Mobile Application Architecture Design

Best Practices for Scalable Mobile Applications

FO Product Center, GS Retail

1. Architecture Design
   - Clean Architecture
   - 3 Layers
   - Presentation Layer
   - Domain Layer
   - Data Layer
2. Test and Lint
   - Unit Test
   - Test Coverage
   - Lint
3. Implementation

1. Architecture Design and Best Practices

The main purpose of implementing the architecture is the **separation of concern (SoC)**.

If a new feature or requirement come to you, in how much change, you can adapt to that feature without breaking existing ones.

The Clean Architecture is powerful solutions that

multiple teams can work on,

independent data layers,

scalable for adding/removing features,

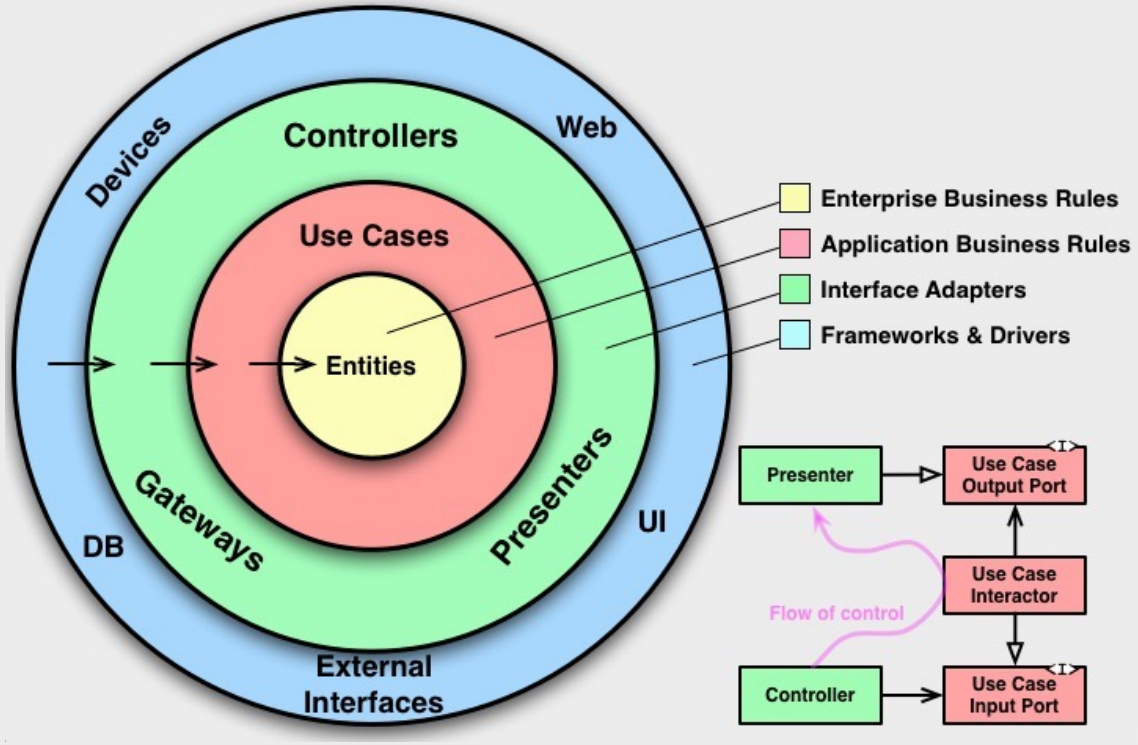
testable,

independent frameworks/tools,

and can be easily maintained at any time.

Clean Architecture

13 August 2012



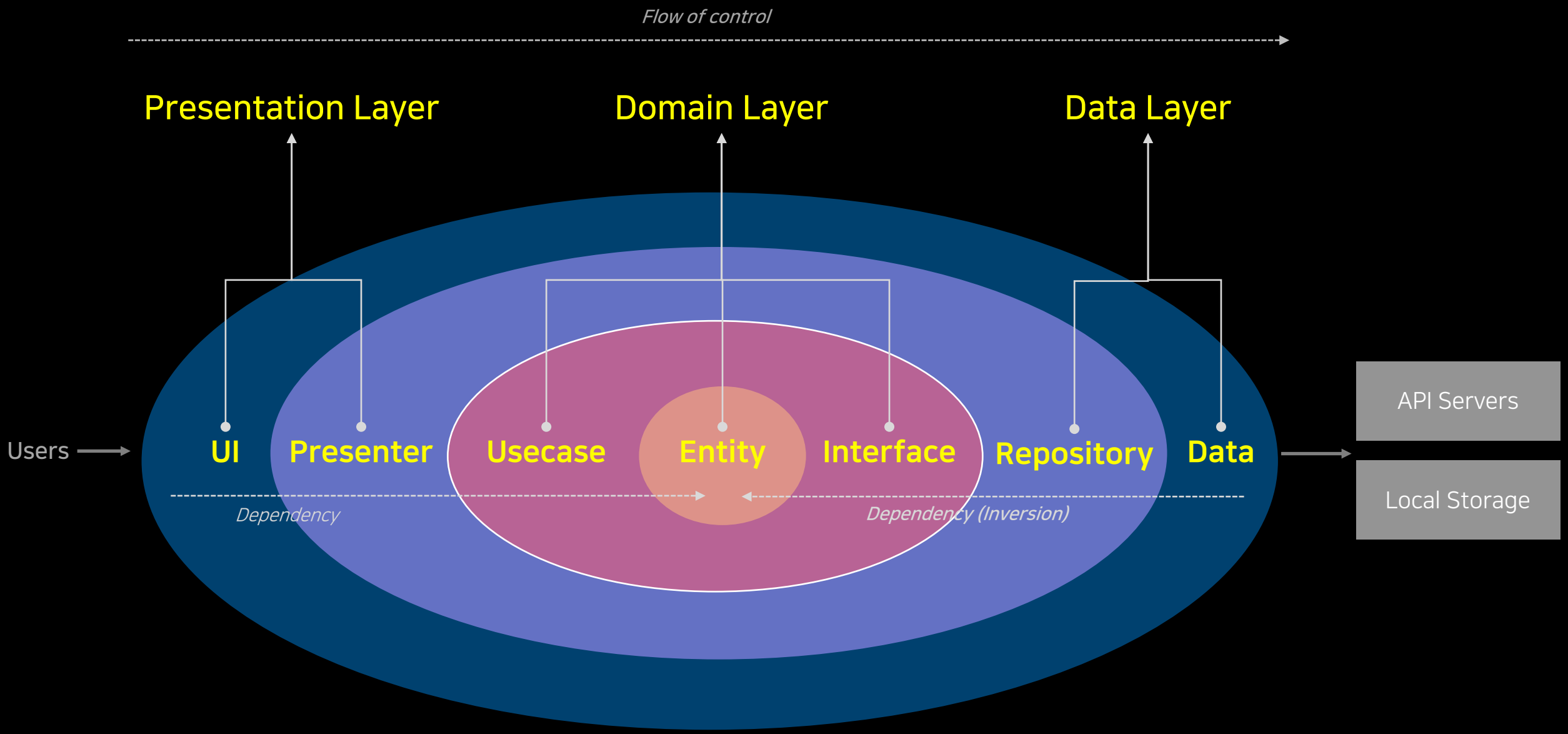
https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

But,

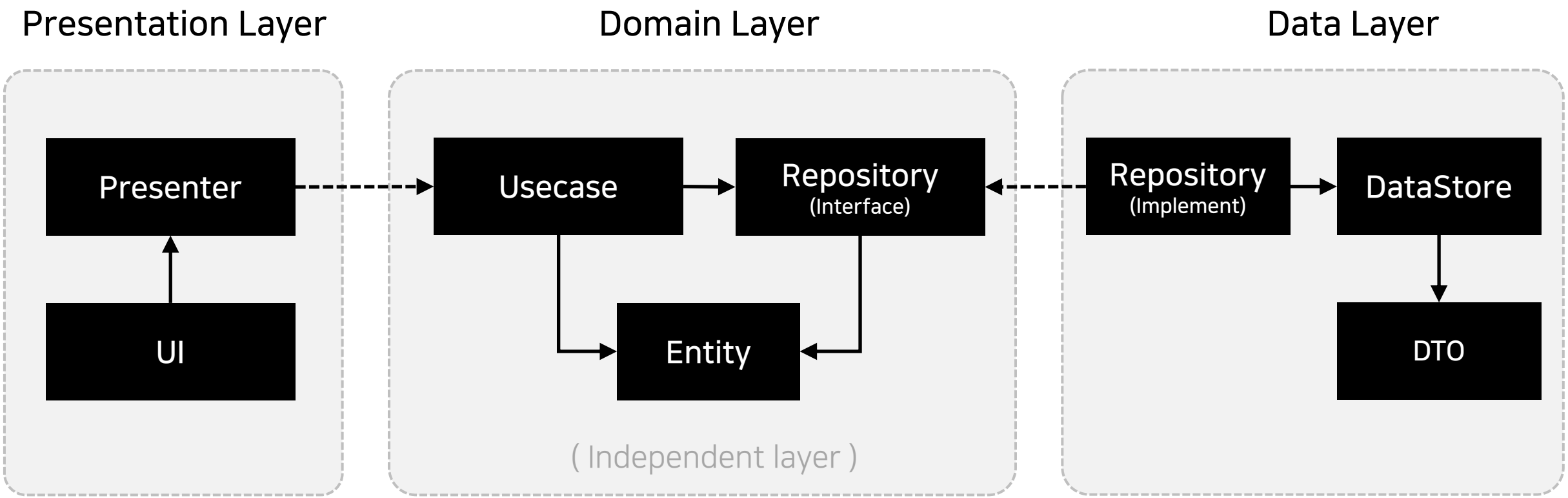
May not be suitable for all kind of projects.

Can be implemented in multiple ways.

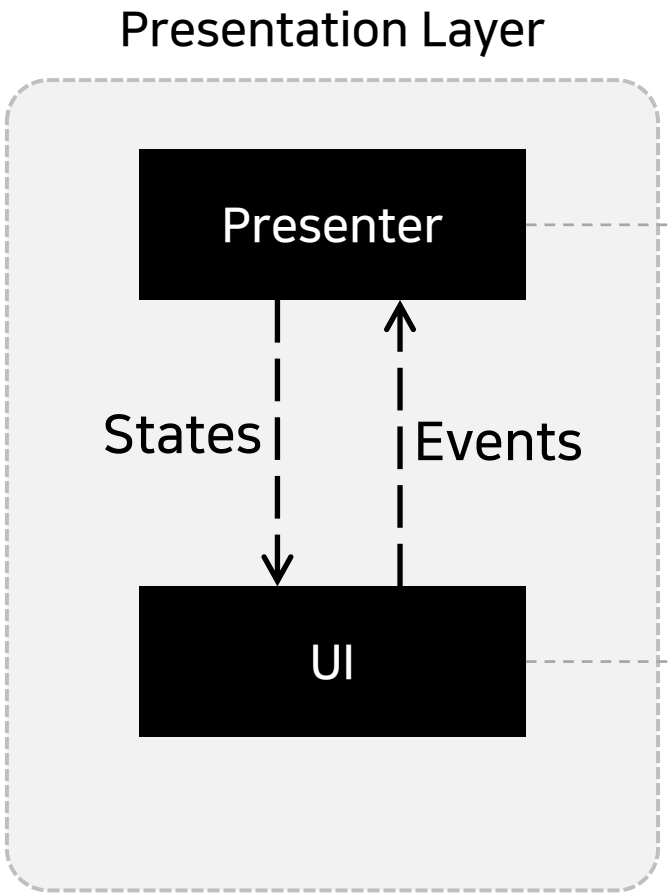
1. Architecture Design



More general & stable stuff in inner layers, more likely to change stuff in outer layers

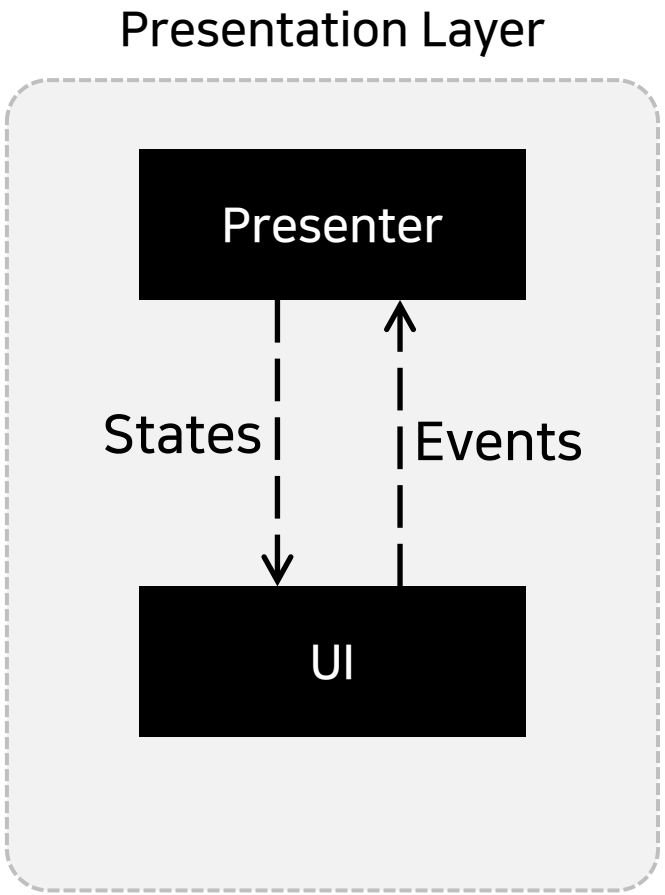


Business rules do not depend on Presentation & Data Layer



- Declarative UI Programming

- Design the user interface based on the data (state) received
- Decouple the business logic (Presenter) of that feature from its presentation (UI)
- Event-driven mechanism



BLoC

https://pub.dev/packages/bloc

- The UI part can listen to the states (stream of states) and do actions, build widgets or anything upon those states.
- Complex application needs for code separation.

Flutter Hooks

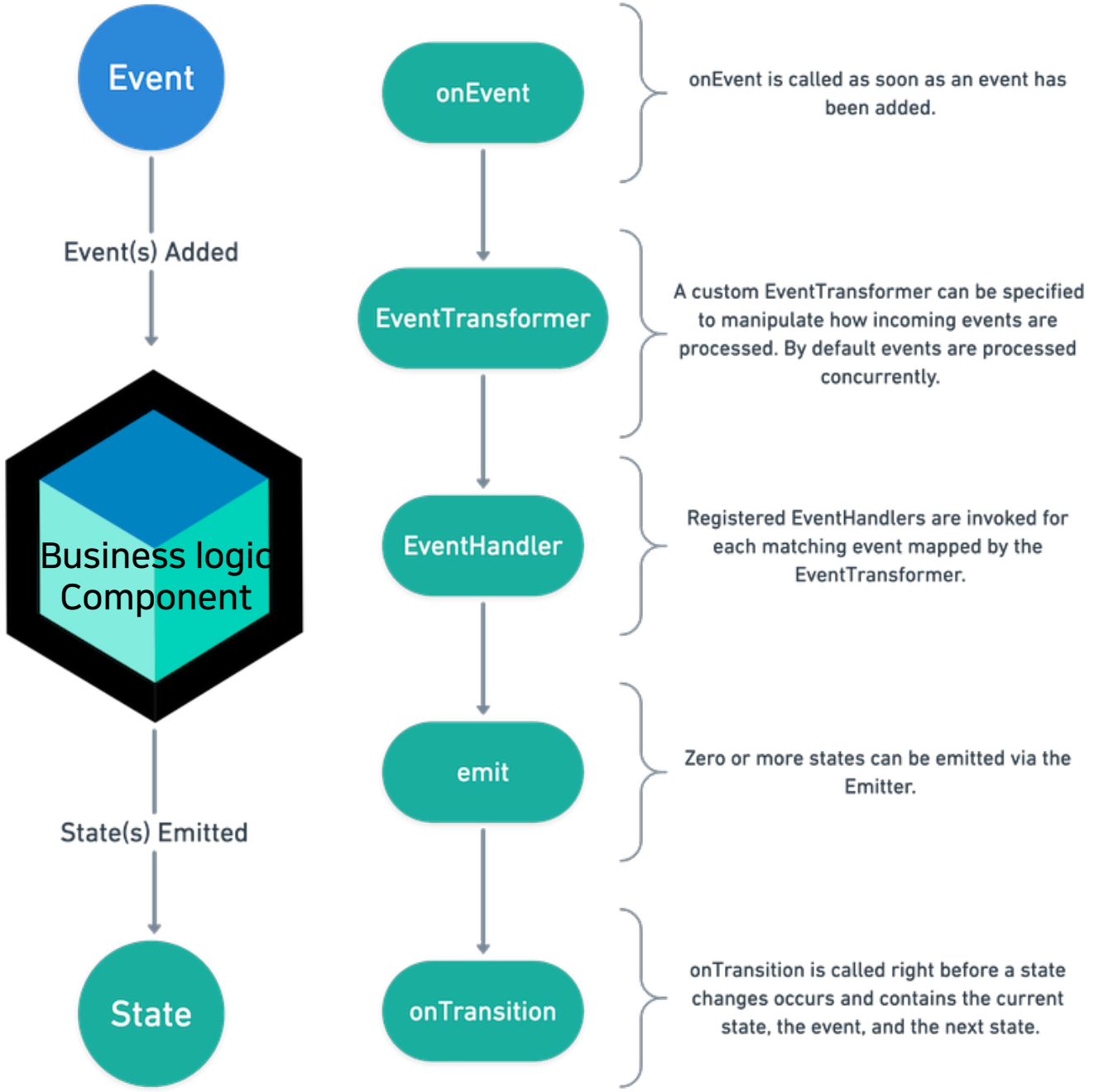
https://pub.dev/packages/flutter_hooks

- Duplicated logic between different components and lifecycle methods.
- *StatefulWidget* makes it difficult for developers to reuse the logic with in *initState* or *dispose*.
- The code must be simplistic and straight to the point.

BLoC is Event-Driven.

In a lot of cases, big products always require keeping track of things, add analytics, transform complex things, reduce debugging time, etc ···

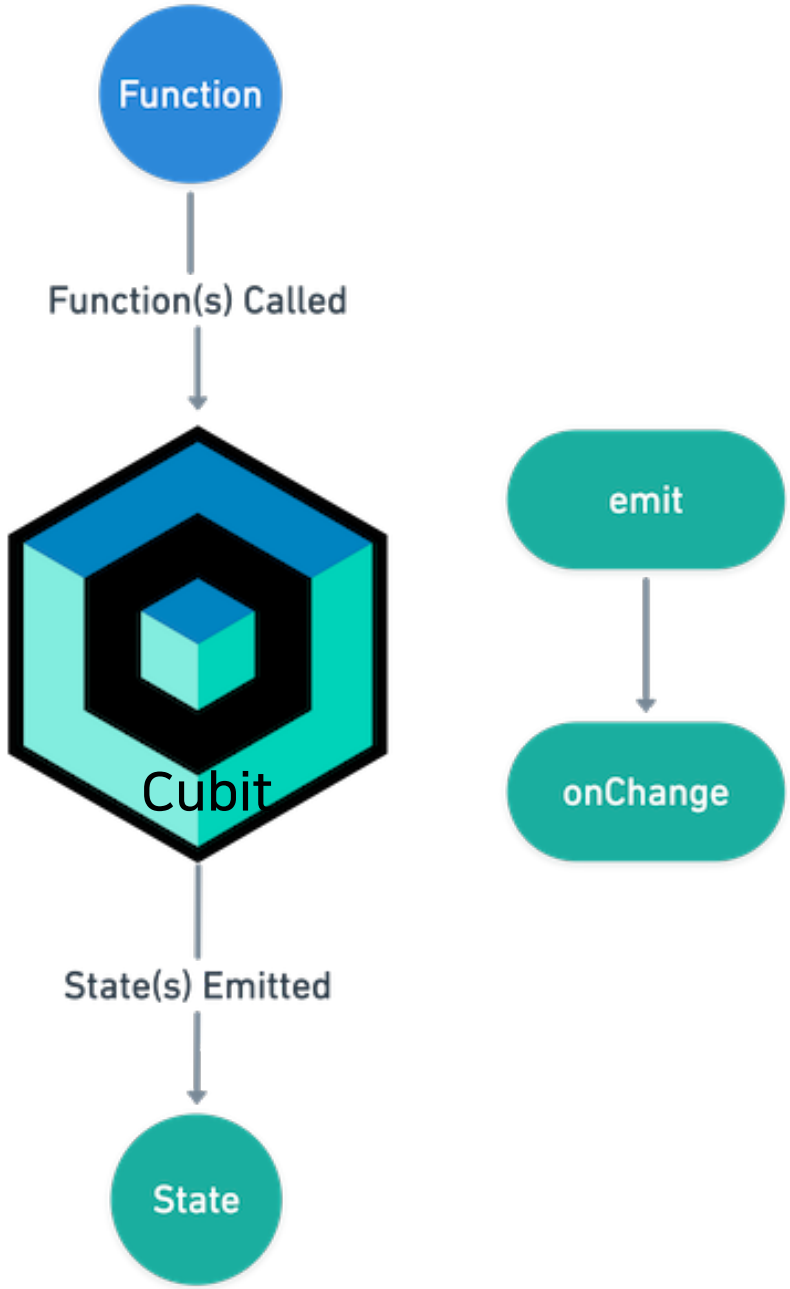
UI State changing, not domain logic.



BLoC is Event-Driven,

but Cubit and Provider is not.

It does not add an event, but directly calls a function containing logic.



Event Concurrency

- Event transformers must be considered.

  : concurrent - process events concurrently

  : sequential - process events sequentially

  : droppable - ignore any events added while an event is processing

  : restartable - process only the latest event and cancel previous event handlers

Using bloc_concurrency

https://pub.dev/packages/bloc_concurrency

Immutable State Object

- Must be immutable (create an instance using *copyWith* method)
- Easily forget to handle some of the bloc states.

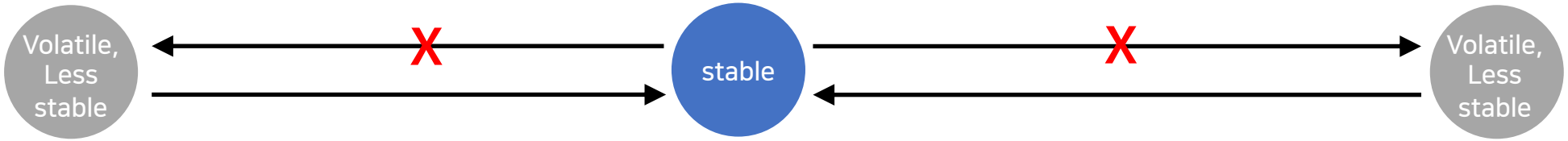
  → Sealed/Union classes

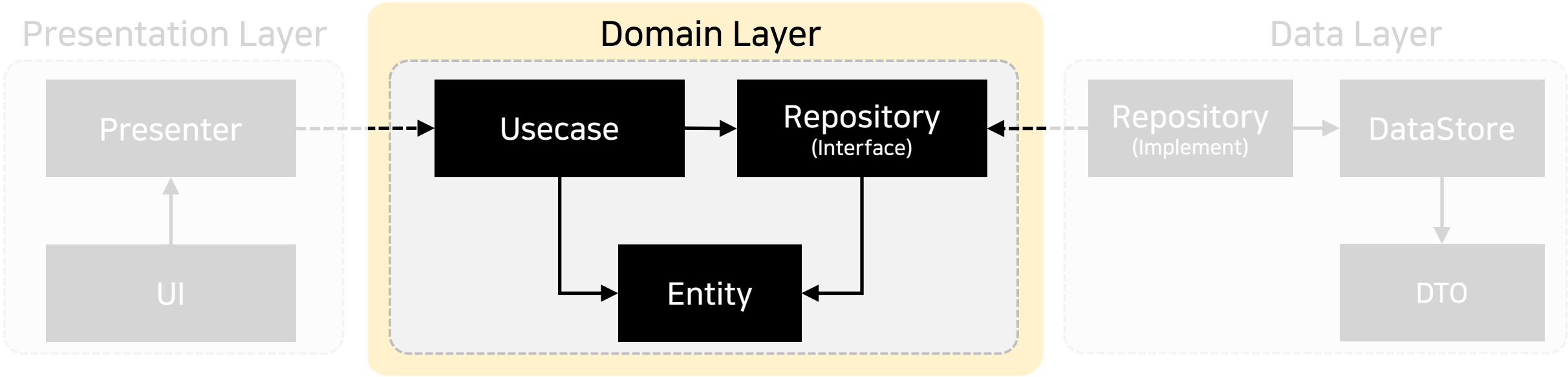
Using Freezed

https://pub.dev/packages/freezed

Stable Dependencies Principle

The dependencies between layers should be in the direction of stability





Fully independent layer which is the code for the business logic

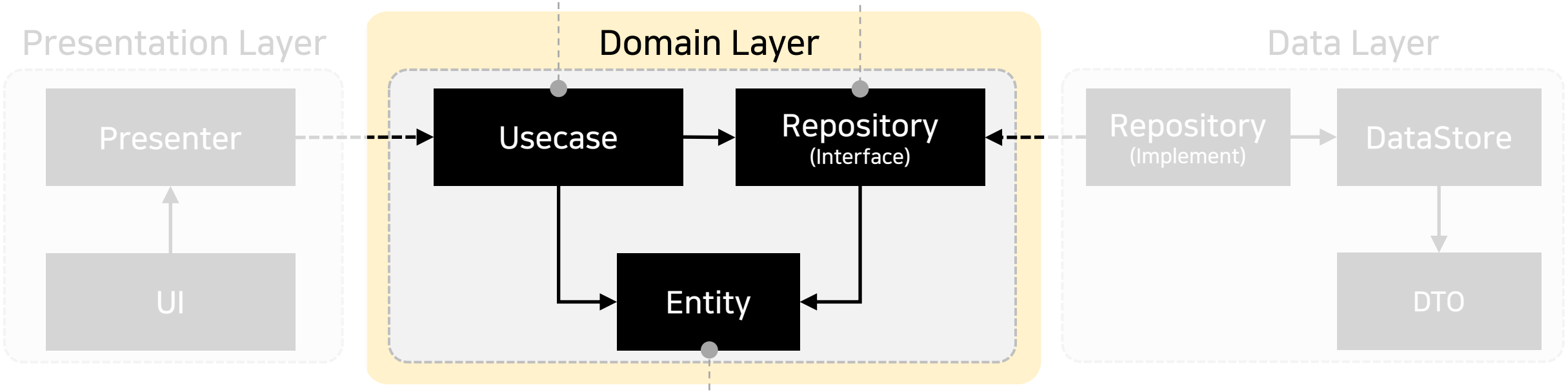
(No dependencies with other layers)

Use Case

- Describe the intent of the applications (Business logic).
- Give use cases access to Presentation Layer.
- Use cases combine data from 1 or multiple Repository Interfaces.

Repository <Interface>

- Only Interface for Dependency Inversion.
- Data layer Implements Repository interface.



Entity

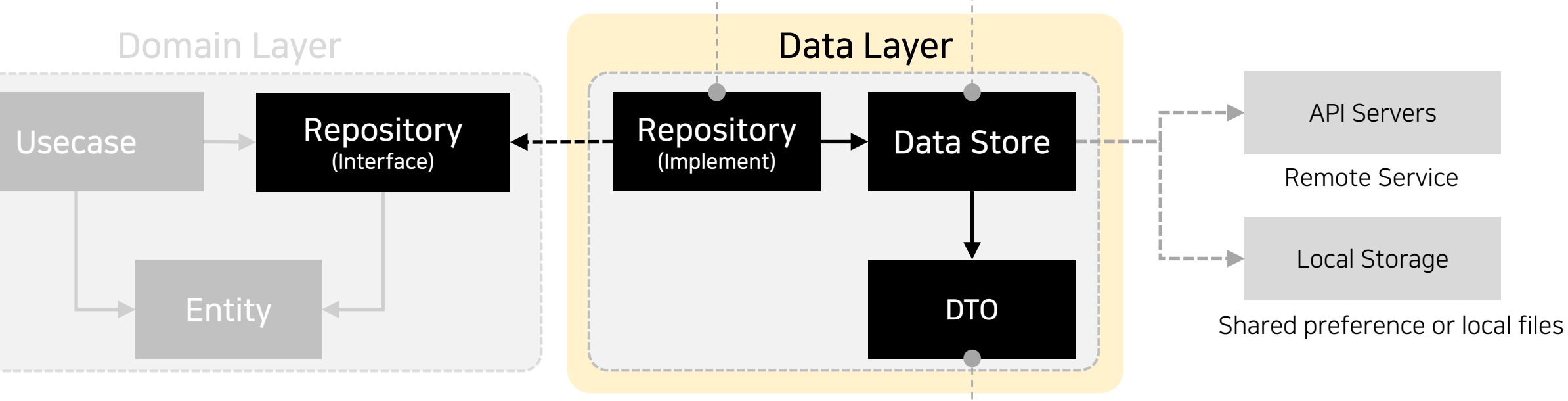
- Give entities access to both Presentation Layer and Data Layer
- Immutable Object

Repository (implementation)

- Implements Repository Interface from Domain Layer.
- Transforms from DTO to Entity
- Repository uses 1 or multiple Data Stores

Data Store

- Communicates with persisted local storages and remote API Servers.
- Only Data Store can access the outer data sources

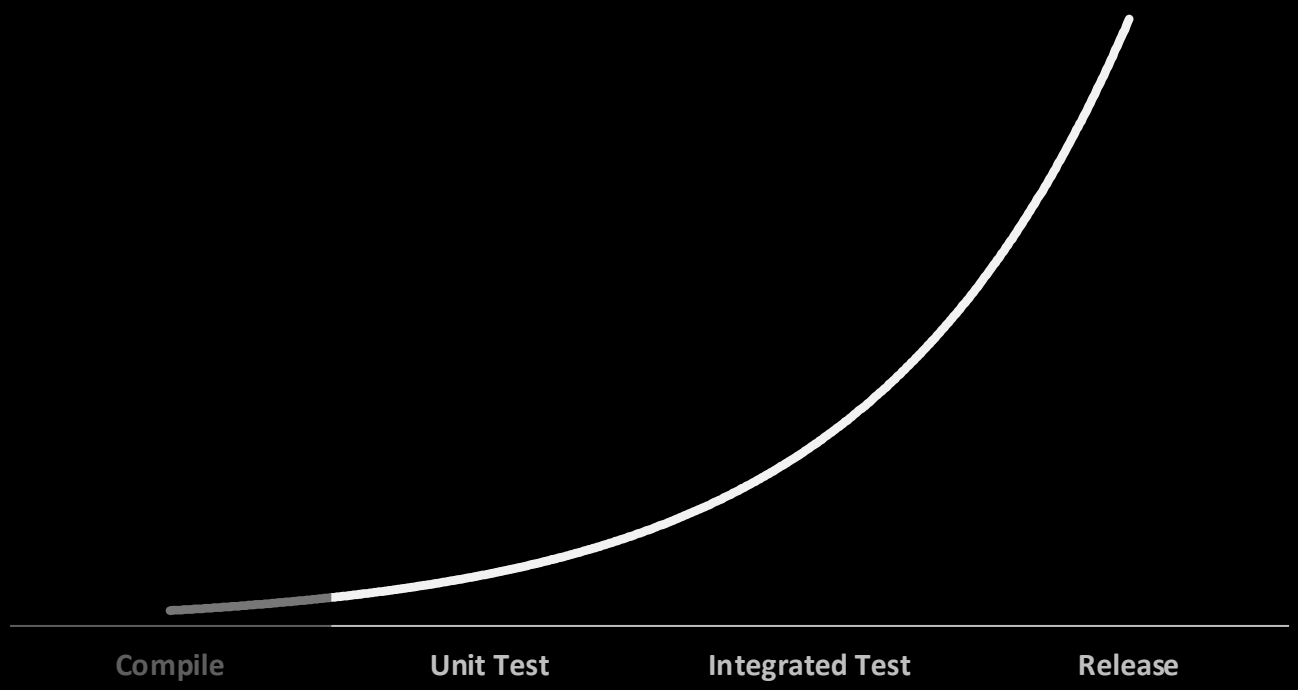


Data Transfer Object

- Serialize & Deserialize JSON from remote API and local storage

2. Test and Lint

2. Test and Lint – Cost of fixing a bug



The cost to fix bugs at Google

Software Testing Phase Where Bugs Were Found	Estimated Cost per Bug
System Testing	$5,000
Integration Testing	$500
Full Build	$50
Unit Testing/Test-Driven Development	$5

Decoupling also makes the process of testing and debugging of code easier and faster.

BLoC Test

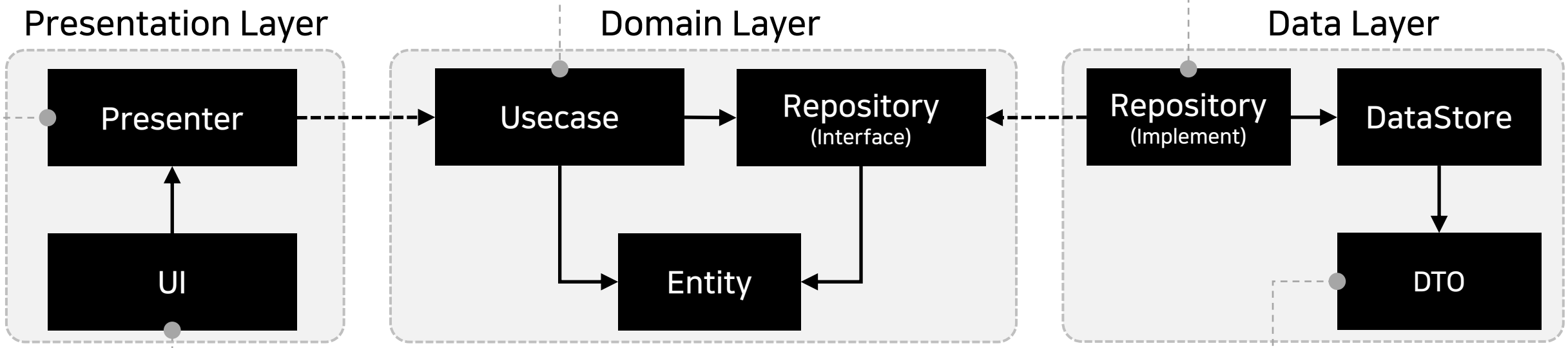
- tests the logics of 'BLoC provider' and states changing.
- mocks 'Use Case' from Domain Layer

Use Case Test

- tests Domain (Business) logics.
- mocks 'Repository' Implementation from Data Layer.

Repository Test

- tests transformation from DTO to Entity
- mocks HTTP Client



Widget Test

- tests various functionalities of the widget (usually individual widgets).
- tests that make sure that your widgets are still pixel-perfect (called 'golden' tests).

De/Serializing Test

- tests DTO serializing / deserializing of the responses from 'Data Store'

Testing strategy

- Business logic should be covered 85-100% in unit/integration tests.

  : Presenter(BLoC), Use Case and Repository implementation.

- Widget tests should cover all the reusable UI components.

  : When individual components are tested properly, you could start testing individual screens but in less detail.

- When the whole UI is ready and implemented, golden tests to ensure that UI is not affected by changes later.

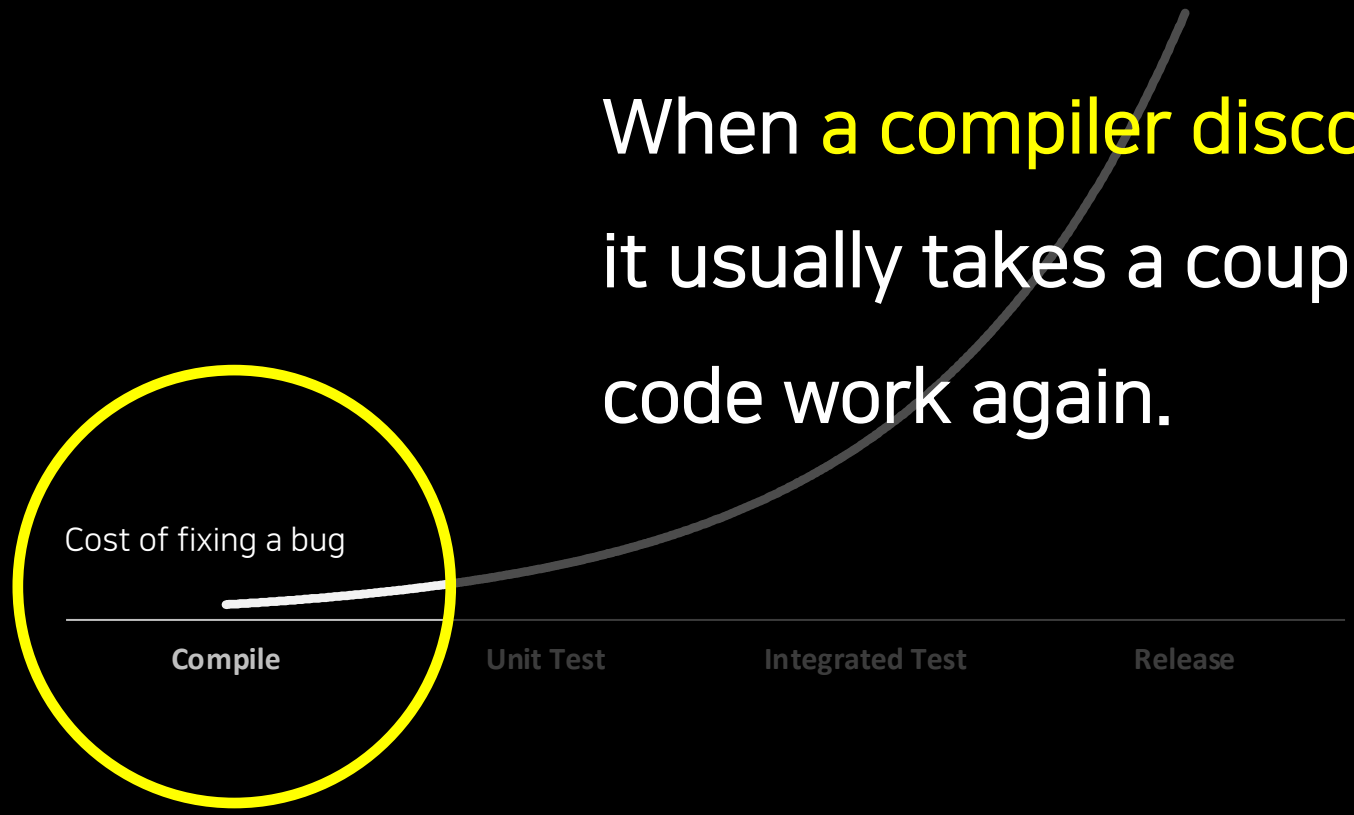
Does the flap of a butterfly’s wings in Brazil set off a tornado in Texas?

Steps in Widget test.

(It doesn't actually run on a physical/simulated device)

1. Set up requisites and create (pump) a widget to test with
2. Find the visual elements on the screen via some kind of property (such as a key)
3. Interact with the elements (such as a button) using the same identifier
4. Verify that the results match what was expected

When a compiler discovers a defect,
it usually takes a couple of seconds to make your code work again.



Static analyzer

```yaml
include: package:flutter_lints/flutter.yaml

linter:
  rules:
    ignore: camel_case_types
    prefer_relative_imports: true

analyzer:
  errors:
    todo: ignore
  exclude:
    - "**/*.g.dart"
    - "**/*.freezed.dart"
    - "**/*.test.dart"
```

analysis_options.yaml

Static analyzer for identifying possible problems in source code.

- More than a hundred linter rules are available.
  : errors - Possible coding errors
  : style - Matters of style (derived from the official Dart Style Guide)
  : pub – Package related rules
- Check anything from potential typing issues, coding style, and formatting.
- Provides official IDE extensions.

https://pub.dev/packages/flutter_lints

https://dart-lang.github.io/linter/lints/

https://dart.dev/guides/language/analysis-options

Dart Code Metrics

```yaml
analyzer:

  ....

  plugins:
    - dart_code_metrics

dart_code_metrics:
  metrics:
    cyclomatic-complexity: 20
    number-of-parameters: 4
    maximum-nesting-level: 5
  metrics-exclude:
    - test/**
  rules:
    - newline-before-return
    - no-boolean-literal-compare
    - no-empty-block
    - prefer-trailing-comma
    - prefer-conditional-expressions
    - no-equal-then-else
  anti-patterns:
    - long-method
    - long-parameter-list
```

Static analyzer that helps you analyze and improve your code quality.

- Rules

  https://dartcodemetrics.dev/docs/rules/overview

- Metrics

  : Cyclomatic Complexity

  : Halstead Volume

  : Lines of Code

  : Maintainability Index

  : Maximum Nesting

  : Number of Parameters

  : Source lines of Code

- Anti-patterns

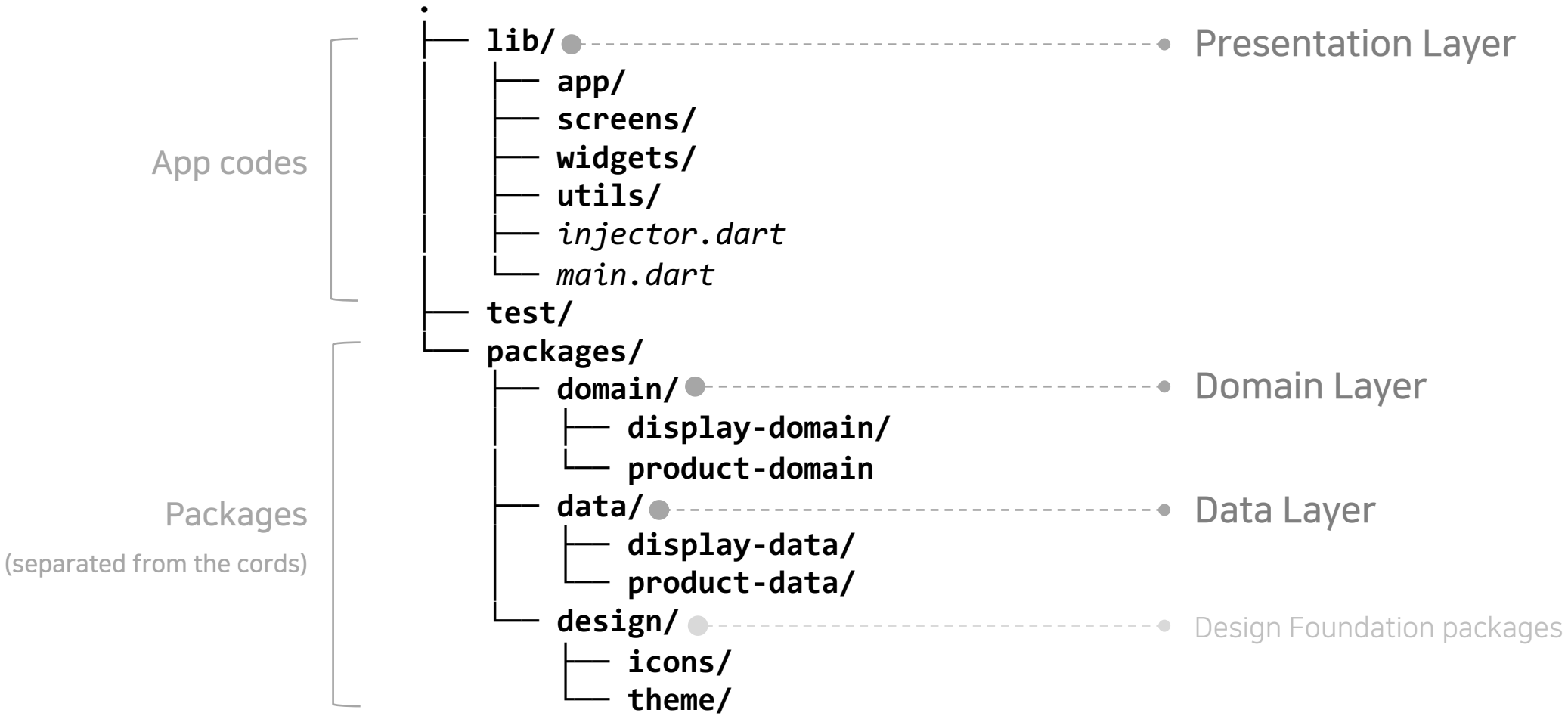
  : Long Method

  : Long Parameter List

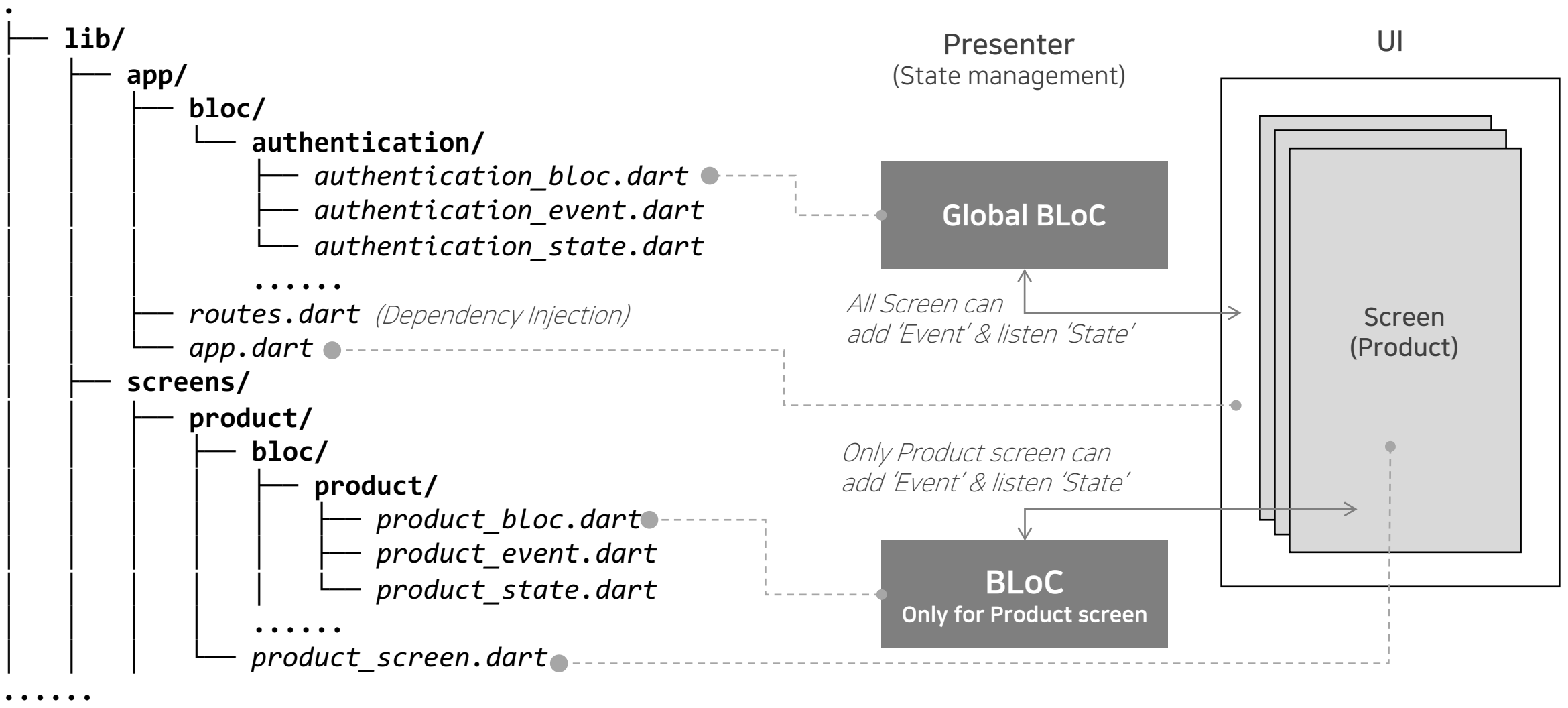
- Integrations with Codemagic and GitHub Action

3. Implementation

3. Implementation – folder structure



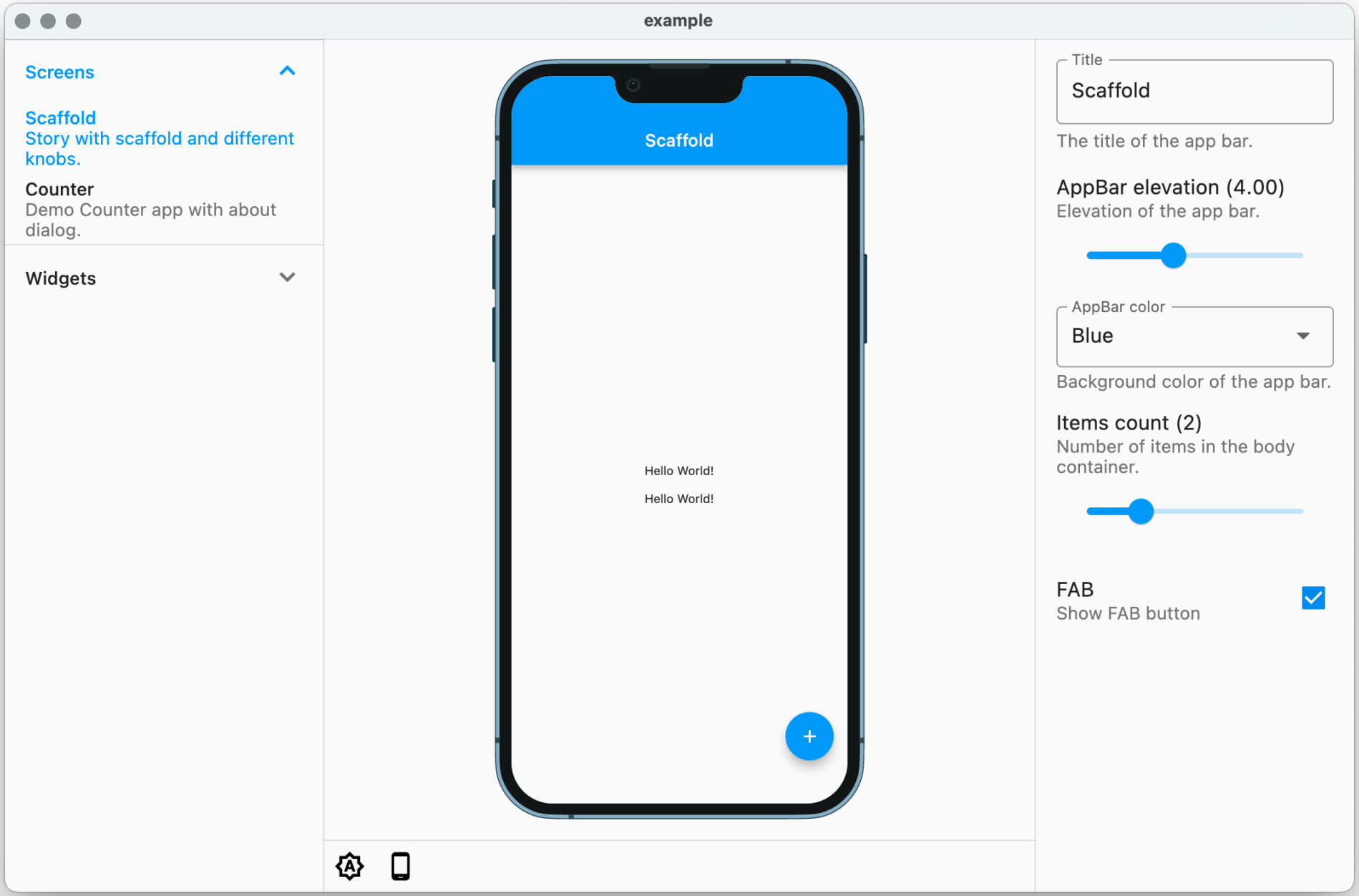
Presentation Layer



Storybook

Showcase UI Widgets

https://pub.dev/packages/storybook_flutter



Collaboration & Code Review

SharedPreferencesRepository 처리 관련 0
#9 · opened 21 hours ago by Shin HyunWook — updated 21 hours ago

rename view widgets into `/lib/widgets/common` CLOSED 1 2 3
#8 · opened 2 days ago by Kim Yohan — updated 21 hours ago

rename view widgets into `/lib/widgets/view` CLOSED 0
#7 · opened 2 days ago by Kim Yohan — updated 23 hours ago

remove constants classes from `/lib/constants` 2
#6 · opened 2 days ago by Kim Yohan — updated 23 hours ago

create new package for handling behavior tracing. CLOSED 0
#5 · opened 2 days ago by Kim Yohan — updated 1 hour ago

rename classes into /utils 0 of 3 tasks completed CLOSED 0
#4 · opened 2 days ago by Kim Yohan Mar 15, 2022 — updated 21 hours ago

create new package for design theme. 0
#3 · opened 2 days ago by Kim Yohan Mar 15, 2022 — updated 1 day ago

rename DTO files from *.entity.dart to *.dto.dart CLOSED 0
#2 · opened 2 days ago by Kim Yohan Mar 15, 2022 — updated 1 day ago

remove icons from `/assets/icons` 1 2
#1 · opened 2 days ago by Kim Yohan Mar 16, 2022 — updated 23 hours ago

Closed · Opened 2 days ago by **Kim Yohan** · Reopen issue · New issue

rename view widgets into `/lib/widgets/common`

It's ambiguous what the `common` means.

I think 'common' already overlaps with the meaning that all files in this `/lib/widgets` folder are used in common.

The codebase must make it predictable.

It is inappropriate to use a directory or file called 'common' in this `/lib/widgets` folder.

Therefore, **I suggest changing** `/lib/widgets/common/timer.widget.dart` **to** `/lib/widgets/timer/timer.widget.dart`.

Edited 2 days ago by Kim Yohan

1 Related Merge Request

!61 **WIP: Resolve "rename view widgets into `/lib/widgets/common`"** Closed

2 0

Kim Yohan @22573 changed the description · 2 days ago

Kim Yohan @22573 created branch `8-rename-view-widgets-into-lib-widgets-common` · 2 days ago

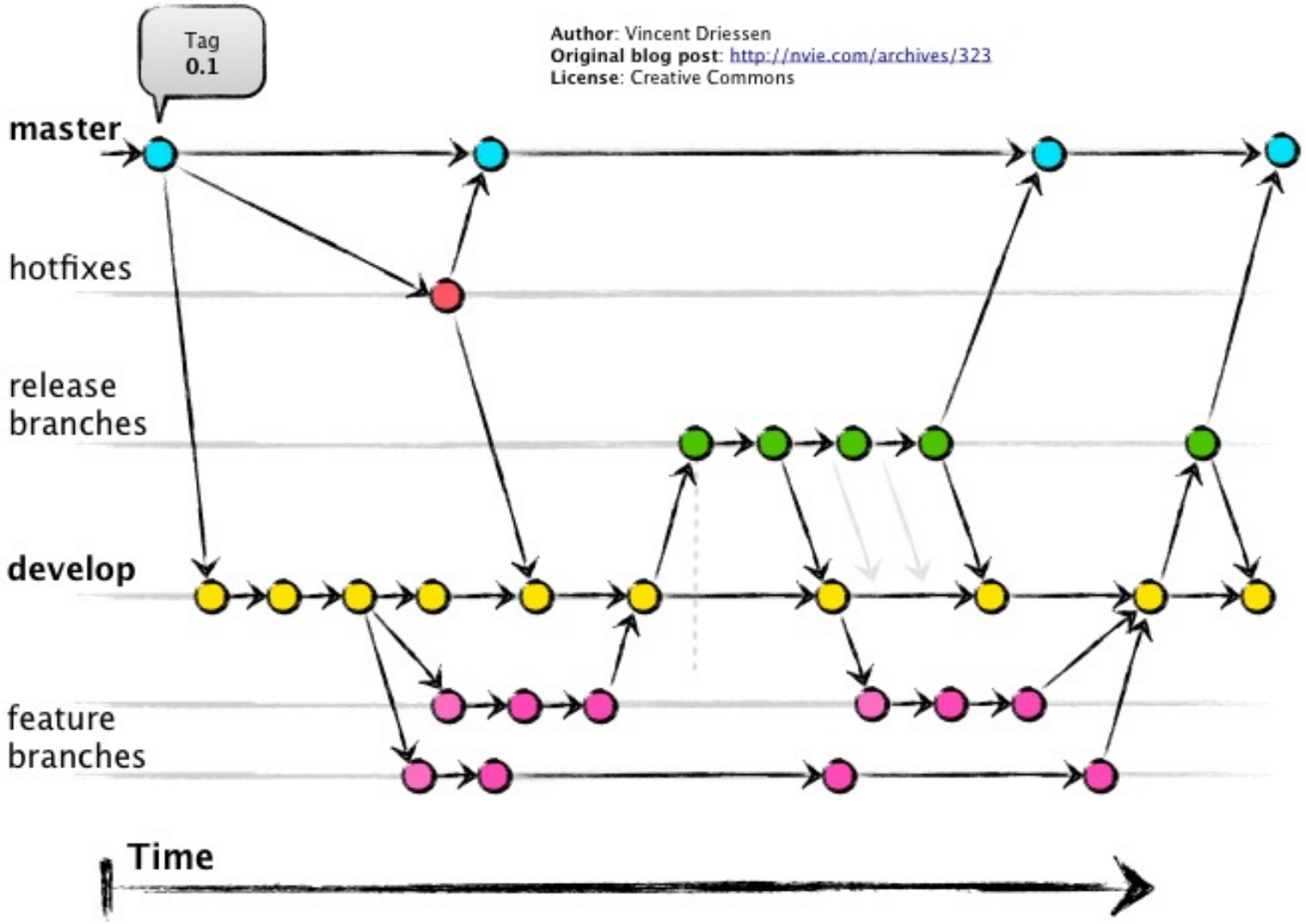
Kim Yohan @22573 mentioned in merge request !61 (closed) · 2 days ago

Shin HyunWook @ae23086 · 2 days ago — Maintainer

일반적으로 패키지 구조가 common을 의미 하기에 추가적으로 common이라는 하위로 나눌 필요가 없다는건 옳은것 같습니다.
위 내용과 별개로 궁금점이 생겼습니다.
기타 공통들, 즉 단일 파일로 구성되거나 코드가 짧은 공통 파일들이 있을 수 있습니다.
이런 것들을 애매하게 각각 패키지로 나누자니 패키지들이 너무 많아지고
common으로 묶어서 전부 넣자니 같은 레벨(widgets같은) 패키지들이 공통이 아닌것 같은 느낌이 드는데

Git-flow

Branch strategy



Finally

app codes

Presentation Layer

GS SHOP

Presentation Layer

GS Fresh Mall

Presentation Layer

?

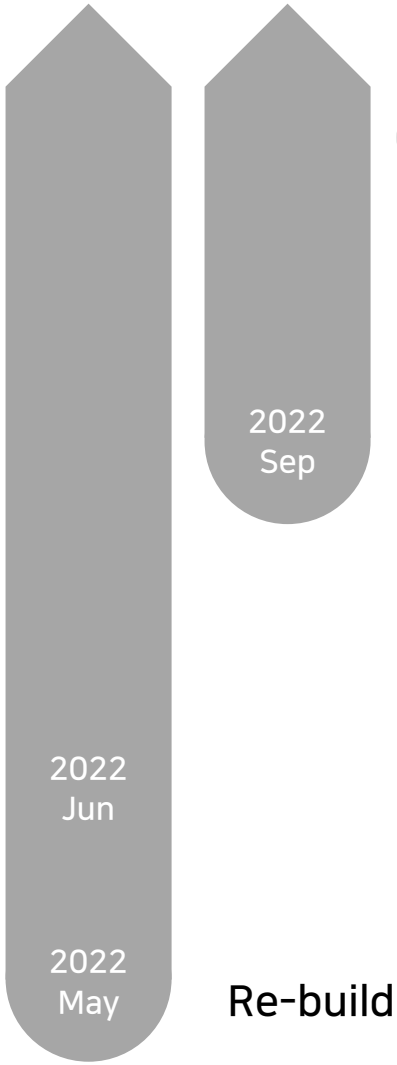
packages

Domain Layer

- Main Display Domain
- Search Display Domain
- Recommend Display Domain
- Product Domain
- Order Domain
- Live Broadcast Domain

Data Layer

- Main Display Data
- Search Display Data
- Recommend Display Data
- SHOP Mobile Product Data
- SHOP Mobile Order Data
- M4 Product Data
- M4 Order Data
- Live Broadcast Data



2023

Re-design Mobile development org.
with Offshore developers

2022
Sep

‘GS SHOP’ app refactoring based ‘Common Domain Layer’

Pair programming
with Offshore Dev Center
(focus on developing “Presentation Layer”)

2022
Jun

Complete architecture design and
documents for developers

2022
May

Re-build ‘GS Fresh Mall’ app with Clean Architecture

Producing boring code is the biggest compliment that an engineering team can receive.

Having a codebase that is predictable, easy to navigate, well tested and properly automated makes it boring. But pleasantly boring!

https://verygood.ventures/blog/boring-code-part-1