

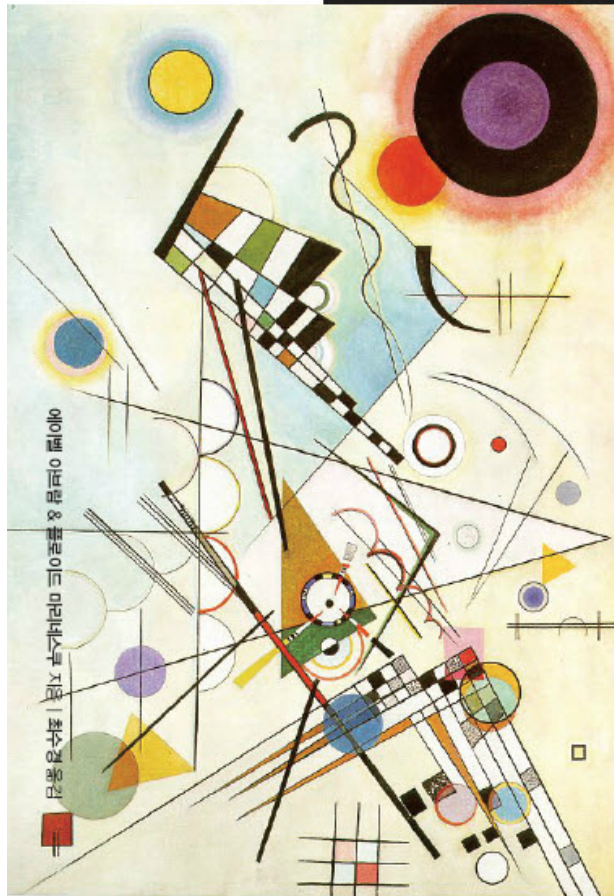
에이벨 이브람 & 클로이드 미라네스루 지음 | 최수경 옮김

Domain
Driven
Design
Quickly

도메인 주도
설계란
무엇인가?

| 쉽고 간략하게 이해하는 DDD |

DDD의 개념을
간략히 소개한
얇은 책입니다.



Domain
Driven
Design
Quickly

도메인 주도
설계란
무엇인가?

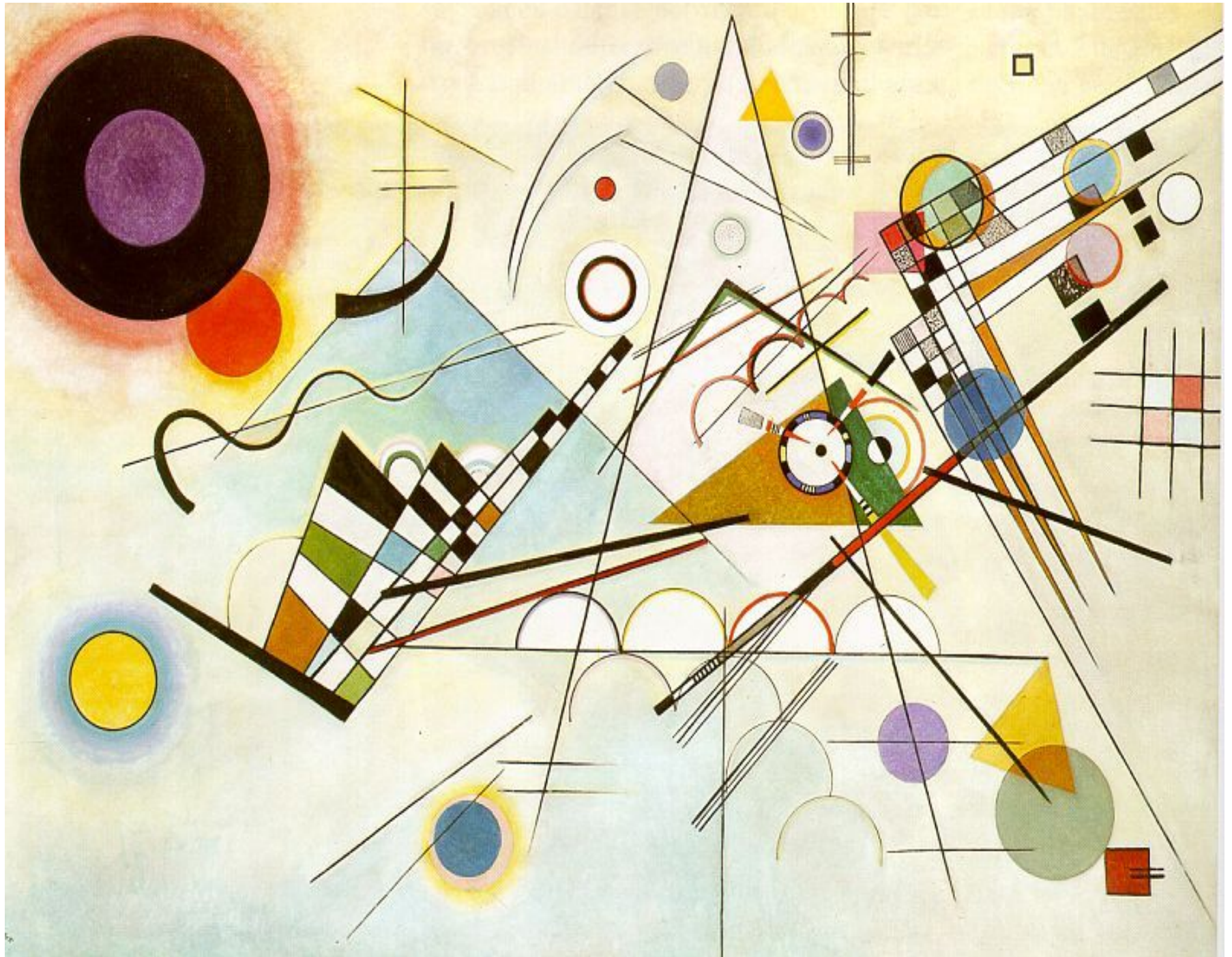
| 쉽고 간략하게 이해하는 DDD |

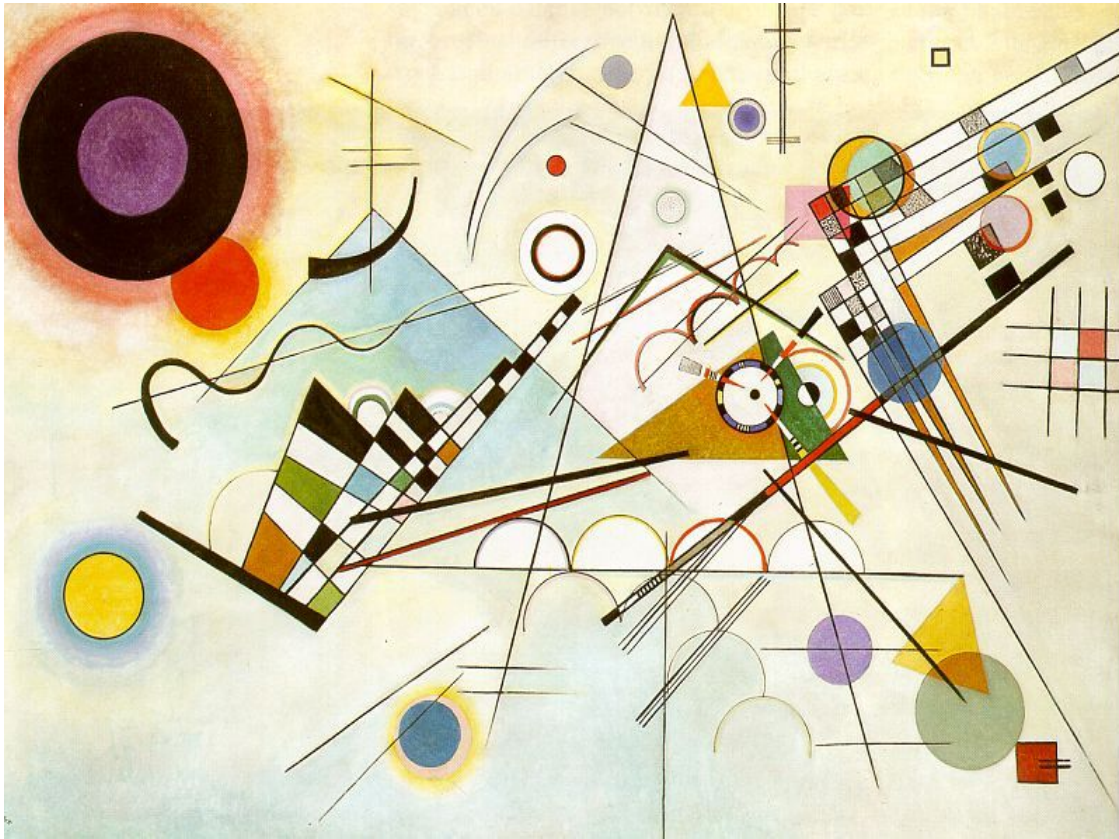
저는 그책의
번역자 이구요.

LG CNS 근무

모델러->강사->QA
기술사/감리사

sookchoi93@gmail.com
cskstory.tistory.com





칸딘스키의 <구성8>

왜 칸딘스키 그림을 표지에 쓴 것일까요? Why This cover?라는 페이지에는 다음과 같은 이유가 있었습니다.

- 혼란과 복잡성 속에서 질서 찾기
- 추상화된 언어 만들기
- 이 추상들을 작업에 포함하기

도메인 주도 설계란 무엇인가?

Domain Driven Design Quickly

소프트웨어의 목적은?

- 현실 세계의 프로세스를 자동화 하거나
비즈니스 문제를 해결하기 위함

도메인이란?

- 자동화된 비즈니스나 현실세계의 문제

소프트웨어와 도메인은 떼려야 뗄 수 없는 관계!

그 관계가 강할 수록 좋은 소프트웨어가 만들어질 가능성이 높음!

소프트웨어와 도메인 사이에는?



모델이 존재합니다.

그런데, 우리가 종종 이 모델을 세분화 하면서...



결과적으로 도메인과 소프트웨어를 멀어지게 합니다.

그래서, 도메인 주도 설계란?

모델이 그 가치를 잃지 않고
소프트웨어 개발에 기여하도록

1. 도메인을 잘! 표현한 모델을 만들고
2. 함께! 끝까지! 그 모델을 바라보자

는 것입니다.

그러자면....

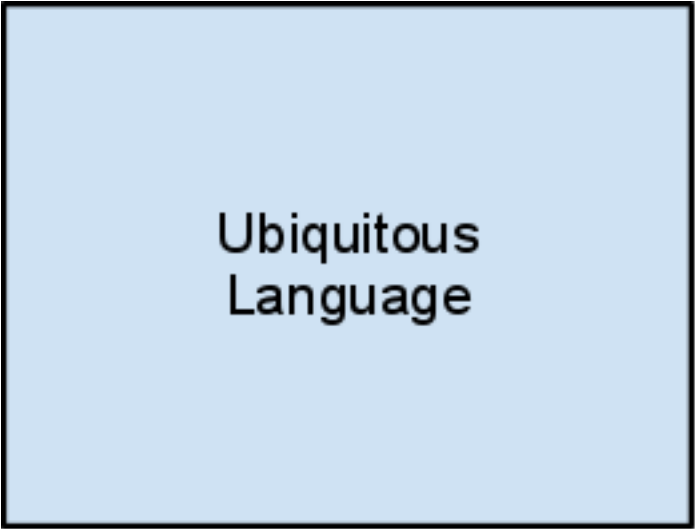
- 모든 사람이 모델 생성에 참여해야 하고
(도메인 전문가, 분석가, 아키텍트, 개발자...)

- 모델은 여러사람이 늘 검증해야 하며
 - 도메인 전문가가 이해할 수 있는가?
 - 모르는 사람이 모델을 보면서 대상 도메인을 이해할 수 있는가?
 - 모델이 구현 가능한가?

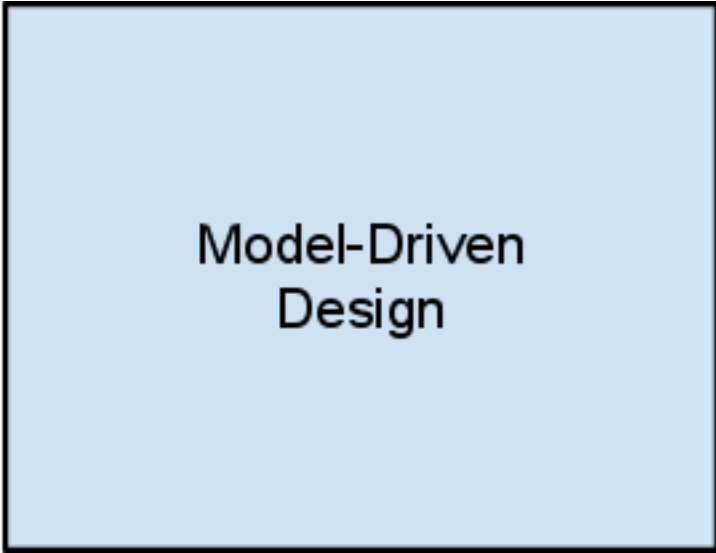
- 모든 사람이 모델에 나타나 있는 같은 언어(용어)를 사용해야 하며
(도메인의 용어를 그대로!)

- 해당 도메인의 정수(핵심!)만이 표현돼야 합니다.

요약하면 이렇게!

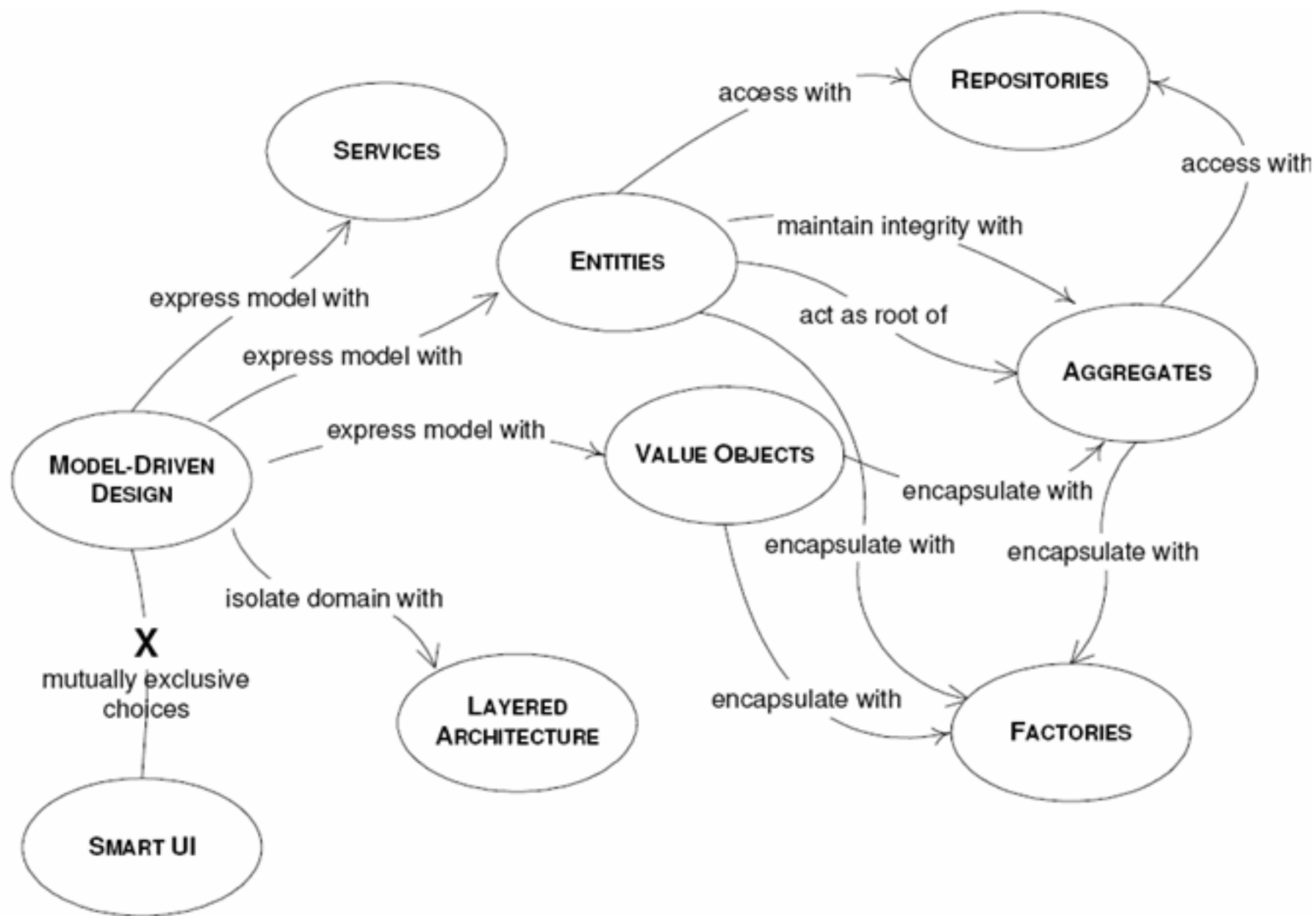


Ubiquitous
Language

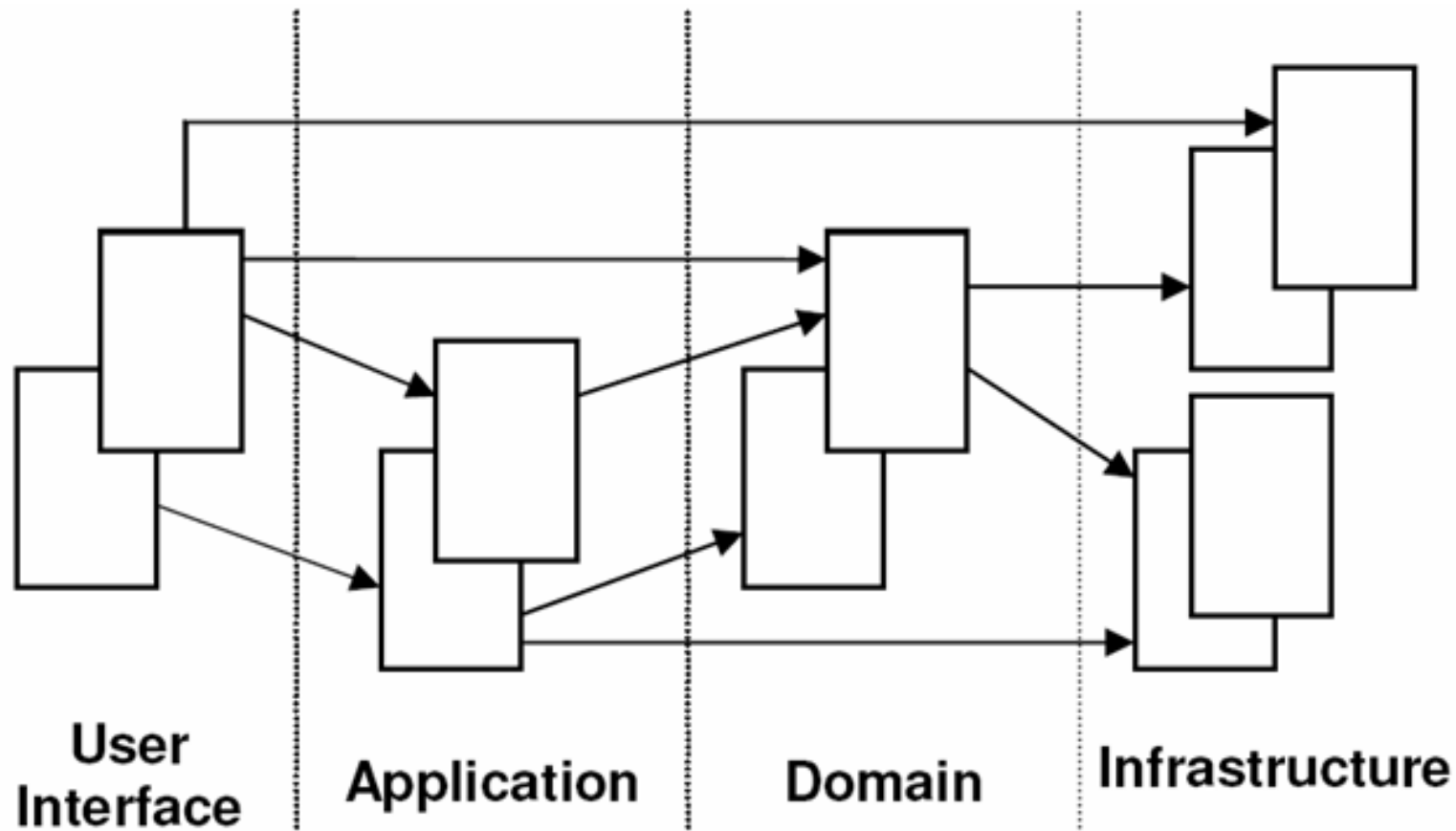


Model-Driven
Design

모델 주도 설계를 위해서는
아래의 패턴을 적용하는 것을 권합니다.



계층구조(Layered Architecture)의 일반적인 모습입니다.



패턴에 대해 간단히 설명하면 다음과 같습니다.

- 엔티티 (Entity)

- 식별자를 가지며 영속성이 필요한 객체

- 값객체 (Value Object)

- 식별자가 없으며 영속성이 필요없는 객체
- 수정할 수 없고 필요한 만큼 복제하여 전달하여 사용

- 서비스 (Service)

- 특정 엔티티나 값객체에 속할 수 없는 도메인의 개념 표현
- 주로 여러객체에 걸쳐서 일어나는 행위를 담당
- 상태정보를 관리하지 않음

- 집합 (Aggregation)

- 객체의 소유권과 경계를 정의
- 데이터를 변경할 때 하나의 단위로 간주되는 관련된 객체들의 집합
- 외부에서 접근할 수 있는 단하나의 창구인 root를 가짐

- 팩토리(Factory)

- 복잡한 객체 생성의 절차를 캡슐화
- 단순한 경우라면 생성자를 사용

- 리파지토리(Repository)

- 객체의 저장을 담당
- 이미 존재하는 도메인 객체의 참조를 얻는 로직을 캡슐화

도메인 모델은 아래와 같은 개념을 적용해 관리하게 됩니다.

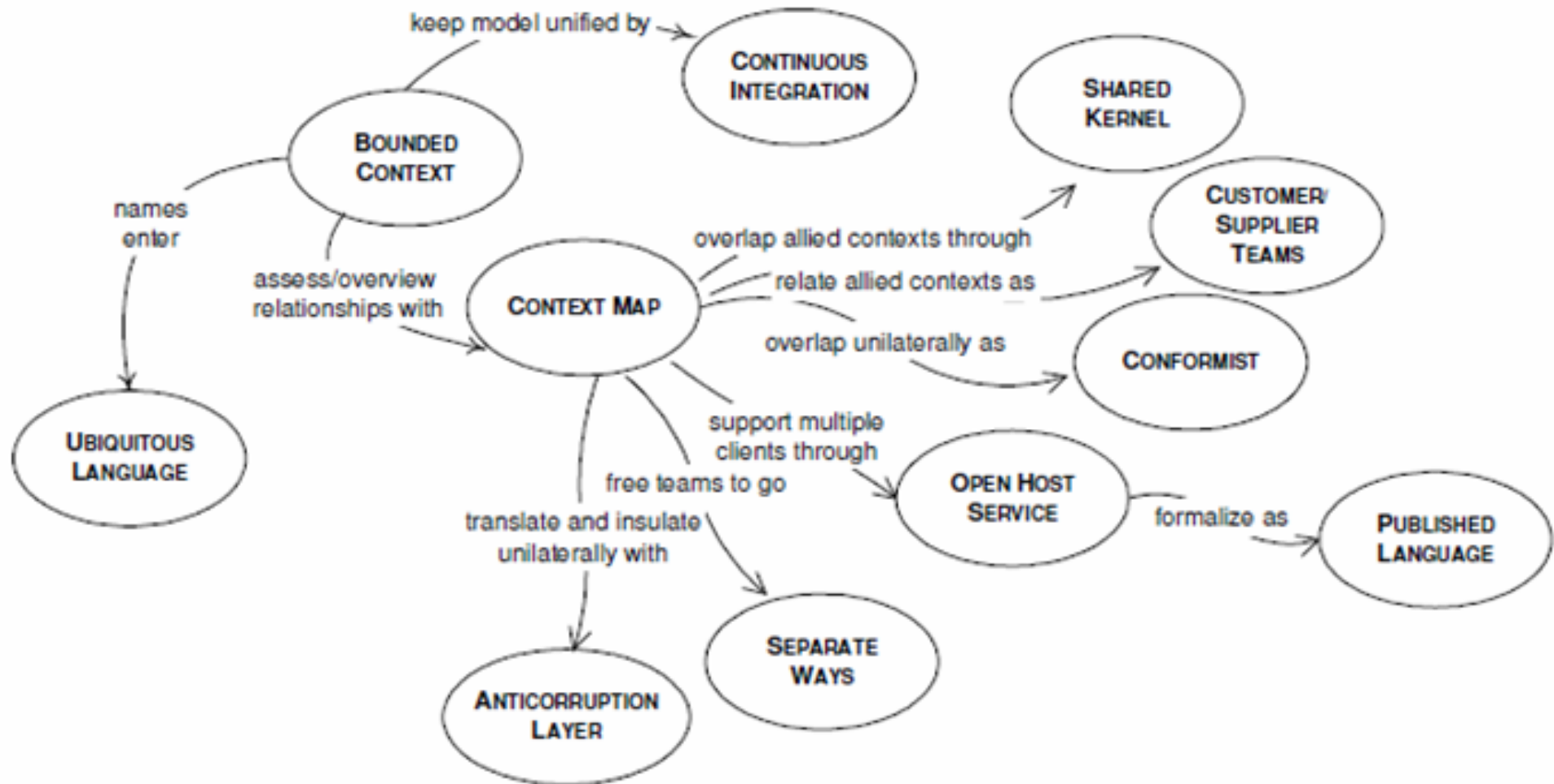
- 모듈

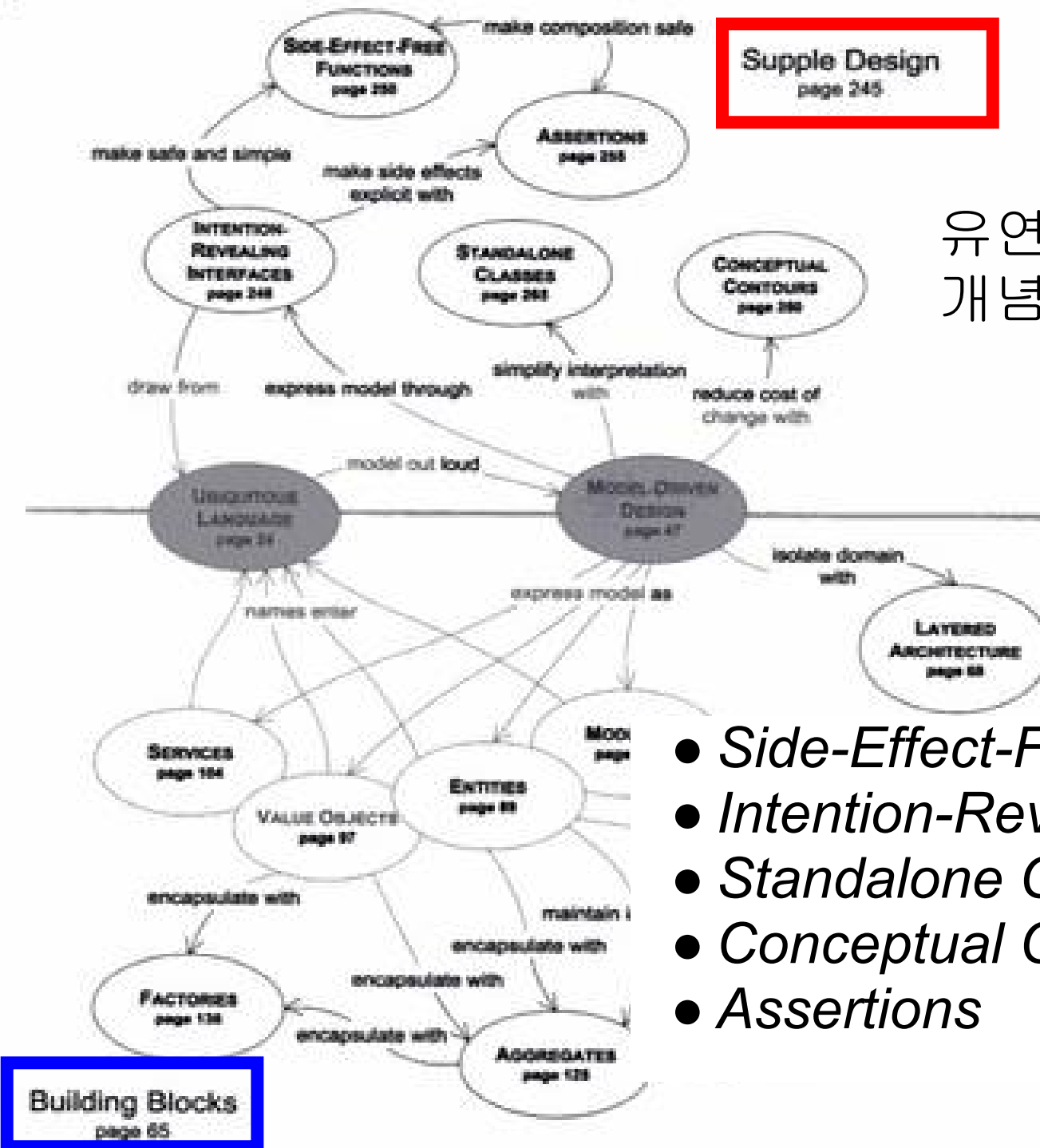
- 결합도 낮게, 응집도 높게

- 리팩터링

- 코드 리팩터링 뿐 아니라, 모델 리팩터링이 중요
- 핵심 개념을 드러나게 하는 고민필요
- **Breakthrough** - 도약의 시점이 온다!
- 언어를 주의깊게 듣고 찾아낸다.
- 분명하지 못한 설계영역을 다시 살펴본다.
- 지나치게 비대한 객체를 의심해본다.
- 상충되는 것을 조화시키기 위한 노력을 해본다.

모델 무결성 보존하기 위한 개념으로
Context를 구분하여 관리하는 것을 권합니다.

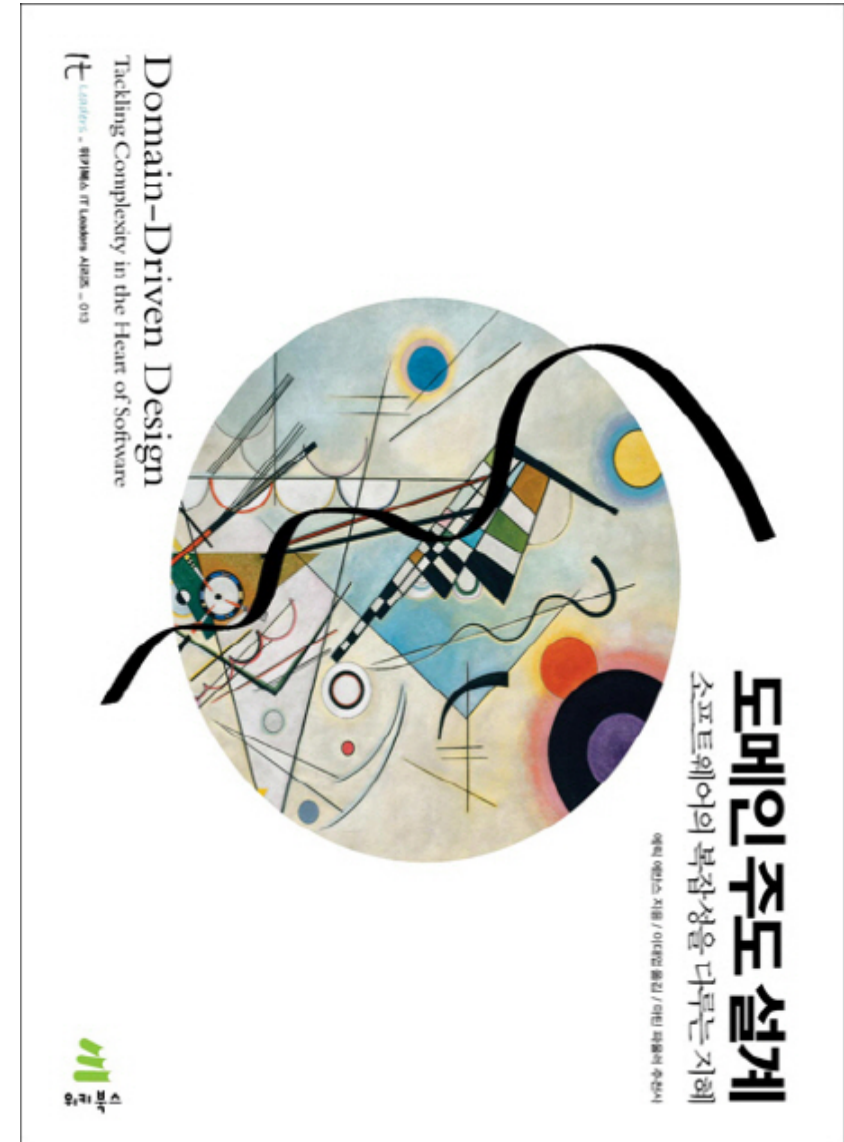
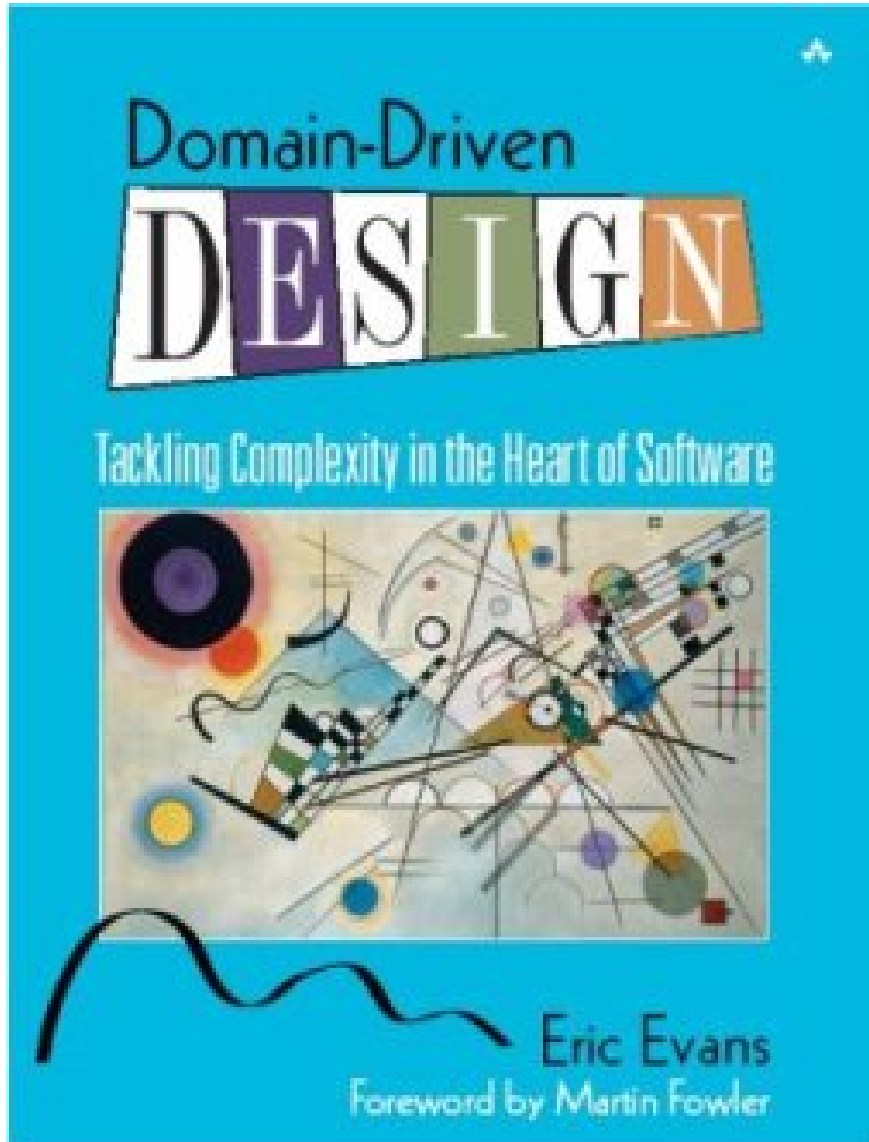




유연한 설계를 위한
개념도 존재합니다.

- *Side-Effect-Free Functions*
- *Intention-Revealing Interfaces*
- *Standalone Classes*
- *Conceptual Contours*
- *Assertions*

DDD에 대한 자세한 내용은 아래 책을 참고하세요.



DDD를 적용할 때는

DDD는 폭포수 개발 방식으로도 가능
그러나 조금씩 발전시켜 나가는 Agile 방식이 더적합

DDD는 절차적 언어로도 가능
그러나 책임을 분할하는 객체지향 언어에 더 적합

DDD 모델은 UML로도 가능
그러나 양이 많아 지거나 행동,제약사항 등의 표현에 한계
DSL(Domain Specific Language) 출현

아래 질문 모두에 Yes 라면 DDD를 적용한 것입니다.

- 모두가 이해하는 도메인 모델이 존재하는가?
(Yes / No)

- 모든 이해당사자가 의사소통에 공통적인 언어(정의된 용어, 즉 Ubiquitous Language)를 사용하는가?
(Yes / No)

- 소프트웨어가 Model Driven Design(모델 수준으로 추상화되어 코드와 연동하는) 방식으로 개발되고 있는가?
(Yes / No)

감사합니다. ^^

Qustion & Answer