

Architecture

Daum Corp.
백명석

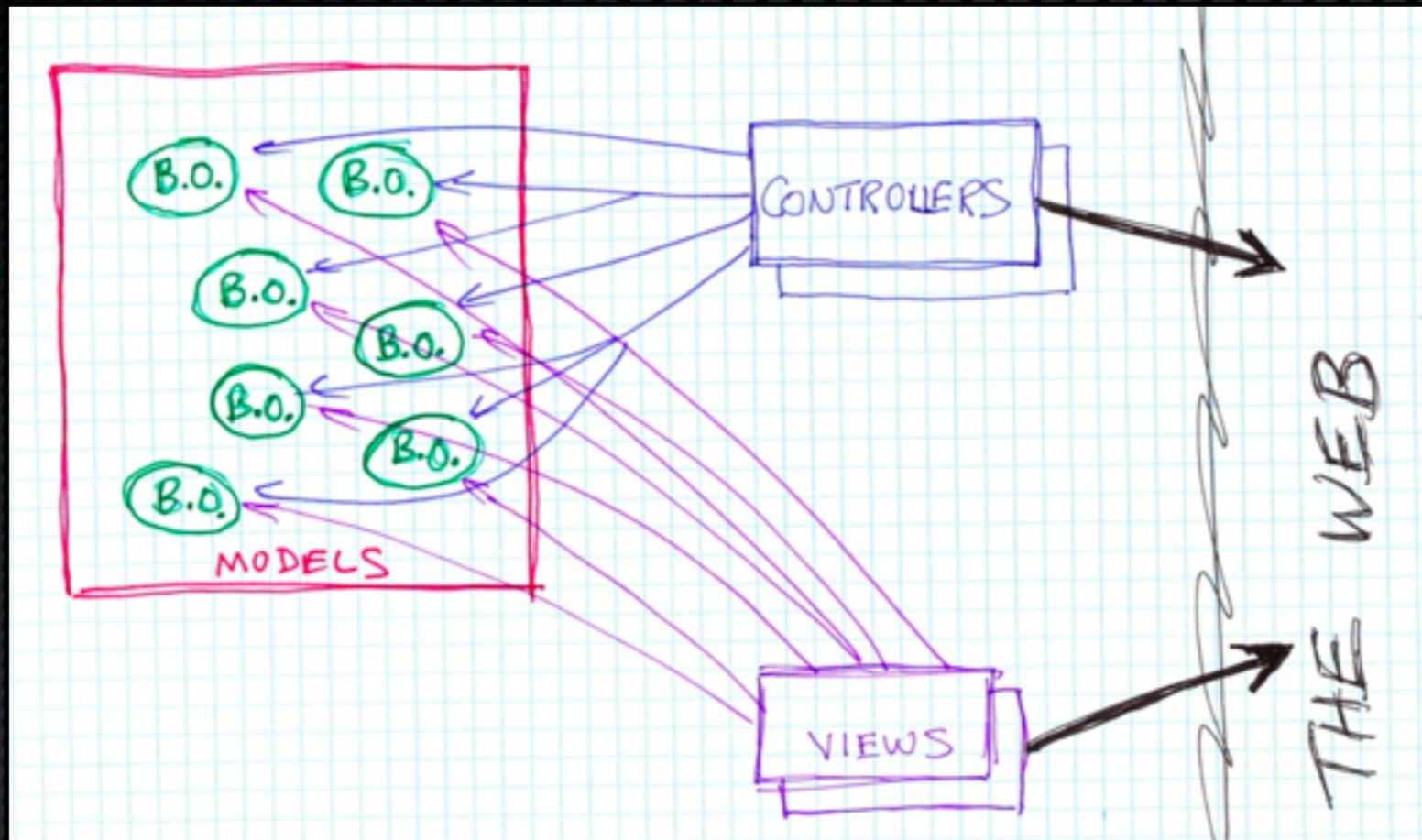
발표목차

- Architecture
- MVC Architecture
- Accounting System Architecture
- Use Cases
- Use Cases Driven Architecture
- Use Case Algorithm
- Partitioning
- Case Study
- Conclusion
- Who is Architect ?

Architecture

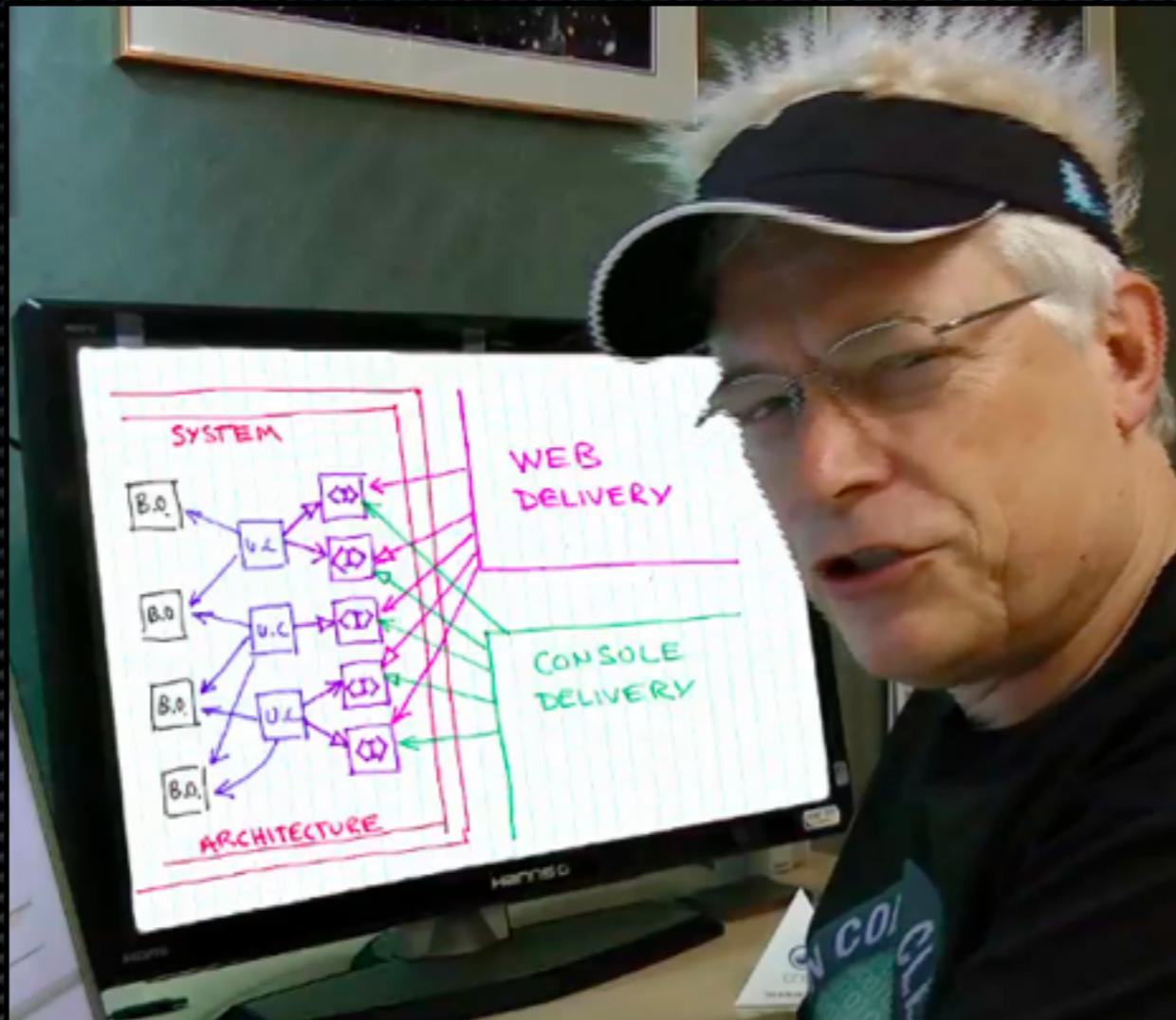
- Web 기반 Accounting 시스템의 아키텍처에서 주목 할 부분
 - Accounting 시스템 ?
 - Web 시스템 ?
- Accounting 시스템이라는 것이 중요
- SW 아키텍처
 - Accounting Issue를 드러내야
 - Web에 대해서는 거의 언급하지 않아야
- 하지만 대개의 Web 시스템은 반대
 - Web Issue에 대해서 고함치고,
 - 비즈니스 의도에 대해서는 거의 언급하지 않음

Web System에 만연하는 MVC Architecture



- View/Controller: 강하게 html과 연관
- Model: Controller에 강하게 연관
- View/Controller: 강하게 모델에 coupling
 - 모델의 구조에 많은 영향을 미친다.

Accounting System Architecture

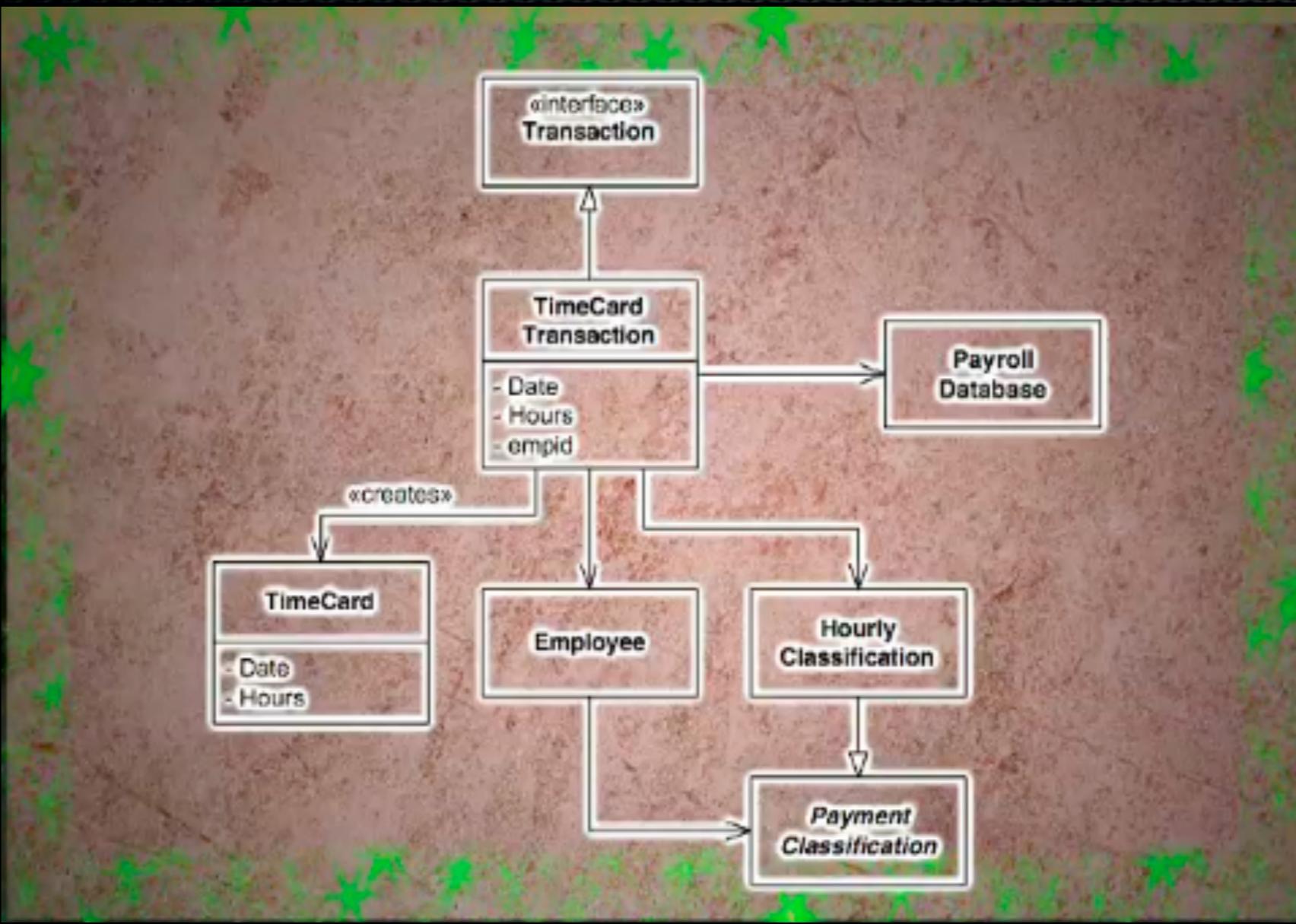


- Architecture 변경 없이 delivery 메커니즘을 변경할 수 있어야
 - 동일한 시스템을 웹과 콘솔에 deliver할 수 있어야
 - 이 두 시스템의 아키텍처는 동일해야

Use Cases

- “Object-Oriented Software Engineering - A Use Case Driven Approach”, Ivar Jacobson
 - Delivery 문제를 Elegant Architectural Solution을 통해 해결
 - Delivery와 무관한 방식으로 사용자가 시스템과 상호작용하는 방식을 이해하는 것
 - 링크, 버튼, 클릭 등의 용어를 사용하지 않고 표현
 - Delivery 메커니즘을 나타내지 않는 용어를 사용
 - 제이콥슨은 이런 상호작용을 **Use Case**라고 했다.
 - The development of application should driven by these delivery independent use cases.
 - use case가 시스템에서 가장 중요한 것

Use Case Driven Architecture



- use case driven 시스템의 아키텍처를 보면 delivery 메커니즘이 아닌 use case를 보게됨
 - What you see is the **intent of the system**.

Use Cases

- Formal description of how user interacts with the system in order to achieve specific goal.
- Goal: Create Order within an order processing system

Create Order

Data:

```
<Customer-id>, <Customer-contact-info>,  
<Shipment-destination>, <Shipment-mechanism>,  
<Payment-information>
```

Primary Course:

1. Order clerk issues "Create Order" command with the above data
2. System validates all data
3. System creates order and determines order-id
4. System delivers order-id to clerk.

• Order clerk gets order-id from system.

• Order clerk receives order and associates order-to

Use Cases

- screen, button, field 등과 같은 웹(delivery 메커니즘)과 같은 것들은 언급하지 않음.
- 시스템으로 들어가는 데이터, 커맨드와 시스템이 응답하는 것만 언급
- Delivery 메커니즘과 무관한 아키텍처를 가지려면 delivery 메커니즘과 무관한 use case로 시작해야
- Use case의 응답도 주목
 - order-id는 완전하게 delivery 메커니즘과 무관
 - human clerk은 order-id를 볼 필요가 없다.
 - 반면 delivery 메커니즘은 팝업을 띠워서 사용자에게 항목을 주문에 추가하라고 요청할 수 있다.

Use Cases

- Notice that the use cases is a essentially algorithm for the interpretation of the input data and generation of output data.
- Use case를 구현하는 객체를 생성할 수 있다는 것을 의미
- 이 예제는 primary course만 있는데 예외상황도 있을 수 있다.

Use Case Algorithm

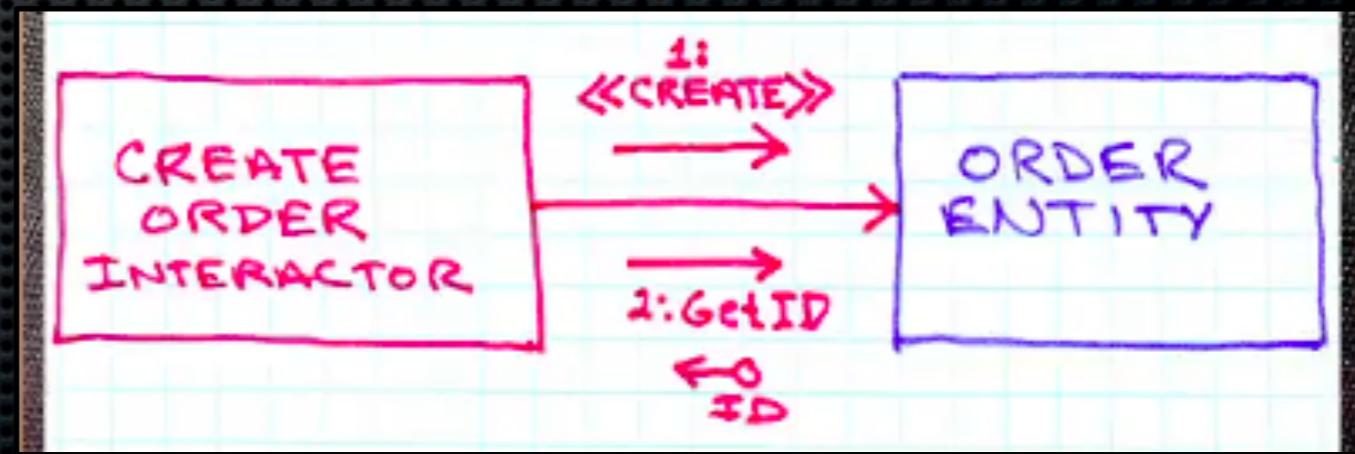
- 다른 비즈니스 객체들(Customer, Order)을 언급
- 알고리즘: use case 정의, 비즈니스 규칙을 내포
- 하지만 이런 비즈니스 규칙은 Customer나 Order 객체에 속하지 않는다.
- 그럼 어디에 비즈니스 규칙을 내포시킬 것인가 ?
- 어떤 객체에 위치시킬 것이며 Use Case 객체를 아키텍처의 어디에 위치시킬 것인가 ?
- 어떻게 우리의 시스템을 partition해서 use case가 central organizing principle이 되게 할 것인가 ?

Partitioning

- 야콥슨: 아키텍처는 3개의 fundamental kinds of object들을 갖는다
 - Business Objects - Entities라고 불리는
 - UI Objects - Boundaries라고 불리는
 - Use Case Objects - Controller라고 야콥슨이 부른, 하지만 MVC와 혼동을 피하기 위해 Interactors라고 부를

Partitioning

- Entities have:
 - Application independent business rules
 - 다른 application에서도 entity들은 사용됨
 - 특정 Application에 종속적인 메소드들을 갖으면 안됨
 - 이런 메소드들은 Interactor 객체로 옮겨야
- Interactors have:
 - Application specific business rules
 - 특정 Application에 종속적인 메소드들은 Interactor 객체에 구현

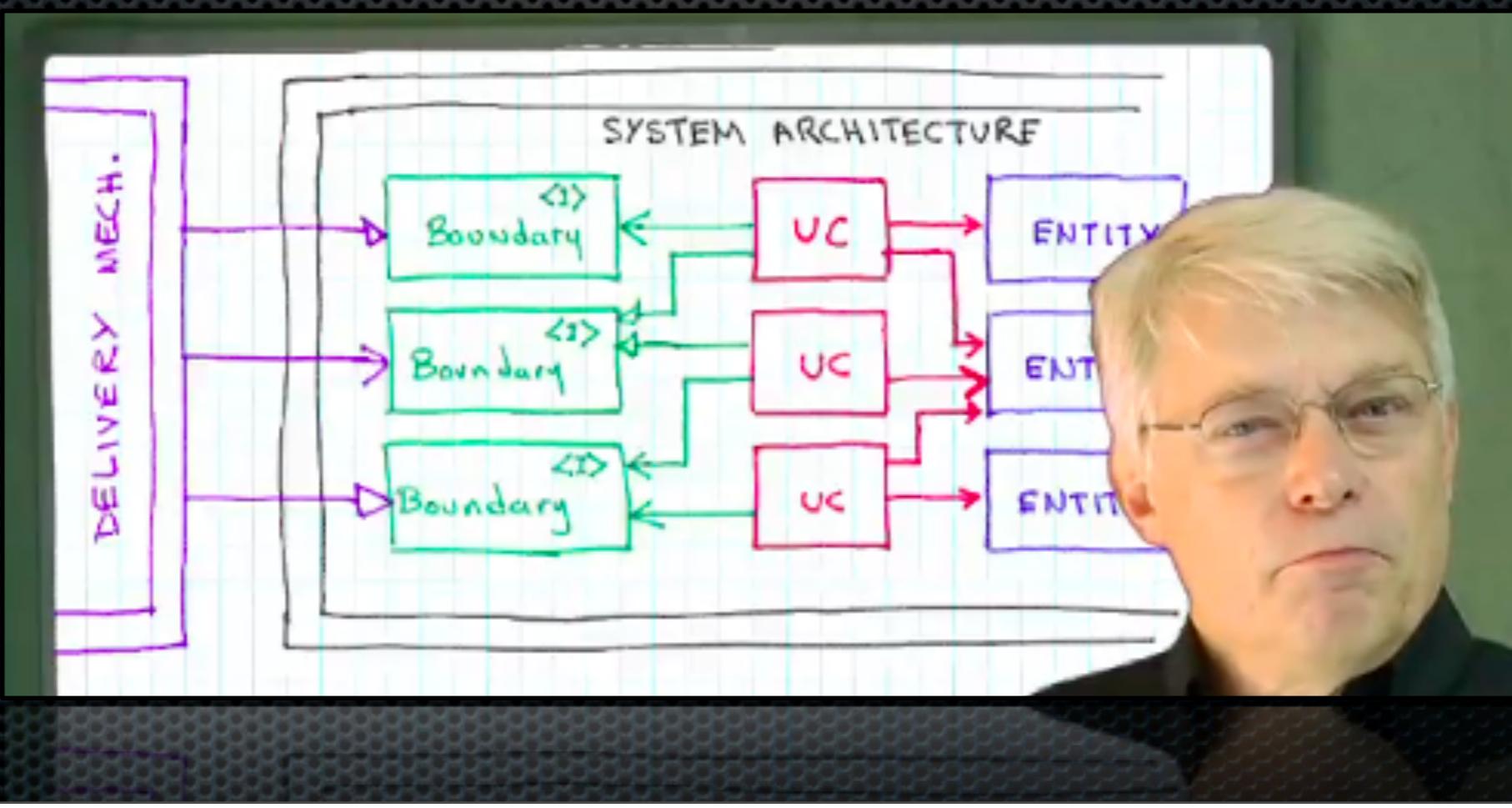


Partitioning

- Interactor achieve their goals with application specific logic.
 - They calls application agnostic logic within the entities.
 - ex. create order interactor invokes the both the constructor and get id method of the order entity.
 - 이 두 메소드는 application agnostic.
 - it's the interactor that knows how to call these methods to achieve the goals of the use case.

Partitioning

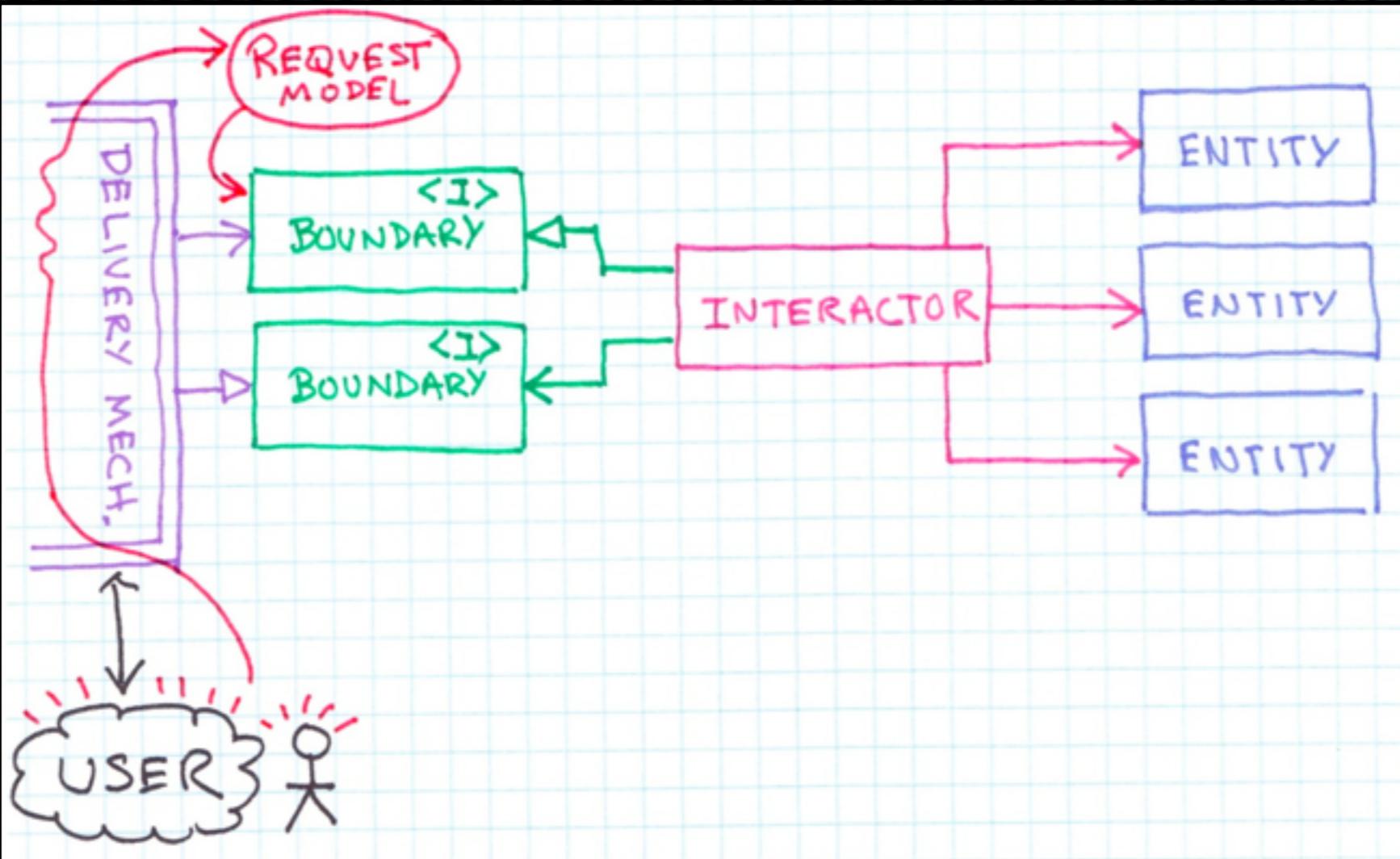
- One of the job of the use case is to accept input data from user and deliver out data back to the user.
- This is the job of third kind of object.
 - boundary object.



Partitioning

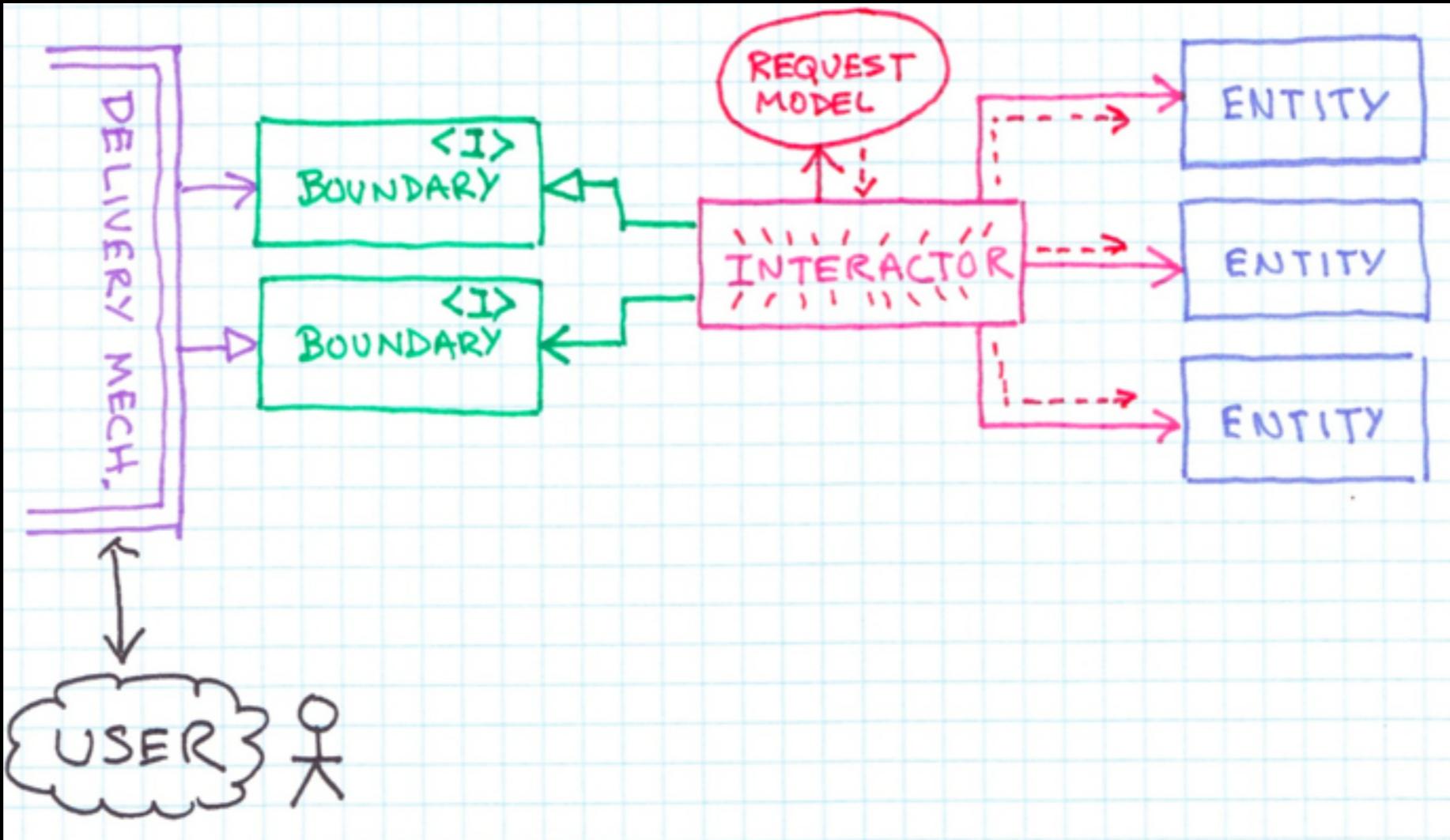
- Boundary object
 - isolate the use cases from the delivery mechanism
 - provide the communications path way between the two.
 - If you got a MVC system or a console system or thick client system all of that delivery stuff is on the far side of the boundary.
 - The use cases are on the other side know nothings about it.

Partitioning - Flow



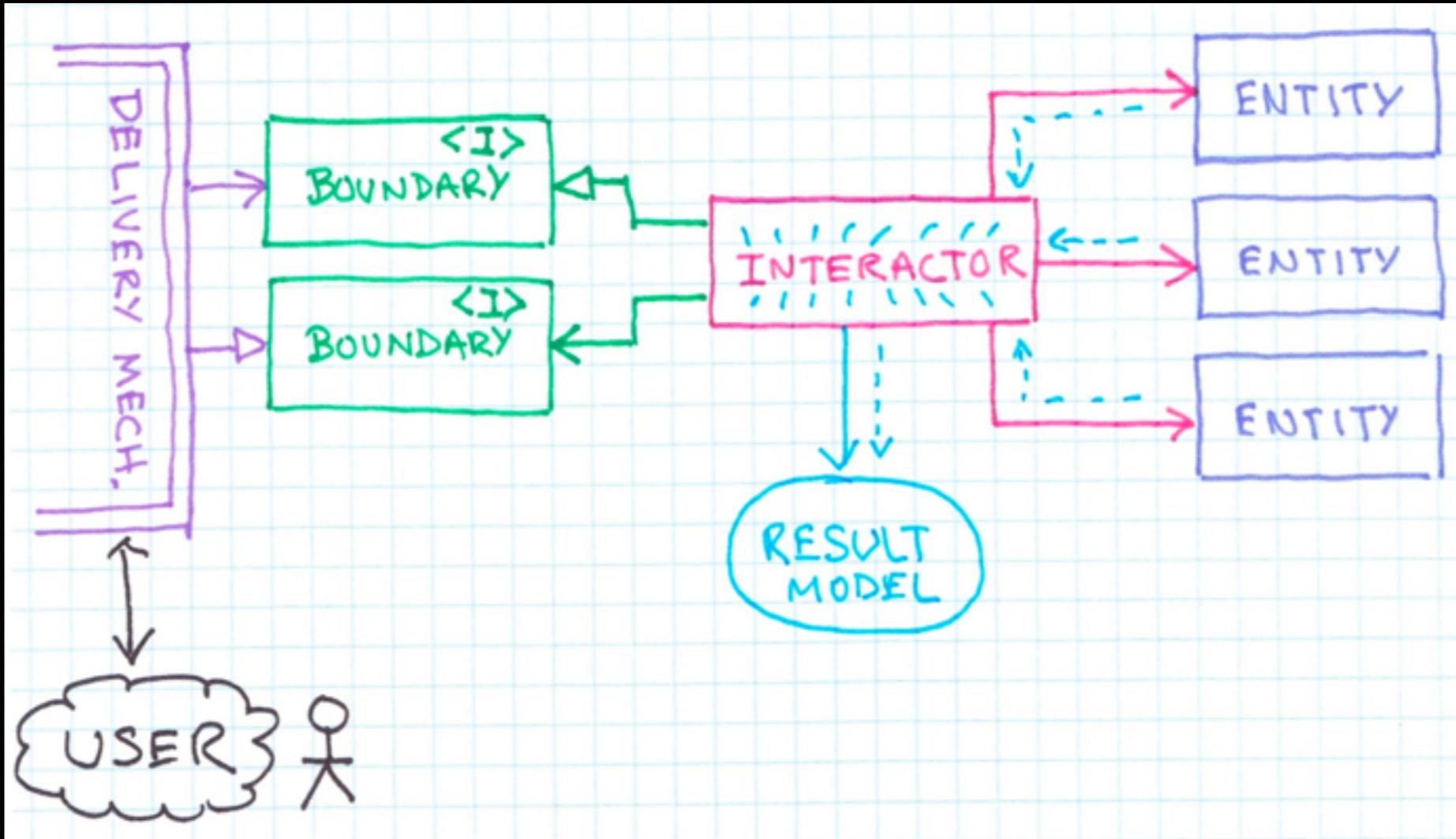
- The delivery mechanism
 - gathers user data
 - wraps it up into nice little canonical form(RequestModel)
 - ships it through the boundary to the interactors.

Partitioning - Flow



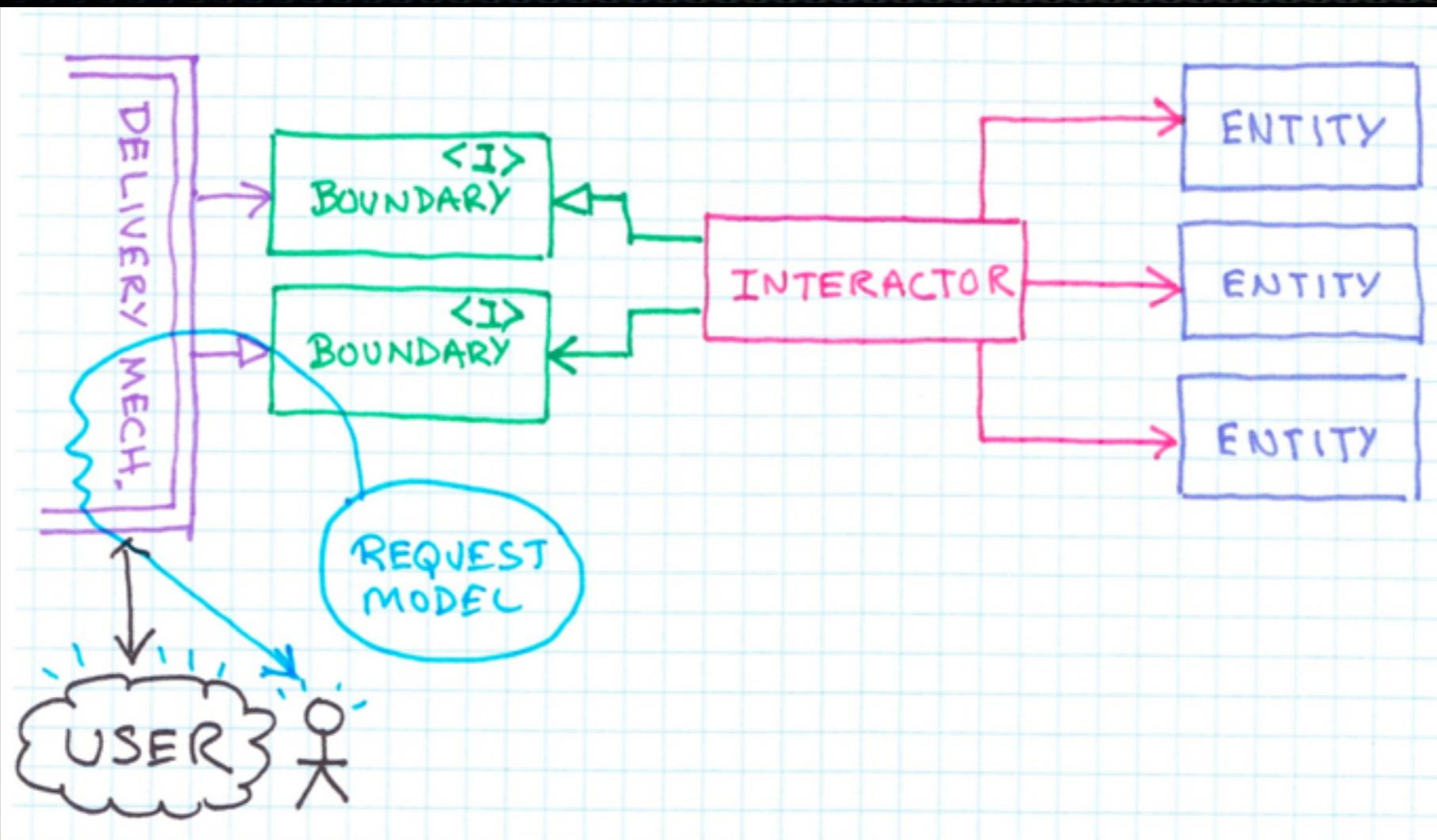
- Then the interactors
 - invoke their applications specific business rules.
 - go on manipulate the entity objects and their application agnostic business rules.

Partitioning - Flow



- Then the interactors
 - finally gathers the results they already gather
 - wrap it up into a nice little canonical form
 - ResultModel

Partitioning - Flow

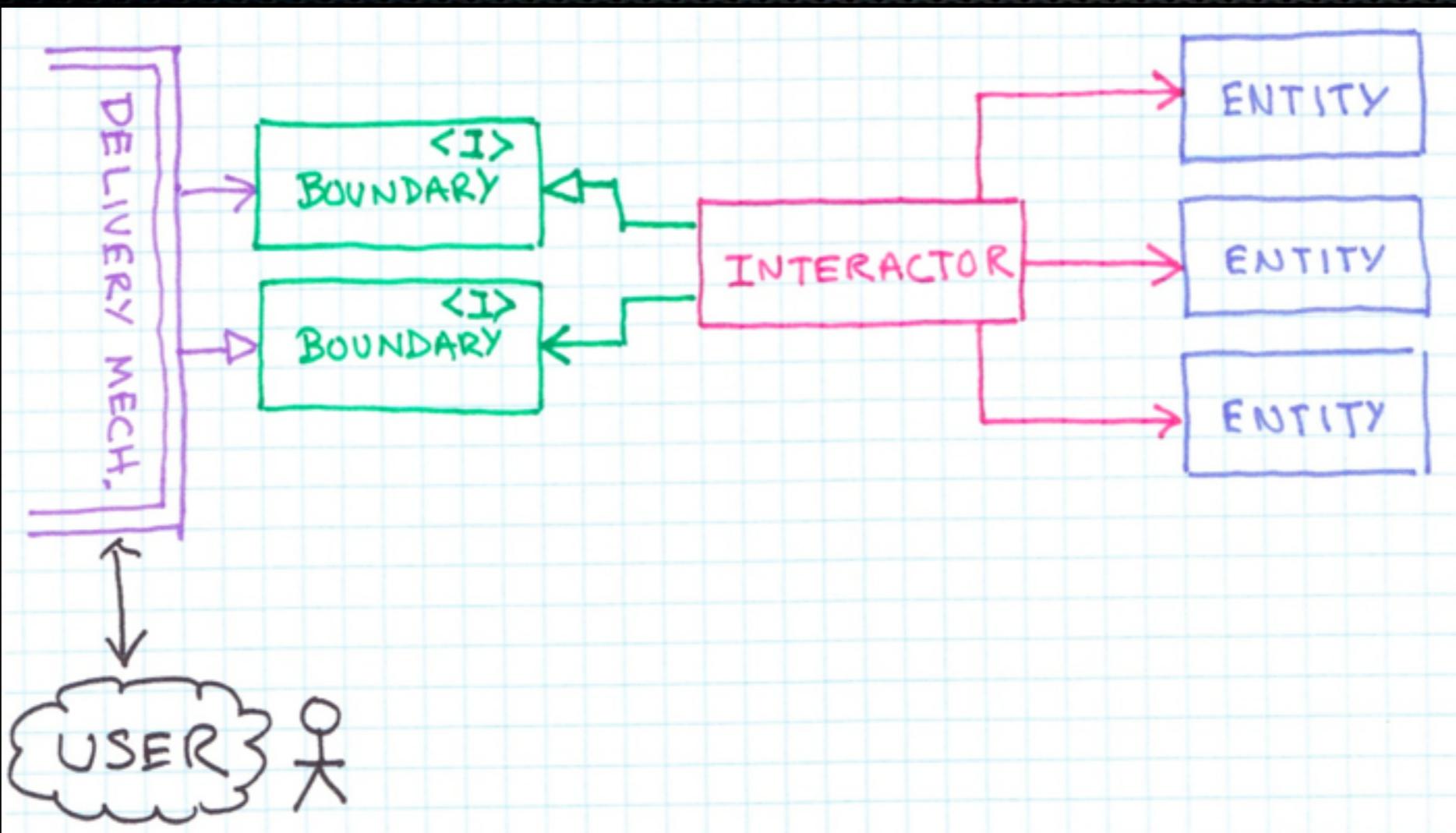


- And the interactors
– ship it back through the boundaries to the delivery mechanism.

Use Cases and Partitioning

- 우리는 시스템의 행위를 use case로 기술한다.
 - Use case에서 application specific한 행위를 interactor 객체로 캡쳐한다.
 - application agnostic 행위를 entity 객체로 캡쳐하고 interactor로 제어한다.
 - UI 종속적인 행위는 Boundary 객체로 캡쳐하여 interactor와 커뮤니케이션한다.

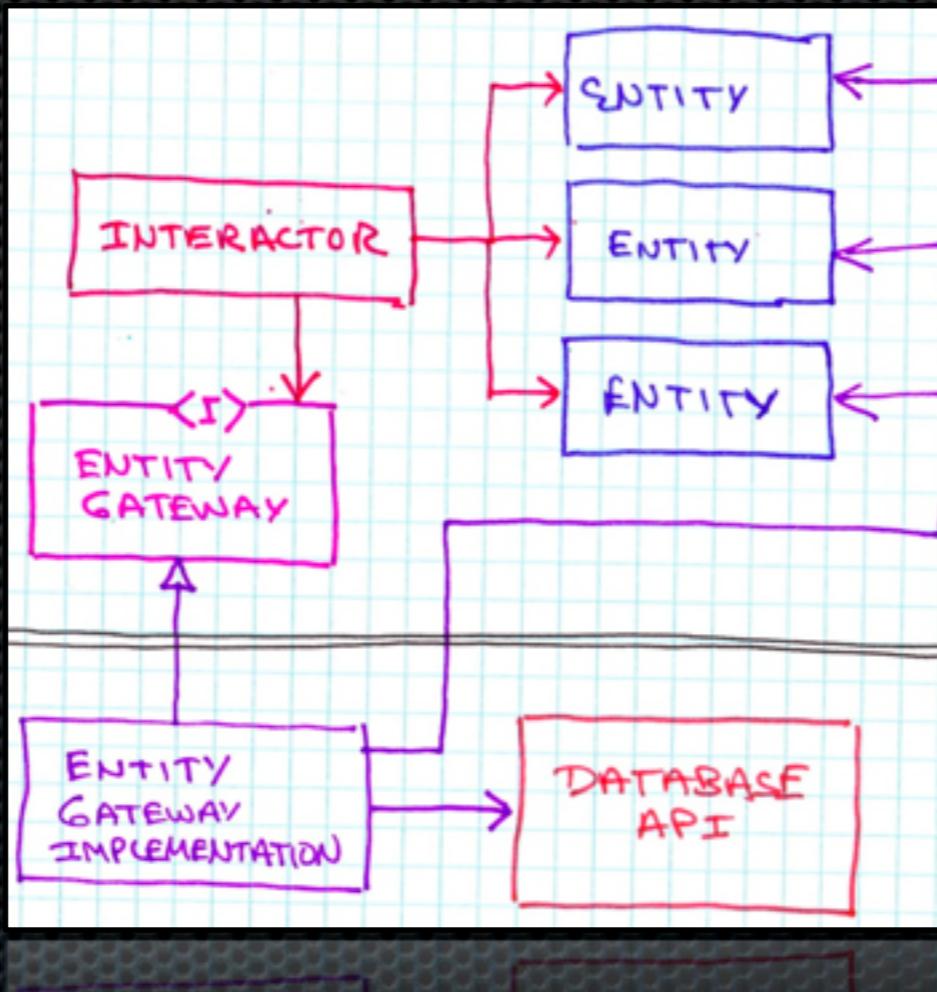
Isolation



- 소스 코드 의존성은 하나의 방향만 유지해야
 - Delivery mechanism을 Decouple하기 위해

Database

- 비즈니스 객체가 아니라 Data Structure를 contain
- 데이터가 저장되는 방식은 Interactor나 Entity가 원하는 방식이 아님
- 그러므로 DB와 Entity 간의 Boundary Layer를 제공해야 함 - Dependency Inversion Principle



Case Study

- “Agile Software Development - Principles, Patterns and Practices”
 - SOLID, Design Patterns, Agile Practices
- Payroll System
 - <http://www.objectmentor.com/PPP/>

The Stakeholder Meeting

- Hourly Employee
 - paid hourly base.
 - submit time card at the end of everyday.
- Sales Employee
 - Commission Base
- Regular Employee
- Union Employee
- Web Based. DB. Web 2.0. Ajax. SOA. By Next Month. Agile Method.

Analysis

- Web 2.0, Ajax, SOA, DB, GUI 등은 무시 - Details
 - We gonna treat all that stuffs as things we can safely defer.
 - It will be the goal of our architecture to make all those things irrelevant.
 - Which means easy to add later if necessary.

Analysis

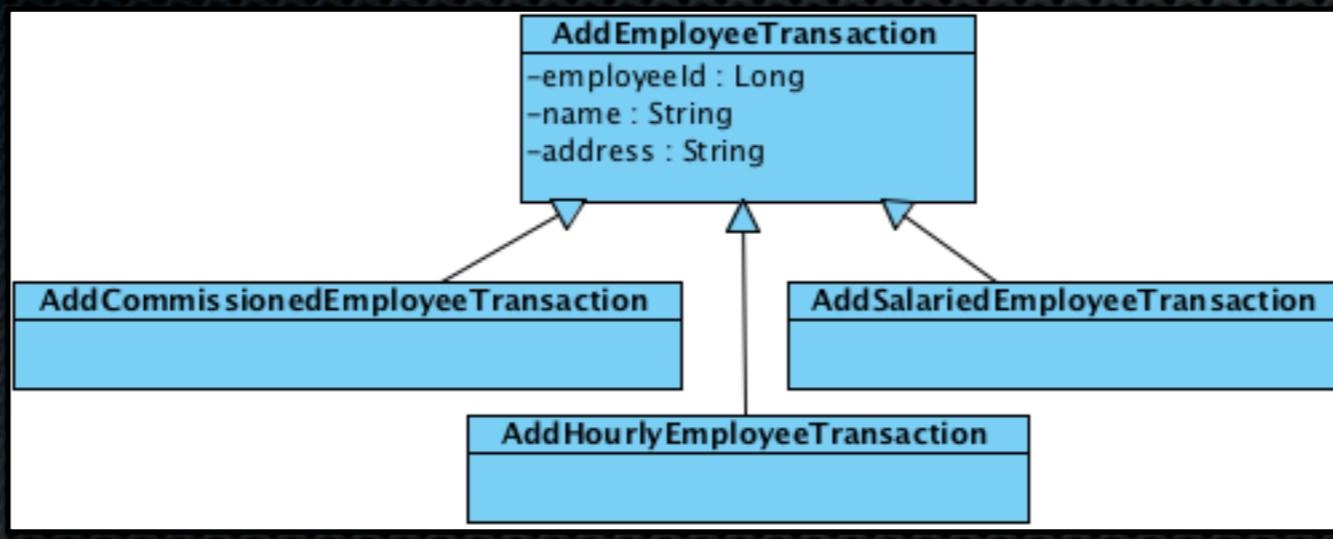
• 물론 Use Case로 시작. - Add Employee

```
1 A new employee is added by the receipt of an AddEmp transaction.  
2 This transaction contains the employee's name, address and assigned employee number.  
3 The transaction has three forms:  
4     AddEmp <EmpID> "<name>" "<address>" H <hourly-rate>  
5     AddEmp <EmpID> "<name>" "<address>" S <monthly-salary>  
6     AddEmp <EmpID> "<name>" "<address>" C <monthly-salary> <commission-rate>  
7 The employee record is created with its fields assigned appropriately.  
8 ┌  
9 Alternative 1: An error in the transaction structure  
10 If the transaction structure is inappropriate, it is printed out in an error message,  
11 and no action is taken.
```

• Notice

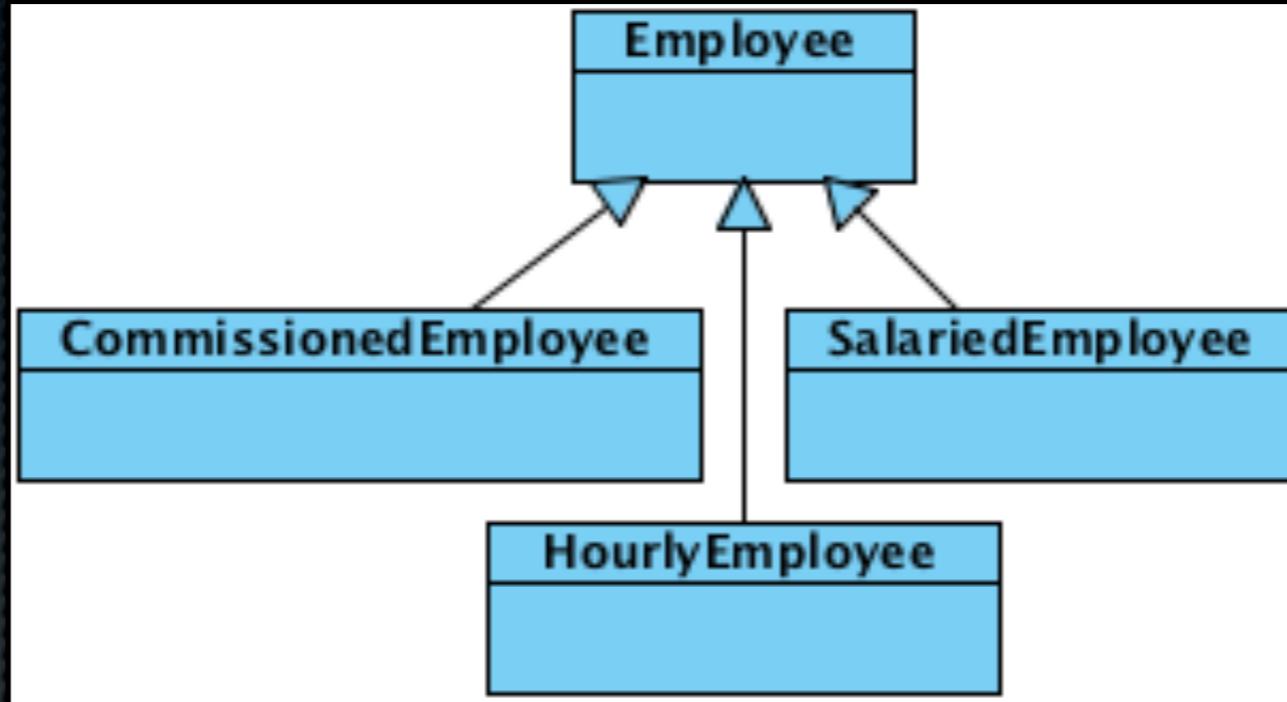
- 3개의 변형
- Use Case와 관련된 데이터들이 존재
- Employee type이 H/S/C로 encoding
- Details에 대한 언급 없음

첫번째 Use Case 설계



- Use case를 Transaction으로 매핑
- 3개의 변형이 존재
 - 3개의 변형에 대한 generic한 부분 캡쳐
 - 3개의 파생 클래스들은 다른 점들을 캡쳐

Entities



- Interactor는 Entities들과 커뮤니케이션함
 - Entities들은 비즈니스 규칙을 표현
 - 이 어플리케이션에서 어떤 종류의 비즈니스 객체들을 가져야 할까 ?
 - Base 클래스 하나와 3개의 employee type을 위한 각각의 파생 클래스(derivative).

Conclusion

- “Agile Software Development - Principles, Patterns and Practices”
- Architectures are not based on tools and frameworks.
- On the contrary good architecture allow you to defer the decisions about tools and frameworks for a very long time.
- In fact a good architecture maximize the number of decisions not made.
- A good architecture does not depend on delivery mechanism, hide the delivery mechanism rather than exposing it.

Conclusion

- If you look at the shape of the system you should not be able to tell that it's a web system.
- 시스템의 use case는 primary abstraction이고 central organizing principles of the system architecture이다.
- Architecture를 보면 UI가 아니라 system의 intent를 볼 수 있어야 한다.

Conclusion

- Use case oriented architecture를 만들기 위한 야콥슨의 Boundary, Interactor, Entity partition을 배웠다.
- Interactor가 use case를 encapsulate하고 entity는 비즈니스 객체를 encapsulate하고, boundary는 UI와 isolation을 제공한다.
- To achieve isolation, we create boundary of interfaces that separate the application from the delivery side.

Who is the Architect ?

- Many company established a high level technical role, architect.
- 가끔 이 역할은 기술적이라기 보다 정치적이거나 관료적이다.
- Sales나 management의 입장을 대변하거나 FW을 옹호하거나 한다.
- 이런 것들이 필요하긴 하지만 architecture랑은 별 상관이 없다.

Who is the Architect ?

- 회사들은 시니어 개발자를 architect로 승진시킨다.
- Architect이 코딩을 하는 것을 원하지 않는다. 좀 더 high level의 원가(디자인, 리뷰 등)를 하길 원한다.
- 이런 정의들은 Architect이 코딩을 안 한다는 것을 가정한다.
- 코딩은 Architect이 하기에는 아주 low level의 일이라고 생각한다.
- Architect don't code quickly become irrelevant.
- 코딩 잘 못하는 Architect의 설계는 무용지물

Who is the Architect ?

- 당신이 Architect이고 효과적으로 그 역할을 수행하길 원한다면
 - you should write code.
 - You should work right along side with programmers
 - Pair program with them
- 100%의 시간을 코딩에 쓸 필요는 없다.
 - 심지어 50%의 시간을 코딩에 쓸 필요도 없다.
 - 하지만 약간의 시간이라도 코딩을 해야 할 때면
 - you should code well.