ontology artifacts live

A predictable repository layout allows tooling, CODEOWNERS, CI filters and junior developers to know where each semantic artifact belongs.

```
chubb-enterprise-ontology/
├── ontology/
│   ├── foundation/core.ttl
│   ├── policy/policy.ttl
│   ├── claim/claim.ttl
│   ├── party/party.ttl
│   └── coverage/coverage.ttl
├── vocabulary/
│   ├── claim-status.ttl
│   └── line-of-business.ttl
├── shapes/
│   ├── policy.shacl.ttl
│   └── claim.shacl.ttl
├── mappings/
│   ├── acord/
│   └── external/
├── competency/
│   ├── CQ-POL-001.rq
│   └── expected/
├── samples/
├── governance/
│   ├── ontology-governance.yaml
│   ├── owners.yaml
│   └── release-state.json
├── changes/
│   └── CHG-YYYY-NNNN.yaml
├── provenance/
├── releases/
├── reports/
├── scripts/
├── tests/
└── .github/workflows/ontology-governance.yml
```

3.1 Rule: one authoritative location

- OWL logic belongs under ontology/.
- Validation constraints belong under shapes/.
- Controlled vocabulary concepts belong under vocabulary/.
- External alignments belong under mappings/.
- Generated JSON-LD, docs or release bundles should be generated; do not maintain duplicate semantic truth manually.

4. Namespace and versioning rules

Purpose	Pattern	Example
Ontology namespace	https://data.chubb.com/ontology/{domain}/	https://data.chubb.com/ontology/policy/
Class/property IRI	stable, no version suffix	.../policy/Policy
Ontology IRI	stable module identity	https://data.chubb.com/ontology/policy
Version IRI	immutable release	https://data.chubb.com/ontology/policy/1.4.0
Shape namespace	.../shapes/{domain}/	https://data.chubb.com/shapes/policy/
Vocabulary namespace	.../vocabulary/{scheme}/	https://data.chubb.com/vocabulary/claim-status/
Resource/A-Box	.../resource/{domain}/	https://data.chubb.com/resource/policy/

		P123

Key principle: Do not create Policy_v2, Policy_2026 or Policy_new for normal evolution. The stable IRI Policy remains the identity; the containing ontology release gets a new version.

5. What triggers when a developer makes a change

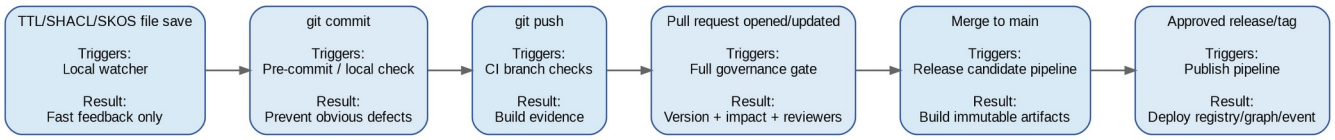


Figure 3 - Trigger map from local save to enterprise release

Developer action	Automatic trigger	What it is allowed to do
Save TTL/SHACL/SKOS	Local watcher	Analyze and report only; never publish production
Commit	Pre-commit checks	Block invalid syntax/namespaces early
Push branch	Branch CI	Run test/gate subset and archive evidence
Open/update PR	Full governance workflow	Semantic diff, impact, version recommendation, reviewer routing
Merge approved PR	Release candidate	Build immutable bundle and provenance
Approve/tag release	Publish/deploy	Update registry/graph/docs and emit release event

Key principle: Automatic reflection means automatic analysis and automatic propagation after approval - not automatic production mutation on file save.

6. Daily developer workflow

6.1 Before editing

6. Create or reference a change request such as ONTO-452.
7. Identify owning ontology module and steward.
8. Pull latest main branch and create a feature branch.
9. Run the baseline governance gate; it must pass before editing.

```
git checkout main
git pull
git checkout -b feature/ONTO-452-cyber-policy
python scripts/govern.py --ci
```

6.2 While editing

- Keep the stable IRI when the same concept is being refined.
- Update OWL, SHACL, SKOS, mappings and competency questions together when required.
- Run the local watcher for immediate feedback.
- Do not edit generated release bundles.

```
python scripts/watch.py
# Edit ontology/policy/policy.ttl and save.
```

6.3 Before push

10. Read the semantic diff report.
11. Confirm the proposed PATCH/MINOR/MAJOR level.
12. Fix all blocking validation failures.
13. Add/update ChangeSet with business reason and ticket.
14. Commit with ticket in the message.

```
git add .
git commit -m "ONTO-452 Add CyberPolicy concept"
git push -u origin feature/ONTO-452-cyber-policy
```

7. Automated CI/CD governance pipeline

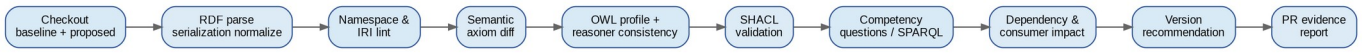


Figure 4 - Pull-request semantic governance pipeline

7.1 Gate behavior

Gate	Purpose	Typical result
RDF parser	Reject invalid Turtle/RDF	BLOCK
Namespace lint	Enforce Chubb IRI/prefix rules	BLOCK
Semantic diff	Compare RDF meaning, not line formatting	REPORT
OWL profile/reasoner	Find logical inconsistency/profile violation	BLOCK
SHACL	Validate sample/contract constraints	BLOCK
Competency questions	Protect expected semantic answers	BLOCK
Impact analyzer	Find affected classes/shapes/mappings/consumers	REPORT/ESCALATE
Version calculator	Recommend semantic release level	BLOCK if under-versioned
Evidence report	Give reviewer human-readable change summary	REQUIRED

7.2 Why semantic diff instead of text diff?

Turtle can be reformatted without changing meaning. Governance should compare normalized RDF statements/axioms and classify changes by semantic role. A textual diff is still useful for source review, but it cannot be the only release decision input.

8. Semantic version decision rules

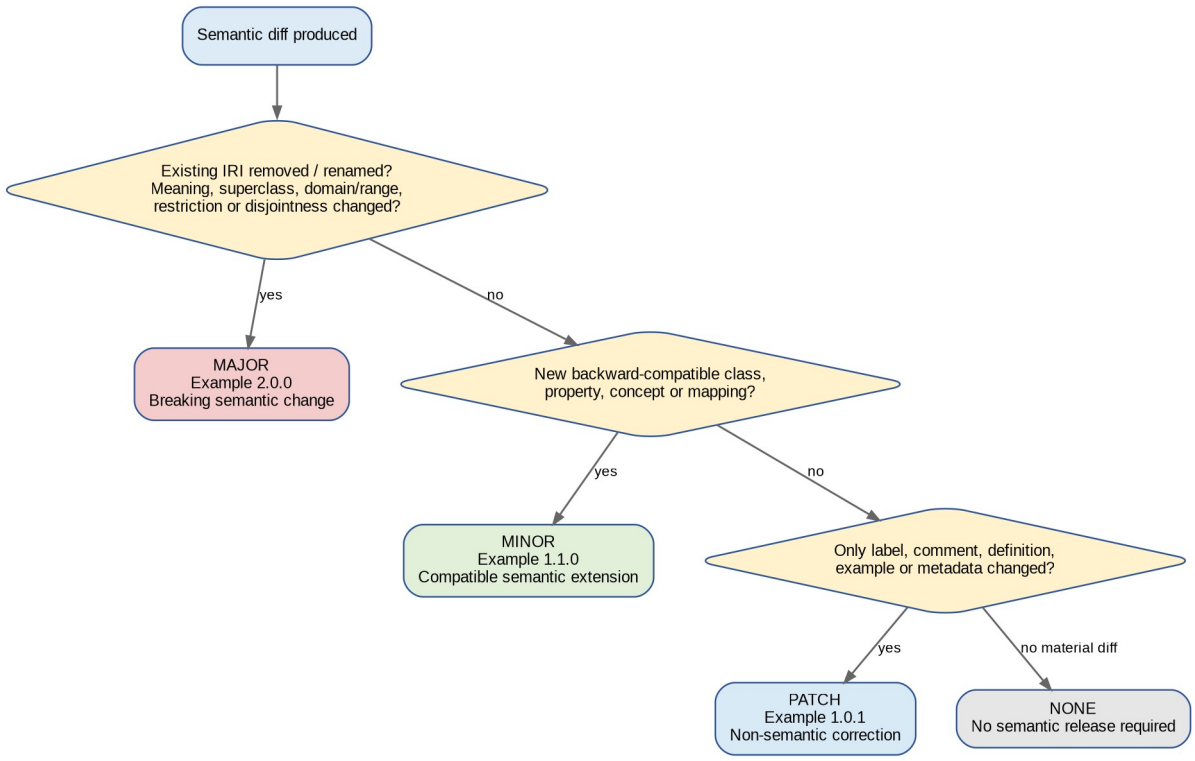


Figure 5 - Semantic version decision flow

Change example	Recommended level	Reason
Correct rdfs:label spelling	PATCH	No semantic behavior change
Add skos:definition or example	PATCH	Documentation/annotation
Add CyberPolicy class	MINOR	Backward-compatible extension
Add hasUnderwriter property	MINOR	New compatible capability
Add new SKOS claim status	MINOR	Vocabulary extension; assess consumers
Remove class/property	MAJOR	Breaking consumer references
Rename term IRI	MAJOR	Identity break
Change domain/range	MAJOR	Can change inferred typing
Change superclass of existing class	MAJOR	Changes taxonomy/inference
Add disjointness/cardinality restriction	MAJOR	Can introduce inconsistency or validation impact

Key principle: The pipeline recommends a version. A human can override only with an explicit governance justification recorded in the ChangeSet.

9. Human approval and RACI

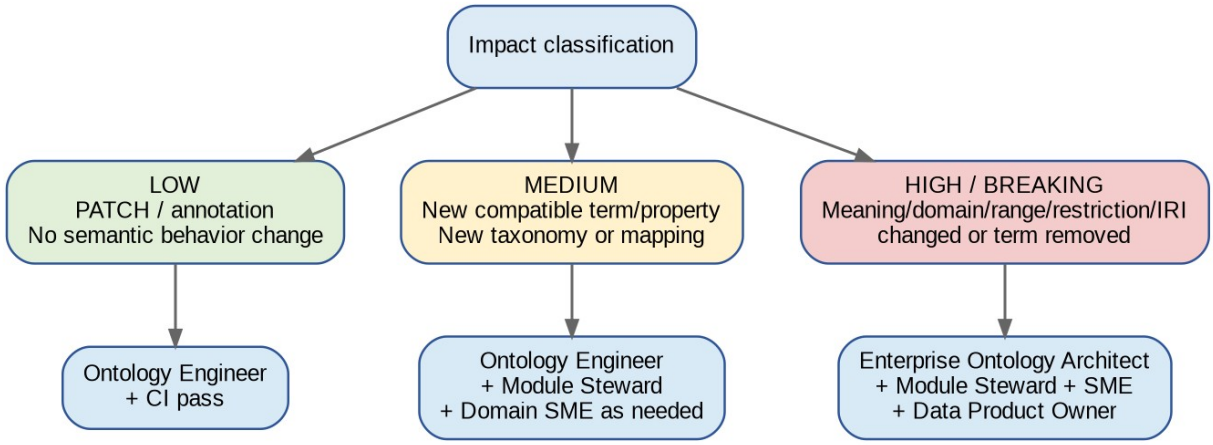


Figure 6 - Approval depth based on semantic impact

Role	Accountability
Ontology Engineer	Model change, tests, ChangeSet, local validation
Module Steward	Own semantic quality and backward compatibility for the module
Domain SME	Verify business meaning such as policy/claim/coverage semantics
Enterprise Ontology Architect	Approve cross-domain/breaking architecture decisions
Data Product Owner	Accept downstream contract/business impact
Semantic Platform/DevOps	Maintain CI, registry, graph deployment and release automation
Data Governance/Security	Approve regulated/sensitive taxonomy and policy implications when applicable

9.1 Reviewer routing rule

CODEOWNERS or equivalent repository controls should automatically request the owner of every modified ontology, vocabulary, shape or mapping directory. Breaking changes additionally require enterprise architecture approval.

10. Release, provenance and audit trail

Every approved release is immutable. Do not overwrite 1.1.0 after publication. Fixes produce a later version such as 1.1.1 or 1.2.0.

10.1 Release contents

```
releases/1.1.0/
├── ontology/
├── vocabulary/
├── shapes/
├── mappings/
├── manifest.json
├── semantic-diff.json
├── change-report.md
└── checksums.sha256
```

10.2 ChangeSet audit fields

Field	Example
changeId	CHG-2026-0452
ticket	ONTO-452
target entity	policy:CyberPolicy
author	corporate identity derived from Git/PR
reviewers	Policy Ontology Steward
reason	Support cyber insurance product semantics
prior version	1.0.0
new version	1.1.0
commit	Git SHA
approval timestamp	2026-08-19T...
impact	MINOR / medium
affected consumers	policy API, mapping pipeline, semantic search

10.3 PROV-O model

```
chg:CHG-2026-0452 a gov:OntologyChangeSet, prov:Activity ;
    gov:changesEntity policy:CyberPolicy ;
    prov:wasAssociatedWith agent:Developer123 ;
    gov:approvedBy agent:PolicyOntologySteward ;
    gov:ticket "ONTO-452" ;
    gov:generatedVersion "1.1.0" ;
    dcterms:description "Add CyberPolicy under CommercialPolicy" .
```

11. Automatic runtime propagation

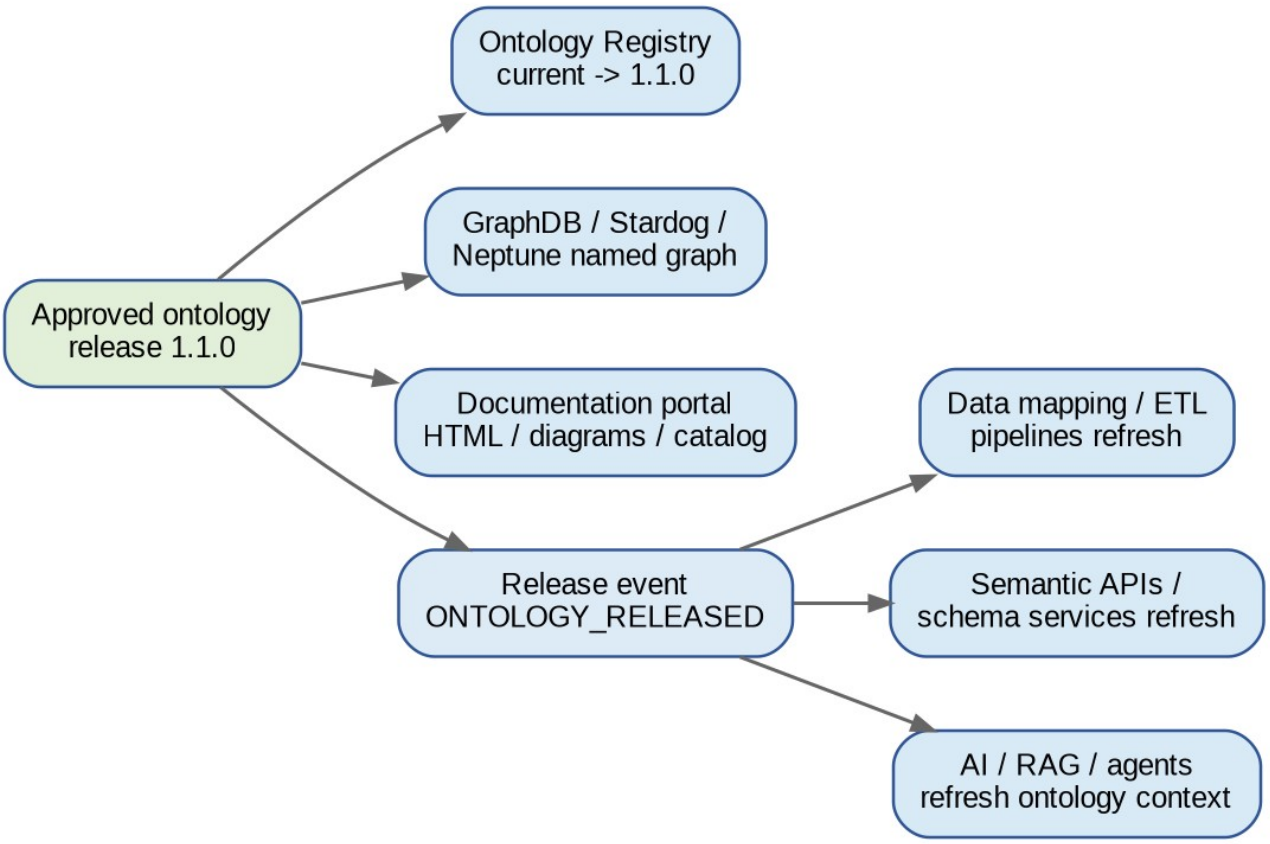


Figure 7 - What automatically reflects after an approved ontology release

The release event is the safe boundary between ontology governance and consuming systems. Consumers should not watch raw Git files in production; they should consume an approved registry version or release event.

11.1 Example release event

```
{
  "eventType": "ONTOLOGY_RELEASED",
  "ontology": "policy",
  "previousVersion": "1.0.0",
  "version": "1.1.0",
  "changeId": "CHG-2026-0452",
  "breakingChange": false,
  "artifactUri": ".../ontology/policy/1.1.0"
}
```

11.2 Consumer behavior

- Graph platform loads the immutable version into a versioned named graph and changes the current pointer after validation.
- Semantic API/schema service reloads approved definitions or invalidates cache.
- Data pipeline mapping service reruns compatibility checks before using a new major release.
- Documentation/catalog generator publishes new term pages and change history.
- AI/RAG/agent services refresh ontology context/index only from approved releases.

12. Rollback and recovery

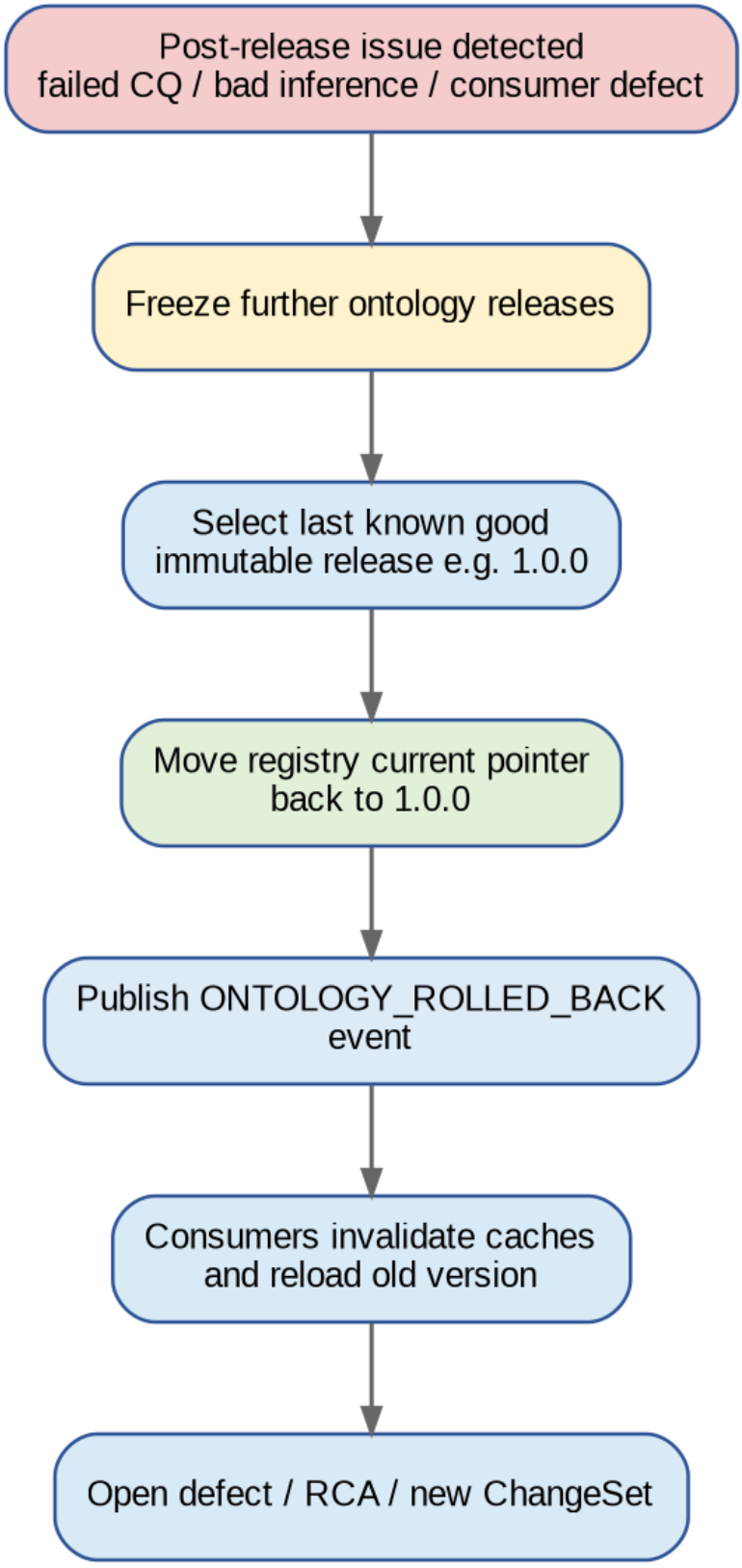


Figure 8 - Rollback path using immutable releases

Key principle: Rollback changes the active release pointer; it does not delete history. The defective release remains auditable and must never be silently rewritten.

12.1 Rollback triggers

- Reasoner inconsistency discovered after integration.
- Critical competency question/regression failure.
- Unexpected API or mapping incompatibility.
- Material downstream data-product defect.
- Incorrect business meaning approved in error.

13. Worked Chubb example: adding CyberPolicy

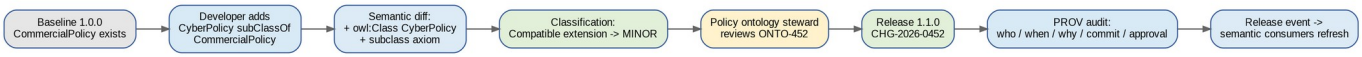


Figure 9 - Worked lifecycle for a compatible Policy ontology extension

13.1 Baseline

```
policy:CommercialPolicy a owl:Class ;
    rdfs:subClassOf policy:Policy ;
    rdfs:label "Commercial Policy"@en .
```

13.2 Proposed change

```
policy:CyberPolicy a owl:Class ;
    rdfs:subClassOf policy:CommercialPolicy ;
    rdfs:label "Cyber Policy"@en ;
    skos:definition "A commercial policy concept used to model cyber insurance coverage."@en .
```

13.3 What automation reports

Finding	Value
Added entity	policy:CyberPolicy
Added axiom	CyberPolicy subClassOf CommercialPolicy
Removed existing semantics	None
Impact class	Compatible extension
Recommended version	1.1.0
Required approver	Policy module steward (+ SME per policy)
Change record	CHG-2026-0452 / ONTO-452
Post-approval event	ONTOLOGY_RELEASED

13.4 Contrast: breaking change

If a developer changes CommercialPolicy from subClassOf Policy to another top-level concept, the meaning of every CommercialPolicy instance changes. The pipeline should classify this as MAJOR and route to enterprise ontology architecture governance.

14. Junior developer hands-on procedure

14.1 Start the repository

```
python -m venv .venv
# macOS/Linux: source .venv/bin/activate
# Windows: .venv\Scripts\Activate.ps1
pip install -r requirements-full.txt
python scripts/govern.py --ci
```

14.2 Run watcher and make a safe sample change

```
python scripts/watch.py
# In another terminal:
python scripts/demo_change.py add-class
```

Expected result: validation PASS, semantic impact MINOR, suggested version 1.1.0. The watcher does not deploy anything.

14.3 Prepare a pull request

```
git checkout -b feature/ONTO-452-cyber-policy
# make and test change
python scripts/govern.py --ci
git add .
git commit -m "ONTO-452 Add CyberPolicy concept"
git push -u origin feature/ONTO-452-cyber-policy
```

14.4 Never do these

- Do not edit main directly.
- Do not rename an IRI to make a label look nicer.
- Do not remove a class because it appears unused without dependency/consumer analysis.
- Do not put validation rules into OWL merely because SHACL is inconvenient.
- Do not edit an already-published release folder.
- Do not bypass a failing competency question to get the PR green.

15. Operational governance cadence

Cadence	Activity	Owner
Every change	PR semantic diff, tests, version, approvals	Engineer + steward
Daily	Review blocked ontology PRs / failed gates	Module stewards
Weekly	Cross-domain change review; upcoming breaking changes	Enterprise ontology architect
Monthly	Ontology quality metrics, deprecated terms, consumer compatibility	Governance board
Quarterly	Namespace/catalog review, external ontology dependency review	Architecture + platform
Per major release	Migration plan, consumer communication, rollback rehearsal	Product owners + platform

15.1 Suggested metrics

- Number of ontology changes by PATCH/MINOR/MAJOR.
- PR validation failure rate and common rule violations.
- Average approval time by impact class.
- Deprecated terms with active consumers.
- Competency-question pass rate.
- Number of consumers not yet compatible with latest major version.
- Unowned terms/modules and missing provenance records.

16. Production implementation checklist

[ ] Approve enterprise namespace policy and prefix registry.

[ ] Define foundation/domain/module boundaries and owners.

[ ] Establish Git repository, protected main branch and CODEOWNERS.

[ ] Implement semantic RDF/OWL diff rather than line-only comparison.

[ ] Implement parser, namespace lint, OWL reasoner, SHACL and CQ gates.

[ ] Define semantic version rules and breaking-change classifier.

[ ] Define ChangeSet schema and corporate identity mapping.

[ ] Generate immutable release bundle, checksum, manifest and PROV-O record.

[ ] Deploy an ontology registry with stable current-version pointers.

[ ] Load releases into versioned graph/named graphs; never overwrite historical graph.

[ ] Publish release/rollback events for downstream consumers.

[ ] Register APIs, pipelines, data products and AI services as ontology consumers for impact analysis.

[ ] Implement approval routing and override justification.

[ ] Implement rollback automation and test it.

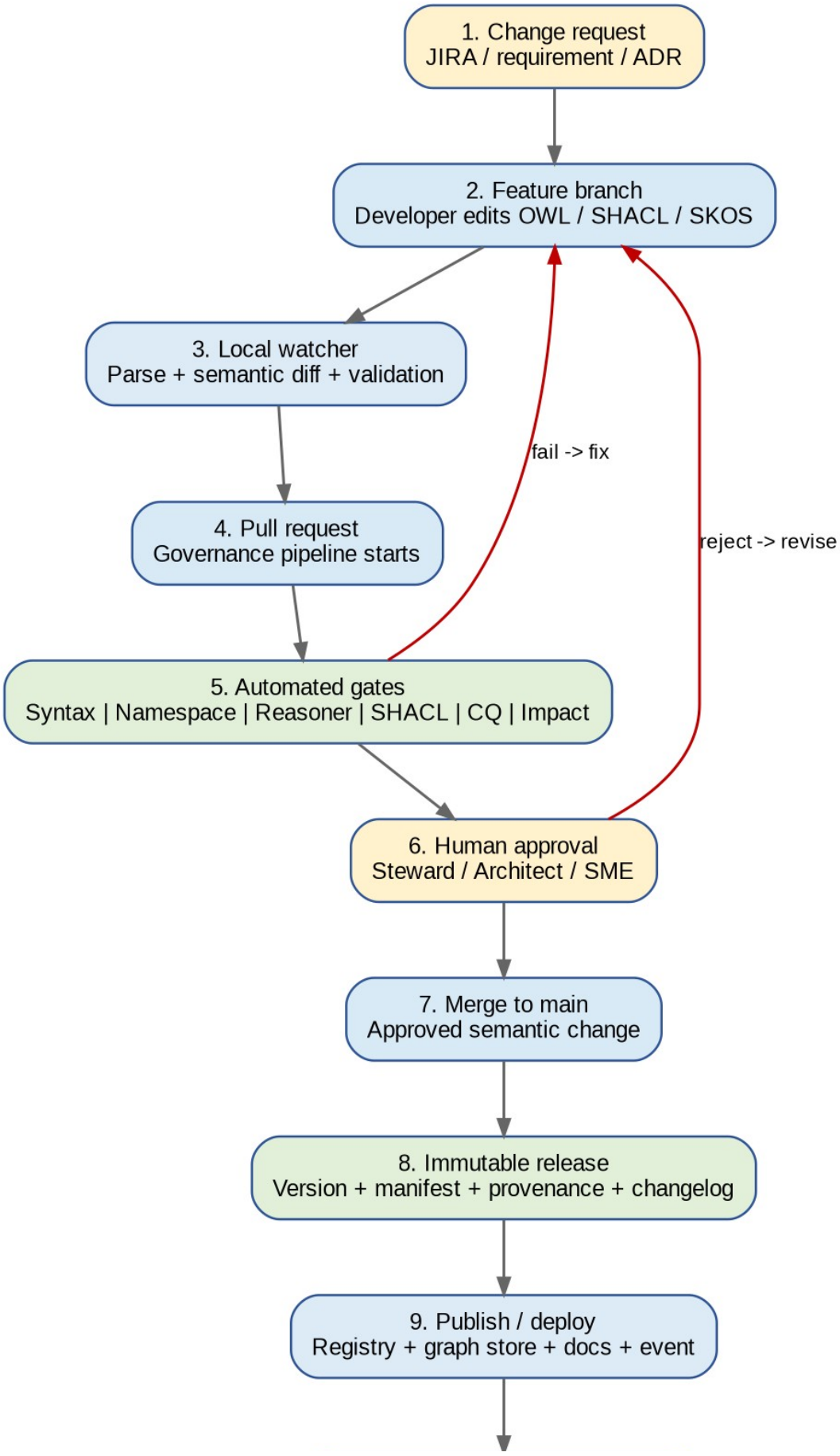
[ ] Train junior developers using the included sample repository and worked CyberPolicy change.

16.1 Final operating principle

Enterprise rule: Detect automatically. Validate automatically. Measure impact automatically. Approve according to semantic risk. Release immutably. Propagate only approved versions. Preserve every audit record and make rollback easy.

Appendix A - One-page quick reference

If you change…	Treat as…	Who reviews…
Label/comment/definition	PATCH	Engineer / standard PR
New class/property/value	MINOR	Module steward + SME as required
Superclass/domain/range/restriction	MAJOR	Steward + Enterprise Ontology Architect + SME
Remove/rename IRI	MAJOR	Architecture + consumer migration plan
SHACL constraint	PATCH/MINOR/MAJOR based on data compatibility	Shape owner + product owner for breaking validation
External mapping	MINOR unless breaking existing mappings	Module steward


1. Change request
JIRA / requirement / ADR
2. Feature branch
Developer edits OWL / SHACL / SKOS
3. Local watcher
Parse + semantic diff + validation
4. Pull request
Governance pipeline starts
5. Automated gates
Syntax | Namespace | Reasoner | SHACL | CQ | Impact
6. Human approval
Steward / Architect / SME
7. Merge to main
Approved semantic change
8. Immutable release
Version + manifest + provenance + changelog
9. Publish / deploy
Registry + graph store + docs + event
fail -> fix
reject -> revise