RFC: Lightweight Documentation

Status	Proposed
Author(s)	Martín San Juan (martin.sanjuan@trafilea.com)
Updated	2024-02-11

What do we understand as documentation?

It's a common misunderstanding to think in a very large document full of detailed information difficult to read or even remember, but we are used to cascade-style documentation, where these types of documents are needed because all the decisions are taken at the beginning of a project, but in an agile context we have a lot of documentation we are not using or we are rejecting even if the agile platform emphasize their usage.

> ***Agile practices encourage the creation of just enough documentation to meet the needs of the team and project***

We have different opportunities to create pieces of software documentation in all the processes of development, we need to recognize those points and take advantage of them.

Aside from Acceptance Criteria, User stories, or task definitions, which should be mandatory documentation on the development process, I like to describe some other opportunities and tools we have to create small, but valuable pieces of documentation. I'll leave out of the scope of this document the Sourcecode Comments due to their extension, but It's also a very important piece of documentation. I leave a different article to cover that in the appendix.

The Importance of Thoughtful Git Pull Request Descriptions

In the collaborative software development landscape, Git pull requests are crucial for proposing and incorporating changes into a codebase. While the mechanics of creating a pull request are well-known, the importance of including a detailed description of the "WHY" behind the proposed changes cannot be overstated.

1. Understanding the Context

A well-crafted pull request description provides context for reviewers and collaborators. It explains the reasoning behind the changes, offering a clear understanding of the problem being addressed or the feature being implemented. This context is invaluable for both current team members and future contributors who may need to revisit the codebase.

2. Facilitating Efficient Code Reviews

By including the "WHY" in your pull request description, you empower reviewers to focus on the essence of the changes. This, in turn, streamlines the review process, making it more efficient and reducing the likelihood of misunderstandings. It enables the team to provide more constructive feedback and catch potential issues early in the development cycle.

3. Documentation and Knowledge Transfer

Pull request descriptions become part of the project's historical record. They serve as a form of documentation, aiding in knowledge transfer across team members and ensuring that the rationale behind each change is preserved. This is particularly beneficial in larger teams or when onboarding new developers.

4. Accountability and Collaboration

Assigning a responsible individual or team to a pull request adds a layer of accountability. Knowing who is overseeing the changes encourages a sense of ownership and fosters collaboration. It becomes easier to communicate and coordinate efforts, leading to a more cohesive and synchronized development process.

5. Effective Project Management

In larger projects, tracking and managing changes can become complex. Associating pull requests with a responsible party and a relevant team helps in project management. It aids in prioritization, and resource allocation, and ensures that everyone is aware of who is working on what.

In conclusion, a Git pull request is not just a technical formality; it is a communication tool. Including a detailed description with the "WHY," and assigning responsibility and team involvement, elevates the pull request from a mere code submission to a collaborative and well-documented step in the software development lifecycle. It not only enhances the quality of the codebase but also contributes to a more efficient and harmonious team dynamic.

C4 Model to understand the context

C4 Model is a framework designed for visualizing the architecture of software systems. It was created by Simon Brown to address the need for a clear and concise way to communicate the architecture of complex systems. The C4 model provides a set of diagrams at different levels of abstraction to represent the static structure of a software system.

The main benefits of the C4 model include:

Benefits

1. Clarity

C4 diagrams are designed to be simple and easy to understand, making it easier for both technical and non-technical stakeholders to grasp the architecture of a system.

2. Consistency

The C4 model provides a consistent way to represent different levels of abstraction, ensuring that everyone involved in the project uses a common language to discuss the architecture.

3. Collaboration

C4 diagrams facilitate communication between different stakeholders, including developers, architects, project managers, and business stakeholders, fostering better collaboration and understanding.

4. Contextualization

The model helps to provide context by breaking down the system architecture into different levels, allowing stakeholders to focus on specific aspects of the system without getting overwhelmed.

Levels

The C4 model consists of **four levels**, but for the sake of lightweight, I'll focus on the first two, the other two contain a deep level of detail and are harder to keep updated, also are easily replaceable for other tools like UML.

Level 1: Context Diagram

- Provides a high-level view of the system within its environment.
- Illustrates how the system interacts with external entities (users, systems, etc.).
- Focuses on the boundaries of the system.

Level 2: Container Diagram

- Expands on the context diagram, breaking down the system into containers.
- Containers represent applications or services that execute code.
- Illustrates the relationships and interactions between containers.

ADR - Architectural Decision Record

An Architectural Decision Record (ADR) is a document that captures and communicates decisions made regarding the architecture of a software system. ADRs are used to record important architectural choices, the reasons behind them, and any associated trade-offs. They serve as a valuable tool for documenting and communicating key architectural decisions within a development team.

Benefits

I believe that ADR has a lot of benefits, but the most important for me are:

Documentation of Rationale

ADRs provide a clear and concise record of the reasons behind specific architectural decisions. This documentation helps future team members understand the context and rationale, facilitating knowledge transfer and maintaining institutional knowledge.

Communication and Transparency

ADRs enhance communication within a team by making architectural decisions transparent. Team members can refer to ADRs to understand the thought process behind certain choices, promoting a shared understanding of the system's architecture.

Decision Traceability

ADRs create a traceable history of architectural decisions over time. This traceability is beneficial for understanding the evolution of the system's architecture, especially when multiple teams or individuals contribute to the project.

Risk Mitigation

By documenting decisions and their associated trade-offs, ADRs help teams recognize potential risks and challenges. This allows for proactive risk management and enables the team to revisit decisions if new information or circumstances arise.

Facilitation of Onboarding

ADRs serve as a valuable resource for onboarding new team members. Newcomers can refer to ADRs to quickly grasp the key architectural decisions and the reasoning behind them, accelerating their integration into the project.

Templates

I prefer the approach of Michael Nygard as a template because It's straightforward and light, but currently, you have many flavors to choose from, just keep it consistent.

Regarding when you need to use an ADR, I think this article at engineering.atspotify.com nails it.

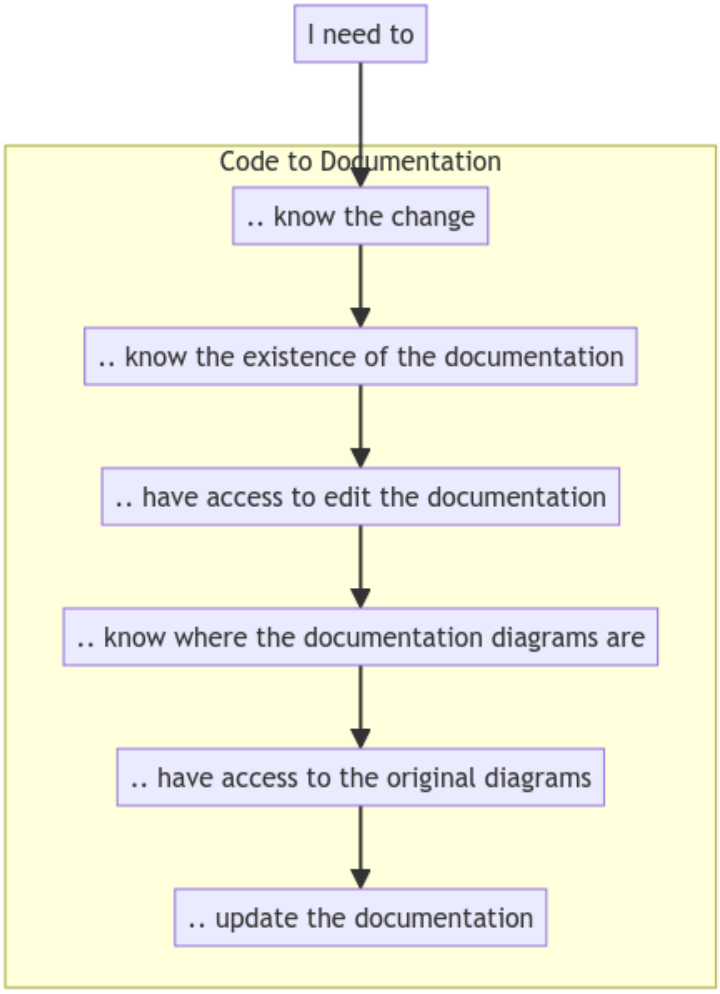
At the end of the document, I will share an example of how we are using this in the Altair project, and here the ADR we are using this.

```
# AWS SNS

## Status: Accepted

## Context

In the context of our AI-powered advertising campaign creation system, we encountered a specific
challenge related to the response time of ChatGPT. The nature of the information provided to ChatGPT,
including details about the product, target audience, and template structure, resulted in significantly
extended response times. The extensive prompt given to ChatGPT led to delays, with processing times
spanning multiple minutes.

## Decision

To address the challenge of prolonged response times from ChatGPT and enable a more responsive system,
we opted to incorporate Amazon Simple Notification Service (SNS) for asynchronous communication between
system components. SNS facilitates the decoupling of services and manages the asynchronous flow of
messages efficiently.

### Justification

#### Asynchronous Management

SNS provides an efficient and scalable publish/subscribe model, which is crucial for handling the
complexity of the asynchronous advertising campaign generation process.

#### Integration with AWS Stack

The company uses the entire AWS stack, and SNS seamlessly integrates with other services in the
platform, facilitating interoperability and cohesion within the ecosystem.

#### Team's Prior Experience

The development team has successfully implemented solutions using AWS SNS in previous projects,
expediting the implementation process and reducing the risk of errors.

## Consequences

### Pros

* Efficient decoupling of system components.
* Facilitates the management of asynchronous processes in the advertising campaign generation flow.
* Natural integration with the existing AWS environment.

### Cons

* Potential costs associated with SNS usage, depending on the volume of messages and subscriptions.
* SNS is designed for mass message publication to many subscribers, which may introduce overhead for a
system with only two elements.
```

How and Where

Often we need to create different diagrams to represent some complex flows inside our system, and we have a variety of different software to do that like Lucidchart, Draw.io, or Excalidraw. Those are great, and for someone who likes to have every box in a perfect line and with the exact amount of space between each other, are perfect, but they have some drawbacks that make us a little heavier for this scenario, for example, the most important ones:

- *Difficult to keep updated*: Each person on the team will use the one that suits them more, and probably export it as an image, and paste it into a document, and that's it. The original… lost. So, the most probable scenario will be that, if we need to update just one system inside the chart, you need to make all the diagrams from scratch.
- *Law of Demeter*: As the same as in software design, we need to reduce the distance between the implementation and the documentation. A common scenario could be the following:



The same weakness is shared by external documentation, like Google Docs or Confluence. You need to know that the documentation for that exists, you need to find it, and you need to have access to it.

- To reduce distance: Create documents on the same repository
- To reduce complexity: Use lightweight formats such as Markdown and Mermeid.
- To increase portability: Use widely supported formats such as Markdown and Mermeid.

To reduce several of these points I choose Markdown and Mermaid as my preferred tools. Markdown is a well-known syntax, but for Mermaid, not so much. You can visit the official documentation to know more about it.

Here are the reasons why use both of these tools:

Speed

Creating a new piece of documentation is easier than creating a file and writing a few lines in the right folder of our project. Marie Kondo said that to keep your drawer in order, It needs to be easy to store your clothes, not just to grab something. If you need to do a lot of things to create your documents, you certainly will not do it.

Everything is in the same place

To gain that speed, we need to remove actors that slow down the process. You are writing code, so documented in the same place. You can do that because it's just plain text.

GitHub & VS Studio Support

Both are supported in GitHub comments and descriptions, so it is really easy to add flows there as side documentation in our PRs. The same we could have in our formal documentation in our repository.

Adding retry logic to club membership subscriptions and refactor to failure flow #443

Edit | Code

Merged | juansef14 merged 7 commits into `develop` from `feature/subscriptions-api/retry-process-club-membership` on Nov 27, 2023

Conversation 14 | Commits 7 | Checks 5 | Files changed 10 | +541 −103

juansef14 commented on Nov 23, 2023 • edited by martin-sanjuan

Write | Preview

@martin-sanjuan PR for Adding retry process and sending Klaviyo comnunication to Hermes at 22 attempts of charging:
Changes in PR:

- [x] Adding retry delay due date process to club membership
- [x] Refactor in PaymentFailureFlow to modularize functionality
- [x] Added validation to send notification at 22 attempts of charging a club subscription
- [x] Test Cases coverage
- [x] Unit tests
- [x] Testing workflow -> PaymentFailureFlow with Payment Source **Subscriptions**

```mermaid
flowchart TD
  G[SQS Schedule Event] -->|Schedule Message| H(Execute)
  A[SNS Order Event] -->|Order Update| B(Post Execute)
  H --> C
  B --> C{Process}
  C -->|Success| D[SuccessFlow]
  C -->|Fail| E[FailureFlow]
  E --> F{PaymentSource}
  F --> |Orders| I[OrdersRetry]
  F --> |Subscriptions| J[SubsRetry]
```

Reviewers: martin-sanjuan
Assignees: No one—assign yourself
Labels: None yet
Projects: None yet
Milestone: No milestone
Development: Successfully merging this pull request may close these issues. None yet
Notifications: Customize | Unsubscribe
You're receiving notifications because you were

Adding retry logic to club membership subscriptions and refactor to failure flow #443

Edit | Code

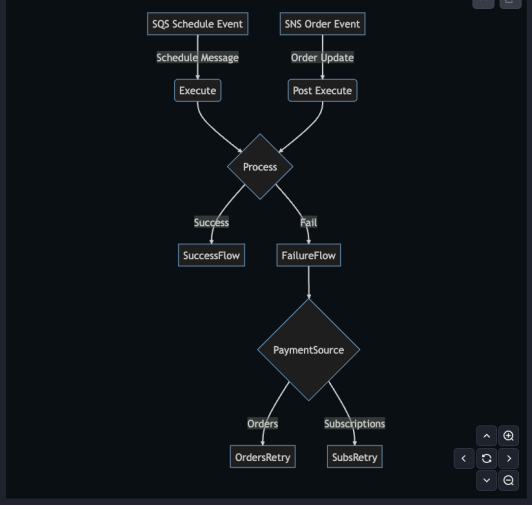
Merged | juansef14 merged 7 commits into `develop` from `feature/subscriptions-api/retry-process-club-membership` on Nov 27, 2023

Conversation 14 | Commits 7 | Checks 5 | Files changed 10 | +541 −103

juansef14 commented on Nov 23, 2023 • edited by martin-sanjuan

@martin-sanjuan PR for Adding retry process and sending Klaviyo comnunication to Hermes at 22 attempts of charging:
Changes in PR:

- [x] Adding retry delay due date process to club membership
- [x] Refactor in PaymentFailureFlow to modularize functionality
- [x] Added validation to send notification at 22 attempts of charging a club subscription
- [x] Test Cases coverage
- [x] Unit tests
- [x] Testing workflow -> PaymentFailureFlow with Payment Source **Subscriptions**



Reviewers: martin-sanjuan
Assignees: No one—assign yourself
Labels: None yet
Projects: None yet
Milestone: No milestone
Development: Successfully merging this pull request may close these issues. None yet
Notifications: Customize | Unsubscribe
You're receiving notifications because you were mentioned.

2 participants

Lock conversation

Portable & Easy to Update

Even if I use an image to embed it in a document, I add the mermaid code as an appendix, so If anyone needs to change it, they just update that code.

If you are working on your documentation on your repository as a markdown you don't need even to get an image version, because Visual Studio and Github will allow you to preview it.

Reduce the size of your modifications

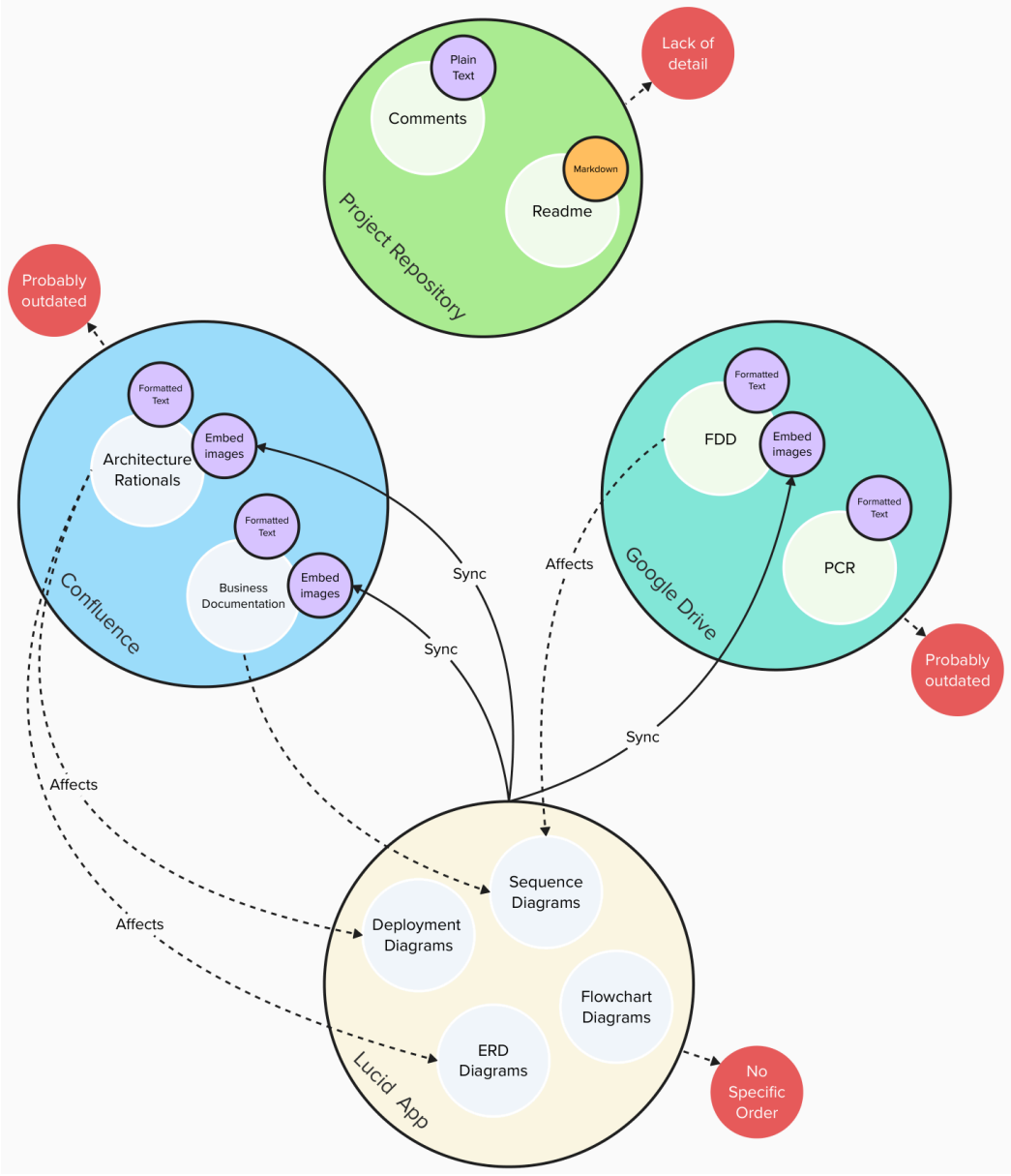
If you can create documentation about your changes in your PR, the one thing you need to update the "formal" documentation is to copy and paste it into the right folder in the form of a description, and ADR or a more elaborate schema in your repository you reduce the amount of work each person needs to do to keep your documentation updated and distribute the responsibility to do it naturally.

Summary

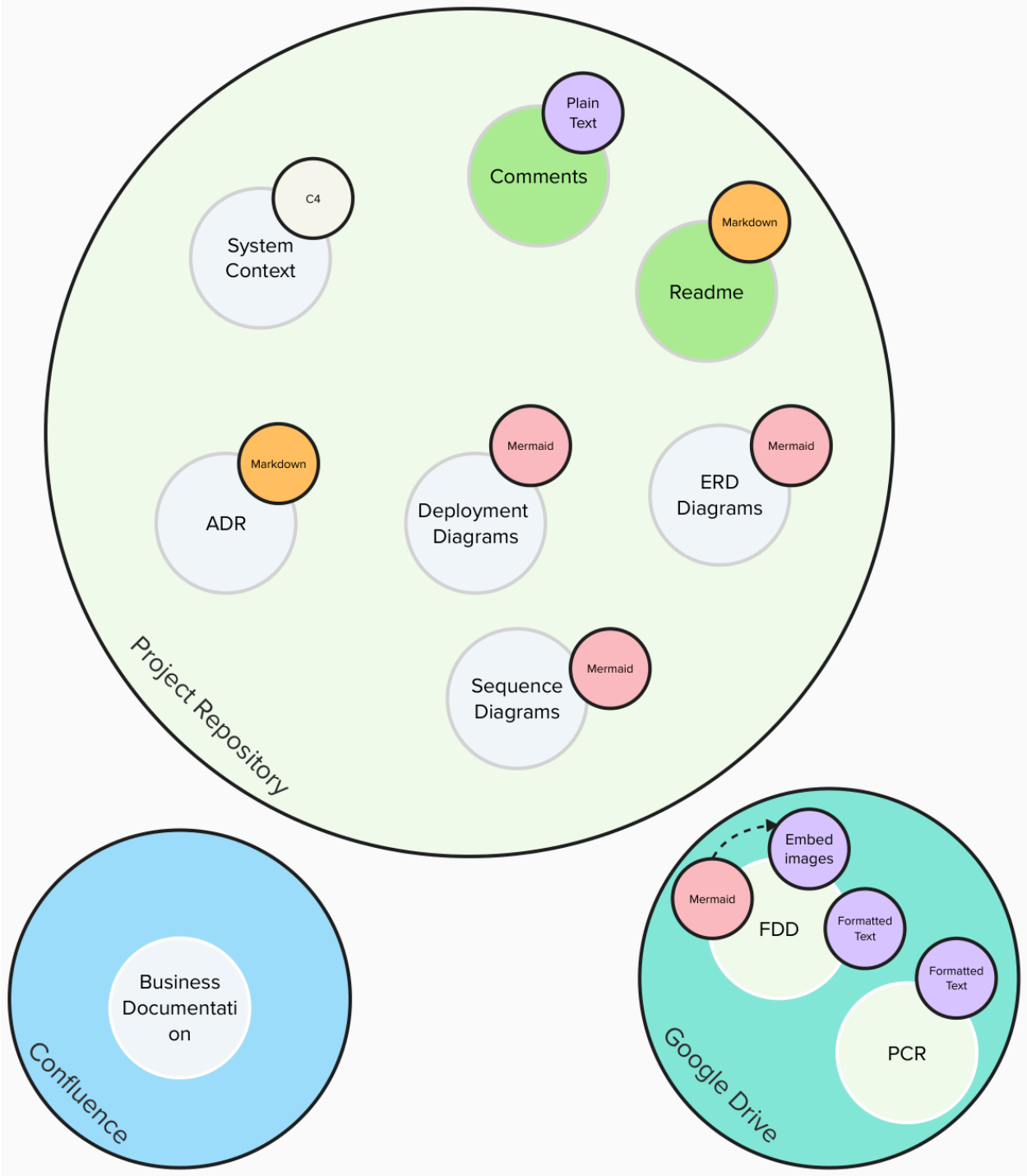
In a nutshell, we can create a lot of documentation inside the same repository with some tools, making it easier to produce and handle, and reducing the distance with the implementation.

This will help us keep the documentation up-to-date, make all the team responsible for generating small pieces of documentation on each step we make, create a history footprint, and make easier the onboarding of new team members and the transfer of knowledge.

Our goal should be moving from here:



To here


Project Repository
C4
System Context
Plain Text
Comments
Markdown
Readme
Markdown
ADR
Mermaid
Deployment Diagrams
Mermaid
ERD Diagrams
Mermaid
Sequence Diagrams
Confluence
Business Documentation
Google Drive
Embed images
Mermaid
FDD
Formatted Text
Formatted Text
PCR

Appendix

Articles

- **When Should I write an architecture decision**: Engineering at Spotify
- **Article about the use of comments**: The Grey Line | Part One & The Grey Line | Part Two

Mermaid Diagrams

Code to documentation Flow

```
flowchart TB
   A[I need to] --> A1
   subgraph Code to Documentation
   A1[ .. know the change]-->A2[.. know the existence of the documentation]
   A2 --> A3[.. have access to edit the documentation]
   A3 --> A4[.. know where the documentation diagrams are]
   A4 --> A5[.. have access to the original diagrams]
   A5 --> A6[.. update the documentation]
   end
```