ADRs and Design Docs

Version MADR 4.0 · verified 2026-11-02

When a decision needs a record, the Nygard and MADR templates, status lifecycle, one-page design docs, review flow, and pointing AGENTS.md at them.

Contents

Mental model 2

When a decision needs a record 2

The Nygard template 3

Status lifecycle 5

MADR: making options explicit 7

The one-page design doc 8

Review flow 10

Where records live, and AGENTS.md 11

Keeping records honest 12

Further reading 13

Mental model

A decision record exists for the person who arrives a year later and asks “why on earth did we do it this way?” — often an AI agent about to undo it. The choice itself is visible in the code; what is lost is the context, the forces, and the options you rejected and why. Write those down while they are cheap to remember, keep the record immutable once accepted, and supersede it when you change your mind.

When a decision needs a record

Signal	Example
Hard or costly to reverse	Database, message broker, cloud
Shapes quality attributes	Sync vs async, caching, tenancy
Crosses team boundaries	API style, auth model
Rejected a popular option	“Why not Kubernetes?”
Keeps being re-litigated	Monorepo vs polyrepo

adr.github.io defines an architectural decision as a justified choice that addresses an *architecturally significant* requirement — one with a measurable effect on the system's architecture and quality. Library upgrades, naming and formatting rules do not qualify; a linter config or a style guide covers them better.

```
Write an ADR if ANY is true:
  - reversing it takes more than a sprint
  - another team must change because of it
  - someone will ask "why not X?" next year
```

> **Tip:** The Y-statement is a one-sentence ADR for small decisions: "In the context of X, facing Y, we decided for Z to achieve Q, accepting D."

The Nygard template

Section	Contains
Title	Short noun phrase, numbered

Section	Contains
Status	proposed, accepted, superseded, …
Context	Forces at play: technical, political, team
Decision	What we will do
Consequences	What gets easier, what gets harder

Michael Nygard's 2011 post popularised this shape, and its brevity is the point: one page, written in an afternoon. Context states facts and constraints without arguing for the answer. Consequences must list the costs too — a record with only upsides is marketing.

```
# 7. Use PostgreSQL for order storage

Date: 2026-11-02

## Status
Accepted
```

```
## Context
Orders need multi-row transactions; the
team already runs PostgreSQL for billing.

## Decision
We will store orders in PostgreSQL.
```

> **Gotcha:** The Consequences section is the one people skip, and it is the one that tells the next reader which pain was expected and which is a bug. Never ship an ADR without it.

Status lifecycle

Status	Meaning
Proposed	Open for review, not binding
Accepted	In force; content is now frozen
Rejected	Considered and declined — keep it

Status	Meaning
Deprecated	No longer applies, nothing replaces it
Superseded by ADR-N	Replaced; link both ways

Once accepted, an ADR is a historical record: you do not edit its decision, you write a new ADR that supersedes it and update the old one's status line only. Never reuse a number, so links and agent references stay valid. `adr-tools` automates the supersede link.

```
adr init doc/adr
adr new Use PostgreSQL for order storage
# later, reverse decision 7:
adr new -s 7 Move orders to DynamoDB
# ADR 7 status now points to the new ADR
```

MADR: making options explicit

MADR section	Adds over Nygard
Decision Drivers	Criteria you judged against
Considered Options	Every option, by title
Decision Outcome	Chosen option, because …
Pros and Cons of the Options	Why the others lost
Confirmation	How compliance is checked

MADR (Markdown Architectural Decision Records) treats the considered options with their pros and cons as essential, because that is the part a future reader needs to avoid re-running the evaluation. It ships full and minimal templates; metadata such as status, date and decision-makers goes in YAML front matter.

```
---
status: accepted
```

```
date: 2026-11-02
decision-makers: [dana, lee]
---
# Use PostgreSQL for order storage

## Considered Options
* PostgreSQL
* DynamoDB

## Decision Outcome
Chosen option: "PostgreSQL", because ...
```

The one-page design doc

Section	Purpose
Context and scope	What exists, what problem
Goals and non-goals	What success is; what is out
Proposed design	Approach, APIs, data model
Alternatives considered	Options and why not

Section	Purpose
Risks and open questions	What you do not know yet

An ADR records one decision; a design doc plans a piece of work that may contain several. Write one before building something that takes weeks, touches several teams, or has a cheaper option someone will propose. Each significant decision inside it graduates to its own ADR once accepted, so the design doc can go stale without losing the why.

```
# Design: Order export to the data lake

## Goals
- Daily export of orders, < 1 h lag
## Non-goals
- Real-time analytics (see ADR-14)

## Open questions
- Who owns the schema contract?
```

> **Gotcha:** Non-goals do more work than goals. Without them, review turns into a list of features you never meant to build.

Review flow

Step	Mechanism
Draft	Branch with ADR in `Proposed`
Review	Pull request; named deciders
Decide	Merge with status `Accepted`
Announce	Link in the team channel

Reviewing ADRs as pull requests puts the discussion next to the text, in history, searchable forever. Name who decides up front — MADR's `decision-makers`, `consulted` and `informed` fields exist for this — and timebox the review so "proposed" does not become a permanent state.

```
git switch -c adr/0012-orders-dynamodb
adr new -s 7 Move orders to DynamoDB
git add doc/adr
git commit -m "ADR 12: move orders to DynamoDB"
gh pr create --title "ADR 12 (proposed)"
```

Where records live, and AGENTS.md

Location	Holds
`doc/adr/` (adr-tools default)	Numbered ADRs
`docs/design/`	Design docs
`AGENTS.md` at repo root	Pointer for coding agents
Nested `AGENTS.md`	Per-package overrides

Keep records in the repository they govern, versioned with the code, not in a wiki that drifts. AGENTS.md is an open format — a README for coding agents — and agents read the nearest one in the directory tree. Point it at the decision log so an agent checks settled decisions before proposing to reverse one.

```
## Architecture decisions
- Read doc/adr/ before changing storage,
  messaging or auth. Accepted ADRs are binding.
- To change one, write a new ADR that
  supersedes it; never edit an accepted ADR.
```

Keeping records honest

Anti-pattern	Instead
Retroactive ADR, written to justify	Write it before or during deciding
ADR as changelog	One decision per record
Editing an accepted ADR	Supersede it
No rejected options listed	List at least two real options
20-page “ADR”	Split; move detail to a design doc

A decision log is only trusted if it is current. Check for ADRs that code no longer follows during reviews, and mark them deprecated or superseded rather than

deleting them — the history of what you used to believe is part of the context.

```
Quarterly decision-log check
  [ ] Any accepted ADR the code contradicts?
  [ ] Any "Proposed" older than 30 days?
  [ ] Any decision made this quarter
      without a record?
```

Further reading

- Documenting Architecture Decisions — Michael Nygard
- adr.github.io
- MADR templates
- adr-tools
- AGENTS.md