

How to Review a Python Programming Project: Purpose, Scope, and Review Mindset



Review is evidence-based assessment of readiness, maintainability, and risk — not style preference



Know your review type: code, architecture, dependencies/security, tests and CI, docs, release readiness



Set the contract first: in scope, out of scope, expected output, decision criteria



Agree on depth before opening the repo: smoke review, maintainability review, or full readiness review



Check intent before rules: does the change do what the ticket asks, and only that?



Why Reviews Fail Without Shared Pass/Fail Criteria

- No explicit criteria → reviews collapse into preference debates, real risks ship
- Green CI only means tools are happy; architecture and intent go unchecked
- A checklist applies judgment to every area, not just what caught your eye
- Treat it as a ledger: PASS, FAIL, or N/A per rule, with file:line evidence
- N/A is a decision; a blank row means the rule was never considered
- Write comments only from FAIL rows, label severity, keep the list short
- AI writes more diffs now — reviewer discipline matters more, not less

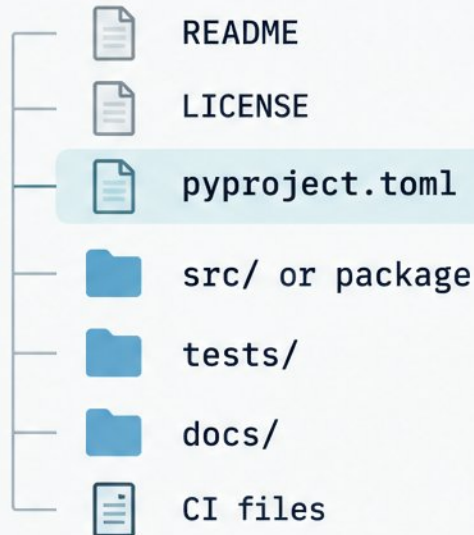


Repository Structure and Packaging Sanity Checks

A healthy project layout and configuration prevents build and release problems.

CLEAN STRUCTURE
RELIABLE BUILDS
HAPPIER DEVELOPERS

PROJECT REPOSITORY (TOP LEVEL)



pyproject.toml (KEY PARTS)

[build-system]

Declared here
Standards-based backend
No setup.cfg metadata

Project metadata

name
version
description
readme
license
authors
requires-python
dependencies

1



Top-level layout: README, LICENSE, pyproject.toml, src/ or package, tests/, CI files

2



pyproject.toml is the single build config — [build-system] declared, standards-based backend, no setup.cfg metadata

3



Core metadata complete and consistent: name, version, description, readme, license, authors, requires-python, dependencies

4



Version defined once; never both static and listed in dynamic — PEP 621 treats that as a hard build error

5



License follows PEP 639: SPDX expression plus relative license-files globs; no deprecated license table or License:: classifiers

6



requires-python checked against the calendar, not just syntax — an EOL floor blocks release

7



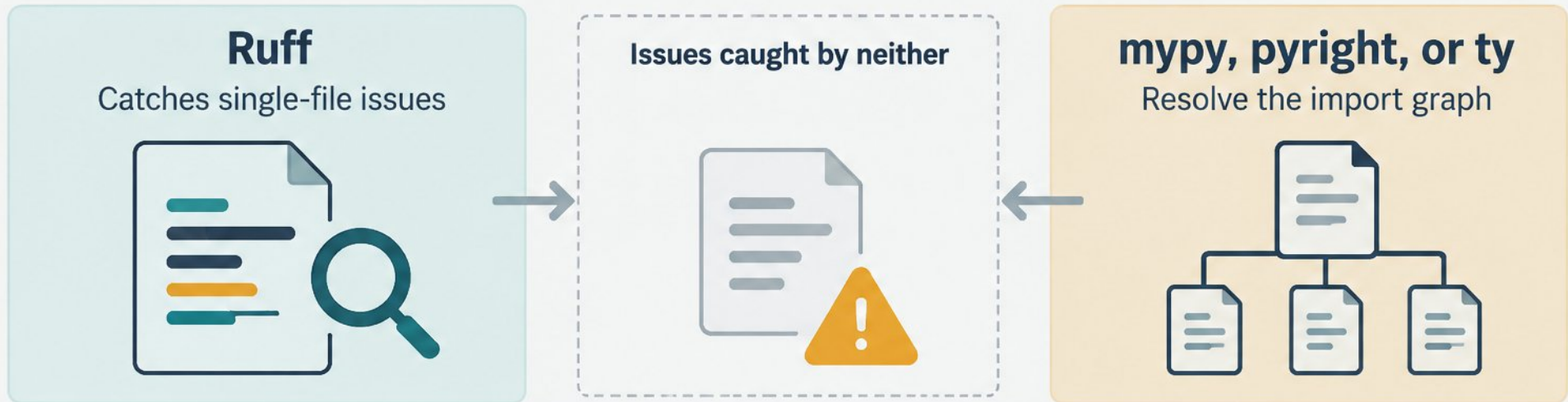
Import boundaries sane: src vs flat layout, no accidental namespace packages, no top-level import side effects

8



Environment reproducible: committed lock file, documented setup, no .venv, build outputs, or secrets

Code Quality, Style, and Static Analysis Signals



- 1 Separate automated enforcement from semantic review: different tools, different questions
- 2 Ruff consolidates flake8, isort, pyupgrade, pydocstyle, and Black-compatible formatting
- 3 Review tool config: selected rule families, pinned target-version, and suppression justification
- 4 Ruff catches single-file issues; type checkers resolve the import graph
- 5 Pick one primary type checker: mypy, pyright, or ty
- 6 Assess naming, function size, cohesion, duplication, error handling, mutable defaults



Automated enforcement

Different tools, different questions



Semantic review

Human judgment

Testing, Coverage, and Real Quality Gates



Assess the test mix: unit, integration, e2e, property-based, regression



Judge quality beyond percentages: assertions, edge cases, failure paths



Mock only at true boundaries; prefer parametrized tests over duplicates



Coverage is a floor: enable branch coverage and read term-missing



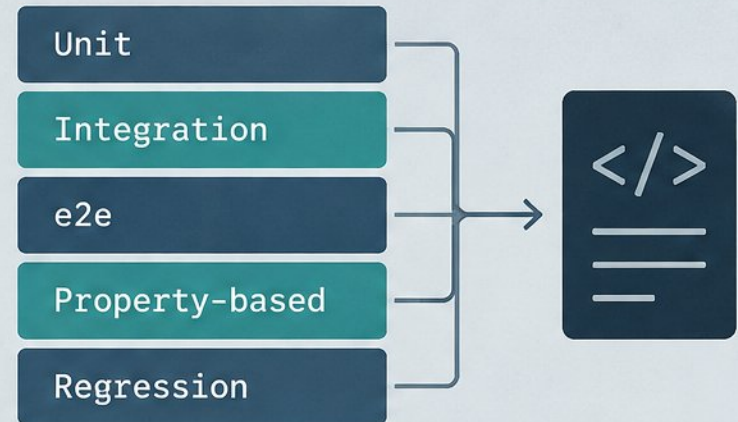
Ratchet the total, require diff coverage, and combine parallel data



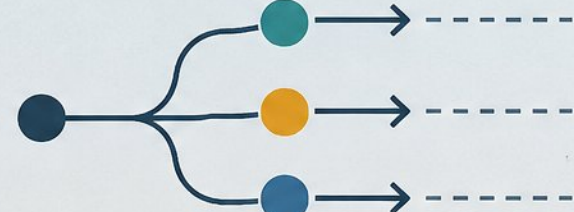
Verify the gate blocks: one stable required check protects the branch



Treat flaky tests as defects: quarantine and fix on a deadline



Branch Coverage



Quality Gate



BETTER TESTS
SAFER CHANGES

HIGHER CONFIDENCE
STRONGER SOFTWARE

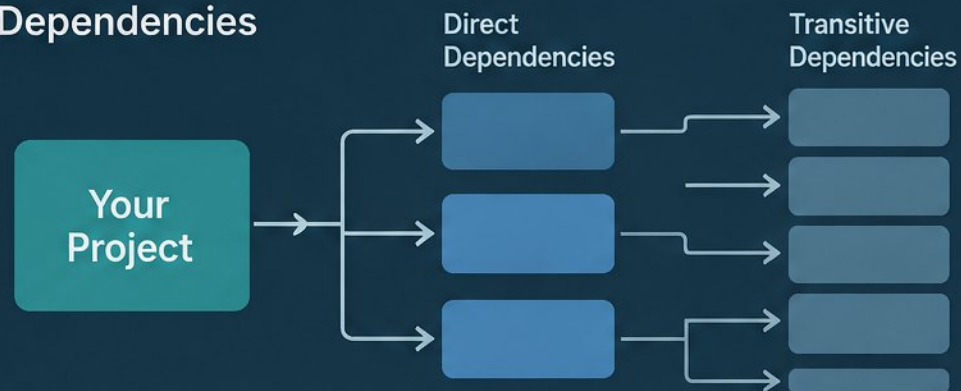
Dependencies, Security, and Supply Chain Review

SAFER PACKAGES. STRONGER SOFTWARE.

REVIEW | IDENTIFY | MITIGATE

- 1 Inventory direct and transitive deps; flag unused, duplicate, abandoned, unjustified additions
- 2 Lock file with exact versions and hashes, committed and enforced in CI
- 3 pip-audit against the resolved set: HIGH/CRITICAL block merge
- 4 pip-audit misses malicious packages, bundled native code, system packages, reachability
- 5 Secrets from env vars or a secrets manager; .env gitignored, .env.example present
- 6 OWASP Top 10:2025 ranks software supply chain failures in the top three
- 7 Durable controls: hash-pinned locks, private mirrors, build-time SBOMs, OIDC trusted publishing

Dependencies



Findings Severity Ladder



Stronger Supply Chains



SECURE DEPENDENCIES. BUILD WITH CONFIDENCE.

PEOPLE | PROCESS | PRACTICE

Architecture, Design, and Maintainability Assessment

CLEANER SYSTEMS
STRONGER TOMORROW



- Map modules and responsibilities: cohesion, coupling, clear boundaries



- Verify dependency direction: domain logic never depends on infrastructure



- Spot architectural smells: cycles, god components, utils dumping grounds



- Write layering rules as build-checkable contracts, not prose notes

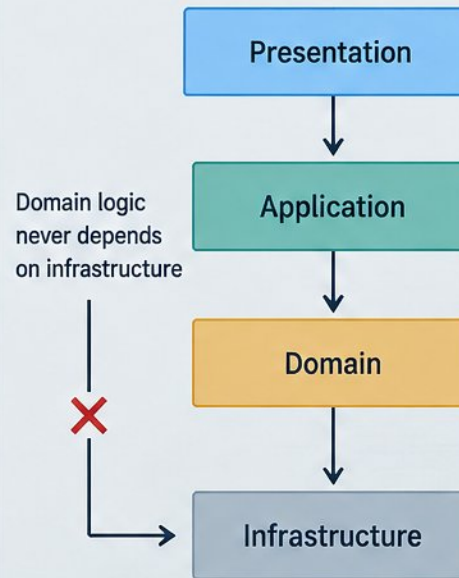


- Judge change cost: how many files does one fix touch?

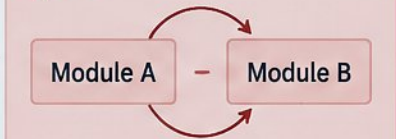


- Classify tech debt as intentional, accidental, or decay

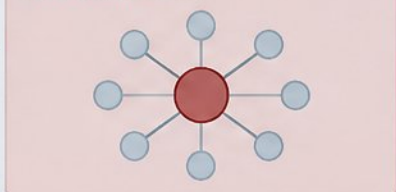
MODULE MAP (EXAMPLE)



Cycles



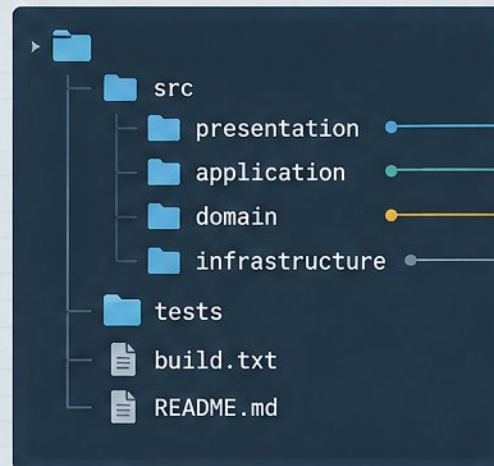
God Component



Utils Dumping Ground



REPOSITORY ANATOMY (EXAMPLE)



- User interface and delivery
- Application services and orchestration
- Core business logic
- External integrations and technical concerns

BETTER ARCHITECTURE
BUILDS BRIGHTER PRODUCTS

Documentation, Onboarding, and Developer Experience

CLEAR DOCS
SMOOTHER CONTRIBUTIONS
STRONGER SOFTWARE



README: purpose, Python version, install, run, tests, env vars



Verify install commands work from a fresh clone — fastest reproducibility signal



Docstrings on public APIs: purpose, args, returns, raises — one consistent style



Docs tooling: Sphinx for API-first, MkDocs/Material for Markdown-first prose



Blank API pages usually mean the package was not installed in the docs build



CONTRIBUTING, templates, and a curated Keep a Changelog with SemVer



Hunt doc drift: test examples with doctest, not by counting pages



Performance, Reliability, and Operational Readiness

BUILD
MEASURE
OPERATE
IMPROVE



Require measurement, not intuition:
no optimization claim ships without a before/after number or profile



Profile with the right tool:
cProfile for sync, yappi for async,
py-spy in production, pytest-benchmark
for regressions



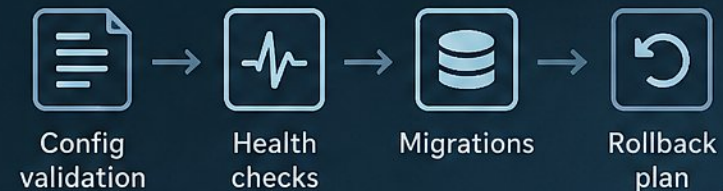
Never block the event loop;
asyncio for I/O, ProcessPoolExecutor
for CPU, threads only to wrap
blocking libraries



Reason about shared state explicitly:
no check-then-act across an await,
use atomic operations or transactions



Operational readiness:
config validation, health checks,
migrations, rollback plan — calibrated
to project type



Review Workflow: Evidence, Severity, and Communication

CLEAR EVIDENCE. CONSTRUCTIVE FEEDBACK. BETTER CODE.



Checklist ledger: verdict PASS, FAIL, or N/A with file:line evidence



Documentation is intent; code, tests, config, and CI are proof



Label findings: blocking, major, minor, nit, question



State issue, impact, and fix; ask when uncertain



Two rounds on a thread, then a call; write outcomes back



Verify fixes in a follow-up pass, never on assurance alone

Review Comment Example

Observed Pattern

```
if (user != null) {  
    return user.getName();  
}
```

Severity
major

Confidence
high

Impact

Possible null pointer at runtime, which can cause a failure.

Suggested Fix

```
return (user != null) ? user.getName() : null;
```

Severity Ladder

blocking

major

minor

nit

question

Reviewing AI-Generated and High-Volume Diffs

- 1 Reviewing an agent's code: same duty, more volume, different defect mix
- 2 You cannot read every line — review risk-first, outside-in, by code flow
- 3 Let linters, strict mypy, security scanners, and coverage clear mechanical noise
- 4 Hunt recurring AI failure modes: missed edge cases, swallowed errors, invented imports
- 5 Flag stale constructs: `datetime.utcnow()`, `typing.List`, `Optional`, deprecated APIs
- 6 Verify rather than assume: check every import, then run the project from scratch
- 7 Tighten gates, not loosen them; record significant decisions for the next reader



Adapting the Review to Context and Depth

SAME PRINCIPLES. DIFFERENT CONTEXTS. RIGHT DEPTH.



Libraries

- Packaging metadata
- API stability
- Typed markers
- Docs depth



Services

- Reproducibility
- Config validation
- Migrations
- Observability
- Rollback



Release candidate



Teaching projects

- README
- Clean setup
- Passing tests
- Safe secrets

- Scale rigor to target: library, service, release candidate, or teaching project
- Match depth to stakes: orientation pass, docs-vs-code audit, full readiness review
- Never claim production readiness without evidence for setup, deploy, security, rollback

REVIEW DEPTHS



Smoke



Maintainability



Full readiness



Never claim production readiness without evidence for setup, deploy, security, rollback.

Capstone: End-to-End Review Walkthrough and Scoring Rubric

REVIEW / ASSESS / EVIDENCE / IMPROVE / RELEASE

A HIGHER STANDARD
FOR BETTER SOFTWARE

- 1 Apply one consolidated rubric across all review dimensions
- 2 Score each dimension with a verdict: PASS, FAIL, or N/A
- 3 Cite file:line or command output as evidence for every score
- 4 Separate release blockers from improvement opportunities
- 5 Produce a concise report with prioritized closure plan
- 6 Re-run gates and confirm original findings no longer reproduce

REVIEW RUBRIC (SINGLE, CONSOLIDATED VIEW)

REVIEW DIMENSION	VERDICT	EVIDENCE (FILE:LINE OR COMMAND OUTPUT)
 Packaging	PASS	src/build.sh:12
 Code Quality	FAIL	src/core.py:87
 Tests	PASS	tests/test_core.py:34
 Dependencies and Security	N/A	pip-audit
 Architecture	PASS	docs/architecture.md:56
 Documentation	FAIL	docs/usage.md:22
 Operations	PASS	scripts/deploy.sh:18
 Release Mechanics	N/A	Makefile:9



PersonWise.

PersonWise • AI digital-human course platform



Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/186ab08b/public