Sustainable Architectural Decision Making

ADR all the things!

The Problem

Have you ever joined a new team or project, looked at the code base, and were like

Have you ever joined a new team or project, looked at the code base, and were like

Tracking why decisions were made during the software lifecycle is difficult.

Teams change.

Requirements change.

Technology changes.

Facing a decision without knowing its original context, you can either

A. Uncritically accept it.

B. Uncritically change it.

Either option has its drawbacks.

The one sensible way to mitigate them is via documentation.

"What? Documentation? But we're $AGILE!"

Being `$AGILE` doesn't mean being against documentation.

It means being against valueless documentation.

The larger the documentation, the more likely it is to become out of date.

Getting out of date means losing value.

Small documentation has a chance to be kept up to date.

It has the chance to become sustainable.

Sustainable Design Decisions

To achieve sustainably documented architectures, we need sustainable design decisions.

Sustainable Architectural Design Decisions [6] defines the following five criteria for sustainable design decisions:

1. Strategic
2. Measurable and Manageable
3. Achievable and Realistic
4. Rooted in Requirements
5. Timeless

1. Strategic

To evaluate the strategic consequences you should consider things such as the decisions' long-term impact.

2. Measurable and Manageable

You should be able to measure and evaluate a decision's outcome over time according to objective criteria.

In order to be manageable, you must limit the decisions' granularity to a certain level of detail.

3. Achievable and Realistic

The rationale for fitting the solution to the problem should be chosen pragmatically and made explicit.

Make sure you are neither over- nor under-engineering.

4. Rooted in Requirements

Decision making should be grounded in domain-specific architecting experience and context.

Make sure you take into account your company's environment, project's constraints, and your team's skill level.

5. Timeless

Decisions should be based on experience and knowledge that won't likely be soon outdated.

When in doubt, choose boring technology.

Now we know how what it means for our decisions to be sustainable.

But how do we make sure our documentation is sustainable too?

Architecture Decision Records (ADRs)

ADRs are typically small text files, each describing a single design decision and rationale.

ADRs can be templated and commonly include three parts: context, decision, and consequences.

Context

The technical, business, social, or political circumstances that directly influence a design decision.

Description

A brief description of the decision itself, outlining the selected course of action for the design.

Consequences

The expected outcomes that result once the decision is applied.

This should include whether and how the decision (or its implementation) can be reversed.

ADR Templates

Nygard (aka "the OG template")

1. Title
2. Context
3. Decision
4. Status
5. Consequences

Y-Statements

1. context
2. facing
3. we decided
4. and neglected
5. to achieve
6. accepting that

Y-Statements (Example)

In the context of the Web shop service, facing the need to keep user session data consistent and current across shop instances, we decided for the Database Session State pattern and against Client Session State or Server Session State to achieve data consistency, accepting that a session database needs to be designed and implemented.

Tooling

adr-tools[1]

"A command-line tool for working with a log of Architecture Decision Records (ADRs)."

It is based on the *Nygard* template.

[1] https://github.com/npryce/adr-tools

Create an ADR directory in the root of your project:

```
$ adr init doc/architecture/decisions
```

Create a new architecture decision record:

```
$ adr new "Implement in PHP as part of the monolith"
```

If a decision is reversed, it should still be kept around. It is still relevant to know that and why it was the decision in the past.

To create a new ADR that supercedes decision `$ID`, use the `-s` option:

```
$ adr new -s "$ID" "Implement as Go micro service"
```

A Definition of Done

We now know that and how to document our architectural decisions in a sustainable way using ADRs.

But when can a decision be considered done?

We need a Definition of Done for ADRs.

Let me propose the following acceptance criteria[2] phrased as questions:

[2] Taken from A Definition of Done for Architectural Decision Making.

1. Are we confident enough that this design will work?

2. Have we decided between at least two options, and compared them (semi-) systematically?

3. Have we discussed among each other just enough and come to a common view?

4. Have we captured the decision outcome and shared the decision record?

5. Do we know when to realize, review and possibly revise this decision?

Conclusion

ADRs enable developers to actively take part in sustainable architectural decision making.

1. They are stored close to the relevant code.
2. They capture context and consequences.
3. They keep the documentation short and relevant.

ADRs' greatest strength is their low barrier to entry. Since anyone on the team can write an ADR, everyone who wants can fill the role of software architect. That anyone can write ADRs creates an opportunity to grow software architects over time.

— Michael Keeling in *Love Unrequited* [7]

Further reading

[1] A DoD for Architectural Decision Making
[2] adr-tools
[4] Architectural Decisions — The Making Of
[5] Documenting Architecture Decisions
[6] Sustainable Architectural Design Decisions
[7] Love Unrequited
[8] Y-Statements

Thank you!