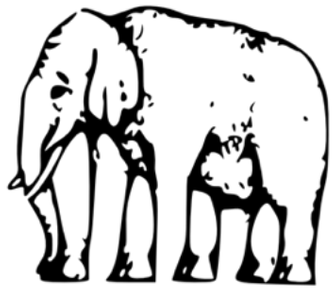
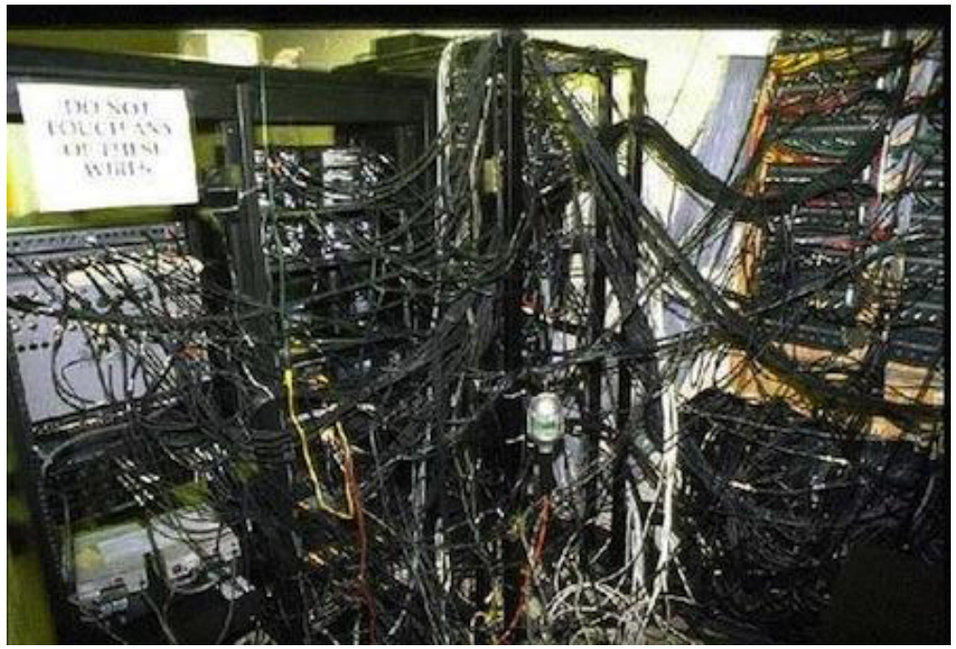
Clean Architecture

für Mac und iOS
inspiriert durch
Robert C.Martin (Uncle Bob)



@m_mlr

Wozu überhaupt Architektur



- BUD? (big upfront design TM)
- Wartung
- Orientierung

Was ist die Architektur einer Software?

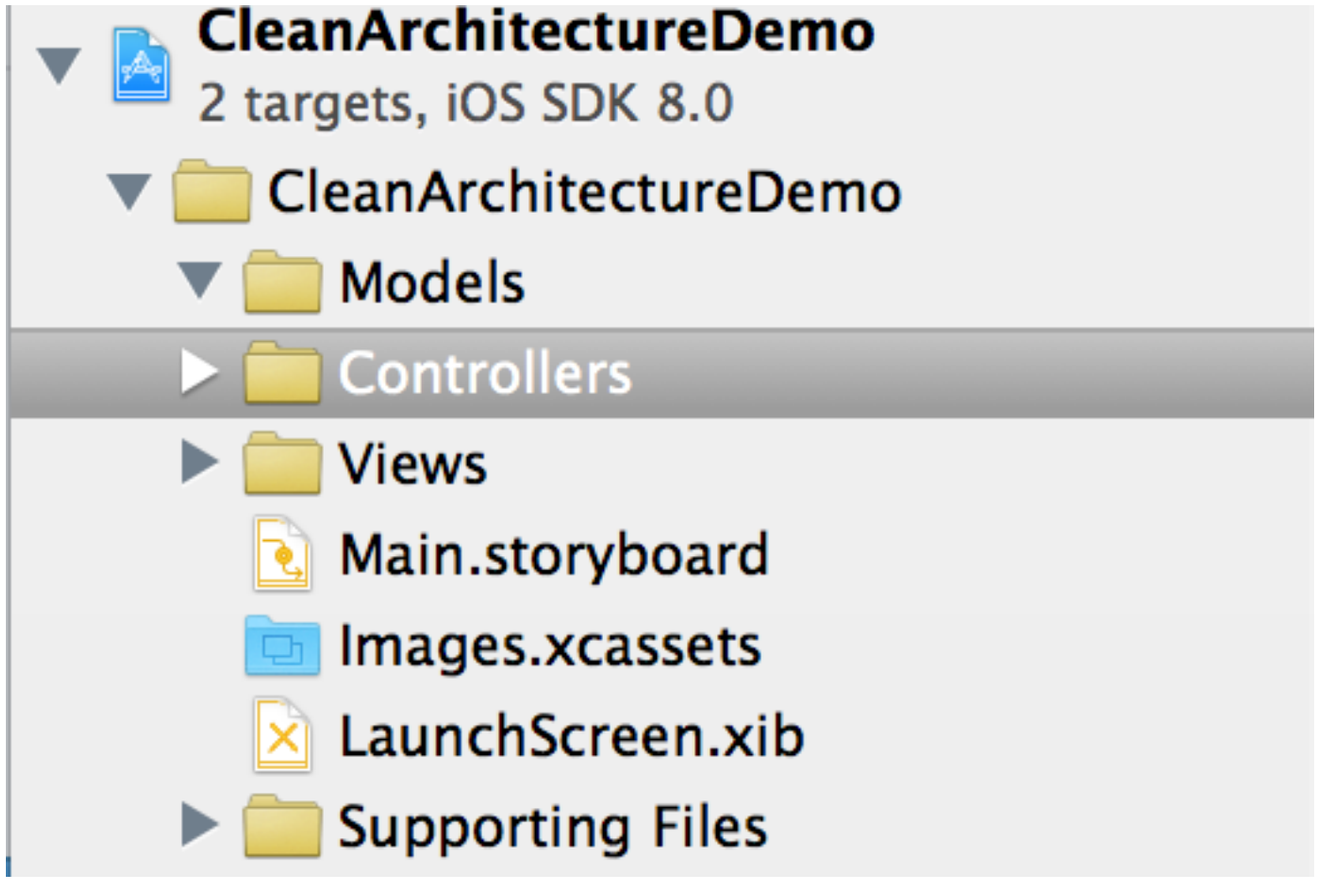
- UI?
- Datenbank?
- Web?

Was macht eine gute Architektur aus?

- einfach zu verstehen
- einfach zu erweitern
- einfach zu implementieren
- einfach zu testen

Cocoa

- Model-View-Controller!
- Einteilung der Komponenten einer Anwendung zu Model, View oder Controller
- NS-/UIView, XIBs, Storyboards
- NS-/UIViewController, NSWindowController
- CoreData


CleanArchitectureDemo
2 targets, iOS SDK 8.0
CleanArchitectureDemo
Models
Controllers
Views
Main.storyboard
Images.xcassets
LaunchScreen.xib
Supporting Files

Wo ist das Problem?

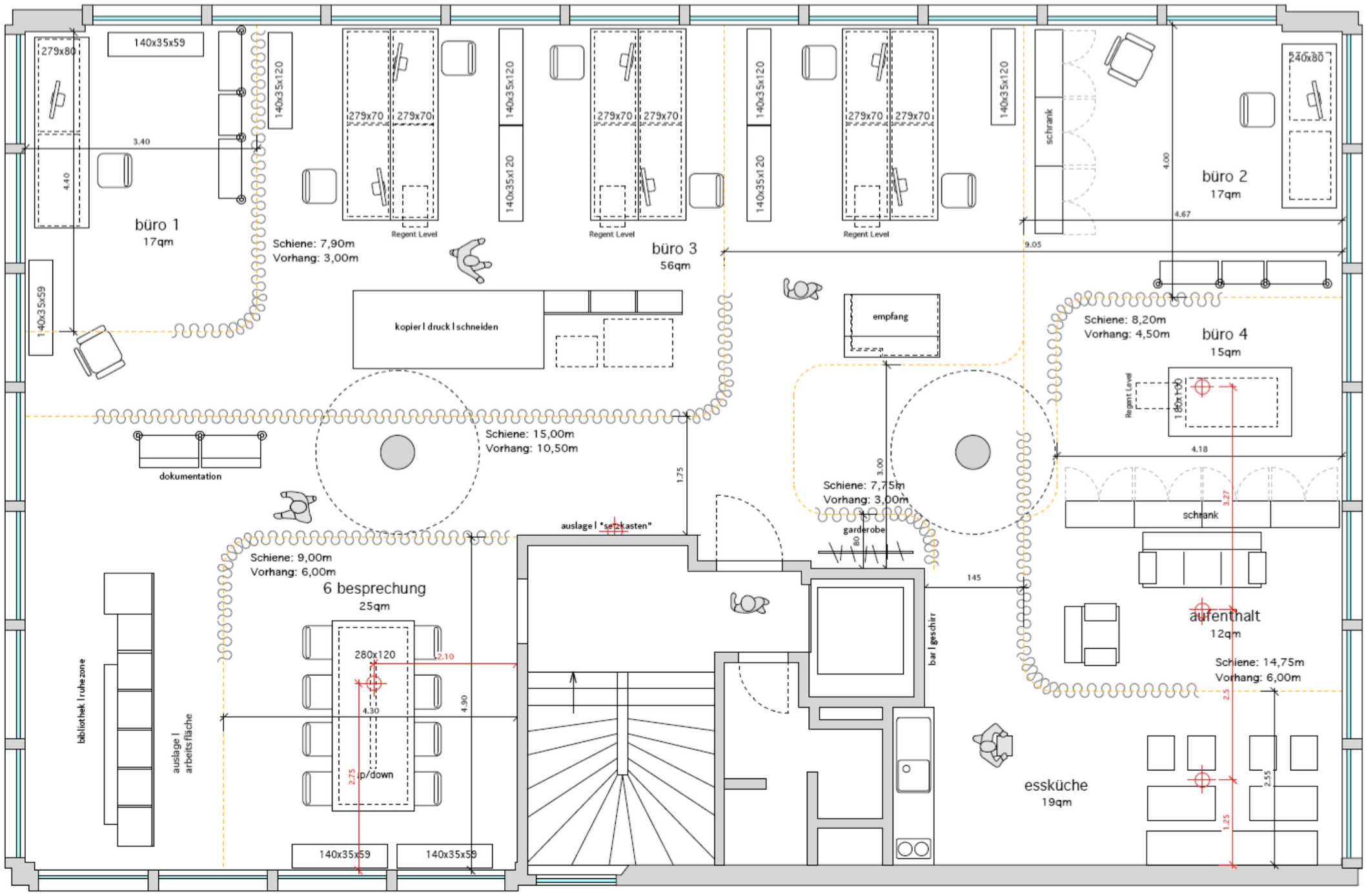
- Architektur hat nichts mit dem benutzen Framework zu tun

- Architektur sollte durch Frameworks unterstützt werden

- Frameworks sollten nicht eine bestimmte Architektur erzwingen

- MVC ist ein Framework-Detail, keine Architektur

Demo



variante : 12

5m

Projekt:	umbau . mobiliar . innenausbau . hilda design
Bezeichnung:	grundriss 5.OG
Bauherr:	HILDA DESIGN MATTERS

Architekt:	grigoleitniederreutherarchitektur . brauerstrasse 45 . 8004 zürich
Datum:	05.04.2008
Massstab:	1:50

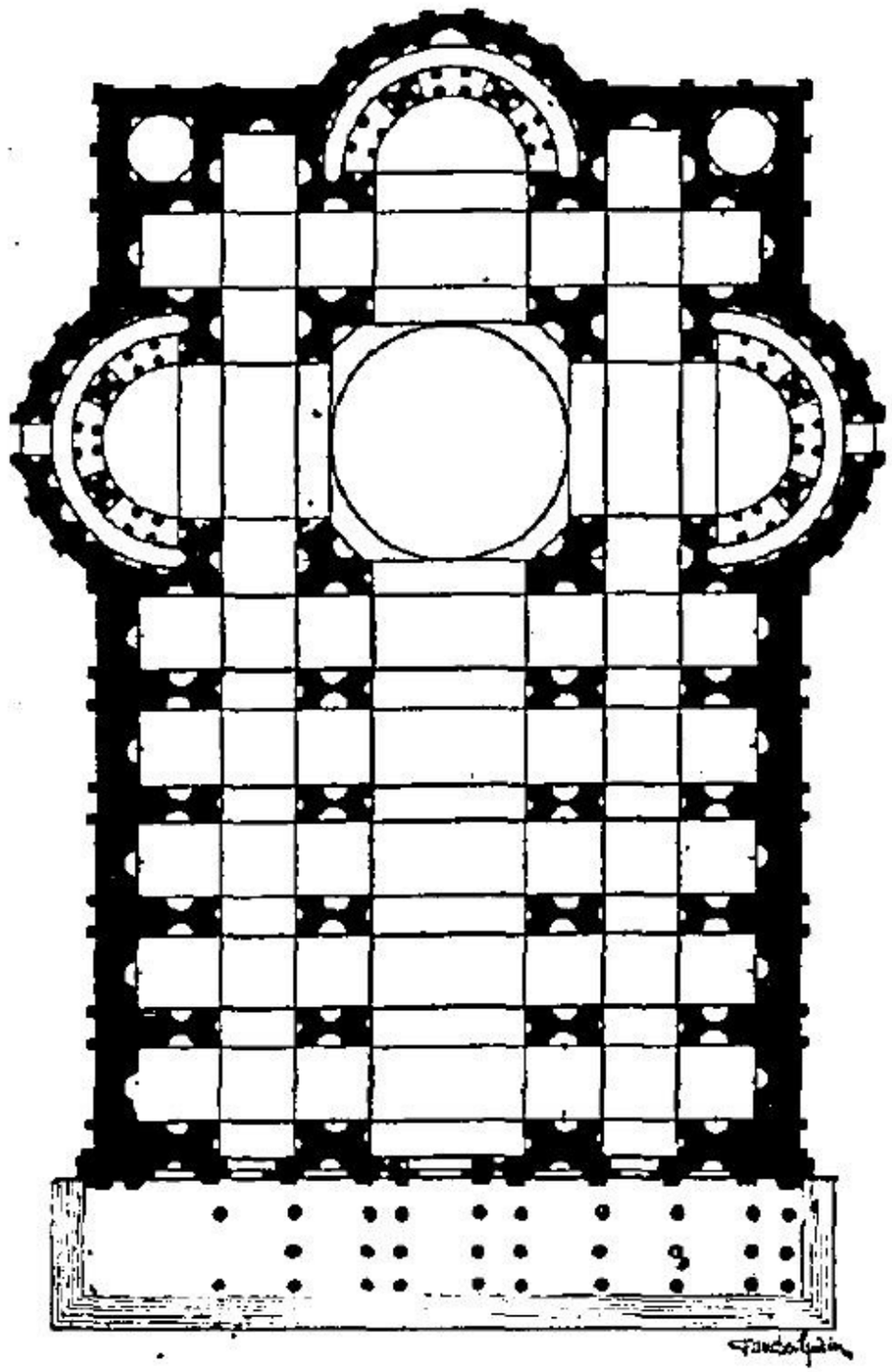
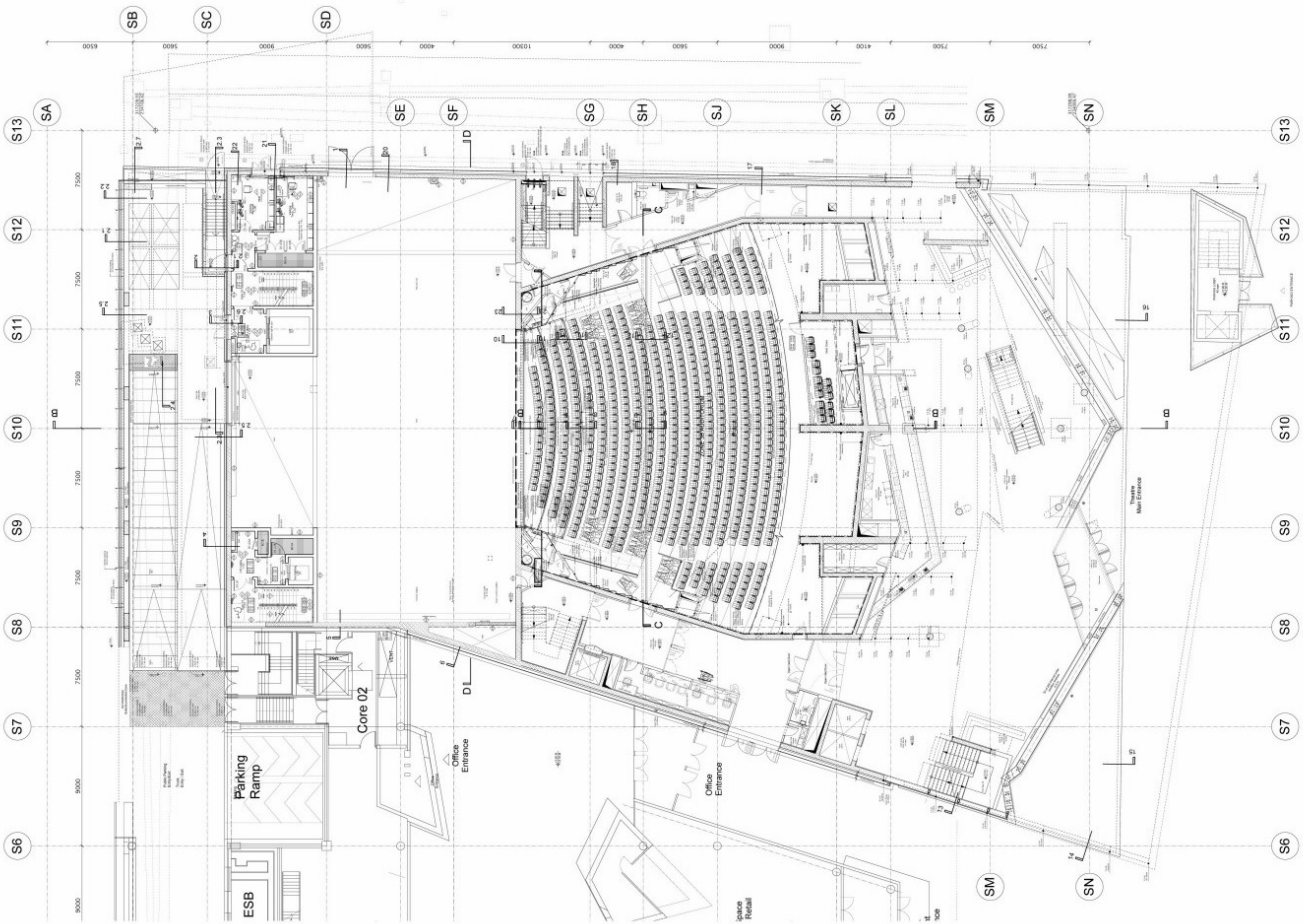


Fig. 8. — Saint-Pierre.

(Plan de Raphaël.)



DRAWING STATUS: INFORMATION

ZONE: ZT GROUND FLOOR

DRAWING Nº	REVISION	SCALE	PAPER SIZE
GCS-ZT-AD-02-10	L	1:100	A0

JOB Nº	DRAWN	CHECKED	FIRST ISSUE	DATE
0404	AB	GB	02.04.2007	28.02.2008

DRAWING TITLE

Ground Floor

PROJECT: GRAND CANAL SQUARE, GRAND CANAL HARBOUR, DUBLIN

ARCHITECT: ARCHITEKT DANIEL LIBESKIND AG
WALCHESTRASSE 9
CH-8006 ZURICH
TEL. +41 44 540 4700
FAX. +41 44 540 4760
E-MAIL. info@daniel-libeskind.ch

CLIENT: RAMFORD LTD.
USHER HOUSE, MAIN STREET,
DUNDRUM, DUBLIN 14
TEL. +353 1 216 4000

0404 GRAND CANAL SQUARE - DUBLIN

- gute Architektur ist unabhängig davon, mit welchen Werkzeugen das Gebäude gebaut wird
- gute Architektur ermöglicht, Detailfragen *später* zu beantworten

Architektur hat viel mit Kommunikation zu tun und sollte etwas über die Benutzung erzählen

- „Mails lesen“, „Text bearbeiten“, „Musik hören“, „Routenplaner“, „Bilder anschauen“

- nicht „CoreData“, „MySQL“, „UIViewController“, „JSON“

Use case driven architecture

- ???
- Use Case entspricht einer „User story“
- Architektur einer Anwendung sollte die use cases „herausschreien“, ähnlich realen Architekturplänen
- „Notizen durchsuchen“
- „Benutzer hinzufügen“
- „Mails der letzte Woche darstellen“

Softwarearchitektur

- die use cases bestimmen die Architektur
- Anwendungsregeln dominieren
- keinerlei Bezug zu Frameworks, Datenbanken, UIs
- gute Architektur entkoppelt den Anwendungskern von den Details

Was ist mit Cocoa (touch)?

- eigentlich sehr gute RAD Werkzeuge (Storyboards, CoreData, Interface Builder)
- Apple-Samples geben oft schlechte Beispiele
- Testing?

Cocoa MVC

- Zu starre Einteilung der Komponenten zu den Schichten Model, View oder Controller
- verleitet zum „Massive View Controller“ 😥
- Zuviel Anwendungslogik landet in den (View-)Controllern
- Der Anwendungskern sollte von der Viewschicht unabhängig sein

SOLID

© Uncle Bob

Single Responsibility Principle

„Es sollte nie mehr als einen Grund dafür geben, eine Klasse zu ändern.“

Open/Closed Principle

„Module sollten offen für Erweiterungen, aber geschlossen für Modifikationen sein“

Liskov substitution principle

“objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.”

Interface segregation principle

„Klienten sollten nicht dazu gezwungen werden, von Interfaces abzuhängen, die sie nicht verwenden.“

Dependency inversion principle

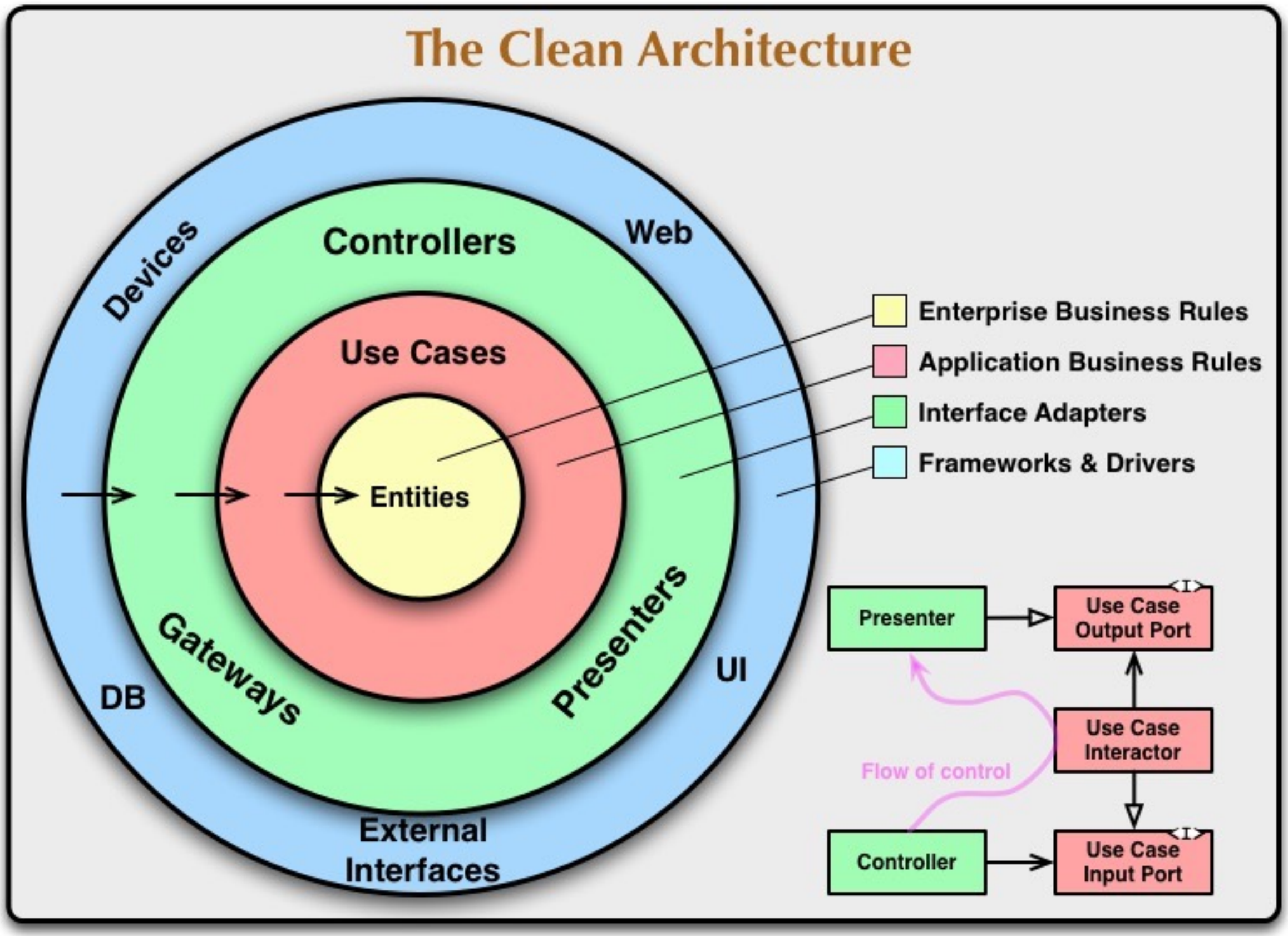
„Module hoher Ebenen sollten nicht von Modulen niedriger Ebenen abhängen. Beide sollten von Abstraktionen abhängen.

Abstraktionen sollten nicht von Details abhängen. Details sollten von Abstraktionen abhängen.“

Clean Architecture

Uncle Bob’s way

The Clean Architecture



Entities

- enthalten anwendungsübergreifende Geschäftslogik

- reguläre NSObject-Klassen unabhängig von jeglichen Framework-Details wie Datenbanken oder Viewschichten

- leicht zu testen

Use Cases

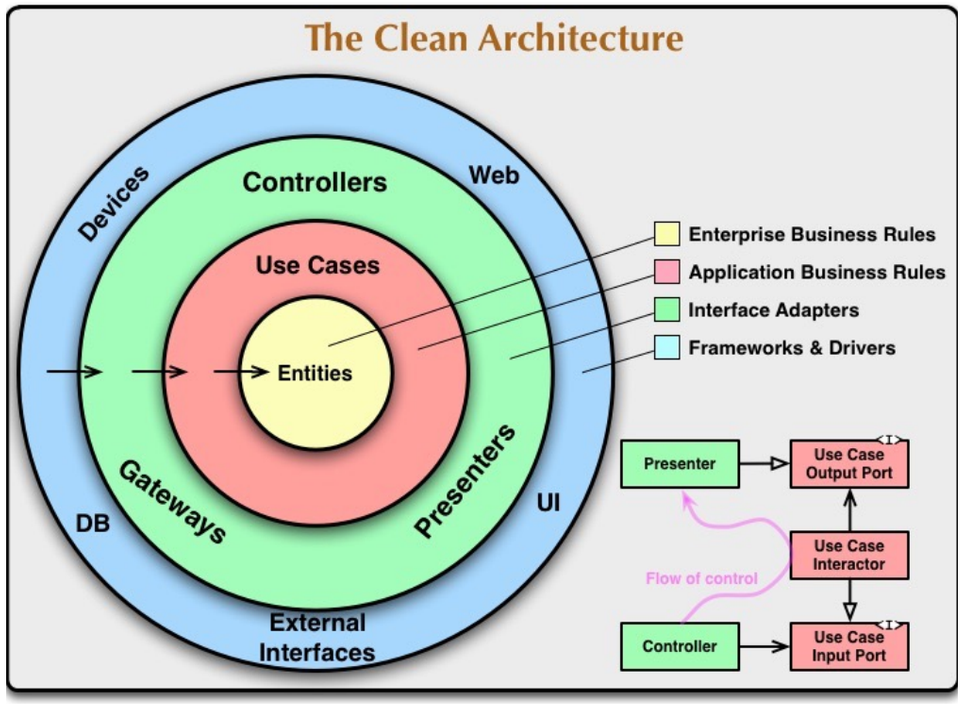
- ein Use Case entspricht einer „User story“
- enthält anwendungsspezifische Logik
- reguläres NSObject
- unabhängig von systemspezifischen Frameworks
- Abhängigkeiten in die Aussenwelt über klare und abstrahierte Schnittstellen (etwa zur Datenbank oder ins UI) mittels value objects oder data-structures
- es werden niemals Entities in die nächst äußere Schicht gereicht
- leicht zu testen

Presenter, Gateways, Controller

- Schnittstelle zwischen Use Cases und der Aussenwelt
- Reguläre Objekte, die die jeweiligen Interfaces implementieren (z.B. zum ORM-Mapper in eine DB oder in die Viewschicht)
- Presenter erzeugen view models (strings, flags ob UI-Element aktiviert wird etc)
- View Models sind einfache data structures
- leicht zu testen

UI, Devices, DB

- Äusserste Schicht welche die Aussenwelt repräsentiert
- Keine Kenntnis von Entities oder Use Cases
- Cocoa-Viewcontroller
- VCs sind „dumm“ und kümmern sich ausschliesslich um die Darstellung der ViewModels (simple data structures)
- MVC, KVO, ReactiveCocoa etc.
- CoreData, Datenbanken, REST, JSON etc.



- Alle Abhängigkeiten zeigen nach innen

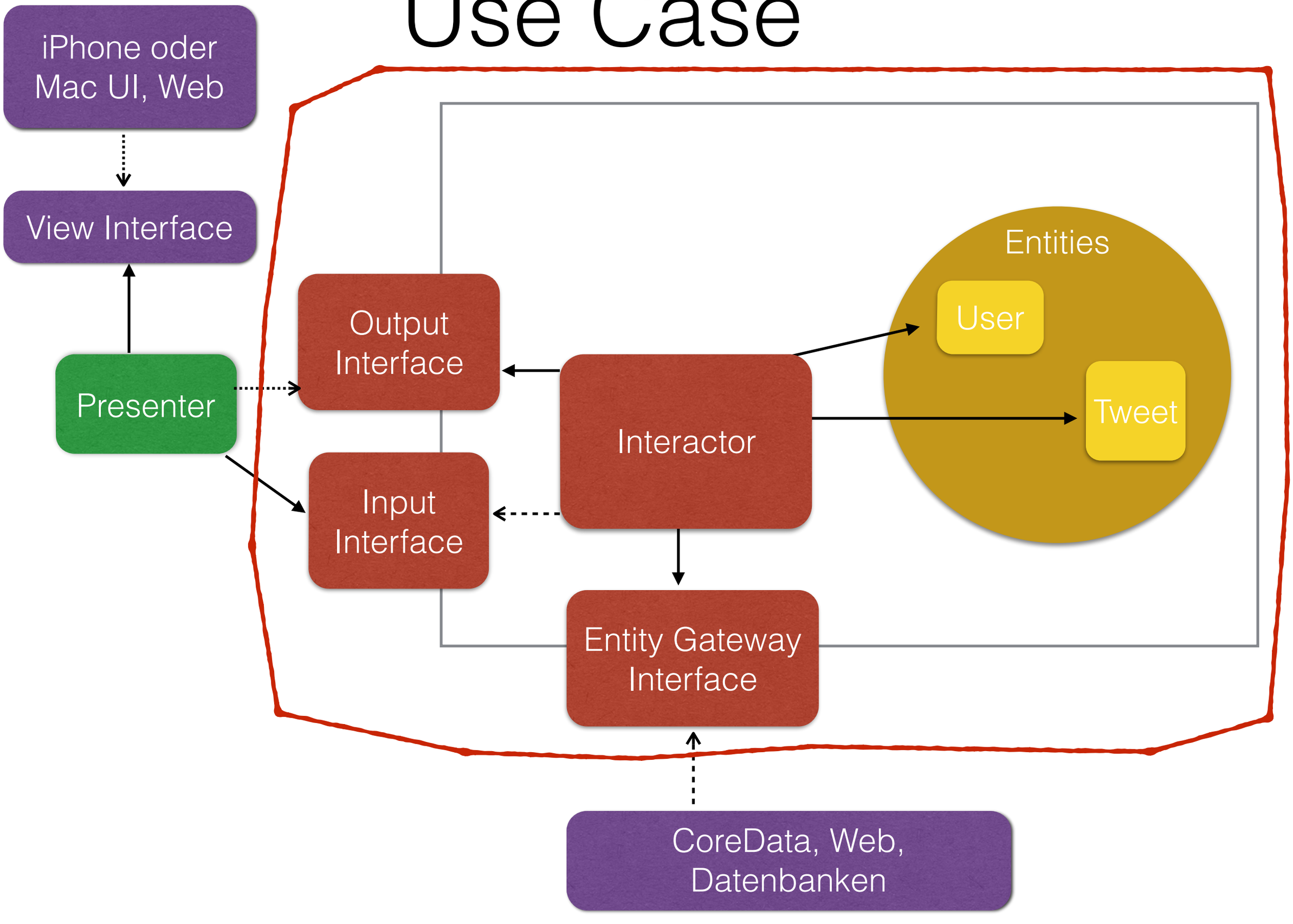
- Entities sowie die Use Cases wissen nichts von Datenbanken oder UIs

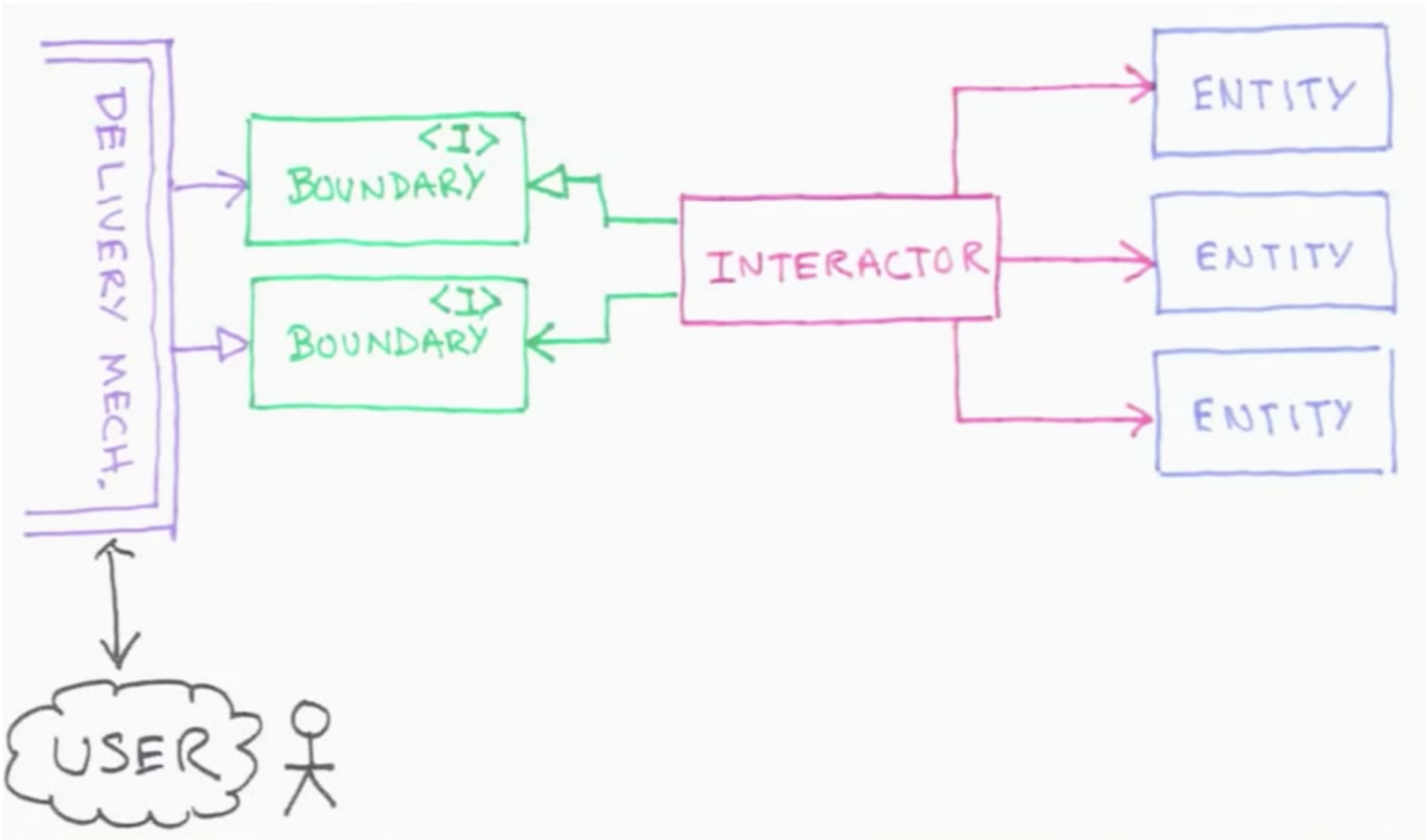
- Ausschliesslich die Use Cases benutzen die Entities

- Kontakt zur „Aussenwelt“ über Schnittstellen mittels Data Transfer Objects / Datenstrukturen (simple Objekte ohne Verhalten)

- Anwendungslogik ist unabhängig von systemspezifischen Details

Use Case


iPhone oder
Mac UI, Web
View Interface
Presenter
Output
Interface
Input
Interface
Interactor
Entities
User
Tweet
Entity Gateway
Interface
CoreData, Web,
Datenbanken



Demo

Vorteile

- klare Zuständigkeiten (SRP)
- Anwendungslogik von UI entkoppelt
- viele, aber einfache und kleine Klassen
- TDD bis in die Viewschicht hinein möglich
- einfache Tests, kaum Mocking nötig

Nachteile?

- erscheint auf den ersten Blick antikonzeptionell zu Cocoa
- Tendenz zum „over engineering“
- Entkopplung mittels value transformern oder KVO/ Bindings möglich

Mehr zum Thema

- http://blog.8thlight.com/uncle-bob/2011/09/30/Screaming-Architecture.html
- http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html
- http://www.objc.io/issue-13/viper.html
- Ivar Jacobson „Object Oriented Software Engineering - A use case driven approach“
- http://www.cleancoders.com
- http://www.confreaks.com/videos/759-rubymidwest2011-keynote-architecture-the-lost-years
- https://github.com/mmllr/CleanTweeter