ADRs: the lost art of documenting important architectural decisions

Marek Dominiak

Marek Dominiak

I'm running training in EventStorming, Architecture, DDD, and TDD

17 years of professional experience

Co-owner and instructor at trainitek

Hands-on Architect and Team Lead at MINTRA

CTO at Sparkbyte Solutions at SPARKBYTE SOLUTIONS

Why to document architecture?

- Because people come and go but the code and the architecture stays.
- For effective communication (e.g. onboarding new team members).
- Curious developers can get answers faster.
- Sustainability of the business.

A story about S3

An innocent question from CTO:

“Marek, could you tell me why in 2018 when moving our services to AWS, we stayed on Elastic File System storage for quite some time instead of moving to S3 storage immediately? It was 10 times more expensive to be on EFS.”

A story about S3

- I started archaeological process of gathering information from:
  - Long Emails threads
  - Confluence documentation
  - Commits history across some projects
  - Jira tickets and comments
  - Coworkers

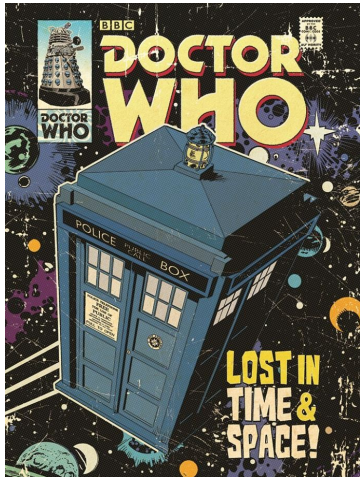
Confluence

GitLab

Jira

A story about S3

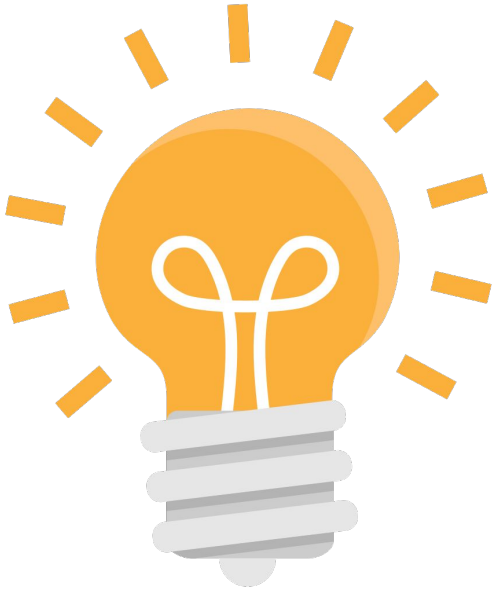
- …and after just **six** hours of



- being lost in time and space

- I successfully recovered context of the decision and found the answer!

Wouldn’t it be easier to have this decision recorded?



Architecture Decision Records

*“An architecture decision record (ADR) is a document that captures an important architectural decision made along with its **context** and **consequences**.”*

Ref 1: First mention of the ADR - https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions
Ref 2: Definition from https://github.com/joelparkerhenderson/architecture-decision-record?tab=readme-ov-file#what-is-an-architecture-decision-record
Note 1: ADR are sometimes referred to Architectural Decision Records

An example of an ADR

1. APP - Pick Unit Testing Framework - Spock

Date: 2024-02-03

Status: Accepted

Context: The project requires a unit testing library that not only enhances code readability but is also intuitive and widely adopted in the industry. Our development team primarily works with Java and frequently encounters issues with the excessive boilerplate code required for testing various use-cases across features. Ensuring efficient testability is a key concern in our ongoing development efforts.

Decision: After evaluating several options, we have selected Spock for our unit testing framework. While Spock is known for its slightly higher learning curve compared to JUnit and TestNG, it offers superior readability and supports data-driven testing effectively (test tables). Spock's integration capabilities with Groovy and its good support community were decisive factors.

…

An example of an ADR

Consequences:

Positive:

- Test tables - support for data-driven tests, which will allow for more comprehensive and maintainable test cases.
- Increased developer engagement due to the preference for using Spock, which can lead to more thorough and creative test scenarios.
- Improved maintainability of test suites due to Spock's readable syntax.

Negative:

- Initial productivity may drop as approximately 30% of the team ramps up their proficiency in Groovy.
- Additional resources will be required to train team members and integrate Spock into our existing Java projects.

Other Possible Solutions:

- **JUnit**: While highly popular and familiar to the team, JUnit was not selected due to its limited support for data-driven tests and less flexible test coding options.
- **TestNG**: Considered less popular and with fewer resources available compared to JUnit and Spock, TestNG was also passed over due to similar limitations in data-driven testing capabilities.

ADR Template

ADR:

- **title**: what is the decision about?
- **date**: when it was taken
- **status**: is the decision proposed, accepted, superseded
- **context**: what are the known architectural drivers?
- **decision**: choice and it’s justification
- **consequences**: known consequences of the decision
- **stakeholders**: group of people responsible for the decision
- **other possible solutions**: why they weren’t chosen
- **investigation:** how we ended up with the decision
- **your own section**: in case you or your team finds it useful

What kind of Decisions should be documented?

- Decisions that are not easily reversible.
- Decisions that are not obvious.
- Decisions that can help us answer the question “Why …?” in the future .
- Decisions that can be used in onboarding.
- Decisions that bother us at night (or in a shower).

What should we **always** include in an ADR?

- "***Architectural drivers*** *are* ***considerations*** *that need to be made for the software system that are architecturally significant.*" - Joseph Ingeno
  - Functional requirements
  - Design objectives
  - Quality attributes
  - Constraints (internal, external)

Refs: Software Architect's Handbook and Software Architecture for Developers- learn about categories of different Architectural

A closer look at an ADR

1. APP - Pick Unit Testing Framework - Spock

Date: 2024-02-03

Status: Accepted

Design / project objective - long term sustainability

Project constraints: Team Skills and Knowledge

Context: The project requires a unit testing library that not only enhances code readability but is also intuitive and widely adopted in the industry. Our development team primarily works with Java and frequently encounters issues with the excessive boilerplate code required for testing various use-cases across features. Ensuring efficient testability is a key concern in our ongoing development efforts.

Quality attributes: increased Testability and Maintainability

Decision: After evaluating several options, we have selected Spock for our unit testing framework. While Spock is known for its slightly higher learning curve compared to JUnit and TestNG, it offers superior readability and supports test tables effectively. Spock's integration capabilities with Groovy and its good support community were decisive factors.

A closer look at an ADR

Consequences:

Positive:

- Enhanced support for data-driven tests, which will allow for more comprehensive and maintainable test cases.
- Increased developer engagement due to the preference for using Spock, which can lead to more thorough and creative test scenarios.
- Improved maintainability of test suites due to Spock's readable syntax.

Negative:

Include both positive and negative consequences

- Initial productivity may dip as approximately 30% of the team ramps up their proficiency in Groovy.
- Additional resources will be required to train team members and integrate Spock into our existing Java projects.

Other Possible Solutions:

- **JUnit**: While highly popular and familiar to the team, JUnit was not selected due to its limited support for data-driven tests and less flexible test coding options.
- **TestNG**: Considered less popular and with fewer resources available compared to JUnit and Spock, TestNG was also passed over due to similar limitations in data-driven testing capabilities.

“Context is king”

“*The journey is more important than the destination”*

How to store ADRs? Architecture Decision Log - ADL

- ADL is just a log of ADRs
- The whole team has access to it
- It can be stored in:
  - Confluence:
    - One main page for all Decision Logs across system
      - One page for ADL per each project / system
      - General rules, system architecture, DevOps team, Service 1, Service2 etc.
    - “Decision page”, “DACI: Decision documentation” for ADR or create your own template
  - Git repository:
    - Keep as simple text files e.g. using https://github.com/npryce/adr-tools (.md files)
    - You can visualise ADRs using Structurizr
    - Keep in mind this can be challenging if you have many repositories
  - Ultimately the choice, is yours (discuss with your team)

Some examples of ADR titles

- Logback is used as a Logging Framework

The title reveals the decision

- Adoption of Cloud Provider - AWS vs. Azure vs. On-Premise - Proposed

We can change the title once it's accepted or rejected

- Documentation language is Norwegian

Not all decisions are technical!

Takeaways 1

- **Clarity is Key:** Ensure all **drivers** and necessary **context** are included

- **ADR Length:** Aim for 1-1.5 pages

- **Managing Length:** Rather than adding more information to one ADR, consider splitting the ADR:
  - E.g. **investigation** part at the end of the document or as an appendix
  - Creating a separate ADR

Takeaways 2

- Add a **periodical "task" in your calendar** to review ADRs and if we need to update / add new decisions

- To speed up the process of writing ADR **you can leverage AI**, e.g. ChatGPT

- Don't be afraid of adding adding new fields like **'Issue Link'** if your team finds it useful

Reminder on why to use ADRs in the team

- ADRs help us to make better decisions (peer-reviewed decisions).
- ADRs are great tool for effective collaboration.
- ADRs are very simple to start with.
- ADRs save a lot of time in the future.
- ADRs can be very quick to write using support of AI tools.
- The effort spent on writing ADRs yields great value.
- Just start using writing them!

References

- Documenting Architecture Decisions , Michael Nygard
- ADR: https://adr.github.io/ - Lots of information
- Software Architect's Handbook, Joseph Ingeno
- Software Architecture for Developers, Simon Brown
- Design IT, Michael Keeling
- Fundamentals of Software Architecture: An Engineering Approach - Mark Richards, Neal Ford
- Facilitating Software Architecture, Andrew Harmel-Law (release in 2025)

Contact / Slides

Slides