EVERYTHING YOU NEED TO START USING ADRs.

What is an ADR?

An ADR (Architecture Decision Record) is a document that captures a decision, as it is said in the presentation, but it also includes the context of how this decision was made, how did we reach that decision, and the possible consequences of adopting the decision. As a consequence, at the end we will have a list of those decisions that are significant, I mean, those that affect the structure, dependencies, interfaces, etc.

Developers and stakeholders are able to see ADRs. To sum up, everyone will be able to see these records, to avoid misunderstoods when trying to understand the motivation for a determined record.

It helps to discover the motivation behind certain decisions, as we will keep track of those topics needed in order to achieve that decision.

Benefits of using Architecture Decision Records.

1) They are especially useful for capturing long-term intentions.
    a) As all the team will be able to see at every time what the key challenges are, and the evolution of our architecture. And we can also document rejected solutions and explain why they couldn't be beneficial for achieving our goals.
2) They are small and modular, which makes the maintenance of a system easier.
    a) They could also be called Lightweight Architecture Decision Records. The main reason is that we try to isolate questions into small pieces, something like dividing EPICs into user stories in an Agile Methodology.
3) As they are records for significant questions, the quantity of valueless information will be minimum.
    a) In these records, we will only cover the aspects that really matter when we are declaring one, for instance, the context, the scope… forgetting about every unessential point, and providing a great approach for knowledge sharing and transparency.

When should we write ADRs?

- Every time we make an architecturally significant decision: When we start planning how to develop a project, we will always have to make decisions that have a large impact on all the topics that an ADR covers. For instance, when we need at a certain point to introduce a new API or library that involves important changes, an ADR is needed.
- When the solution of a problem is not documented yet. Saving the solution for a problem could save our life's many times. We do not know when a problem could appear, and keeping track of how to solve a question, is always worthwhile.

Alejandro Aguirre López
Alberto Díez Bajo
Juan Rodríguez de la Fuente

- When we have a problem and a solution, but it involves changes (or not). Even though a lot of questions that have no impact could remain undocumented, but an accumulation of unsignificant (referring to the architectural aspect) could make a mountain out of a molehill, that will be very complex to solve in the future.

Taking decision while writing an ADR

If a decision is reversed, we will keep the old ADR, but mark it as superseded. It is still relevant to know that it **was** the decision but is **no longer** the decision.

Now we are going to see the main structure of an ADR according to Michael Nygard:

- Title: Short noun phrases.
- Context: This section describes the forces at play, including technological, political, social and project local. These forces are probably in tension, and should be called out as such. The language in this section is value-neutral. It is simply describing facts.
- Decision: This section describes our response to these forces, the chosen option
- Status: Specifies the state of this decision (proposed, accepted, rejected or superseded)
- Consequences: This section describes the resulting context, after applying the decision. Both positive and negative consequences

The final size of the document should be about one or two pages.

We will write each ADR as if it is a conversation with a future developer, so we should apply a good writing style, with full sentences organized into paragraphs.

Where should ADRs be stored?

ADRs are part of the project documentation, so they can be stored in the same directory as the rest of the documentation. As they are just text files, we can also keep them in a Version Control System (VCS), like git, along with the project source code. This way we can follow the same workflow as the rest of the project, with version history, pull requests, issues, etc… Additionally, they can be kept synchronized with the project, and the commit history can provide us an overview of the context in which the ADR was made.

Tooling

Due to their simple text file nature, you can just use any text editor by creating the files and editing them. But there exist programs to automate tasks like creating ADRs following a template, updating their status, superseding previous decisions, linking decisions or generating a table of contents. One such program is adr-tools, that follows Michael Nygard's proposed format, which we explained in a previous slide.

The ADR format is not a fixed standard, so there exist different templates apart from Nygard's one, like MADR, which specifically uses Markdown. Or the one for business cases, which includes cost analysis for each option.

Alejandro Aguirre López
Alberto Díez Bajo
Juan Rodríguez de la Fuente