Архитектура ПО на уровне компонентов

Мы пишем код для того, чтобы ...?

Мы пишем код для того, чтобы *решать проблемы клиента*

Зачем что-то проектировать?

Зачем что-то проектировать?

- Лучше решать проблемы клиента
- Не создать клиенту новых проблем

Где место архитектуры в цикле разработки?

Проблема => решение => архитектура => итеративная разработка

На что опираться?

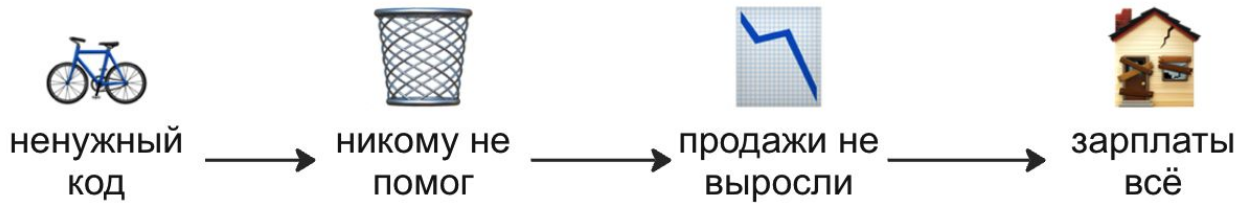
Из проблемы и предлагаемого ПМ/ПО решения определить набор функциональных и нефункциональных требований

YAGNI - You Aren't Gonna Need It

- Если пишете код, то будьте уверены, что он вам понадобится. Не пишите код, если думаете, что он пригодится позже.

- Потому что:
  - На написание кода тратится время
  - Написанный код нужно сопровождать
  - Написанный код ограничивает то, что может быть сделано в будущем, — ненужные новые функции могут впоследствии помешать добавить новые нужные

YAGNI - You Aren't Gonna Need It



KISS - keep it simple, stupid

- Не придумывайте к задаче более сложного решения, чем ей требуется

- Почему? - В простом классе/системе значительно легче обеспечить надежность

- «Понятный» код лучше «умного»

Пара хороших правил:

- *Правило 80/20 - 20% усилий дают 80% результата, соответственно эти 20% усилий нужно сделать в первую очередь!*

- *Pragmatic, not dogmatic*

Писать только код,
помогающий решать
определенную проблему
клиента*

*(сейчас или в будущем)

Что такое компонент и архитектура на уровне компонентов?

- Компонент - составная часть ПО, выполняющая определенную задачу (модуль/библиотека/проект/пакет)

- Архитектура в рамках одного приложения, сфокусированная на зависимостях компонентов внутри этого приложения

Почему важно думать о зависимостях?

Почему важно думать о зависимостях?

- A зависит от B => любое изменение B может сломать A

Если не читали про SOLID - почитайте =)

SOLID, очень коротко

- Single responsibility
- Open closed
- Liskov substitution
- Interface segregation
- Dependency inversion

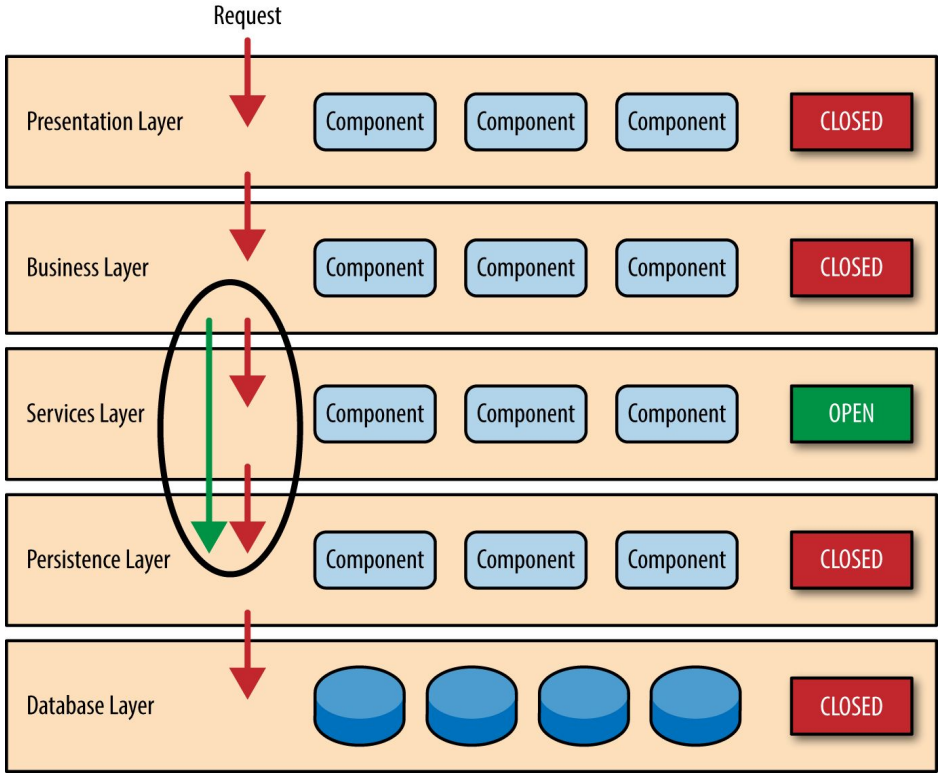
Распространенные подходы к архитектуре на уровне компонентов

Какие компоненты есть в веб-приложении?

Layered Architecture

+ Простота разработки
+ Высокая тестируемость

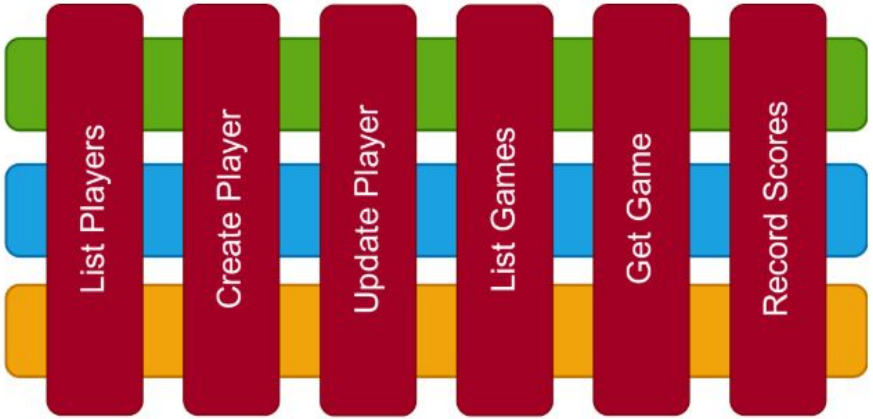
- Производительность*
- Масштабируемость*
- Гибкость*
- Зависимости*



Vertical Slice Architecture

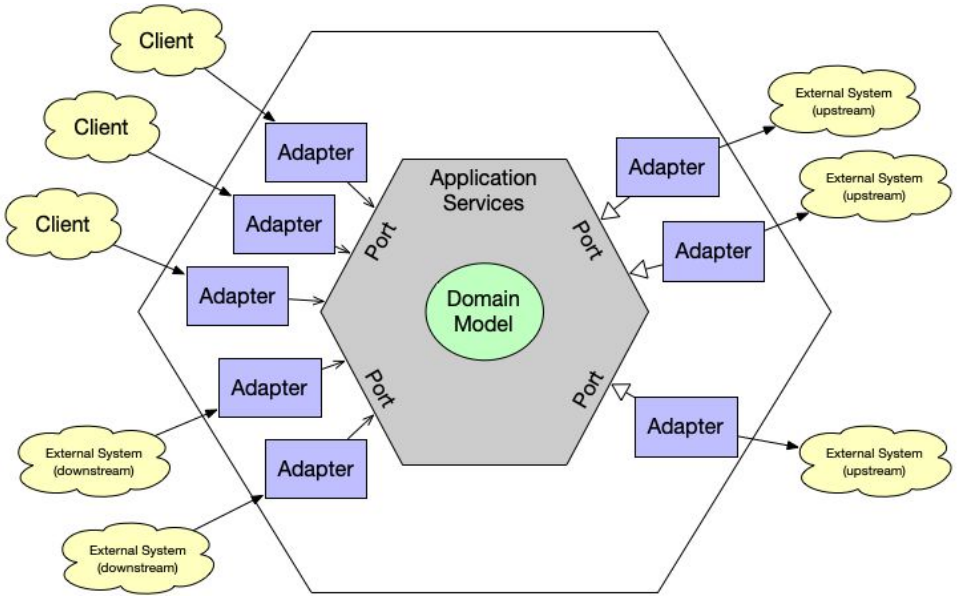
+ Гибкость
+ Масштабируемость

- Выше требования к навыкам рефакторинга, знаниям DDD

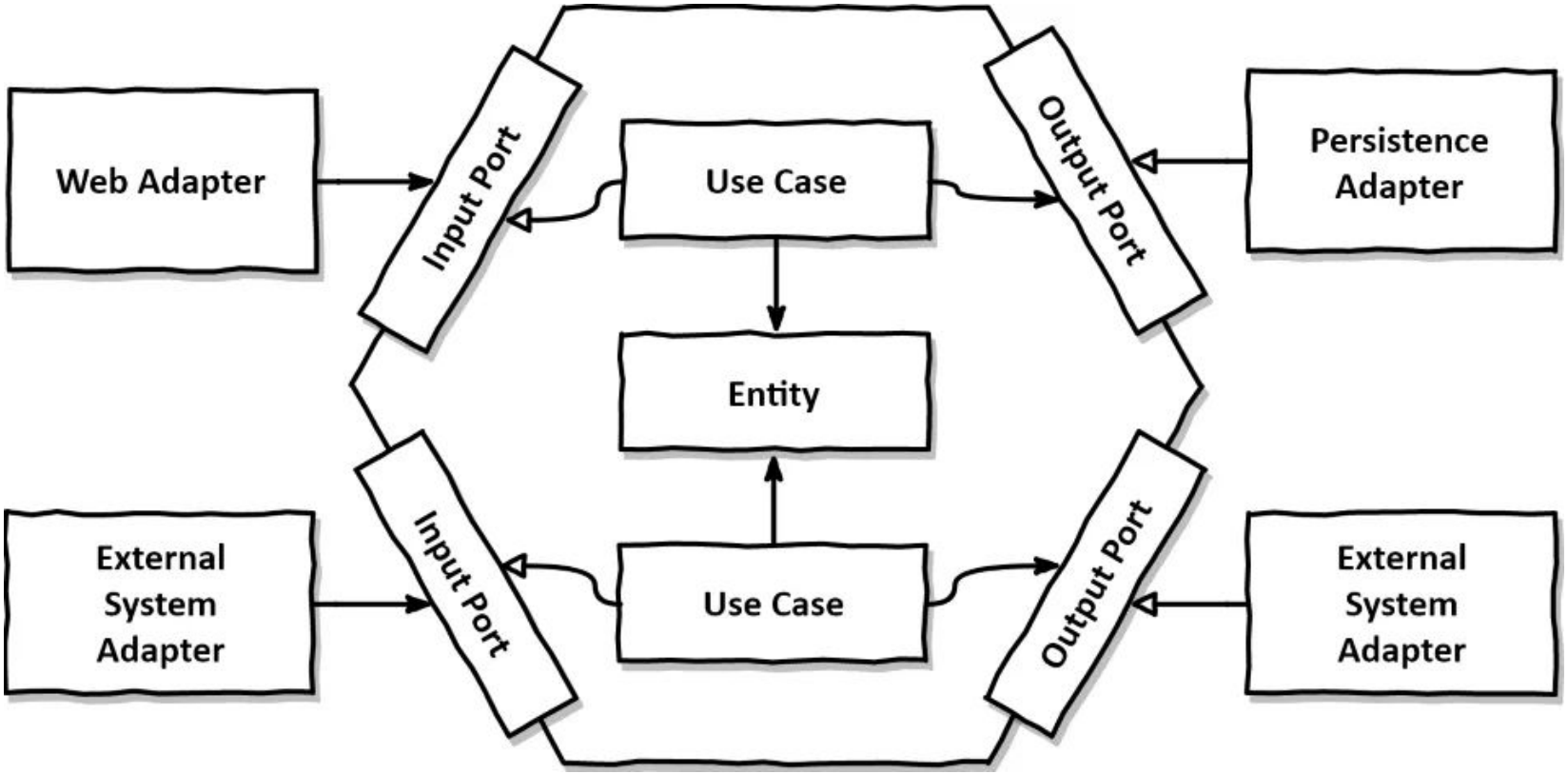


Какие компоненты должны быть наименее изменчивыми?

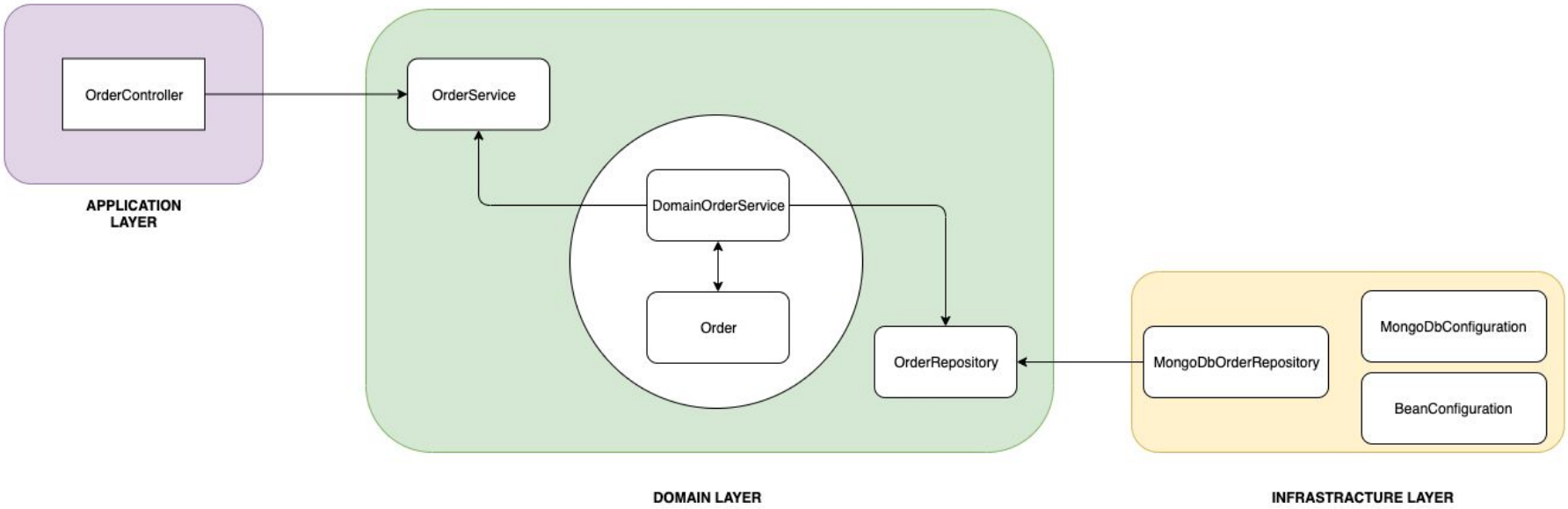
Hexagonal (Ports & Adapters) Architecture



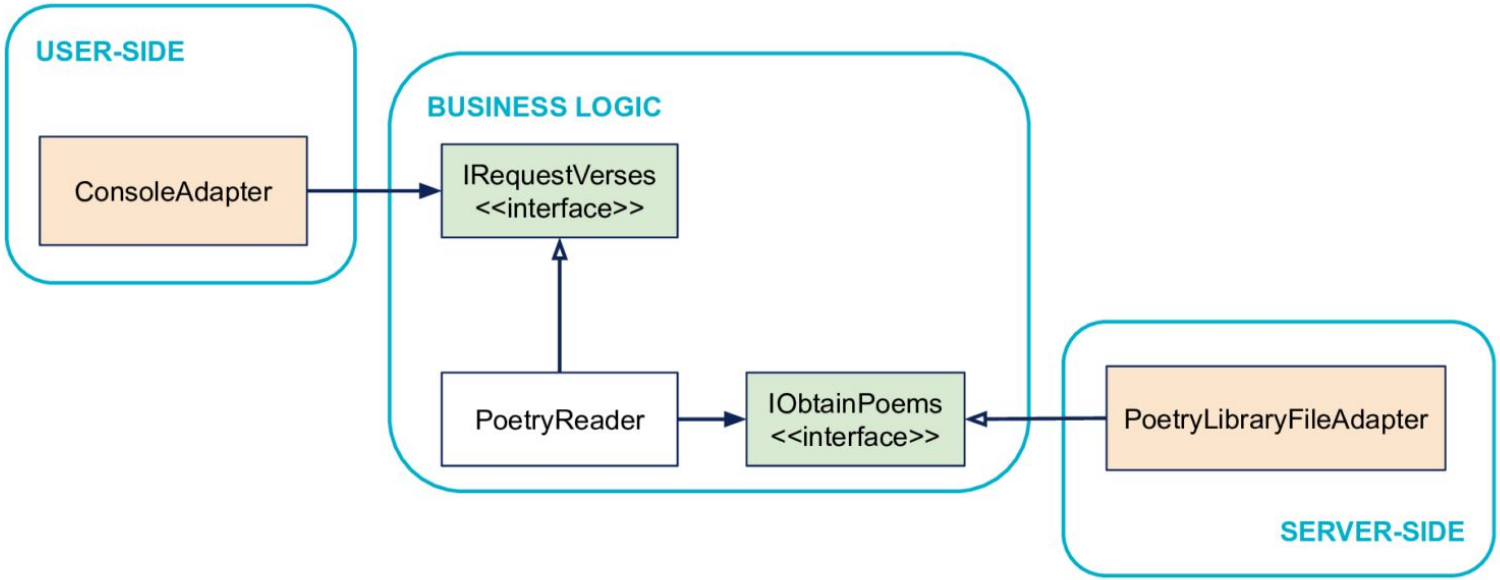
Hexagonal (Ports & Adapters) Architecture



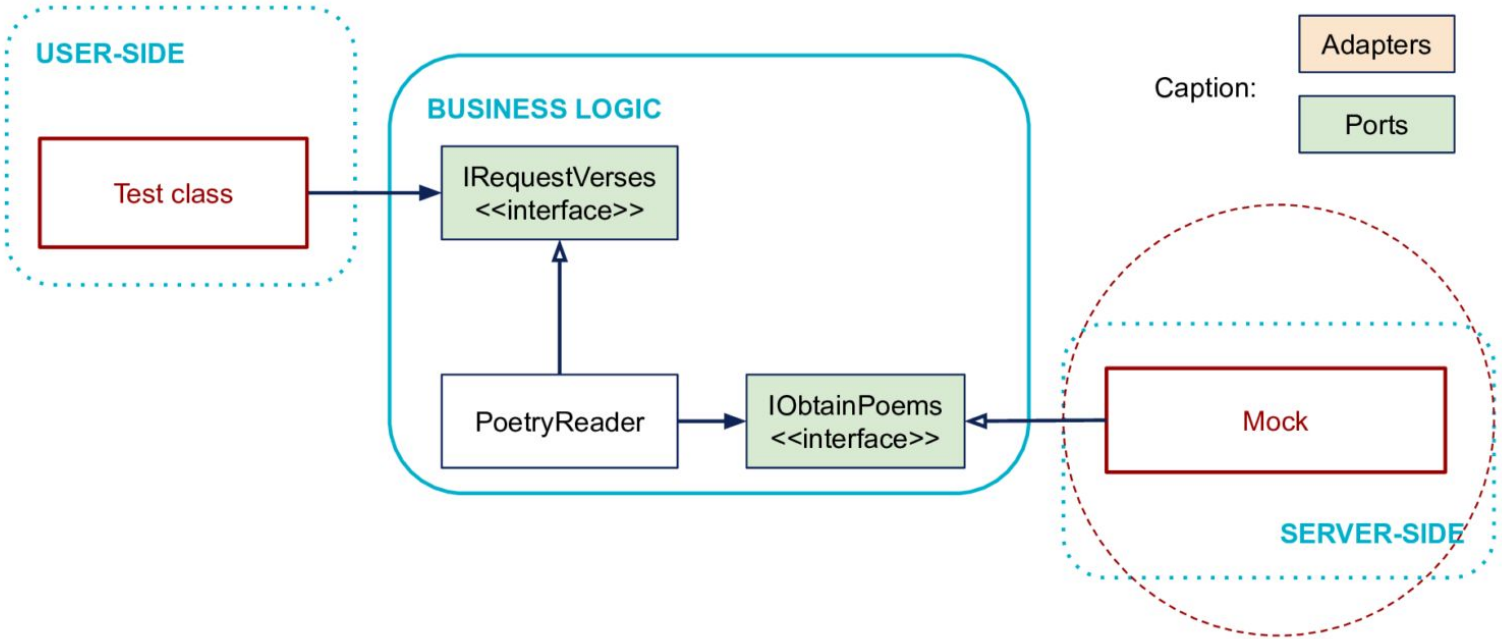
Hexagonal (Ports & Adapters) Architecture



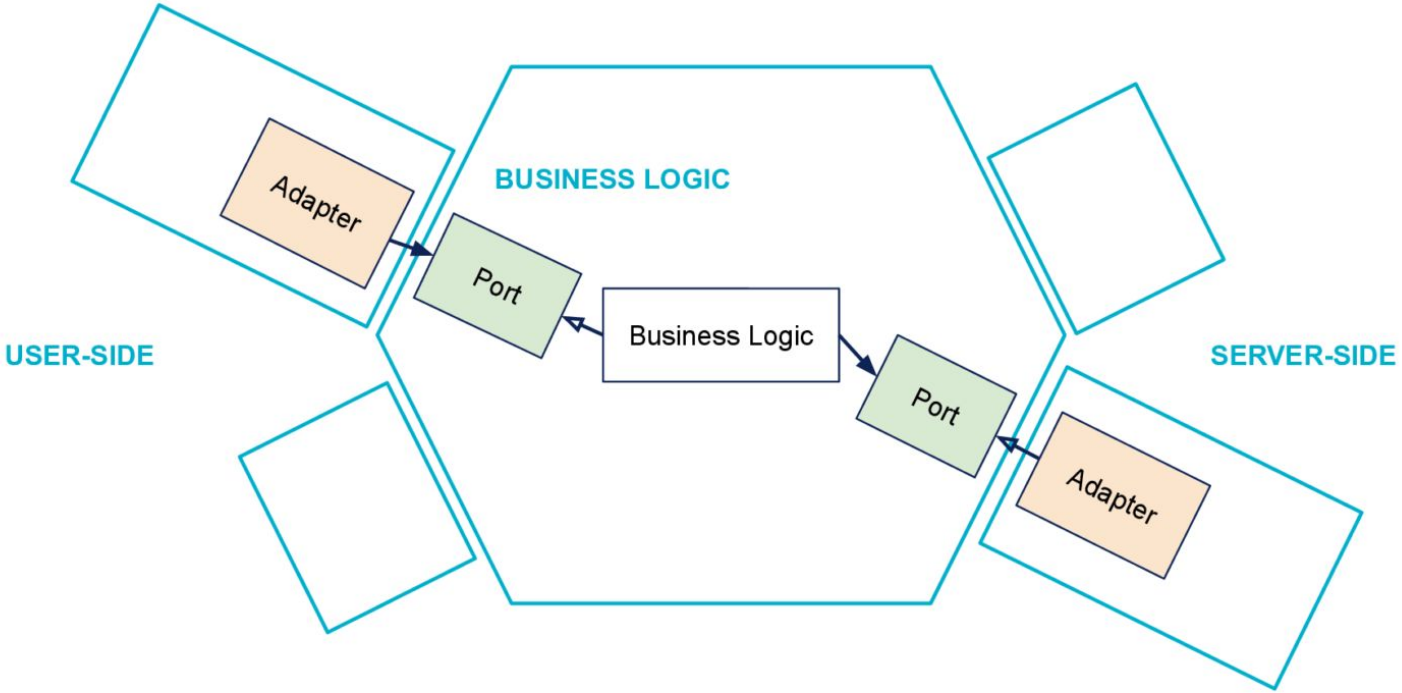
Hexagonal (Ports & Adapters) Architecture



Hexagonal (Ports & Adapters) Architecture

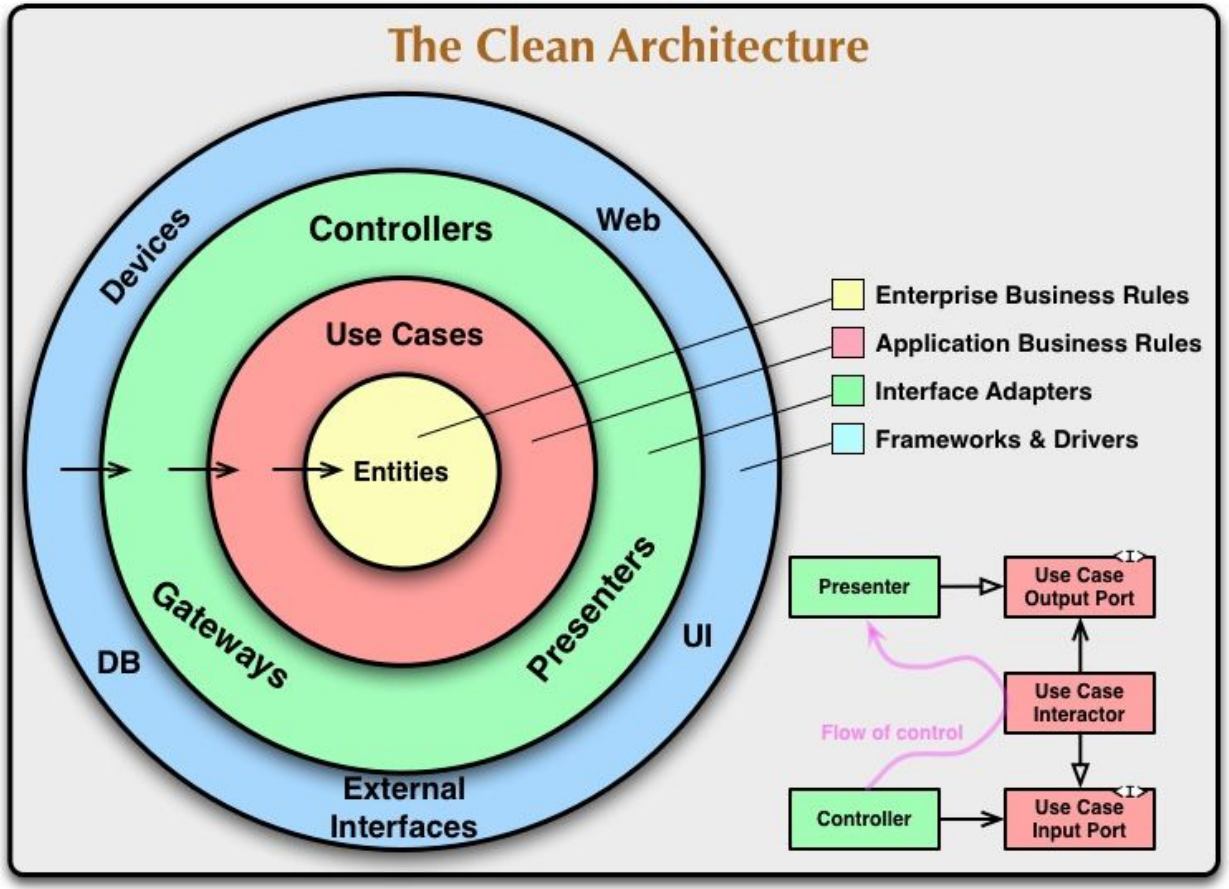


Hexagonal (Ports & Adapters) Architecture

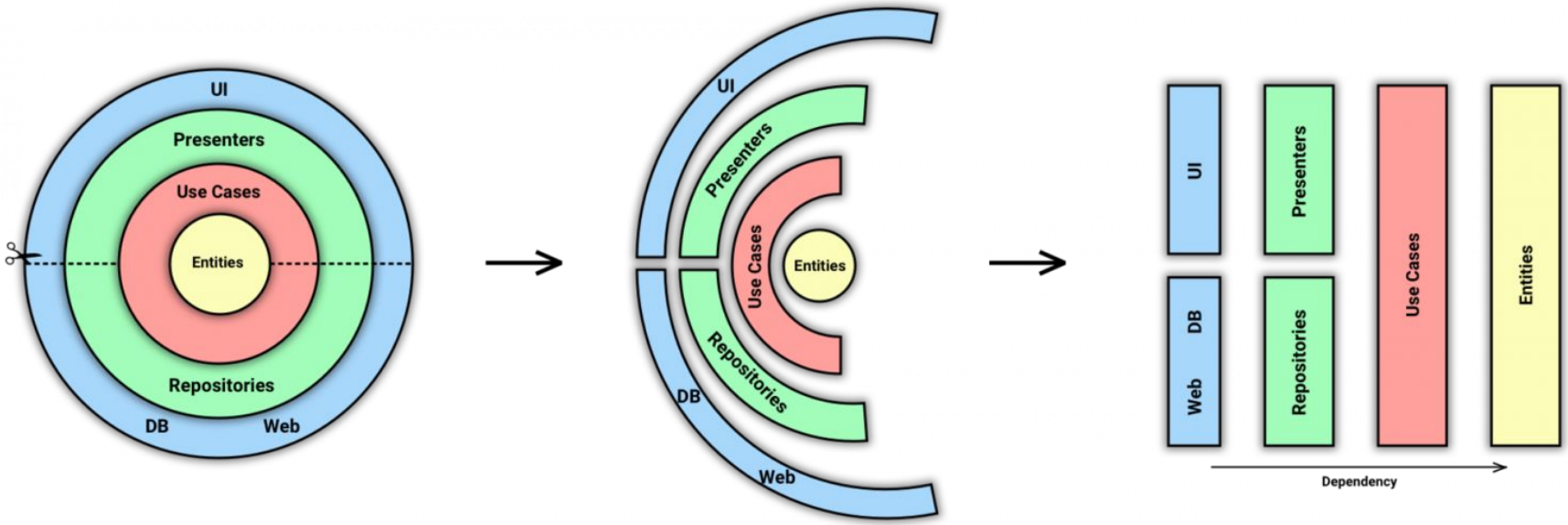


Clean architecture

Зависимости по стрелкам



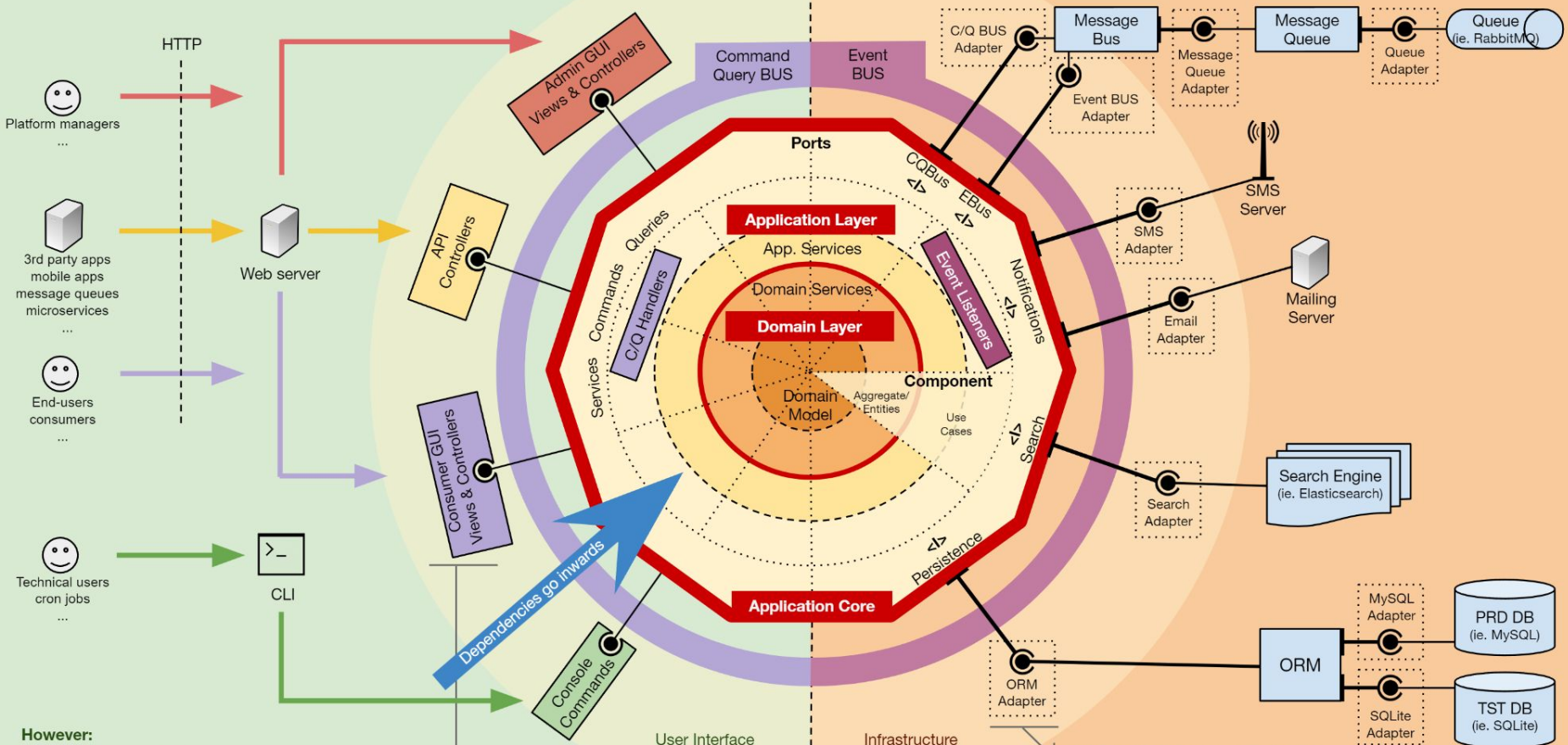
Clean architecture



Explicit Architecture

Primary/Driving Adapters

Secondary/Driven Adapters



However:

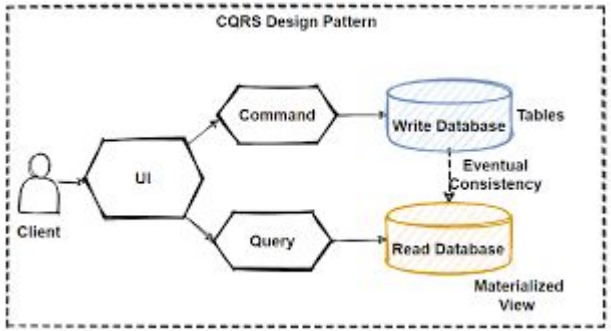
- The map is not the territory.
- Plans are worthless, but planning is everything.
- Understand all of this, but use only what you need.
- The actual architecture is driven by the project requirements.

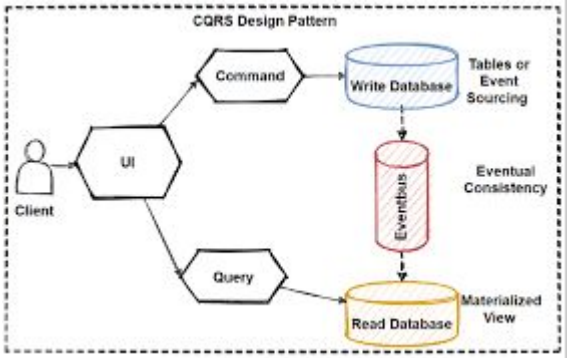
Primary adapters wrap around a use case and adapt its input/output to a delivery mechanism, ie. HTTP/HTML, HTTP/JSON or CLI.

Secondary adapters wrap around a tool and adapt its input/output to a port, which fits the application core needs and can simply be an interface or a complex set of interfaces and objects.

Время вопросов

CQRS


CQRS Design Pattern
Command
Write Database
Tables
UI
Eventual
Consistency
Client
Query
Read Database
Materialized
View


CQRS Design Pattern
Command
Write Database
Tables or
Event
Sourcing
UI
Eventbus
Eventual
Consistency
Client
Query
Read Database
Materialized
View

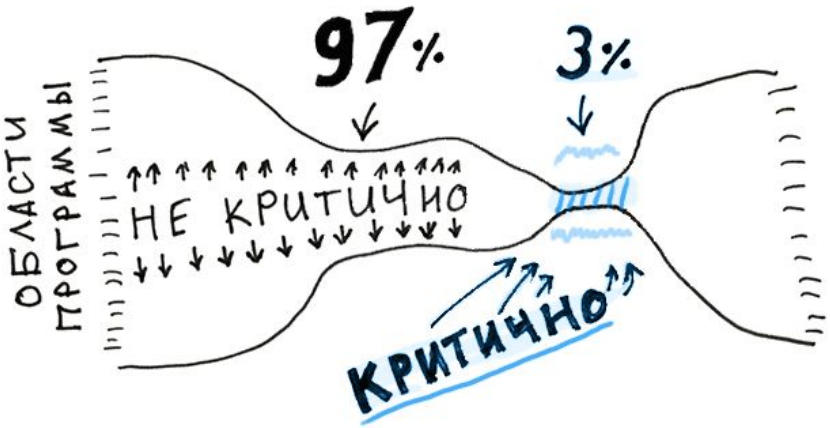
Еще немного полезного

Используй готовые решения, когда это возможно

Когда нужно думать об оптимизации?

Когда нужно думать об оптимизации?

Программисты тратят огромное количество времени, размышляя и беспокоясь о некритичных местах кода, и пытаются оптимизировать их, что исключительно негативно сказывается на последующей отладке и поддержке. Мы должны вообще забыть об оптимизации в, скажем, 97% случаев; более того, ***поспешная оптимизация является корнем всех зол.*** *И напротив, мы должны уделить все внимание оставшимся 3%. (с) Дональд Кнут*



Когда нужно рефакторить код?

Как правильно рефакторить код?

Полезные ссылки

- Видео о слоеной гексагональной и чистой архитектурах (и как они объединяются)

Время вопросов