

Documenting Decisions in Python Programming



Capture the reasoning behind significant Python design choices, not just code



Python's flexibility means many valid designs exist; code alone hides the why



For developers, tech leads, maintainers, and mentors inheriting long-lived code



Roadmap: vocabulary, formats, writing, code linkage, tooling, workflow, pitfalls



Outcomes: capture rationale, link decisions to code, sustain a lightweight log



Why Python Projects Lose Rationale

- Many idiomatic designs fit one problem; the choice is a decision, not a default
- Dynamic imports, plugins, and runtime flexibility hide intent
- Team turnover erodes oral history faster than code changes
- Refactoring blind to context reverses deliberate trade-offs
- Docstrings say what code does; decision records say why
- Audit, compliance, and onboarding demand traceable rationale

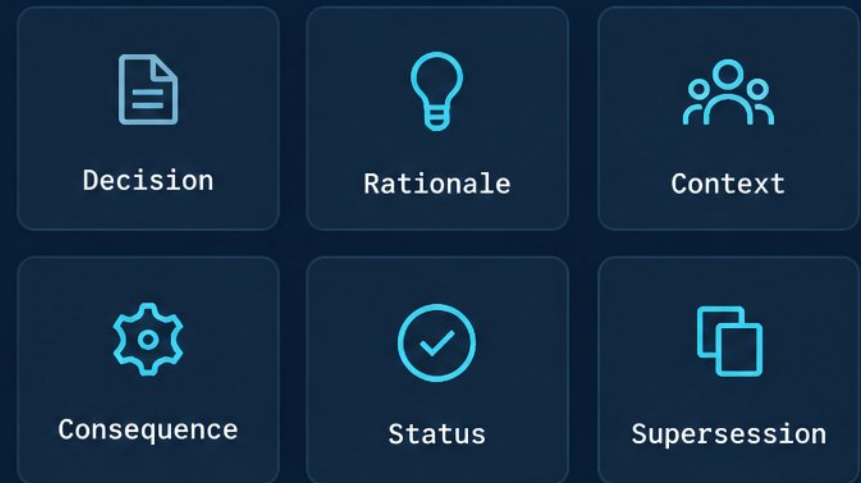


Core Concepts and Vocabulary

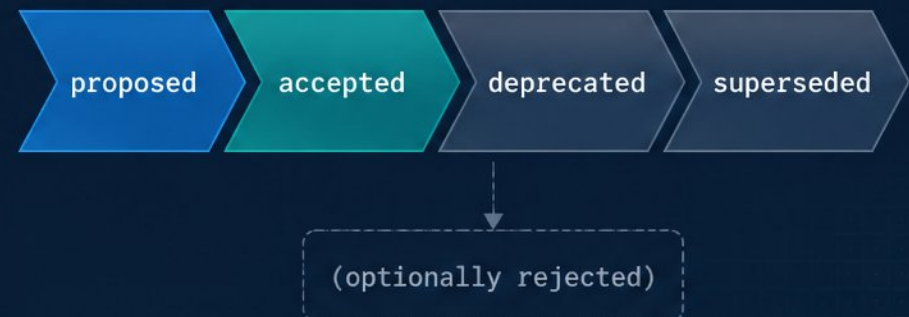
CLEARER DECISIONS
STRONGER SYSTEMS

- Decision, rationale, context, consequence, status, supersession: the working vocabulary
- An ADR is one short, dated, immutable record of a significant decision
- The decision log is the collection of those records
- ADRs record decisions after the fact: design docs describe what was built
- RFCs propose and gather input before a decision lands
- Status lifecycle: proposed, accepted, deprecated, superseded (optionally rejected)
- Accepted records stay immutable; update additively, supersede rather than rewrite
- Issues, PRs, and commits supply evidence, but are not durable decision records

THE WORKING VOCABULARY



STATUS LIFECYCLE



Choosing the Right Format and Granularity

SAME DECISIONS. DIFFERENT LEVELS OF DETAIL.



Y-statement



Minimal MADR



Full MADR



Nygard

- Match format to impact: Y-statement, minimal MADR, full MADR, or Nygard
- Y-statement: one sentence capturing context, problem, decision, goal, trade-off
- Nygard's five parts — title, status, context, decision, consequences — are easiest to start
- MADR adds decision drivers, options with pros and cons, outcome, confirmation
- Write when hard to reverse or cross-team; skip version bumps and formatter debates
- Guard against over-documentation — two paragraphs often beat four pages

Writing Decision Records That Stay Useful

CLEAR RECORDS
BETTER DECISIONS



Lead with context and forces:
constraints, quality attributes, pressures



Record at least two options
and why each was rejected



List honest consequences:
easier, harder, riskier, more expensive



Use plain, neutral language —
no marketing adjectives



Link issues, benchmarks, or
prototypes; add a review trigger



Aim for 200–500 words,
readable in one review sitting



Worked Examples: Python-Specific Decisions

Realistic choices. Clear trade-offs. Better software.

Dependency choice



- Dependency choice: stdlib vs. third-party, with lock-in and supply-chain costs

Async patterns



- Async patterns: sync vs. async I/O; threadpool vs. asyncio trade-offs

Packaging



- Packaging: hub-and-internals layout, re-export hubs over direct imports

Type hints



- Type hints: gradual vs. strict typing, plus migration cost for each

Test strategy



- Test strategy: coverage targets, fixtures, pytest plugins vs. custom harnesses

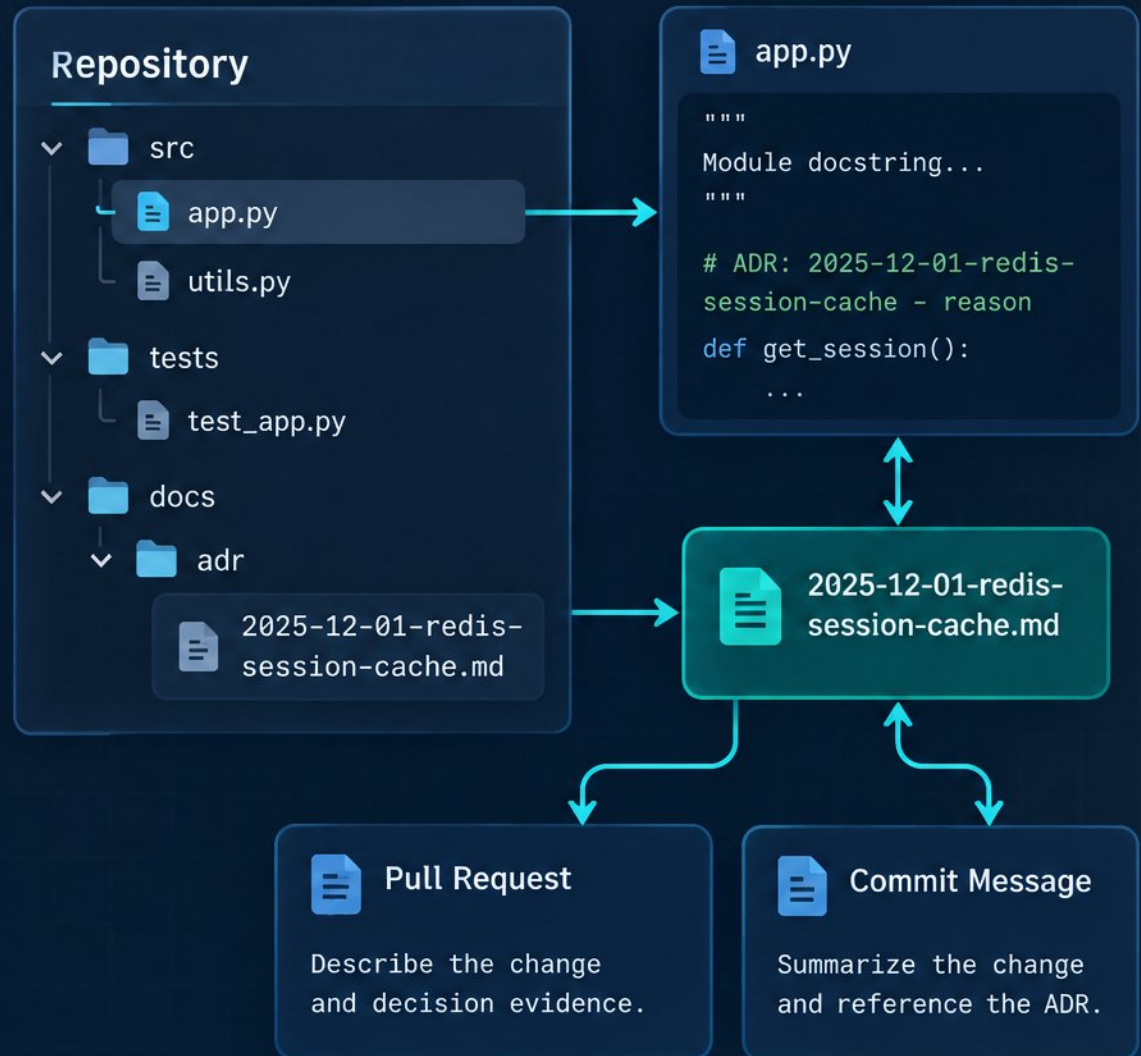


Each record shows context, alternatives, decision, and accepted trade-offs.

Embedding Decisions in Python Code and Repositories

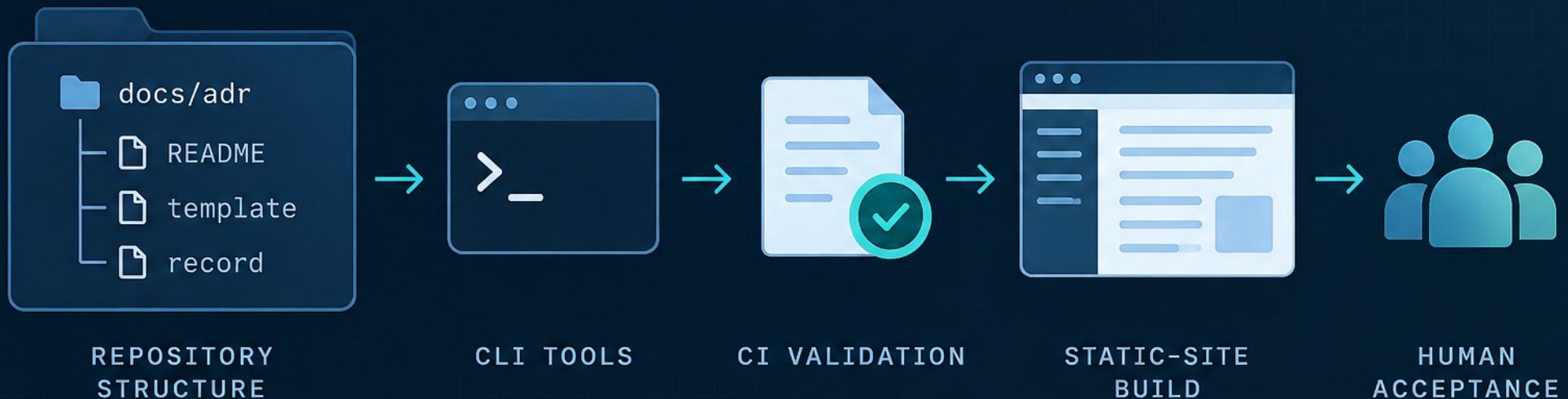
CODE ↔ CONTEXT ↔ LONGER-TERM CLARITY

- Docstrings, module headers, and targeted inline comments anchor rationale
- Convention: ``# ADR: 2025-12-01-redis-session-cache - reason``
- Commit messages and PR descriptions supply decision evidence
- Bidirectional traceability: code cites the ADR, ADR lists implementing files
- Store records next to source under ``docs/adr/``, versioned with the code
- Cross-package decisions: one authoritative record plus local notes



Tooling and Automation

- Start minimal: numbered docs/adr directory, template, and a README convention
- CLI tools scaffold, number, index, and wire supersession links
- Prefer zero-dependency, standard-library Python tools
- CI validates required sections and broken supersession links
- Static-site builds make the log browsable by non-engineers
- AI can draft records; humans own acceptance and trade-offs

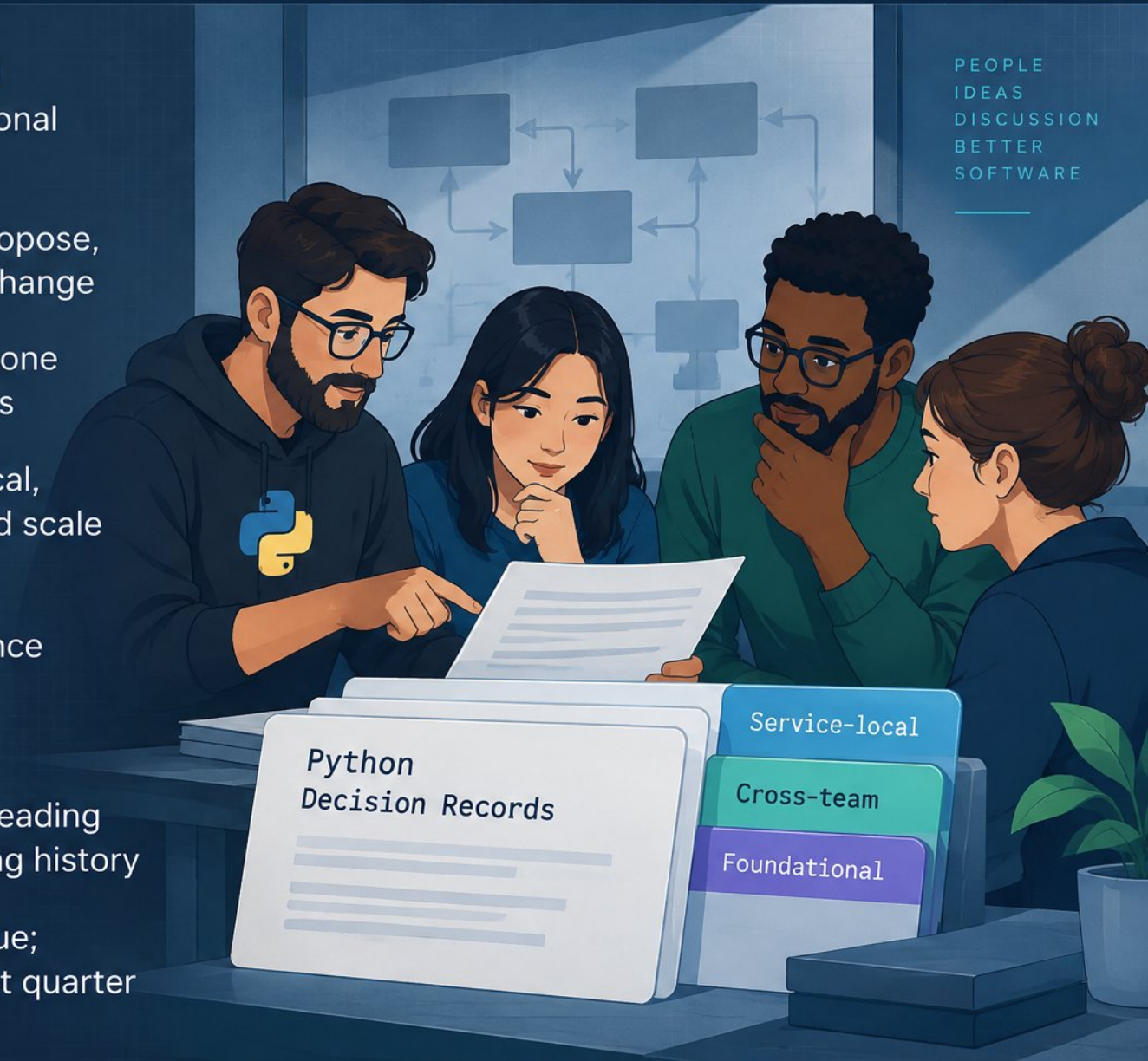


Team Practices, Review, and Culture

BETTER
DECISIONS
STRONGER
TEAMS

- Make decision records a normal design-review step, not an optional follow-up
- Default to PR-based review: propose, discuss inline, merge with the change
- Assign one decision owner: anyone may write, one person maintains
- Classify by scope — service-local, cross-team, foundational — and scale review depth
- Prefer consent-based governance (merge unless blocked) over consensus-required
- Reuse the log for onboarding: reading key records beats reconstructing history
- Watch for documentation fatigue; most practices die after the first quarter

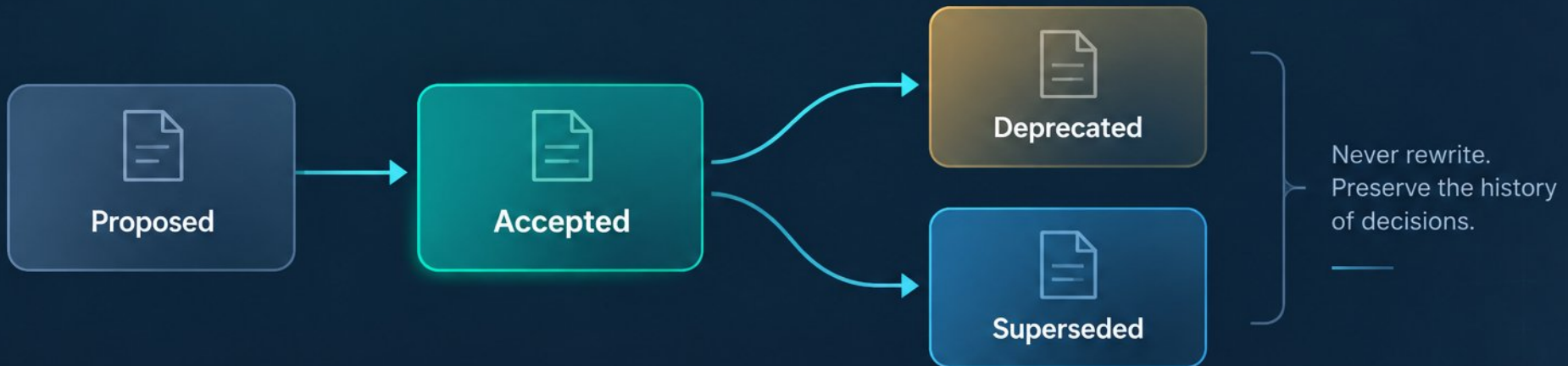
PEOPLE
IDEAS
DISCUSSION
BETTER
SOFTWARE



Maintenance, Supersession, and Long-Term Stewardship

CLEAR RECORDS. STEADY PROGRESS.

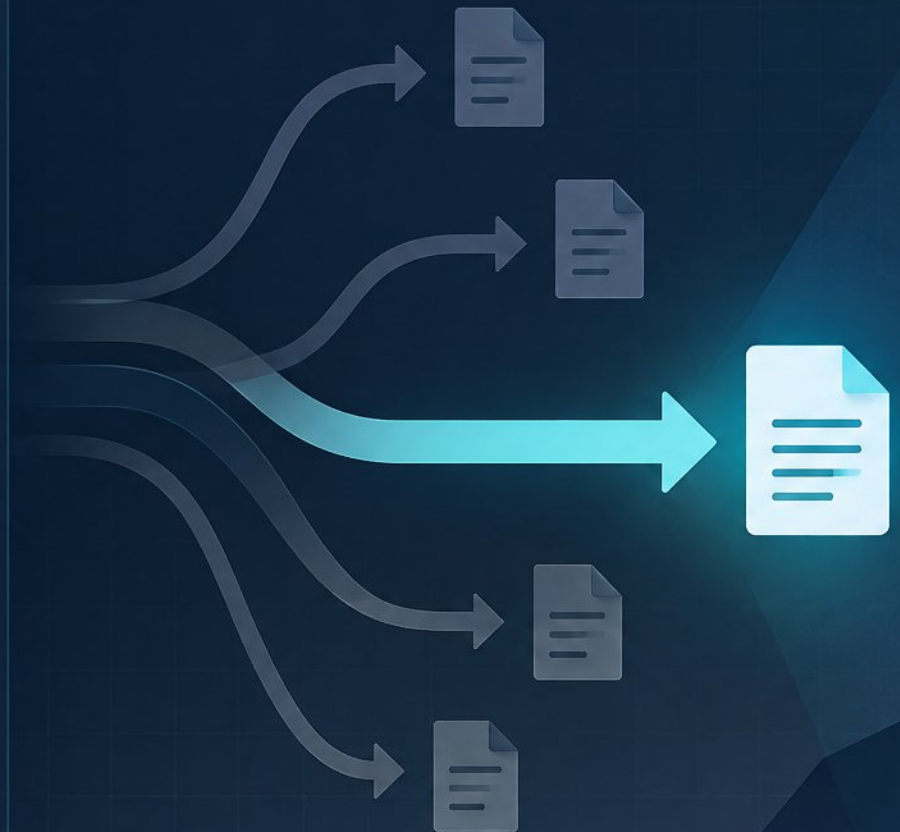
DECISIONS
EVOLVE.
CONTEXT ENDURES.



- Keep statuses honest: accepted, deprecated, or superseded — say what is currently true
- Supersede, don't rewrite: new record, bidirectional links, status flipped in one PR
- Deprecate when the context dissolved; supersede when a specific new decision replaces it
- Chain of supersession preserves evolution; never delete an accepted record
- Review quarterly, plus after migrations, incidents, or vendor changes
- Reconciling drift beats hiding it: verify implementation matches the recorded state

Common Pitfalls and Anti-Patterns

- Writing records after the fact turns reasoning into justification
- Essays instead of records conflate multiple decisions; split them
- Skipping alternatives and negative consequences hides the real trade-offs
- Quietly editing accepted records destroys the audit trail
- Records buried in wikis go unread; keep them in the repo
- Unlinked records stay invisible exactly when needed
- Rotten statuses mislead newcomers; avoid treating records as approval gates
- Trivial records pile up while load-bearing decisions go unrecorded



Hands-On Exercise: Write or Improve a Decision Record

- 1 Pick a real Python design choice a newcomer would question
- 2 Draft context, alternatives, decision, consequences, review trigger
- 3 Add a Y-statement summary line for PRs and stand-ups
- 4 Reference the ADR in the module docstring or inline comment
- 5 Peer review: concrete alternatives, honest trade-offs, word budget
- 6 Commit to docs/adr and agree on the next record to capture



Adoption Roadmap and Recap

From the first decision record to lasting impact

DECISIONS
CLARITY
PROGRESS



- Start small: one record for the next significant decision in docs/adr, linked from the pull request
- Don't backfill history; prioritize load-bearing decisions and current architectural seams
- Pick the lightest format, adding structure only when a real gap appears
- Pair the log with one current-state view so history stays distinct from reality
- Use records in onboarding, design review, postmortems, and migrations
- Measure time-to-onboard and avoided re-litigation, not document count



Recap: capture rationale, link to code, review like code, supersede rather than rewrite

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/7f793878/public