what is an

ADR

and what do we use them for?

An ADR is:

- an *A*rchitecture *D*ecision *R*ecord
- "a collection of architecturally significant decisions"

We use ADRs for:

- Just kidding, we don't.
- But what *could* we use them for?

Five Truths about Software Development

1. The docs are terrible;
2. Nobody actually reads the docs;
3. Code is rarely self-documenting;
4. It's never clear *why* code exists;
5. The docs are still terrible.

RFCs are Great

- A venue for proposing a change;
- A means to collect input on a proposed change;
- A way to propose multiple ideas.

RFCs are also Terrible

- RFCs are not docs!
- RFCs are proposals for a change + comments;
- RFCs never get finished;
- RFCs are usually huge.

Huge docs are also Terrible

- Large documents are hard to write
- Large documents are never read
- Large documents are never updated
- Large documents are usually out of date

How do you *make* architectural decisions?

1. Create an RFC?
2. Somebody takes the general consensus?
3. Someone else implements it?
4. Done?

How do you *record* architectural decisions?

1. "lol, the RFC is good enough!"

How do you *follow* a architectural decisions?

Some other developer has made a decision.

You can either:

1. Blindly accept the decision;
2. Blindly change the decision.
3. Wander through outdated docs, old emails and chat archives for a while, then choose one of the above.

There must be a better way...

A is for *Architecture*

- This is for a change to your architecture only:
  - "We are switching to Python 3 from Python 2"
  - "We are replacing our logging system with a new service"
  - "We are going to build a new tool to frobnicate the biz baz"

A is for *Architecture*

- This is not for:
  - "We are going to start using tabs instead of spaces"
  - "We are going to stop writing huge RFCs"
  - "We are going to roll standups, backlog estimation, review and retro into one big four-hour long meeting"

A is for *A*rchitecture

- These are not necessarily client-facing at all;
- ...but they might be;
- Usually to solve a problem with or improve some aspect of the architecture.

D is for *Decision*

- This is the *decision* made as the result of an RFC:
  - "We are going to replace this closed-sourced service with our own open-source implementation"
  - "After doing some sophisticated analysis, we've decided that Perl 6 is 120% more than Perl 5"

R is for *Record*

- These are meant to be historical documents!
- Especially for disparate teams:
  - "Why does the mobile team use React Native and not something else?"
- They will *never change* once created;
- But can be superseded.

How-To ADR

1. `git init`
2. Make a directory, maybe `./docs/adrs/`
3. Create your first ADR! `adr-001.md`
4. It will have six sections:

How-To ADR (sections)

- Title
- TL;DR
- Context
- Decision
- Consequences
- Status

How-To ADR: Title

- The title should be short and to the point:
- Examples from before:
  - "ADR-001: Adopting Python 3"
  - "ADR-123: Replacing logging with LogFlume"
  - "ADR-101: Building a Frobnicater"

How-To ADR: TL;DR

- Even though our ADR should be short, we should give a summary:
  - "We decided to upgrade to Python 3 to support Unicode better"
- Should be super short and concise.

How-To ADR: Context

- This describes the state of the system when the decision was made;
  - "We are currently using Python 2 everywhere"
- Other external forces that come into play
  - "The client wants to start sending us Unicode characters"
- Should eventually become out of date, but still be contextually relevant!

How-To ADR: Decision

- Full sentences
- Active voice
  - "We will migrate to Python 3 for services X, Y and Z"

How-To ADR: Consequences

- All consequences (not just the benefits);
- What could go wrong;
- Likely to become the context for the next ADR!

How-To ADR: Status

- One of:
  - Proposed;
  - Accepted;
  - Deprecated;
  - Superseded;
- This is the only time when an ADR should be modified after creation.

ADRs ♥ Agile

- Not all decisions can be made at once!
- We need an iterative process for making decisions;
- ADRs can build on each other.

Taking it a step further: APOs

- In the Perl community, an "APO" stands for:

APOCALYPSE!!

"What I will be revealing in these columns will be the design of Perl 6. Or more accurately, the beginnings of that design, since the design process will certainly continue after I've had my initial say in the matter. I'm not omniscient, rumors to the contrary notwithstanding. This job of playing God is a little too big for me. Nevertheless, someone has to do it, so I'll try my best to fake it. And I'll expect all of you to help me out with the process of creating history. We all have to do our bit with free will."

— Larry Wall, Apocalypse 1

http://perl6.org/archive/doc/apocalypse.html

Thanks!