Clean Architecture

R.C.Martin

시스템 = 정책 정책 정책 ⋮

아키텍쳐 개발 => DAG로 구성하기

Node = 동일한 수준의 정책을 포함하는 Component

수준(Level)

입력과 출력까지의 거리가 멀수록 고수준의 정책

데이터 흐름 !== 소스 코드 의존성

```
function encrypt () {
    while(true)
        writeCahr(translate(readChar))
}
```

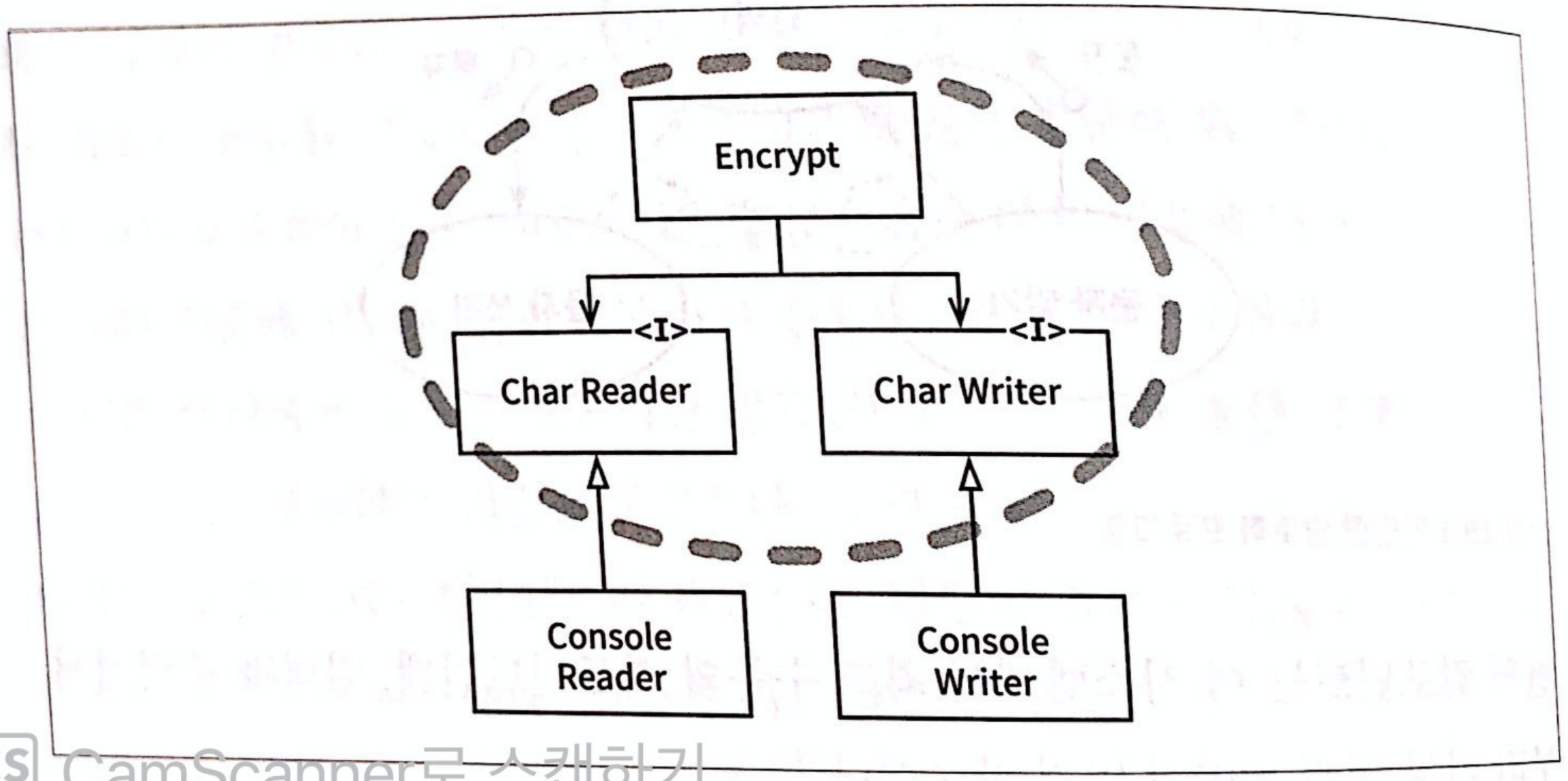


그림 19.2 시스템의 더 나은 아키텍처를 보여주는 클래스 다이어그램

Critical Business Rule

사업적으로 수익을 얻거나 비용을 줄일 수 있어야 함

Critical Business Data

Biz Rule + Biz Data == Entity

Entity

Small 핵심 Biz rule 구체화

DB, UI, Third Party에 오염 X

클래스일 수도 있지만, 하나의 모듈!

UseCase

Less Pure Biz Rule

Application-specific Biz Rule

UseCase가 Entity를 제어

아키텍처

UseCase! UseCase! UseCase!

프레임 워크는 도구일 뿐?

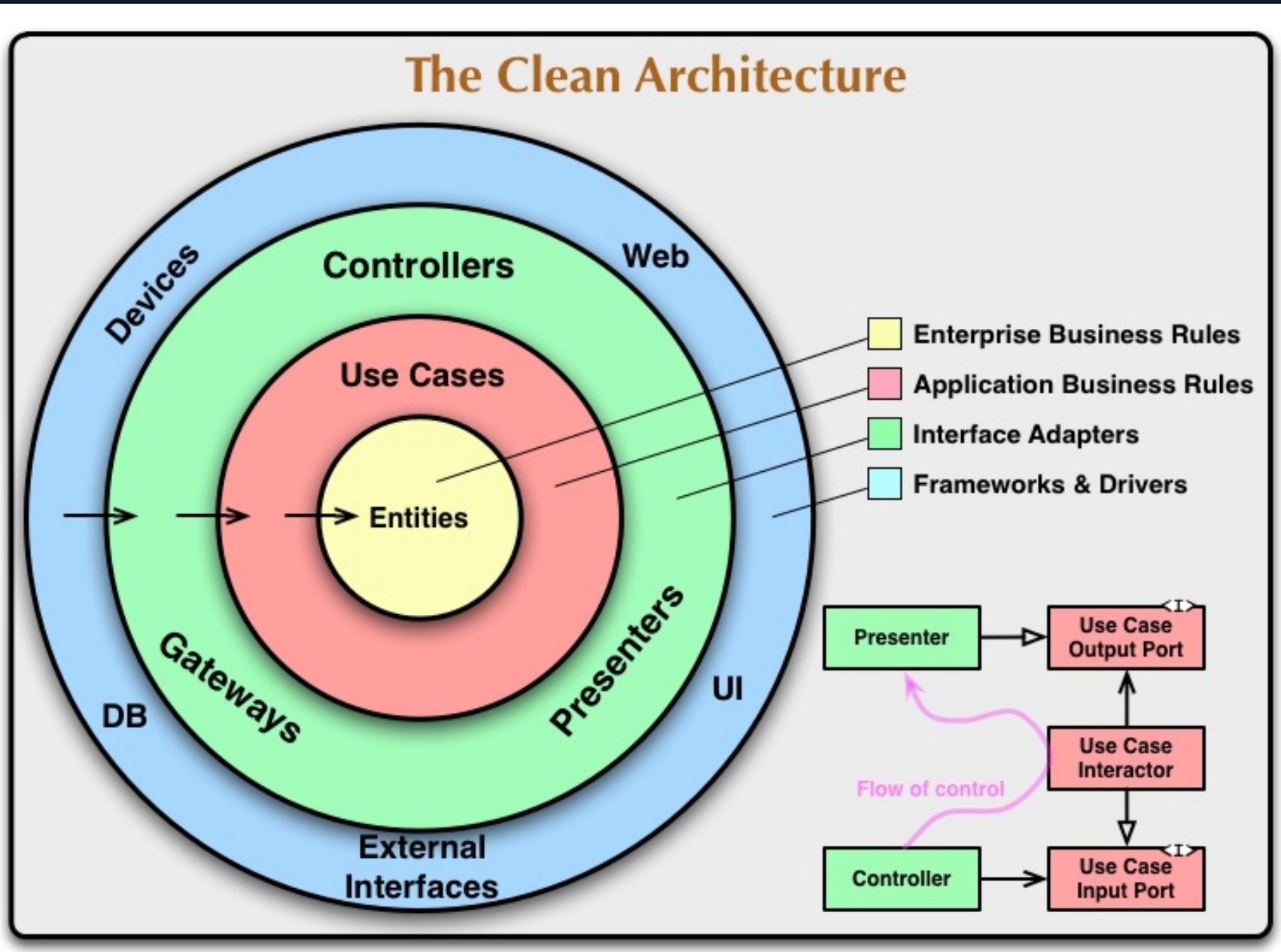
지엽적인 문제들은 미뤄두자

아키텍처의 목적

SOC(Separate Of Concerns)

소스코드 의존성은 반드시 안 쪽으로, 고수준의 정책을 향해야 한다

DTO, 내부의 원에서 사용하기 편하게


The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
<I>
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port
<I>

Humble : 보잘 것 없다

Humble Object Pattern

테스트 용이성

테스트하기 어려운 행위와 쉬운 행위의 구분

테스트 하기 어렵다 = Humble

아키텍처 경계

완벽한 경계 = Too much cost

이건 lean하지 못한 방식 아닌가요?

부분적 경계를 구현하자

부분적 경계

마지막 단계를 건너뛰기

일차원 경계

퍼사드