



# Introduction to Organizing a Codebase

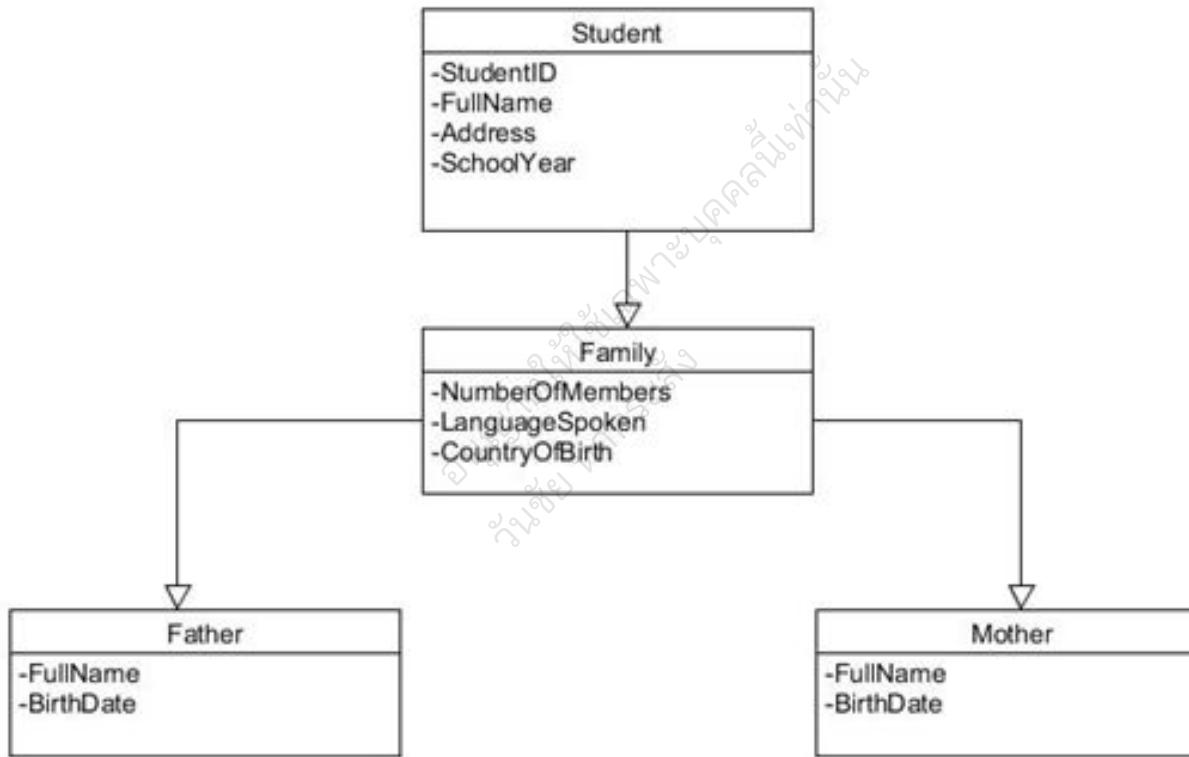
© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.



We started from OOP

รุ่นที่ ๑  
วิชานี้ การสอน  
ภาษาต้องใช้เฉพาะบุคคลนั้น



## **Why classes is not enough?**

- The number of classes can grow
- In Facebook Apps, there are >18,000 classes
- “We have a network problem, can you fix it”
  - Which class does it belong to one?



## Conclusion:

We need to have a group of classes

# Let's grouping it

อนุกรรมการให้เชิงวิชาการและคุณท่าน  
ร่วมช่วย ห้ามระลัง

Feed

Comment

Notification

User

Payment

Data

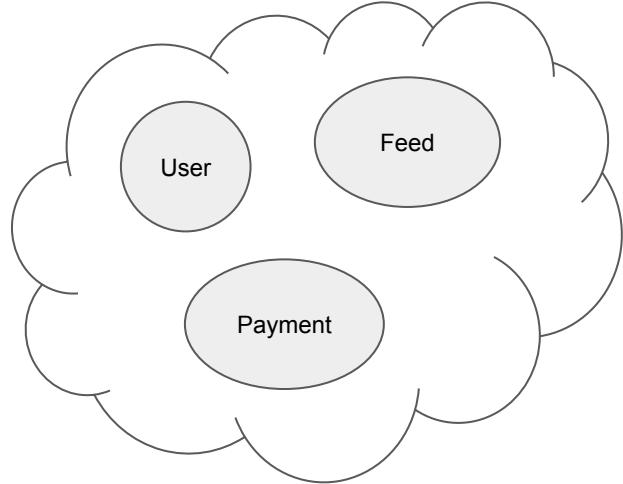
Likes

Networking

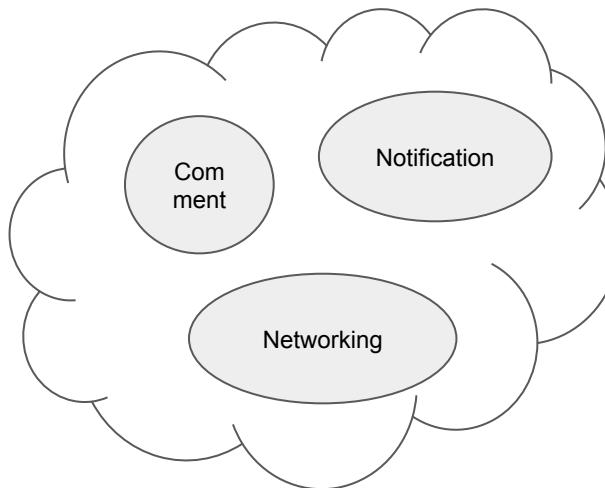
Shop

គម្រោង និង ការបង្កើតរចនាសាស្ត្រ

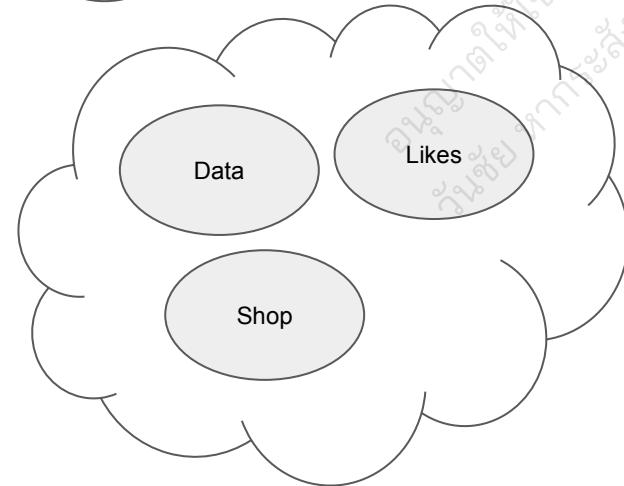




Thanos

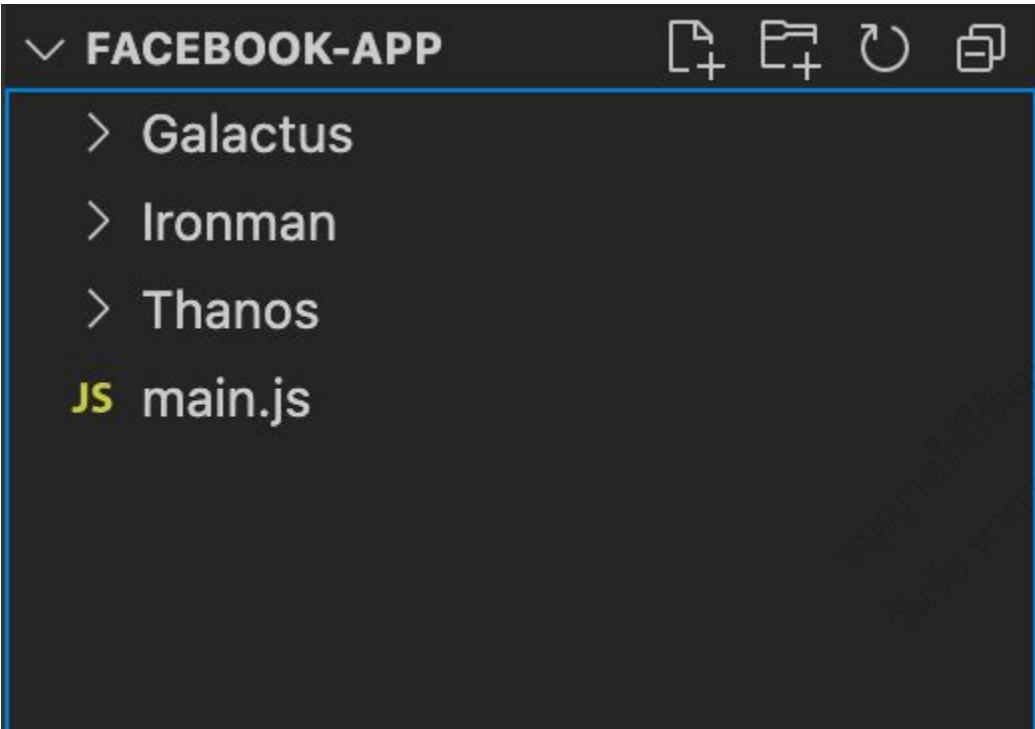


Galactus

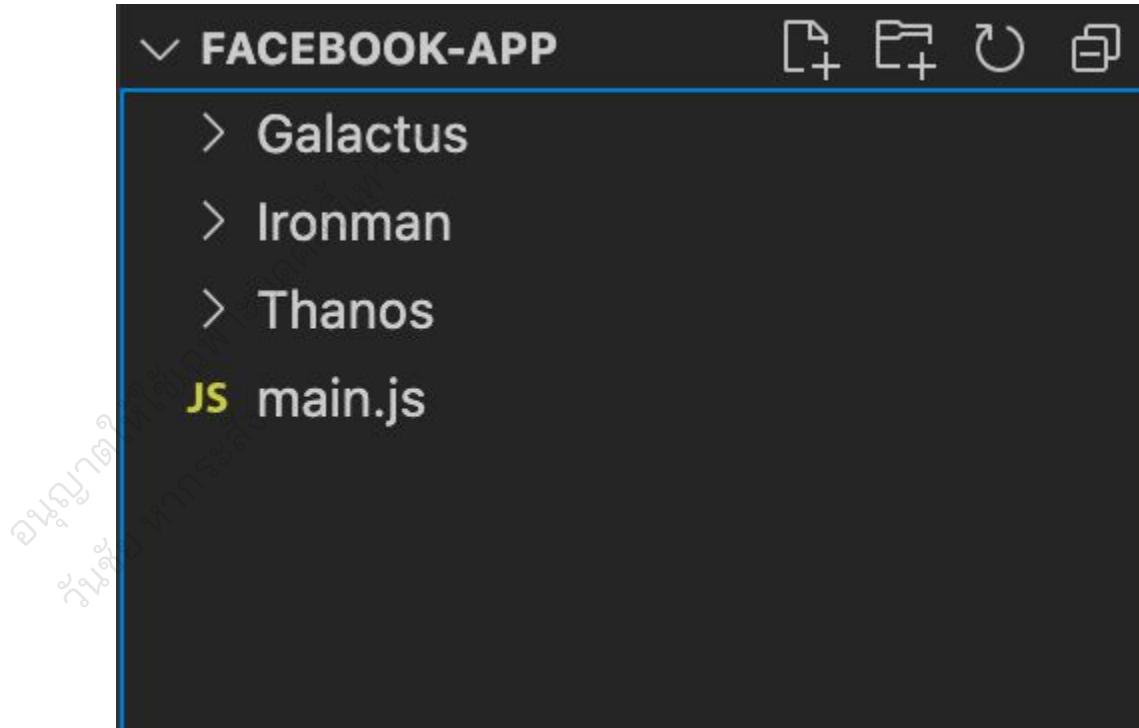


IronMan



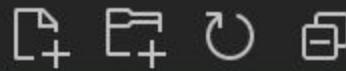


“Perfectly balance, as all thing should be”

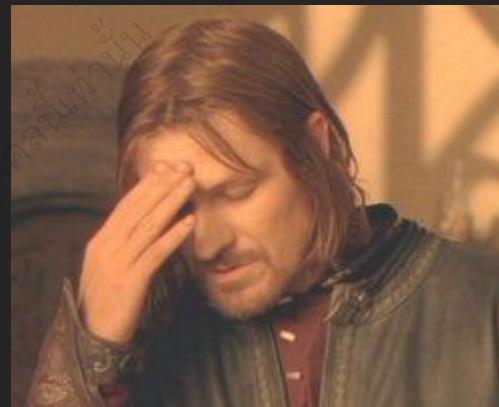


Ok, We got a network problem. Let's fix that

✓ FACEBOOK-APP



- > Galactus
- > Ironman
- > Thanos
- JS main.js**



## **Randomly grouping does not help**

- As you can see, grouping for the sake of grouping does not help the case
- Logically sound grouping

อนุญาตให้ใช้สิ่งพำนัช  
วันนี้ การที่จะ



**One software architect works is about  
“a way to make a sense out of system”**

The big question:  
How should we group it?

## **How should we group it?**

Criteria: Grouping must make a logical sense

The grouping must introduce a benefit, otherwise, why?

The grouping must have a clear logical rules and justification.

- Not “Thanos, Ironman, Galactus”

It should help make some logical sense and enable better understanding



## **How should we group it?**

Making logical sense to who?

- Just me?
- People involved

Who are people involved?

- Technical team members
- Business/Product/Project manager

Specialist Collaboration

Business Collaboration

Organization Collaboration



## **How should we group it?**

But grouping is very domain specific right? Yes.

- The grouping of Facebook and Salesforce should be totally different

Still, there are some heuristics that are made by software engineers working in many type of software.

- N-Tier for Specialists
- Domain-Driven design for Business and Team

These heuristics provide some specific benefits and trade-offs, and we will learn about that.





# N-Tier Architecture

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.

อนุญาตให้ใช้สิ่งที่  
บันทึกไว้ในห้องเรียน

# What is an N-Tier architecture?

รูปแบบ Arquitectura N-Tier  
การรักษาความเป็นส่วนตัวของข้อมูล  
การจัดการความซับซ้อนที่เพิ่มขึ้น

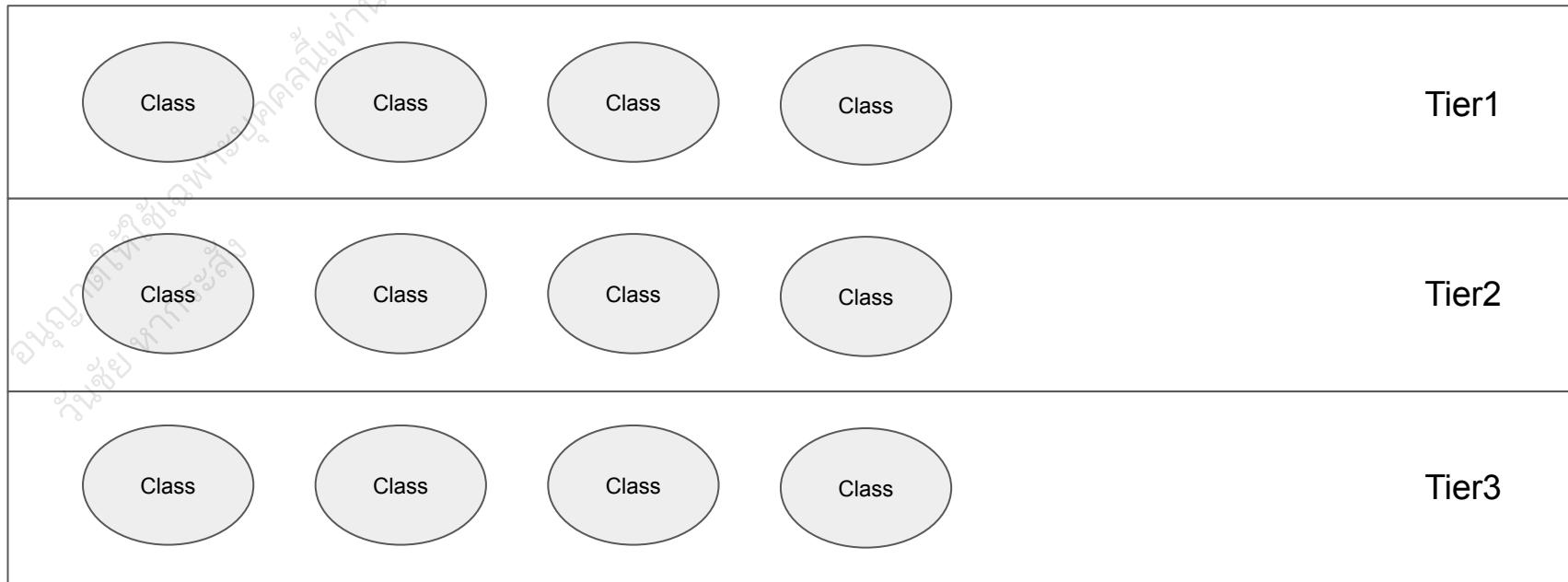
## N-Tier architecture

- N-Tier architecture is a way to group things together.
- N-Tier: You should layer the app
- N-Tier itself is just a concept of layering the app. There is no concrete manifestation

อนุญาตให้ใช้สิ่งที่บันคุณทำนั้น  
วันนี้ยัง การะลัง



# N-Tier architecture



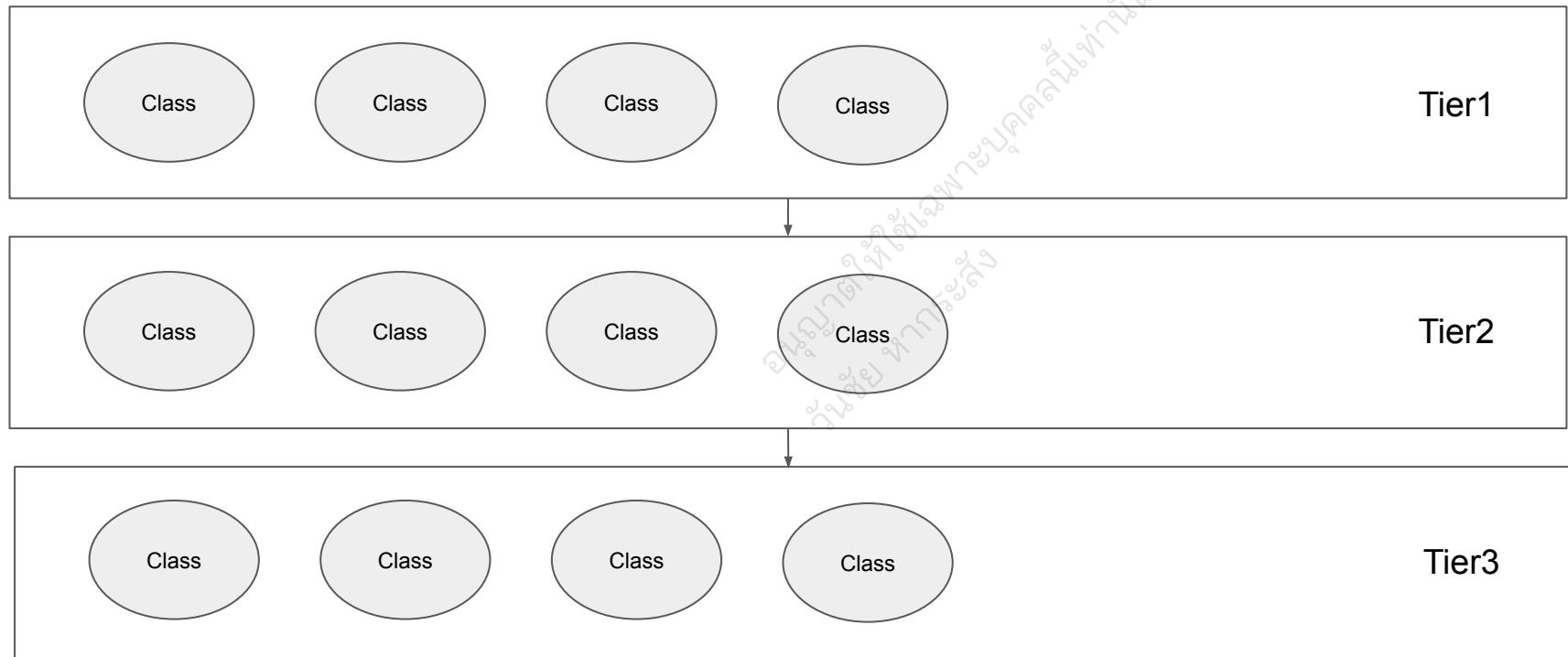
## N-Tier architecture

- What's the difference between N-Tier and "Galactus, Thanos, Ironman"?
- N-Tier rules: Internal layer does not depends on external layer

รุ่นที่ ๑  
การสร้าง  
เว็บไซต์ให้เข้าใจง่าย



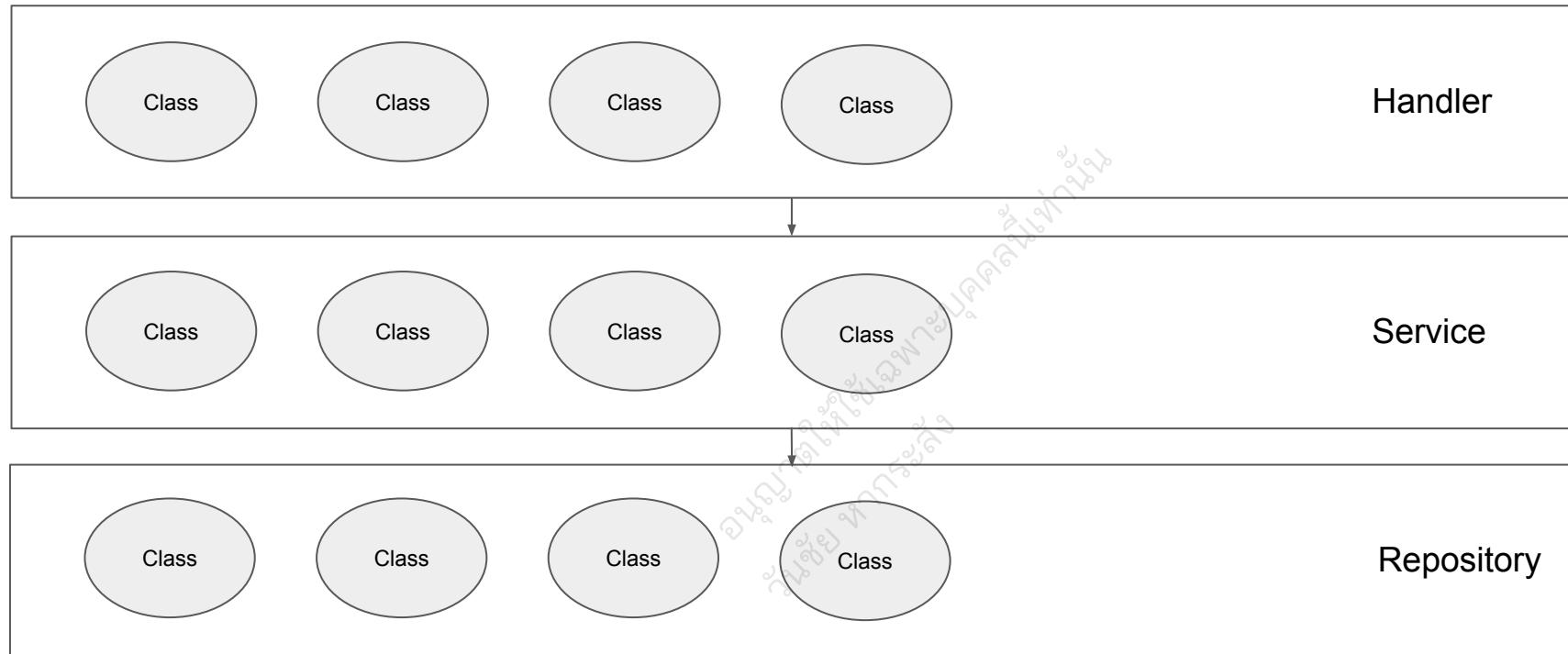
# N-Tier architecture



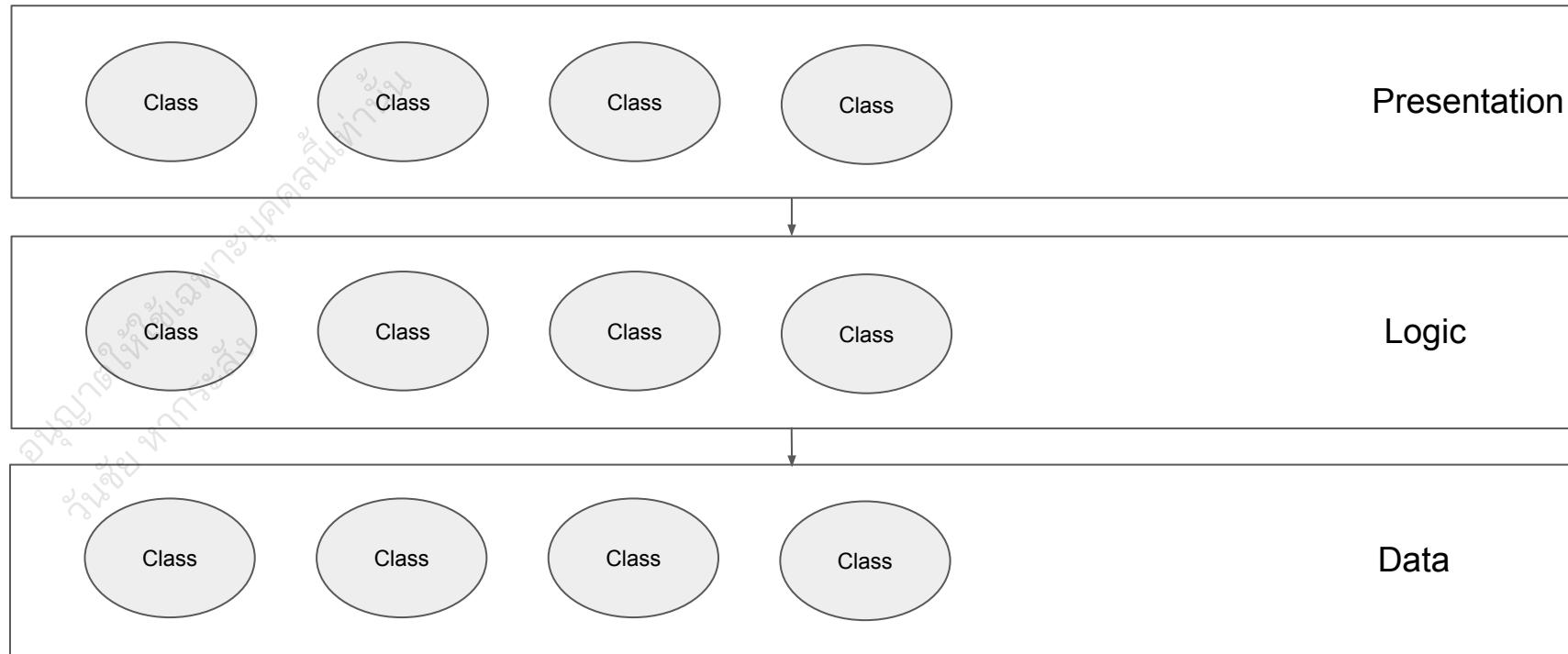
อนุญาตให้เข้ามาพำบุคคลนี้ท่าน  
วันนี้ยัง ทำการลับ

# Example

# N-Tier architecture



# N-Tier architecture



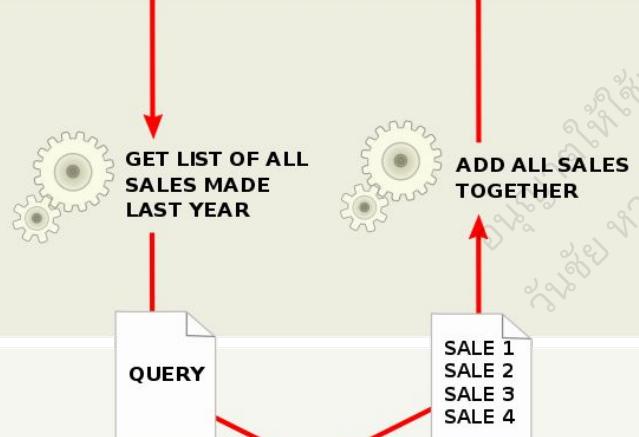
## Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.



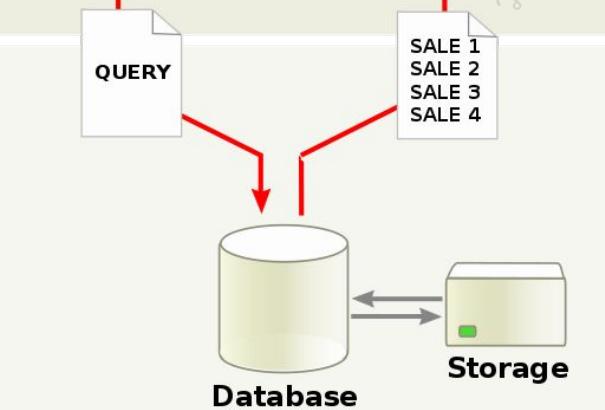
## Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.



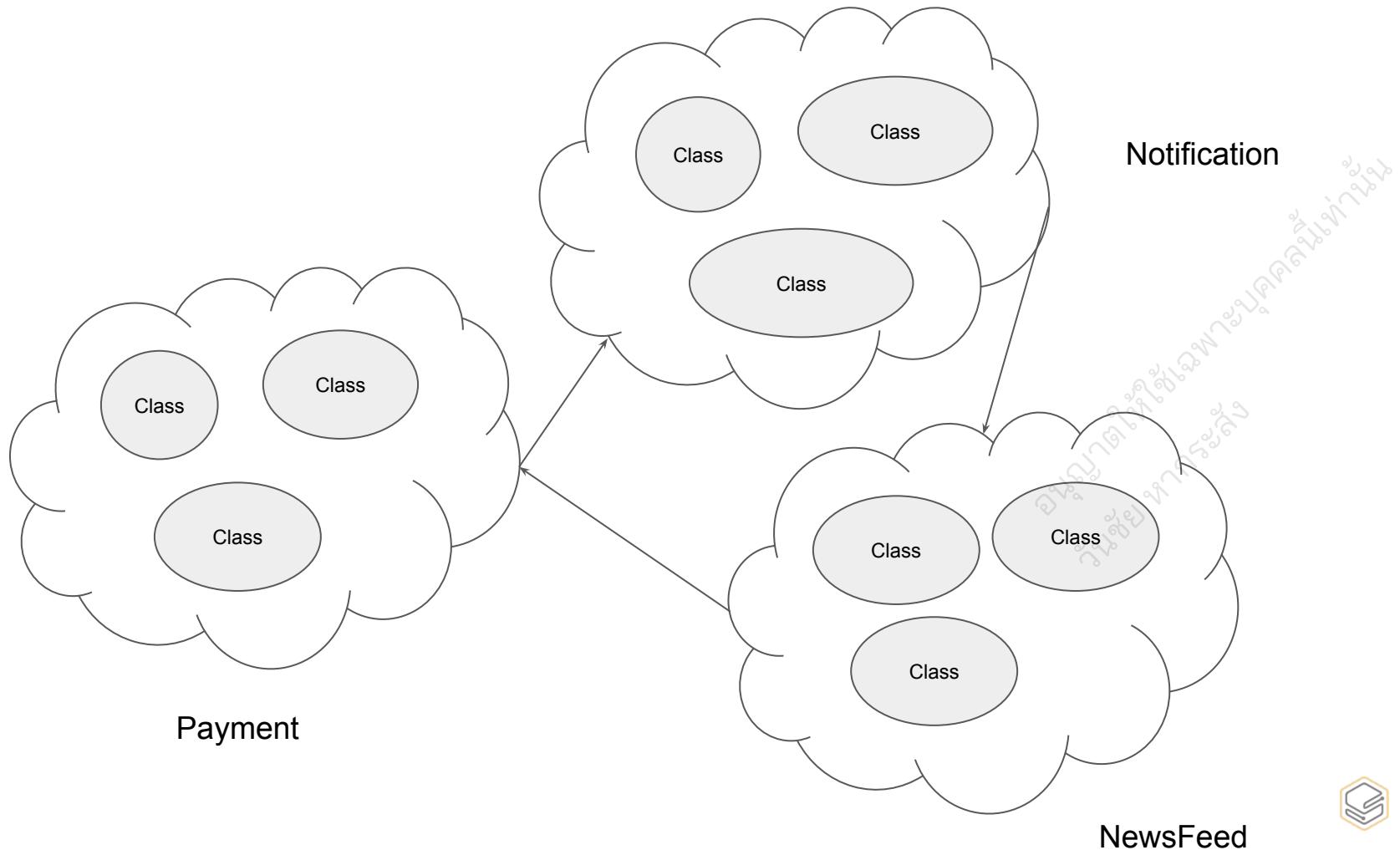
## Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.



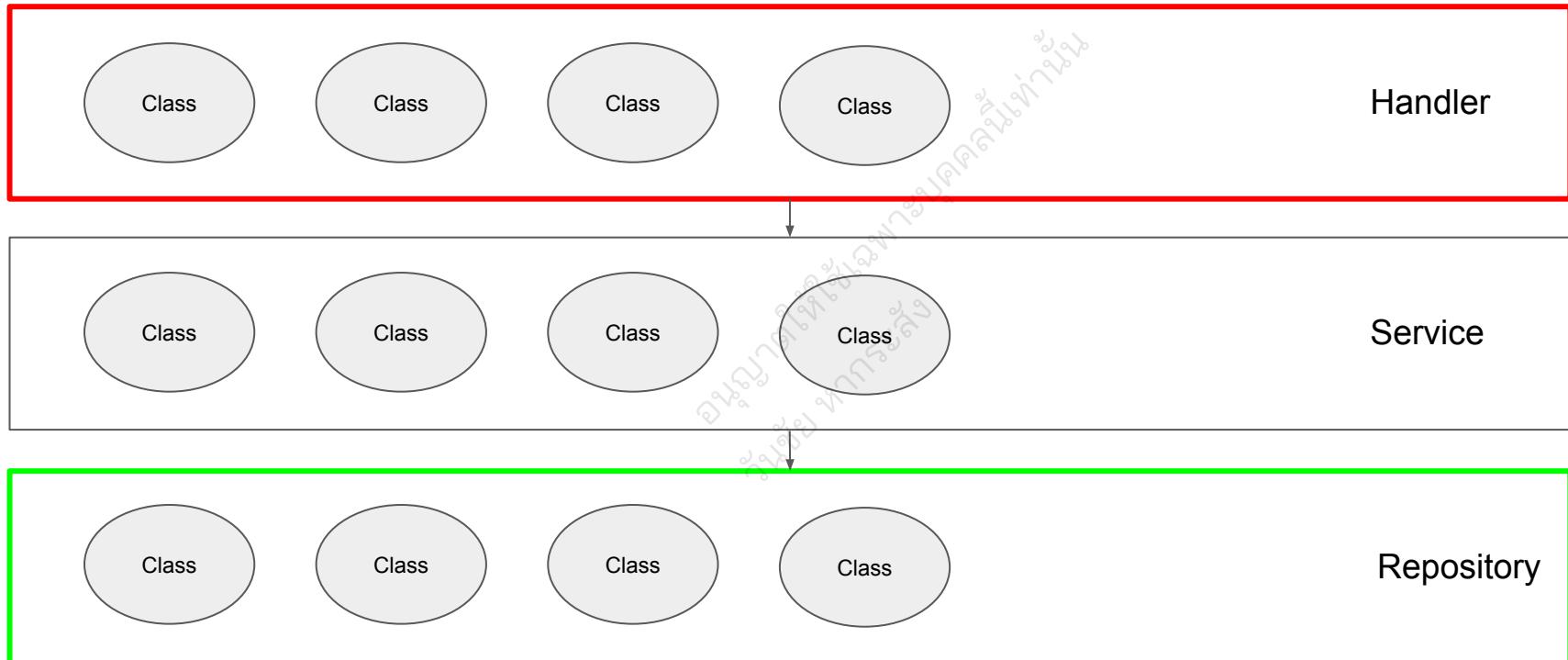
# What is not N-Tier

สถาบัน  
วิจัย  
การศึกษา



The main benefit of N-Tier is the dependency tracking

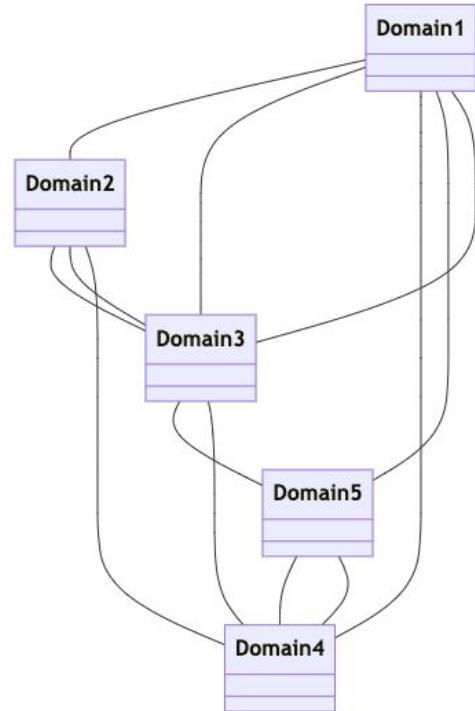
“Fix your request handler, add a rate limit”



The requirement will never affect this



อัลกอริทึมเชิงพารามิติก  
วิธีการลับ



## **Popular Manifestation of N-Tier architecture**

- MVC
  - Spring, Rails, Django, .NET MVC, etc.
- Clean architecture
- Java EE



# Summarize

- N-Tier is an architectural style on how to organize module
- The basic principle:
  - Group modules into multiple layers
  - The higher layer depends on lower layer
- Examples:
  - 3-tier architecture
  - MVC
- Benefit: Dependency tracking
  - We know the impact scope of change
- Trade-offs
  - We can't talk about trade-offs in vacuum. Let's learn another style of architecture first

อนุญาตให้เข้าชมเพียงบุคคลที่  
กับผู้สอนเท่านั้น





# Demystify N-Tier benefit

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.



# Demystify N-Tier architecture benefit claim

We said that benefit of N-Tier is dependency tracking

What about others benefits?

[Microsoft Docs](#)

[DZone](#)

[IBM](#)

Why?

## Benefits

- Portability between cloud and on-premises, and between cloud platforms.
- Less learning curve for most developers.
- Natural evolution from the traditional application model.
- Open to heterogeneous environment (Windows/Linux)

### What Are the Benefits of N-Tier Architecture?

There are several benefits to using n-tier architecture for your software. These are scalability, ease of management, flexibility, and security.

- **Secure:** You can secure each of the three tiers separately using different methods.
- **Easy to manage:** You can manage each tier separately, adding or modifying each tier without affecting the other tiers.
- **Scalable:** If you need to add more resources, you can do it per tier, without affecting the other tiers.
- **Flexible:** Apart from isolated scalability, you can also [expand each tier](#) in any manner that your requirements dictate.

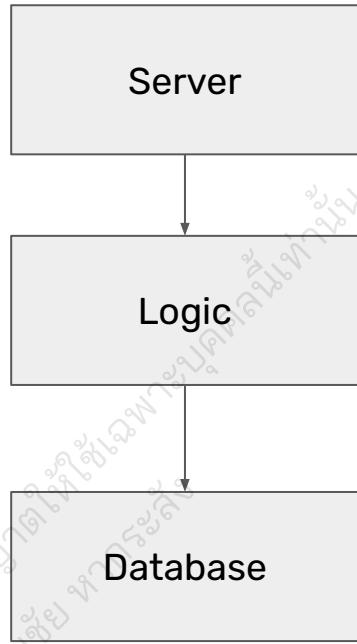


# Let's talk about common benefit N-Tier Claim

- Secure
- Easy to manage
- Scalable

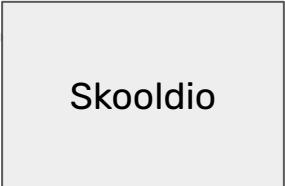
อนุญาตให้เข้ามาดูคลังท่าน  
กับเราทุกวัน





N-Tier



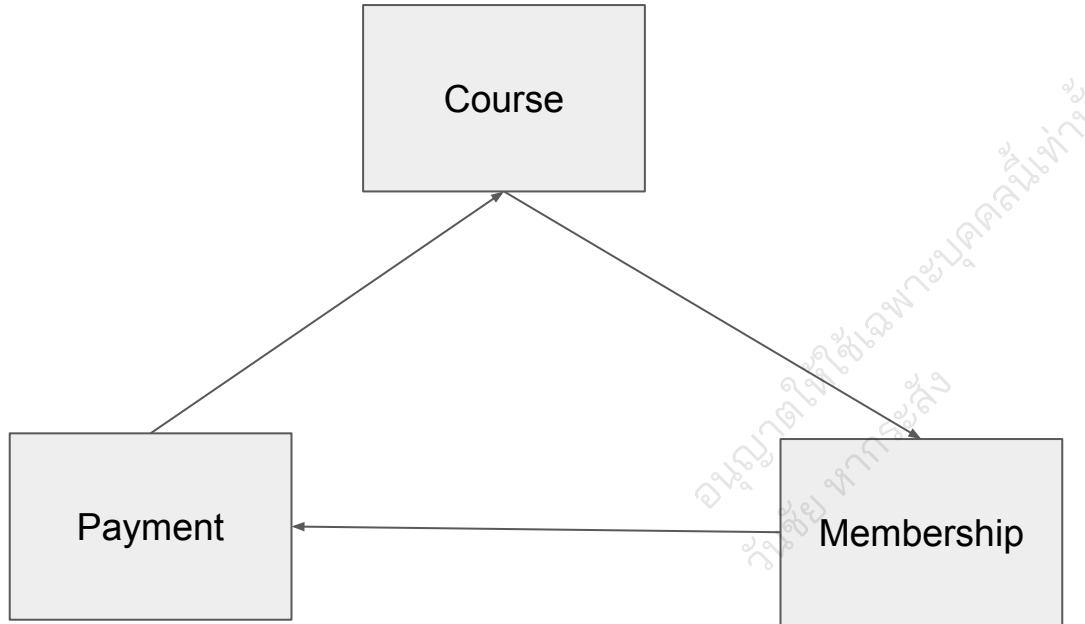


Skooldio

អំពើការបង្កើតរបស់គម្រោង  
និងរបៀបបង្កើតរបស់គម្រោង

As opposed to: God class



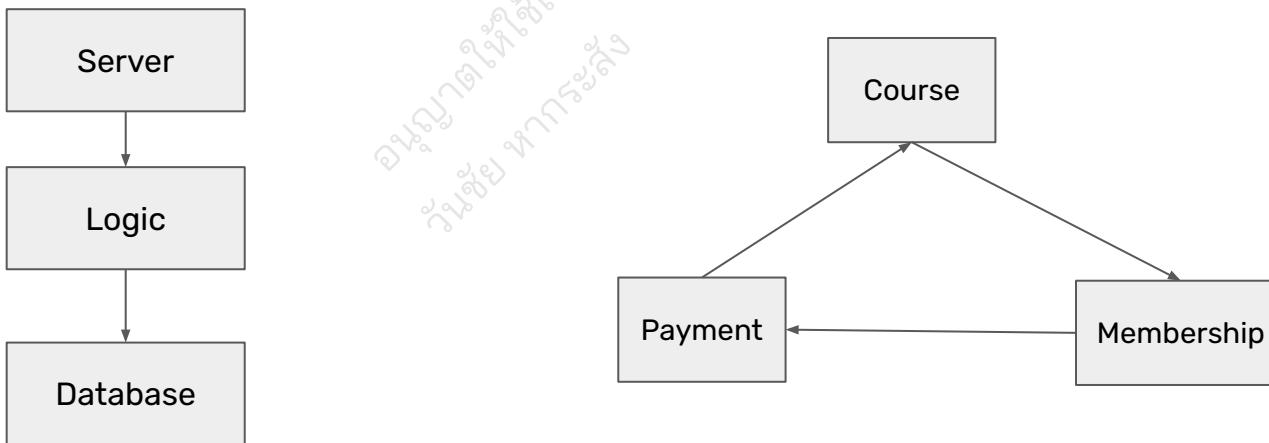


As opposed to: Domain



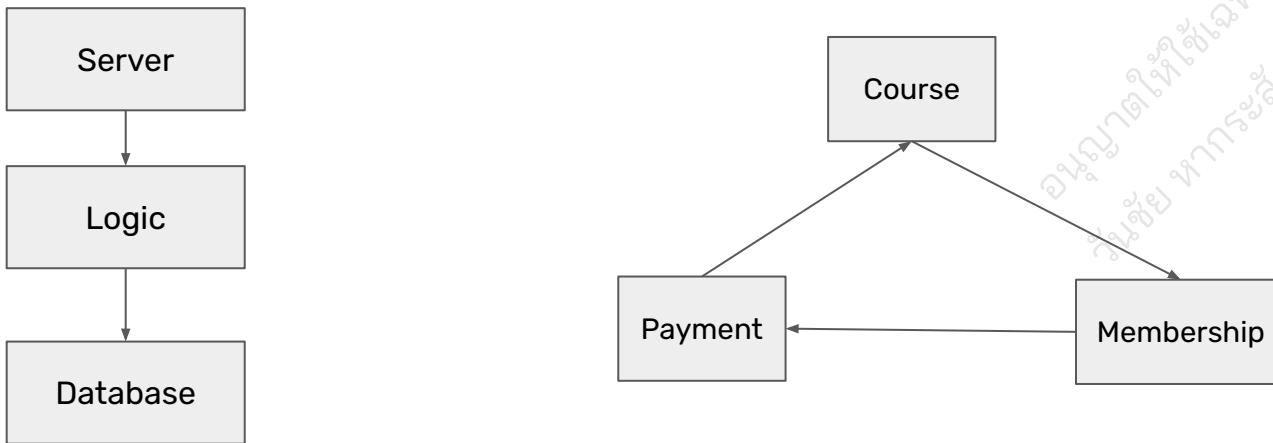
# Let's talk about common benefit N-Tier Claim

- **Secure**
  - Compared to god class, much better
  - “Protecting from DDOS, SQL Injection” vs. “Protecting our payment module”
- Easy to manage
- Scalable



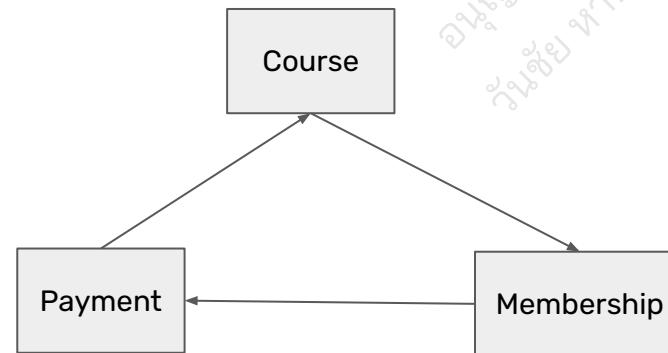
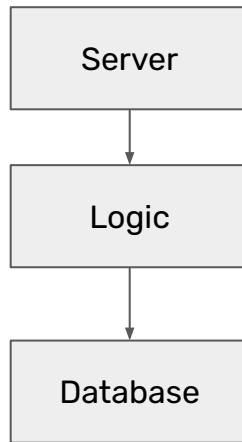
# Let's talk about common benefit N-Tier Claim

- Secure
- **Easy to manage**
  - Technical changes vs. Business requirement changes
  - “Increase query performance” vs. “Add new payment method”
- Scalable



# Let's talk about common benefit N-Tier Claim

- Secure
- Easy to manage
- **Scalable**
  - Easy to scale server, database, etc.
  - “Scaling database” vs “Scaling courses”



# N-Tier benefit

It depends

- Are you compare to God class or Domain?
- What is the nature of work?

That is why I reduce to just dependency tracking



# Choosing architectural style

No architecture benefit exists in vacuum

Consider these

- What is your environment and context?
- What do you pitch against?





# MVC #1: Requirement

ឧប្បត្តិវេទ្យរបាយកម្មគោលនយោបាយ  
រៀងចំ ការសំគាល់

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.

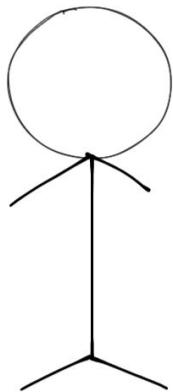
Something we need  
to build here

E-Learning system

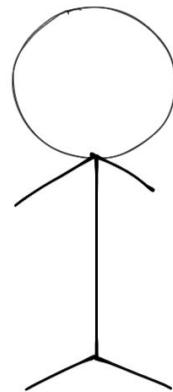
รุ่นใหม่ที่ใช้เฉพาะบุคคลเท่านั้น  
วันนี้เป็นวันแห่งการลับ

# What about the team?

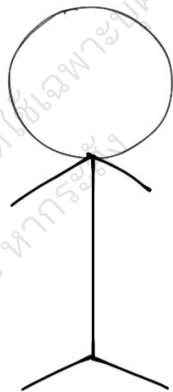
Web  
developer



E-Learning  
expert



Web  
developer



อนุญาตให้ใช้สิ่งพกพาในคลัง  
วันซ้อมทางการศึกษา

# Let's design for this



# MVC #2: What is MVC?

อนุญาตให้ใช้เพื่อทางบุคคล  
วัสดุย ห้ารับสั่ง

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.

# MVC History

- Trygve Reenskaug introduced MVC into Smalltalk-79 while visiting the Xerox Palo Alto Research Center (PARC) in a year 1978
- It was a pattern to solved UI interaction

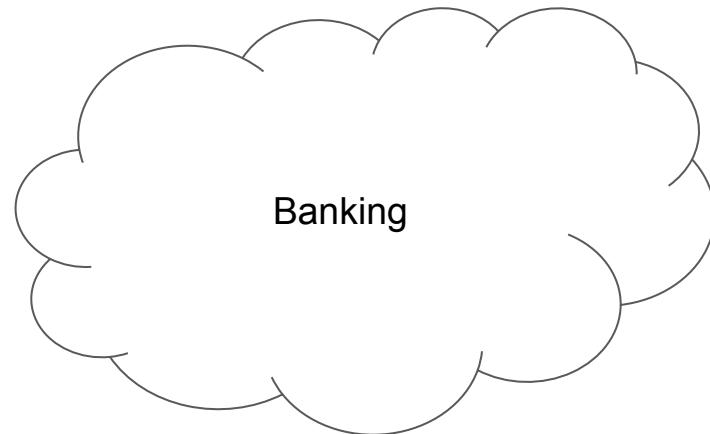
อนุญาตให้ใช้สิ่งที่ทางบูรณาคุณท่าน  
วันนี้เป็น หางรังส์



Let's find out how did we come up with  
MVC

วันนี้จะพูดคุยเรื่อง MVC  
การสร้างระบบแบบ MVC

รุ่นที่ ๑  
ธนาคารเพื่อการส่ง  
ออกบูรณาภรณ์



# MVC History



UI interaction for core banking

## Input

- Keyboard input has its own interface
- Network input has its own interface
- Terminal input?
- Mouse? (Inventing)

Same program, multiple input-output



# MVC History



## Output

- Terminal Screen
- Network (to ATM machines)
- Cashier screen
- Each screen has its own format, and required appropriate layouting
- Layouting change frequently

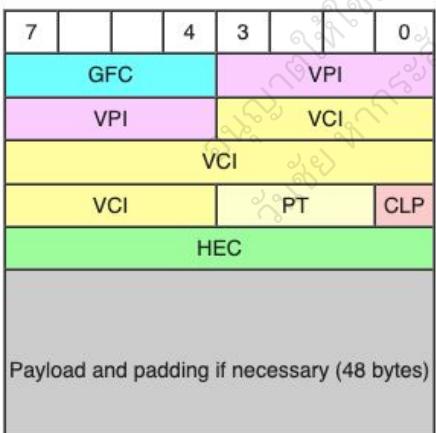


## Cell structure [edit]

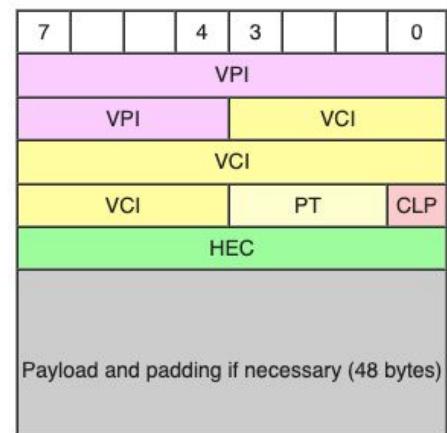
An ATM cell consists of a 5-byte header and a 48-byte payload. The payload size of 48 bytes was chosen as described above.

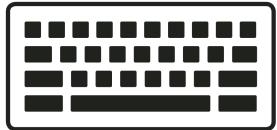
ATM defines two different cell formats: [user–network interface \(UNI\)](#) and [network–network interface \(NNI\)](#). Most ATM links use UNI cell format.

**Diagram of a UNI ATM cell**



**Diagram of an NNI ATM cell**

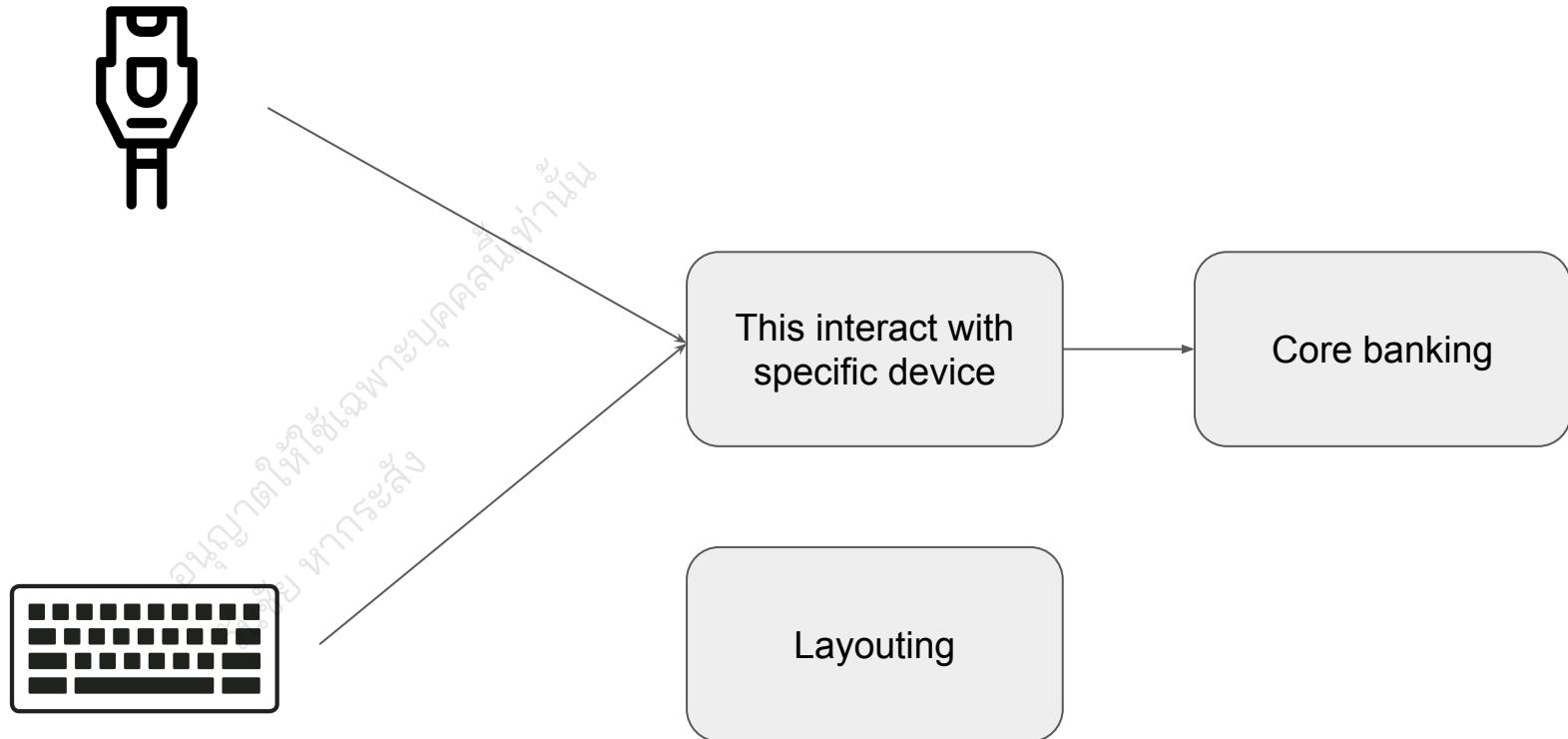




This interact with  
specific device

Core banking

Layouting

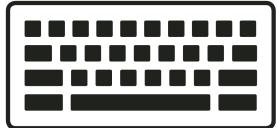




Here comes the input "0000111"

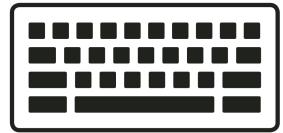
This interact with  
specific device

Core banking



Here comes the input  
"0110111"

Layouting

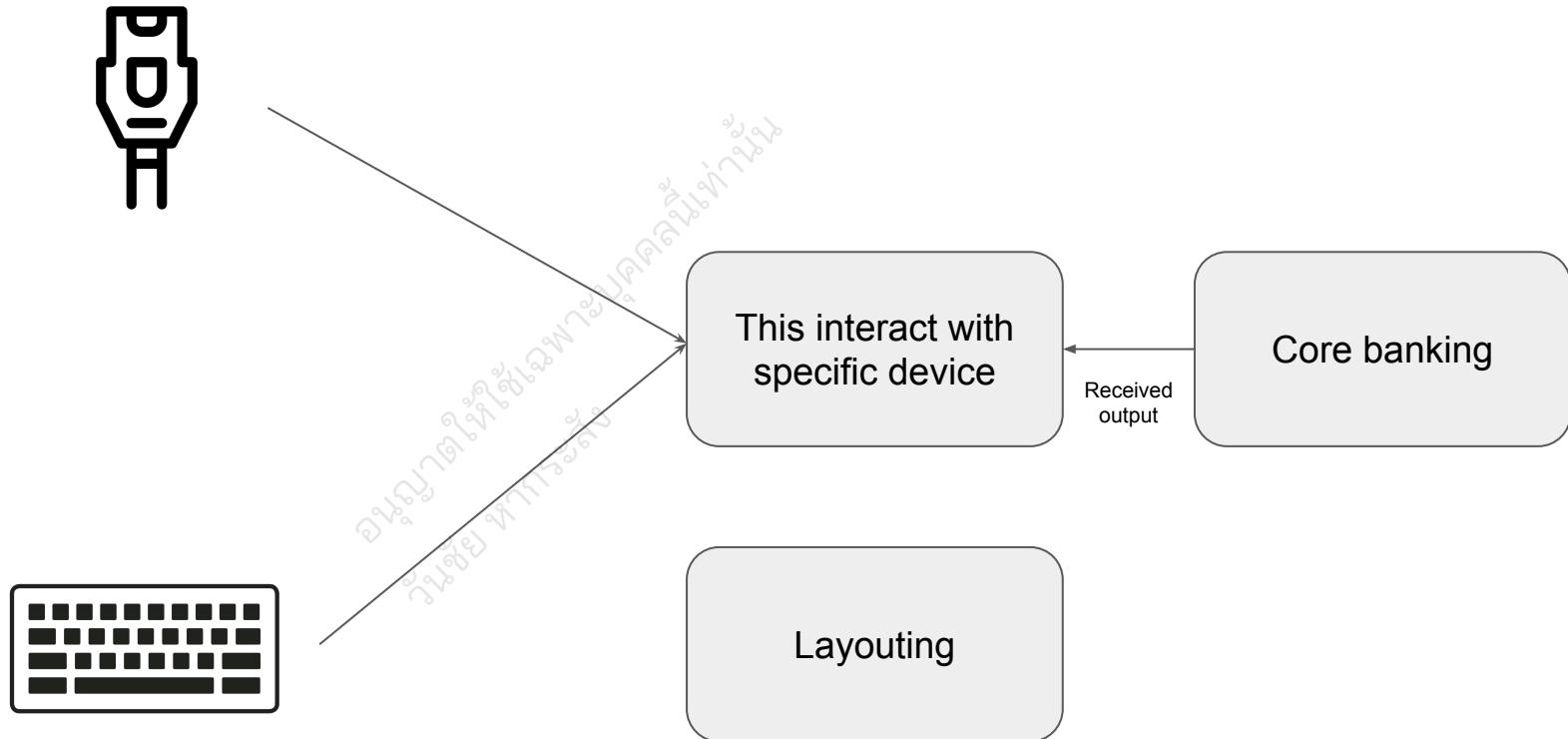


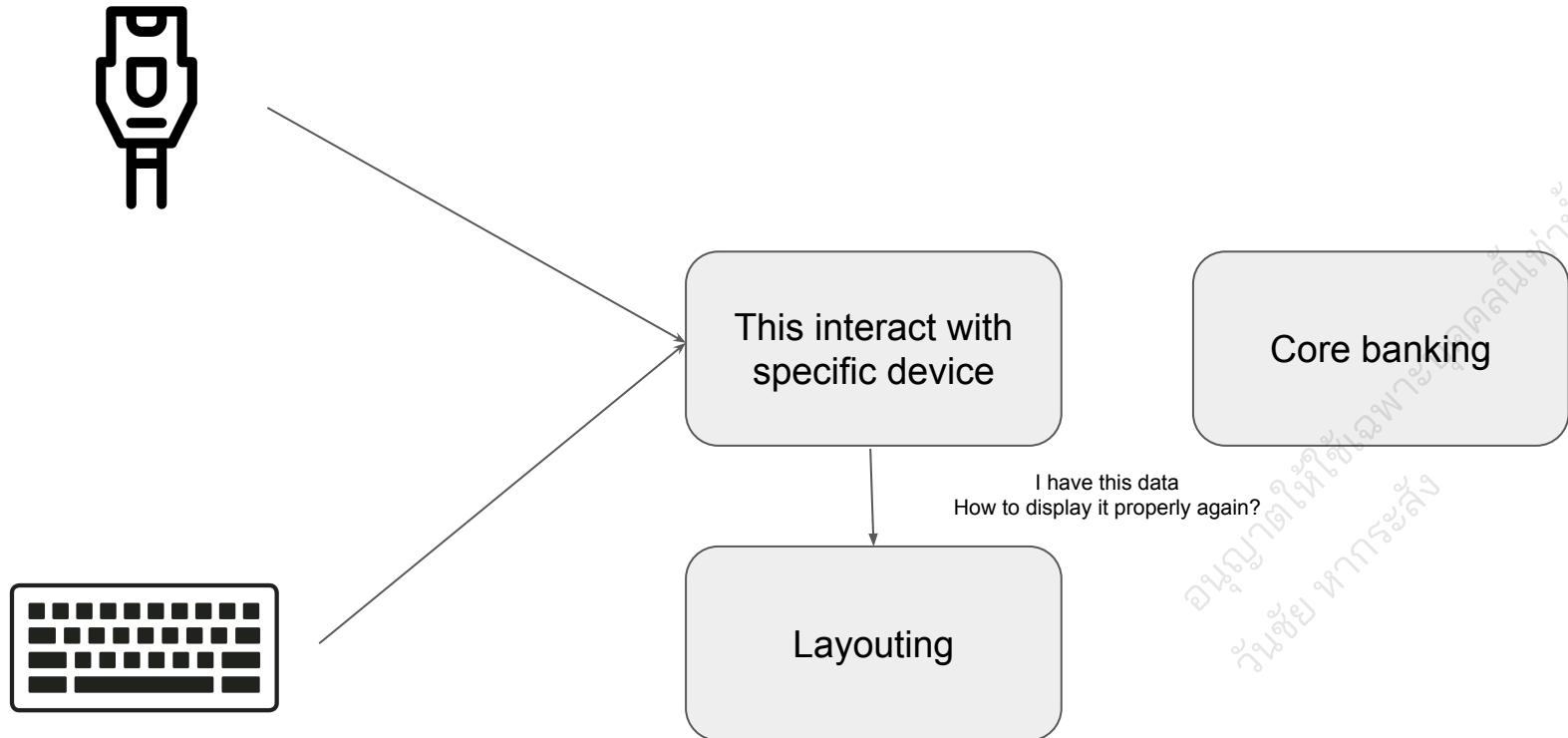
Layouting

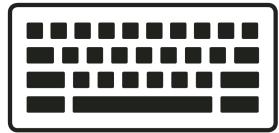
This interact with  
specific device

Core banking

Interpreted  
input







This interact with  
specific device

Layouting

Core banking

Here you go

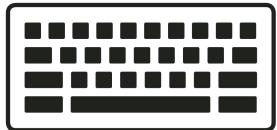


ឧប្បត្តមន្ត្រី  
រៀបចំការងារ

Let me think about  
what to say to others

This interact with  
specific device

Core banking

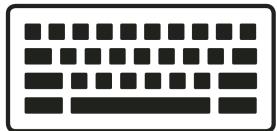


Layouting



Here you go

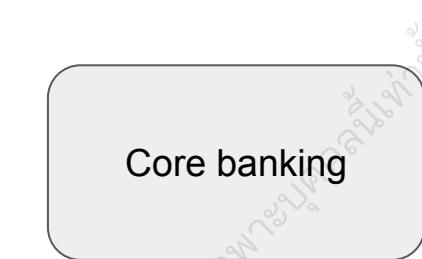
This interact with  
specific device

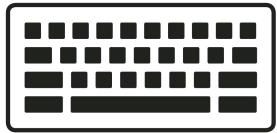


Here you go

Layouting

Core banking





องค์กรที่ใช้สถาปัตยกรรม  
วัสดุ ห้องน้ำ

Controller

Model

View



**MVC**

- Controller handle input-output
- View tell how to display
- Model is the business and internal logic

And that's MVC

อนุญาตให้ใช้สิ่งพิมพ์บนคลิป  
วันชัย ห้ามระลัง



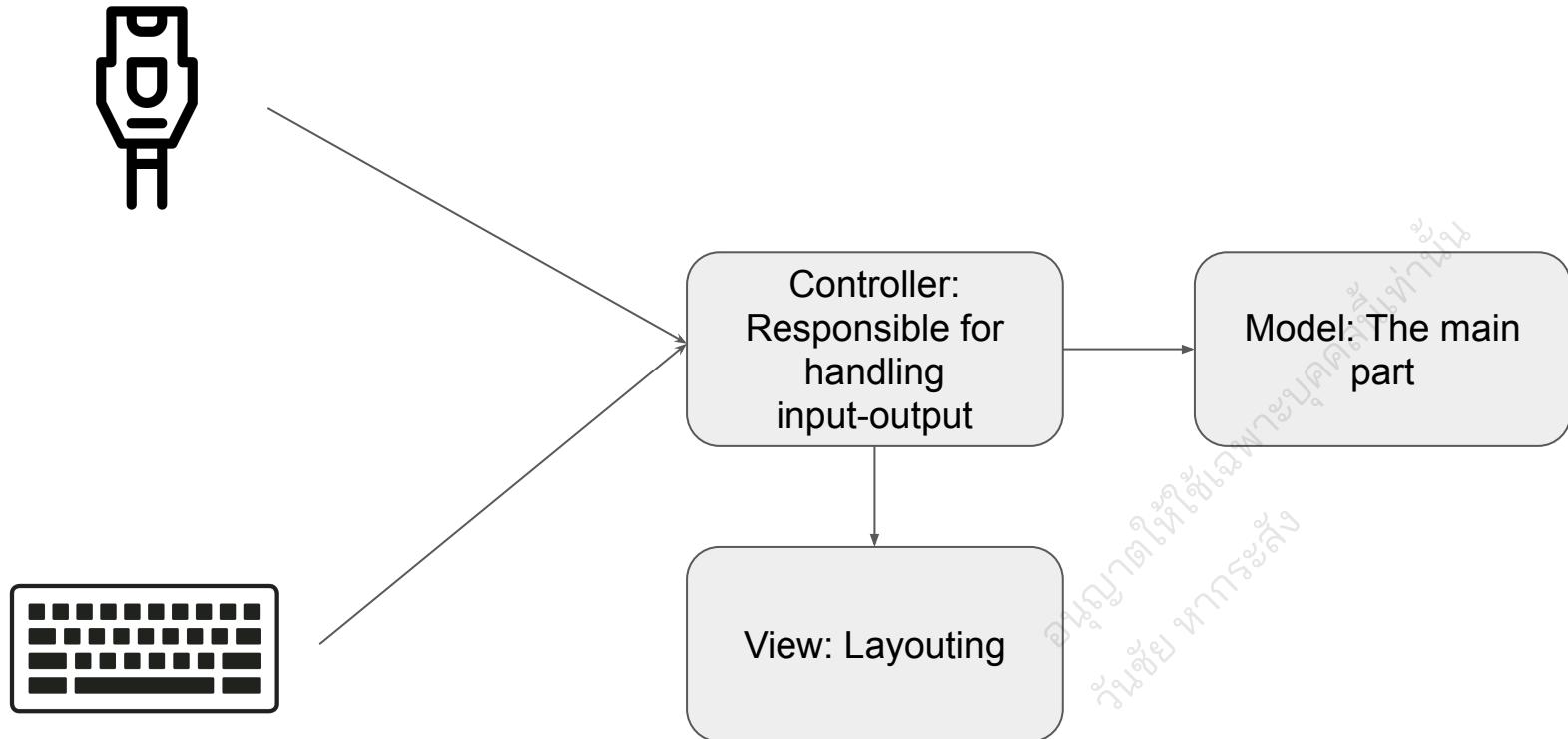
# Misconception

You might heard: “Model is about a database”

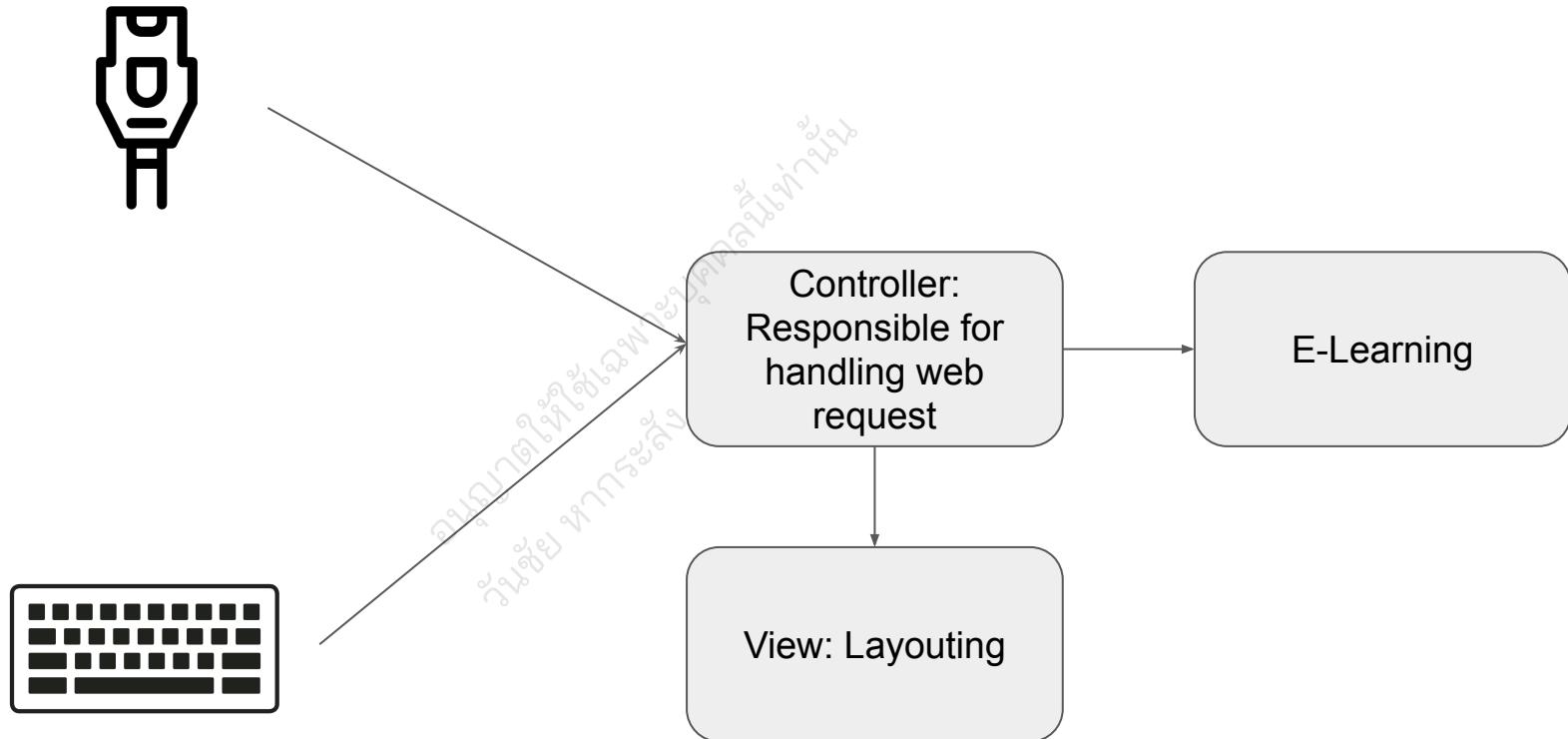
That statement is both correct and incorrect



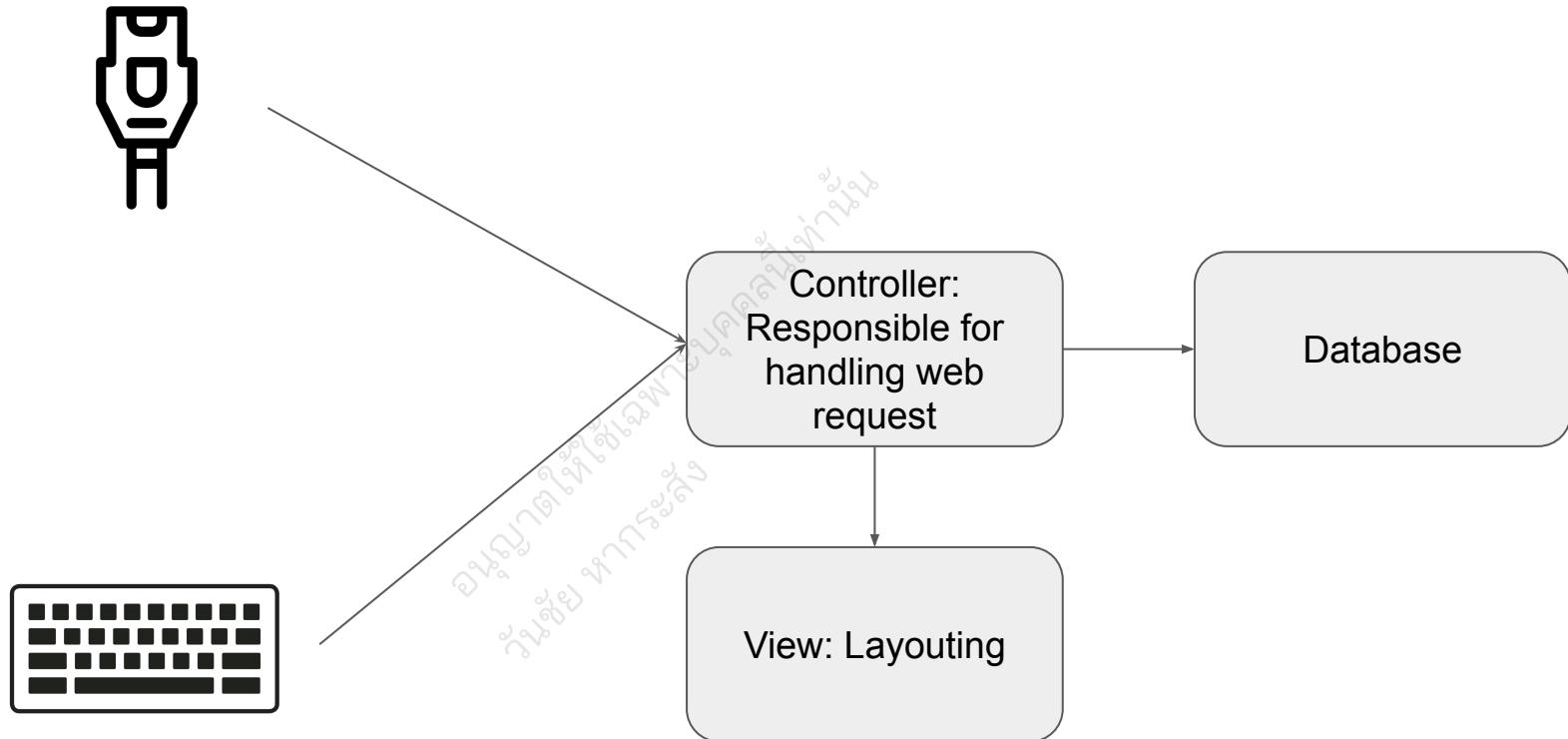
# MVC Concept



# Our application



# Typical Web application



# Misconception

Since in a typical modern web application, we normally don't work on top of legacy system.

Database is a third-party in this case.

More accurate statement:

"In a standalone MVC web application, the model is usually handle the database."





# MVC #3

## Applying

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

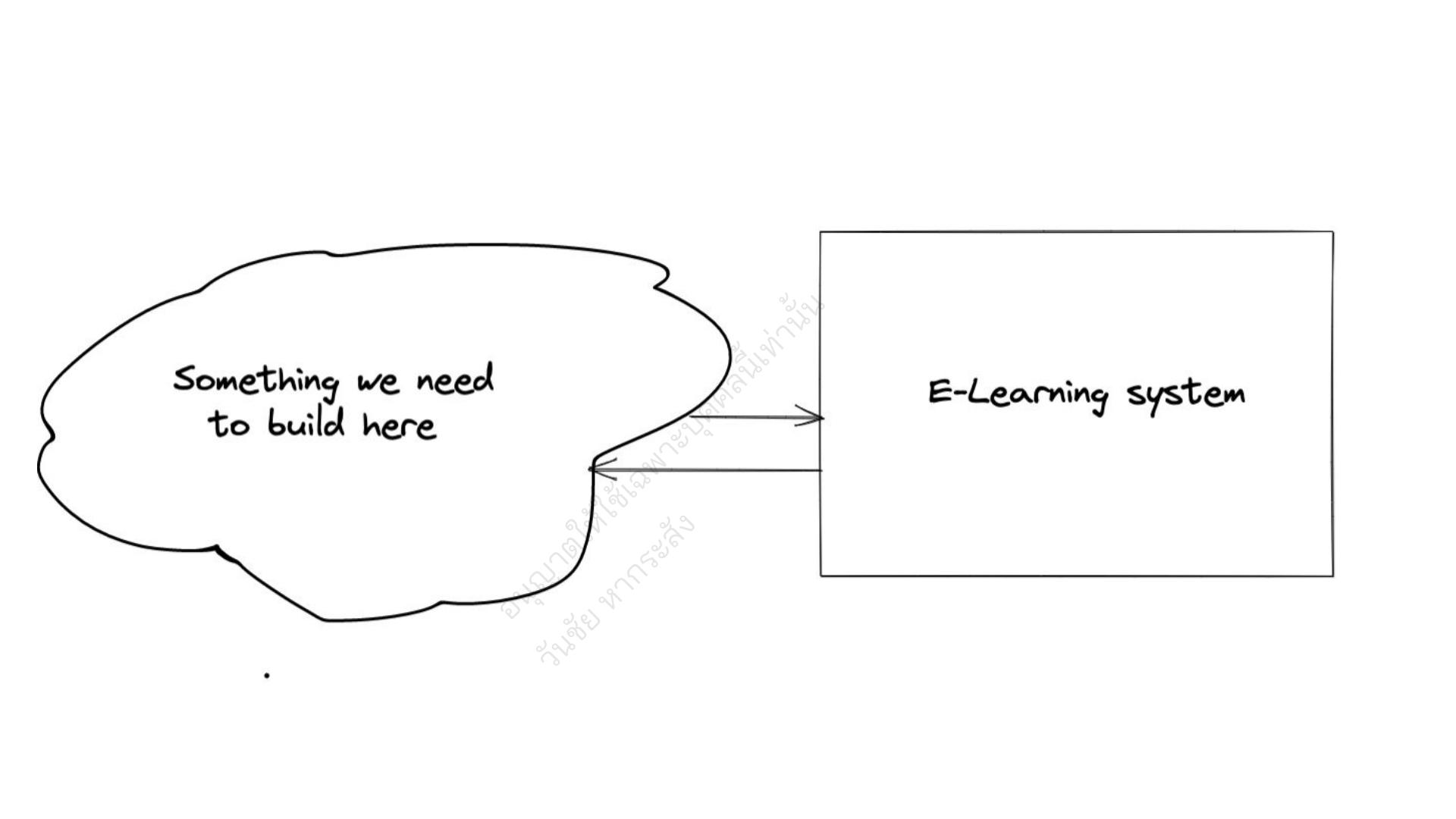
© 2021 Skooldio Co., Ltd.  
All rights reserved.



# Our requirement

ความต้องการที่ต้องพำบุคคลนักเรียน  
ร่วมช่วยเหลือรักษาสิ่งแวดล้อม





Something we need  
to build here

E-Learning system



```
class SkoolDioApp {  
    public void BuyCourse( ) {  
        // Interprete input  
  
        // Call third party  
  
        // Create an layout  
    }  
}
```

## Naive implementation

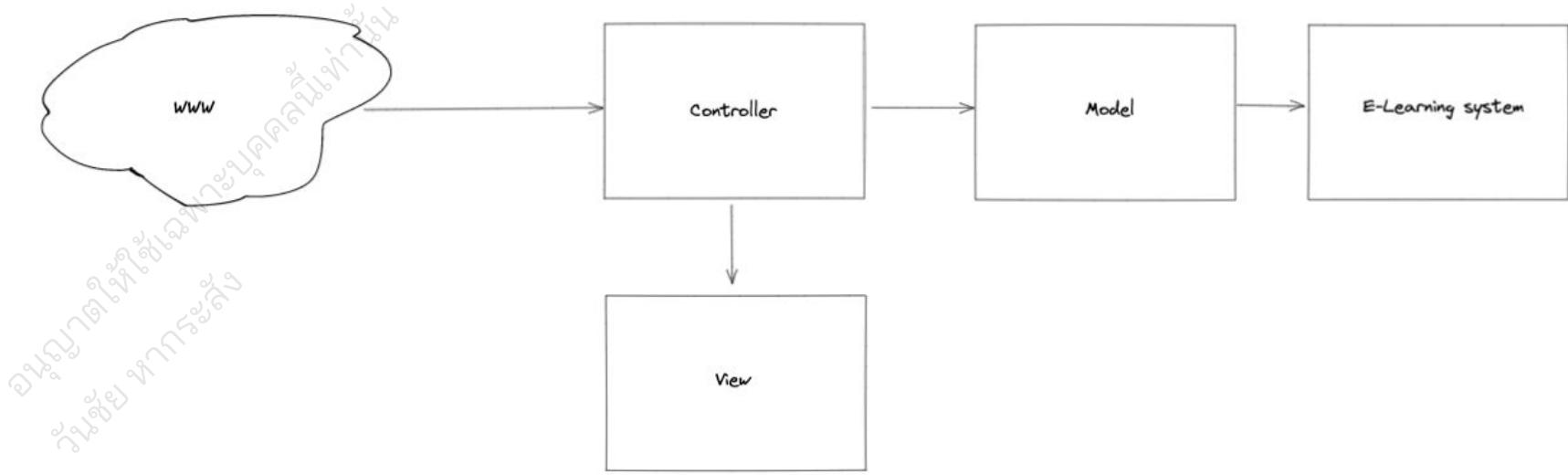
# MVC Usage

อนุญาตให้เข้าชมเฉพาะบุคคลเท่านั้น  
ก่อนซื้อ ห้ามระบุชื่อ

## Naive implementation

- One big class that do everything
- Everyone need to learn every stack and become proficient in everything to contribute
- Will I break the code above?







```
class SkooldioController {
    public void BuyCourse(Request request) {
        BuyCourseData b = new BuyCourseData(request.body, request.header);
        BuyCourseResult result = this.CourseModel.Buy(BuyCourseData);
        View v = new BuyCourseView(BuyCourseResult);
        return v.DisplayString();
    }
}
```

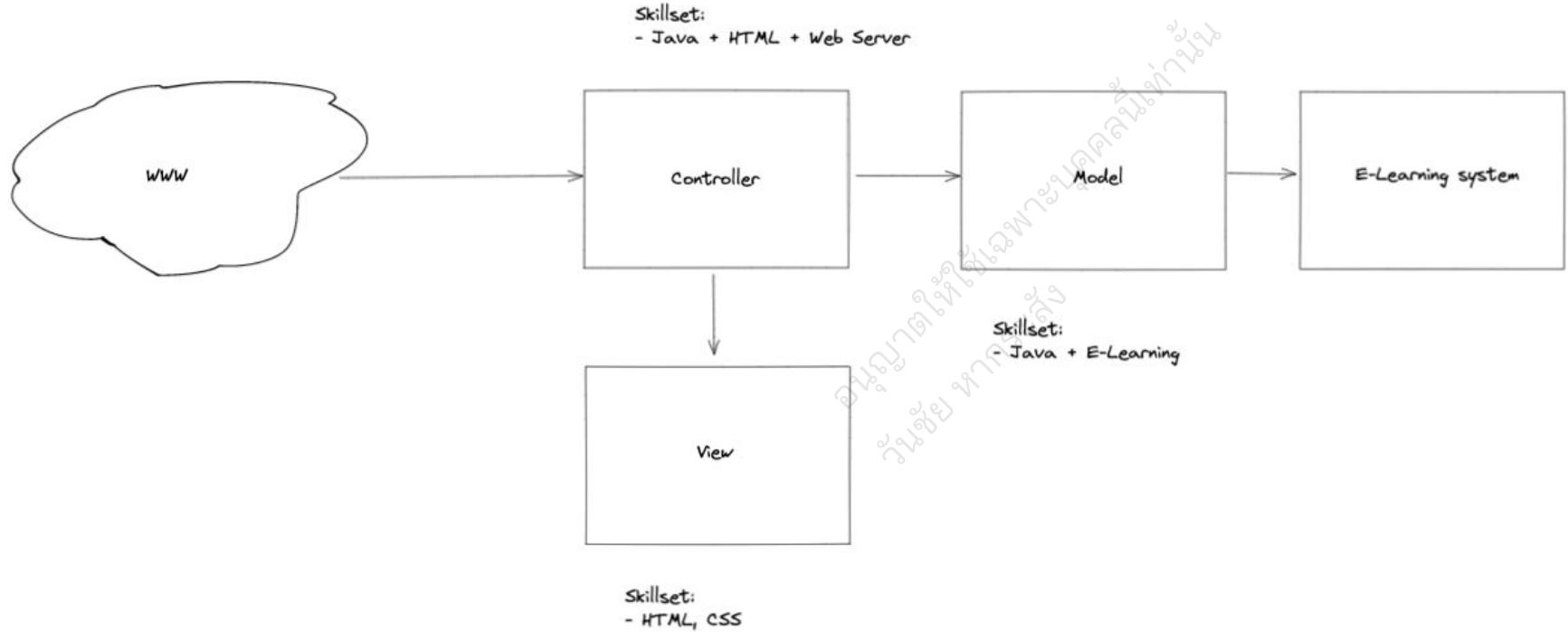


```
class BuyCourseView {
    public string DisplayString() {
        return "Buy result:" + (this.BuyCourseResponse.Success ? "Success" : "Failed");
    }
}
```

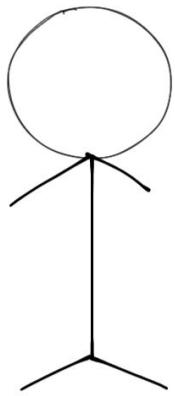


```
class CourseModel {  
    public BuyResult Buy(BuyCourseData input) {  
        // .... Connect to E-Learning here  
    }  
}
```

Let's look at the team & skill required

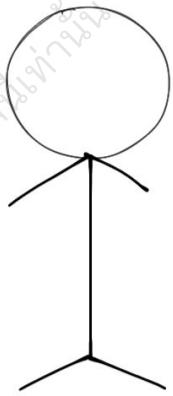


Web  
developer



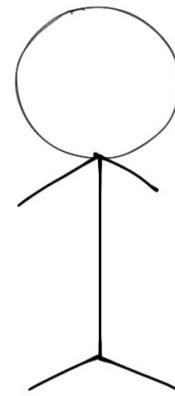
HTML, CSS,  
Java

E-Learning  
expert



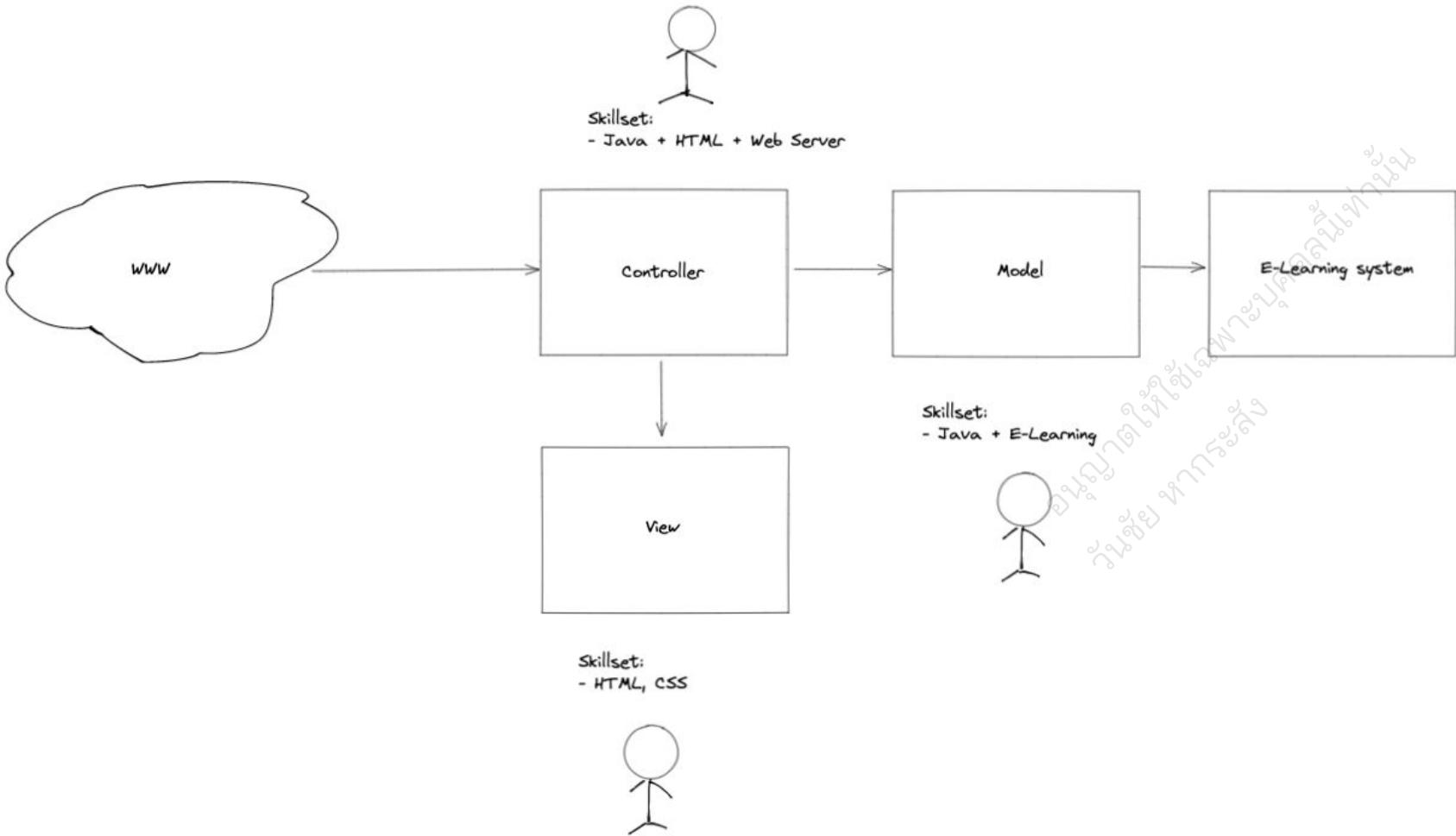
E-Learning  
API, Java

Web  
developer



HTML, CSS,  
Java

อนุญาตให้ใช้เเพ侈บุคคลเดียว  
วันนี้ ห้ามลาก่อน



**It's fit together**

อนุญาตให้ใช้เพื่อการศึกษาเท่านั้น  
วันที่ 9 มกราคม พ.ศ. 2562



**MVC**

อนุญาตให้ใช้สิ่งพิมพ์ทางบัณฑิต  
วุฒิชั้น ห้ารุ่นสูง

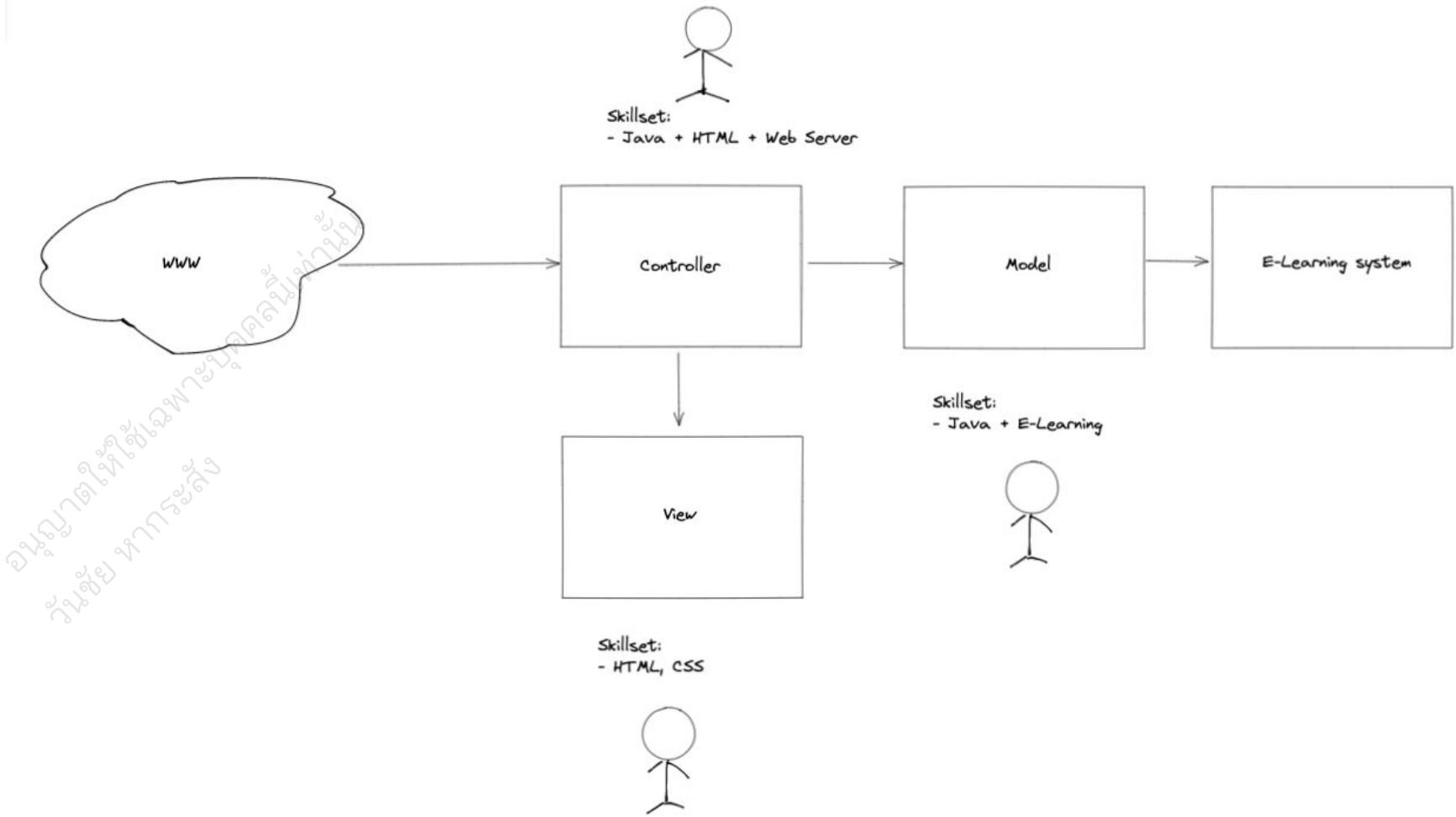
MVC allow enabled better collaboration between E-Learning expert and Web developer

It's tell us exactly where can each of them contribute

Collaboration of specialist

- Internal system specialist
- Device specific specialist
- User interface specialist





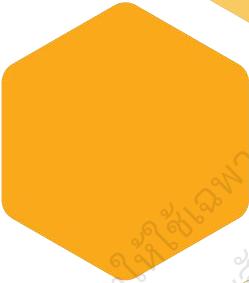


# Clean Architecture #1

## Requirement

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.



# Let's expand Skooldio

ฉบับภาษาไทยเพื่อพากบุคคลทั่ว  
โลกช่วยห้ารรับ

Something we build

Certification?

E-Learning system

Payment

**Let's design for this**



# Clean Architecture #2

## What is Clean Architecture?

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

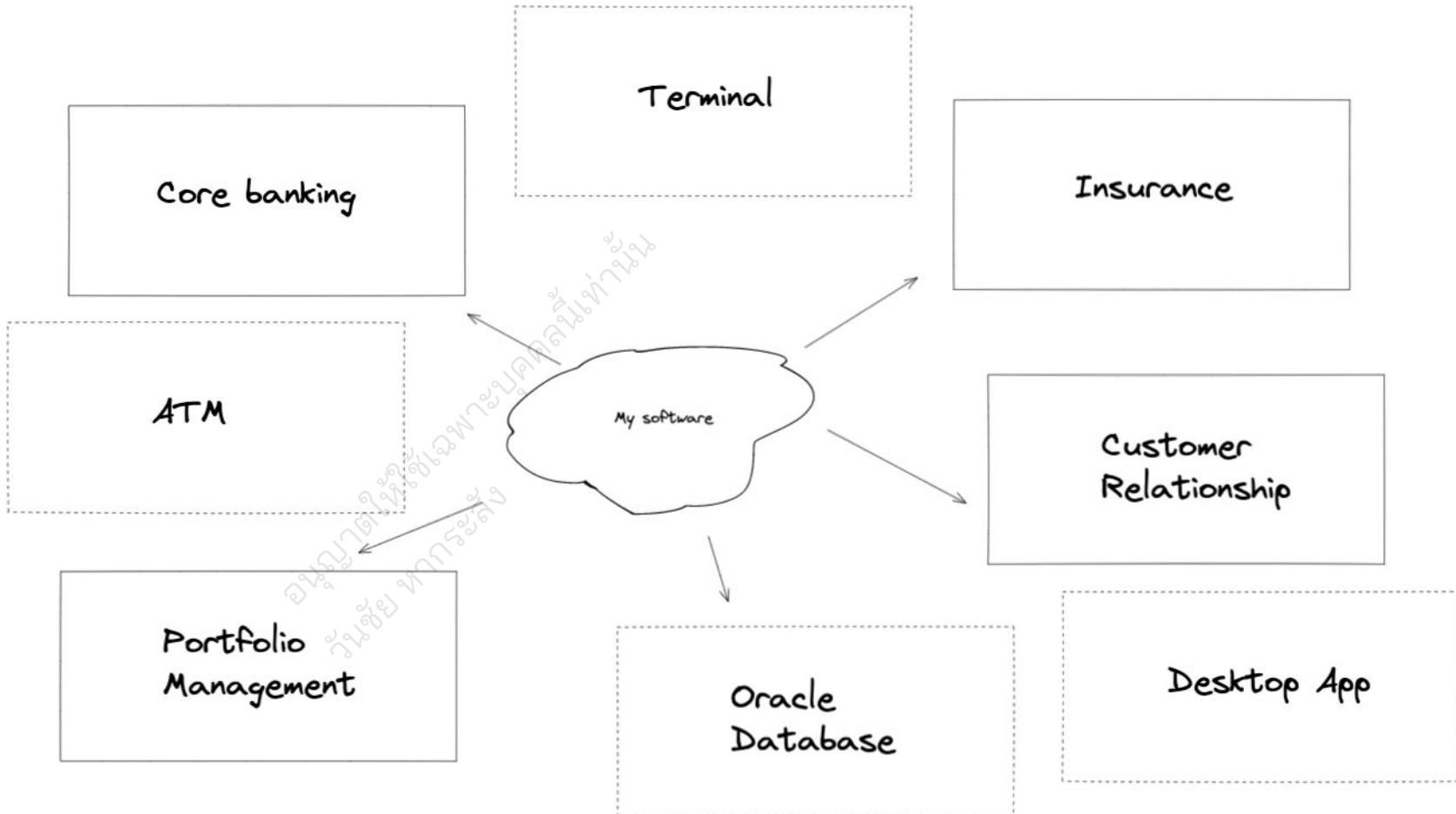
© 2021 Skooldio Co., Ltd.  
All rights reserved.



# **Clean Architecture**

- Hexagonal Architecture was introduced by Alistair Cockburn
- Clean Architecture was introduced by Bob Martin (Uncle Bob)
- An attempted to solidify the idea of “architecture independent of framework”
- The nature of work is a big enterprise with many legacy systems



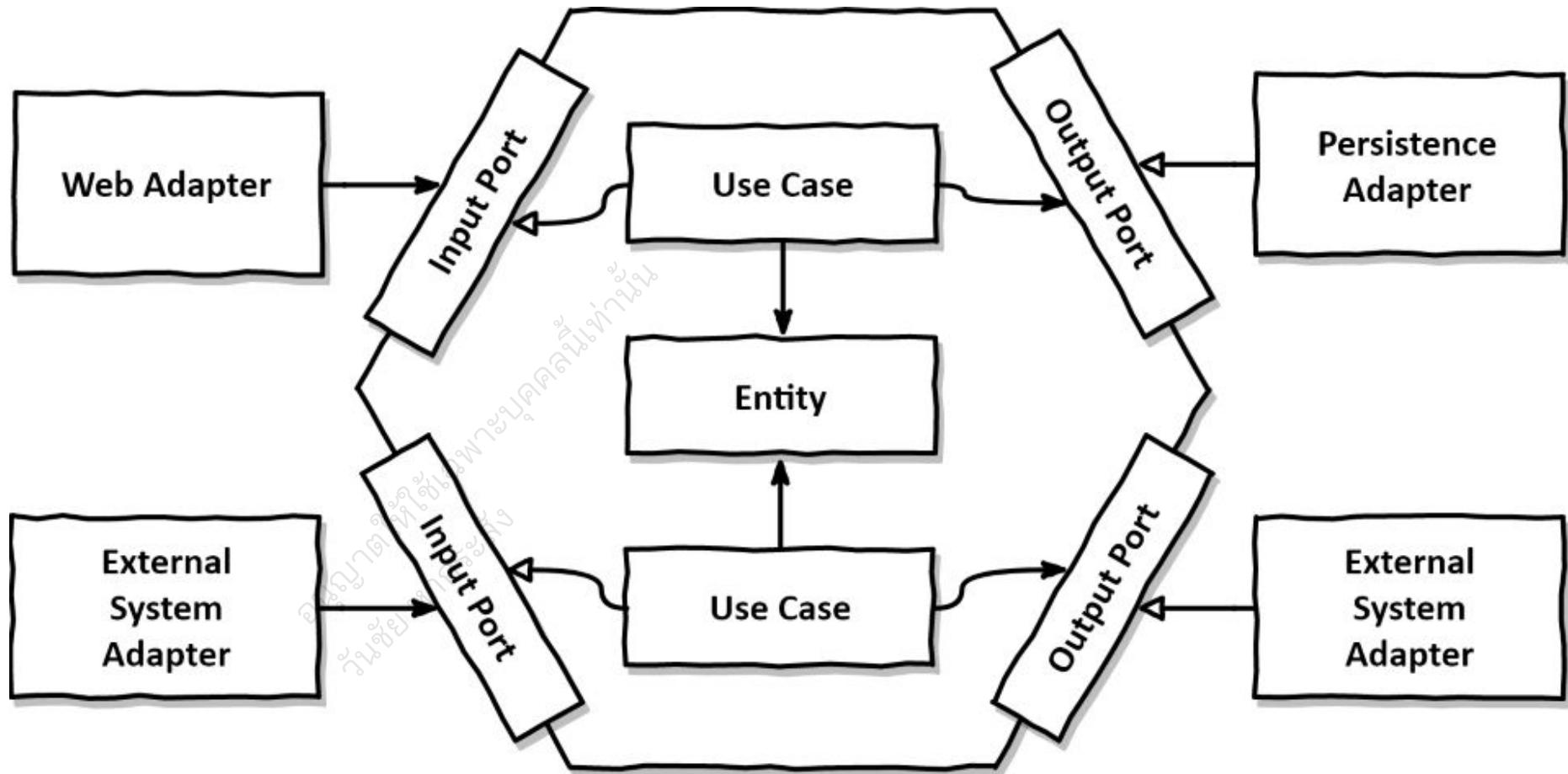




# **Clean Architecture**

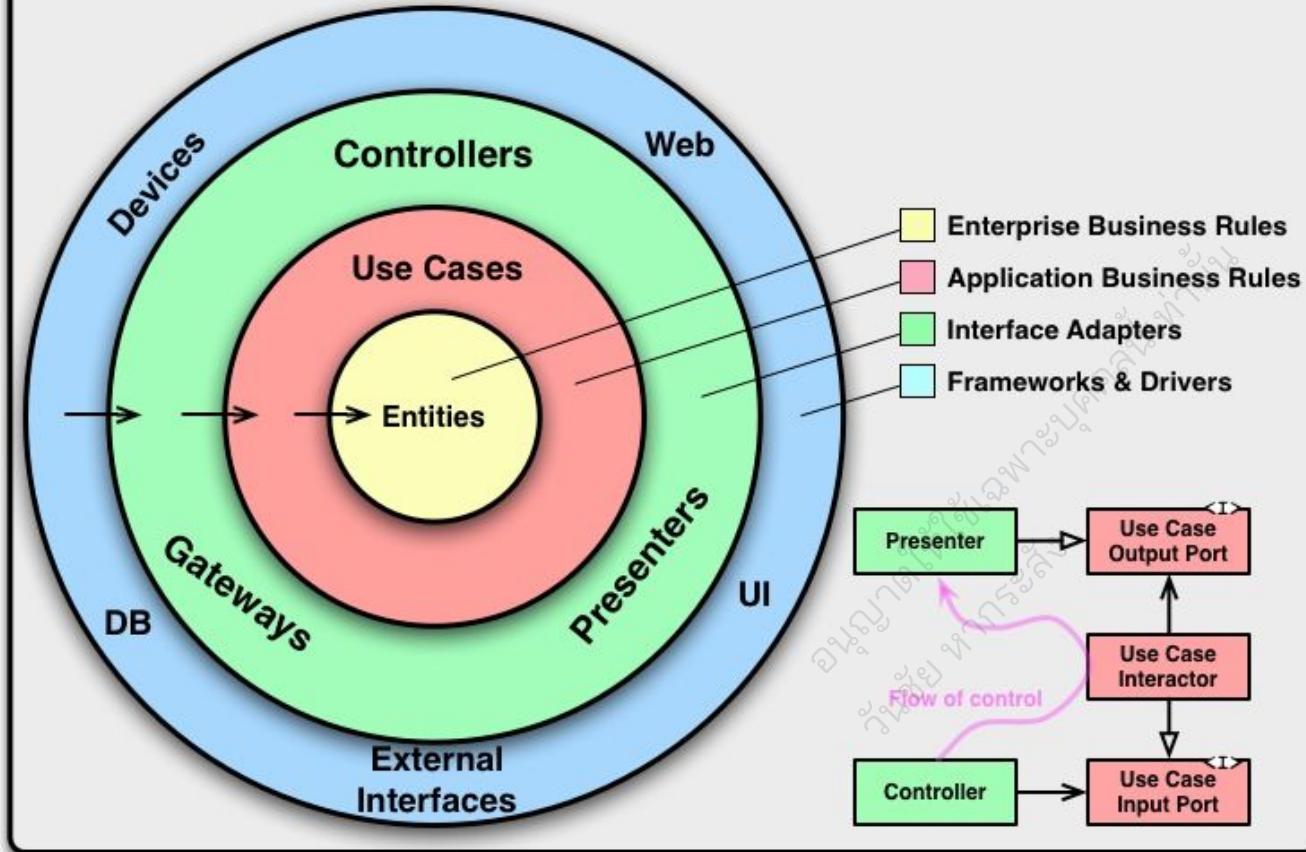
- Traditional architecture is not enough
- What is my software?
- What is my business logic vs. reliance on legacy?
- The core idea is to have a clean separation between what is the core of my software and what part do we rely on others





Alistair Cockburn presenting “Hexagonal Architecture”

# The Clean Architecture



Uncle bob adapt to “Clean Architecture”

# Entities

Shared business rules

บัญชีรายรับ-จ่าย  
วันชัย ห้าระลัง



# Use-case

- Application specific business rules

อนุญาตให้เข้ามาทำงาน  
วันนี้ ทำการตั้งค่า



# Interface Adapter

The software in this layer is a set of adapters that convert data from the format most convenient for the use cases and entities, to the format most convenient for some external agency such as the Database or the Web



# Framework & Driver

- Connecting to all 3rd-party
- Handle input/output

ระบบ  
อัตโนมัติ  
สำหรับ  
การสั่ง



## Misconception #1:

Clean Architecture is not all about these details

## Only Four Circles?

No, the circles are schematic. You may find that you need more than just these four. There's no rule that says you must always have just these four. However, *The Dependency Rule* always applies. Source code dependencies always point inwards. As you move inwards the level of abstraction increases. The outermost circle is low level concrete detail. As you move inwards the software grows more abstract, and encapsulates higher level policies. The inner most circle is the most general.

**It's about dependency rules**

## The Dependency Rule

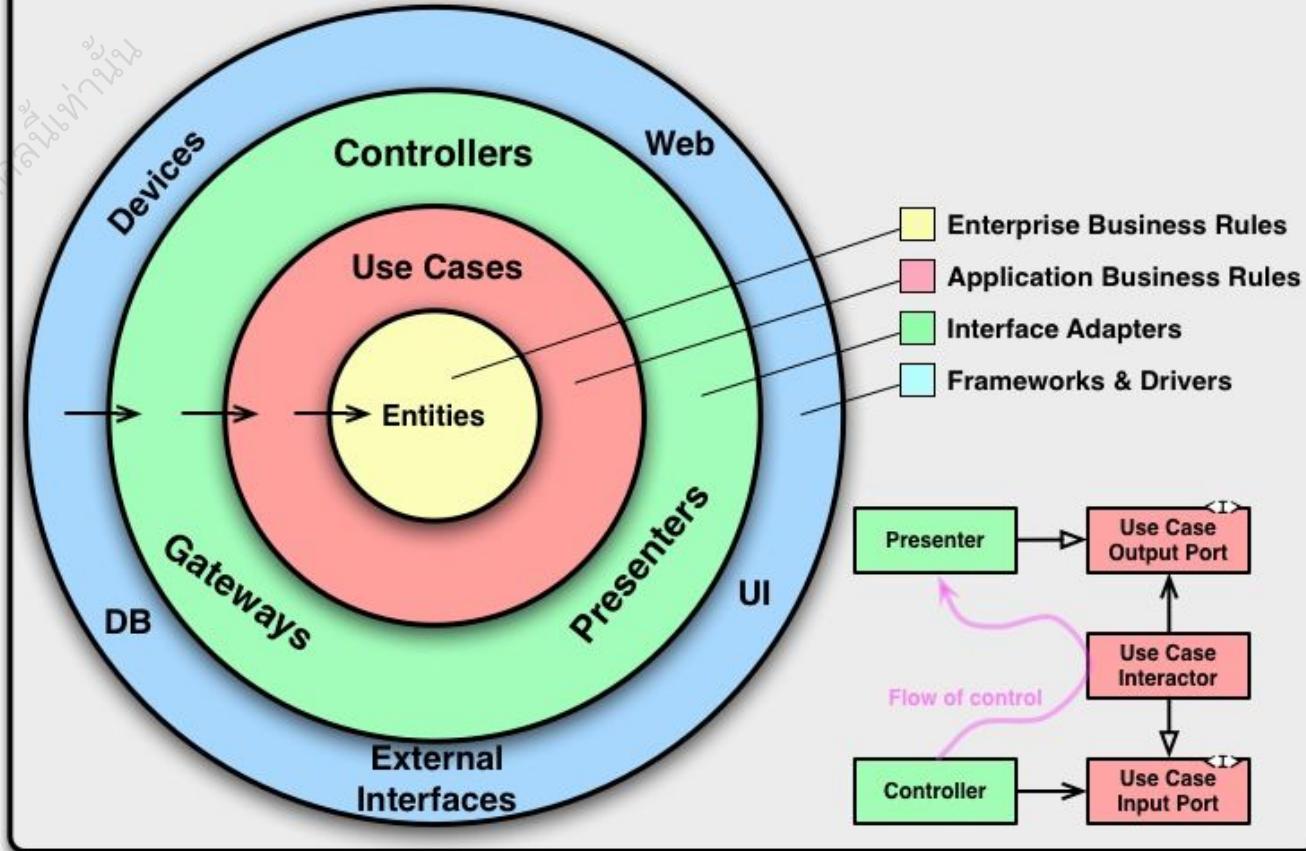
The concentric circles represent different areas of software. In general, the further in you go, the higher level the software becomes. The outer circles are mechanisms. The inner circles are policies.

The overriding rule that makes this architecture work is *The Dependency Rule*. This rule says that *source code dependencies* can only point *inwards*. Nothing in an inner circle can know anything at all about something in an outer circle. In particular, the name of something declared in an outer circle must not be mentioned by the code in the an inner circle. That includes, functions, classes. variables, or any other named software entity.

To sum up:

Core codebase should not have to know  
about all the 3rd-party

# The Clean Architecture





# Clean Architecture

- You separated all 3rd-party out using dependency rules
- You have a core codebase that is independent of any third-party
- That's it
- It's all about dependency rules
- Dependency Inversion might be needed





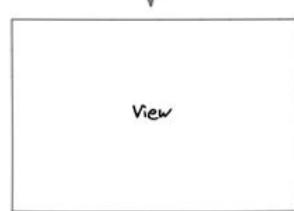
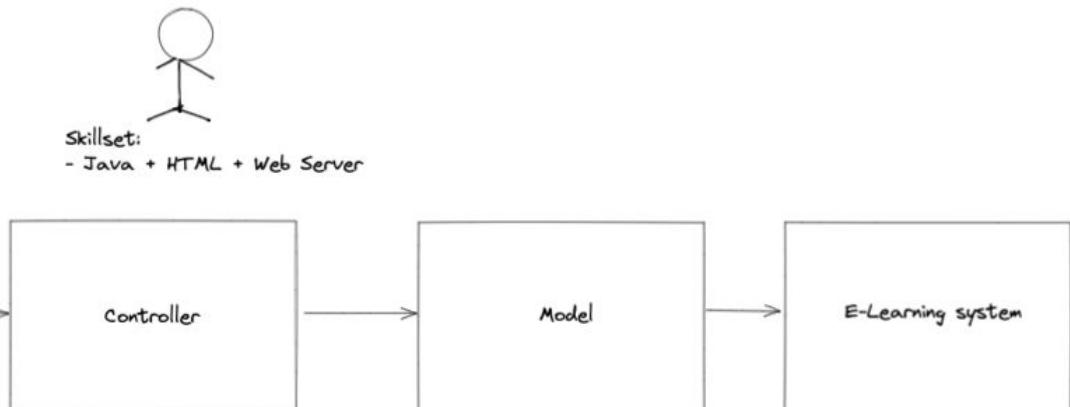
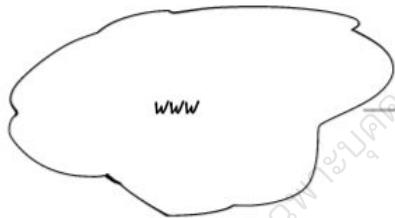
# Clean Architecture #3

## Applying

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.

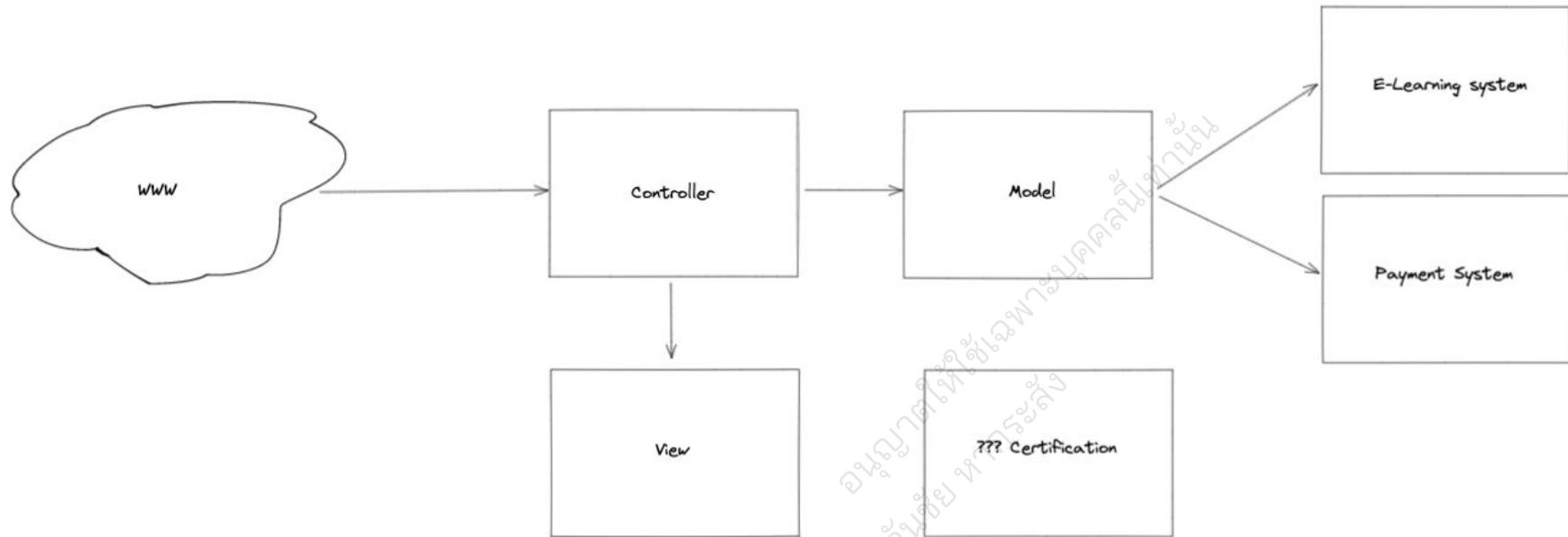




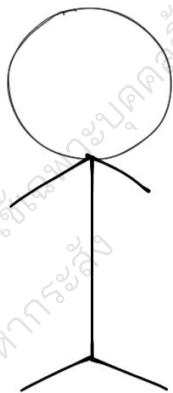
Skillset:  
- HTML, CSS



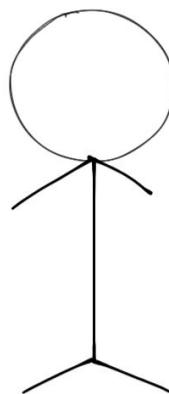
Skillset:  
- Java + E-Learning



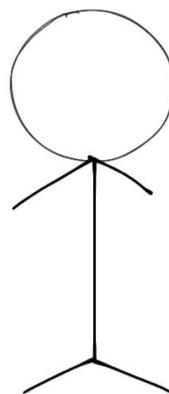
Web  
developer



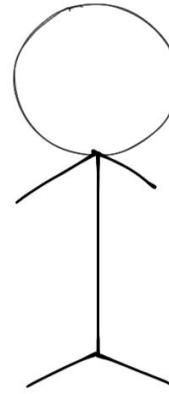
E-Learning  
Expert ???



Payment  
Expert ???



Web  
developer



# Separating 3rd-Party

อบรมการให้เชื่อพำนุกคลน์ห้าม  
วันชัย ห้ากระลัง

ມີມາດໃຫ້ເຂົ້າມືກອນມາ  
ບໍ່ມີມາດໃຫ້ເຂົ້າມືກອນມາ

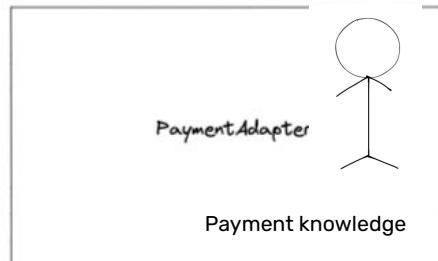
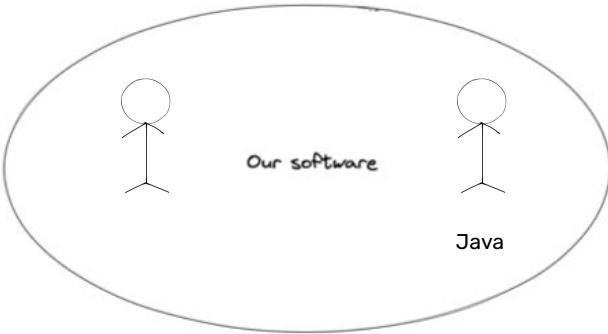
www Adapter

Our software

Payment Adapter

CertificationDatabaseAdapter

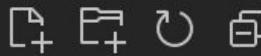
ELearningAdapter



Everyone need to know domain knowledge

อัมดูตได้รับการอนุมัติ  
วันนี้ยังคงอยู่

**That's it. This is a Clean Architecture**



## SKOOLDIO-CLEAN

### Adapters

J CertificationDatabaseAdapter.java

J ELearningAdapter.java

J PaymentAdapter.java

### Entities

J Course.java

J Payment.java

J Student.java

### Sdks

### UseCases

J CoursesUseCases.java

J PaymentUseCases.java

●

8

```
package UseCases;

import Adapters.ELearningAdapter;
import Adapters.PaymentAdapter;
import Entities.Course;
import Entities.Student;
import Entities.Payment;

public class CoursesUseCases {
    private PaymentAdapter paymentAdapter;
    private ELearningAdapter eLearningAdapter;

    public CoursesUseCases(PaymentAdapter paymentAdapter, ELearningAdapter eLearningAdapter) {
        this.eLearningAdapter = eLearningAdapter;
        this.paymentAdapter = paymentAdapter;
    }

    public void BuyACourse(Course course, Student student, Payment payment) {
        if (!course.IsActive()) {
            throw new CourseNotActiveException();
        }
        if (!student.Blacklisted()) {
            throw new CannotBuyException();
        }
        if (this.paymentAdapter.Pay(payment) == PaymentResult.Success) {
            this.eLearningAdapter.RegisterCourse(course, student);
        }
    }
}
```

```
package UseCases;

import Adapters.ELearningAdapter;
import Adapters.PaymentAdapter;
import Entities.Course;
import Entities.Student;
import Entities.Payment;

public class CoursesUseCases {
    private PaymentAdapter paymentAdapter;
    private ELearningAdapter eLearningAdapter;

    public CoursesUseCases(PaymentAdapter paymentAdapter, ELearningAdapter eLearningAdapter) {
        this.eLearningAdapter = eLearningAdapter;
        this.paymentAdapter = paymentAdapter;
    }

    public void BuyACourse(Course course, Student student, Payment payment) {
        if (!course.IsActive()) {
            throw new CourseNotActiveException();
        }
        if (!student.Blacklisted()) {
            throw new CannotBuyException();
        }
        if (this.paymentAdapter.Pay(payment) == PaymentResult.Success) {
            this.eLearningAdapter.RegisterCourse(course, student);
        }
    }
}
```

```
package UseCases;

import Adapters.ELearningAdapter;
import Adapters.PaymentAdapter;
import Entities.Course;
import Entities.Student;
import Entities.Payment;

public class CoursesUseCases {
    private PaymentAdapter paymentAdapter;
    private ELearningAdapter eLearningAdapter;

    public CoursesUseCases(PaymentAdapter paymentAdapter, ELearningAdapter eLearningAdapter) {
        this.eLearningAdapter = eLearningAdapter;
        this.paymentAdapter = paymentAdapter;
    }

    public void BuyACourse(Course course, Student student, Payment payment) {
        if (!course.IsActive()) {
            throw new CourseNotActiveException();
        }
        if (!student.Blacklisted()) {
            throw new CannotBuyException();
        }
        if (this.paymentAdapter.Pay(payment) == PaymentResult.Success) {
            this.eLearningAdapter.RegisterCourse(course, student);
        }
    }
}
```

Wait.

Use cases should not depend on  
adapters right?

# Dependency inversion

อนุรักษ์ให้ใช้เฉพาะบุคคลเท่านั้น  
ก่อนซ่อน หากจะซ่อน

# Our app

Still, buying a course required us to do these

- Connect to E-Learning system
- Pay

How can we claim that use-cases does not depend on 3rd party?

Answer: Dependency inversion



```
interface PaymentAdapter {
    public void Pay();
    // Others ...
}

interface ELearningAdapter {
    public void ViewCourse();
    // Others...
}

public class CoursesUseCases {
    private PaymentAdapter paymentAdapter;
    private ELearningAdapter eLearningAdapter;

    public CoursesUseCases(PaymentAdapter paymentAdapter, ELearningAdapter eLearningAdapter) {
    }

    public void BuyACourse(Course course, Student student, Payment payment) {
        // Use adapters
    }
    // Methods
}
```



```
public class CoursesUseCases {  
  
    public CoursesUseCases( ) {  
  
    }  
  
    public void BuyACourse(Course course, Student student, Payment payment) {  
        // Use adapters  
        ELearningAdapter = new ELearningAdapter(setupData);  
    }  
    // Methods  
}
```

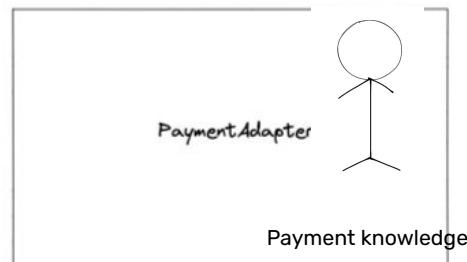
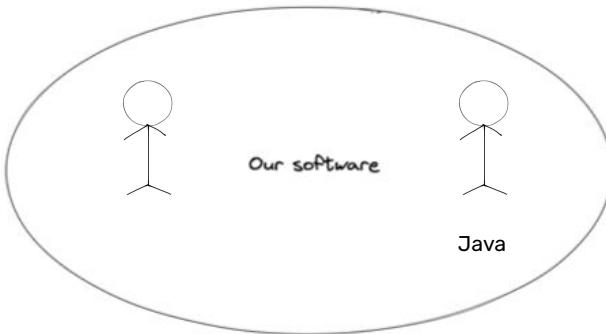
# Dependency inversion

- Interfaces of adapter is declared by core module
- 3rd-party need to implement a class according to the interfaces
- Core does not know what 3rd-party actually do, just use the interface

Use-case does not know any internal detail except for interface



```
class CourseController {  
    public BuyResult Buy() {  
        BuyCourseData buyCourseData = BuyCourseData.fromRequest(request);  
        return this.courseUseCases.BuyACourse(buyCourseData);  
        // Or map to response  
    }  
}  
}
```



Everyone need to know domain knowledge

Adapters use entities as a way to communicate

```
package UseCases;

import Adapters.ELearningAdapter;
import Adapters.PaymentAdapter;
import Entities.Course;
import Entities.Student;
import Entities.Payment;

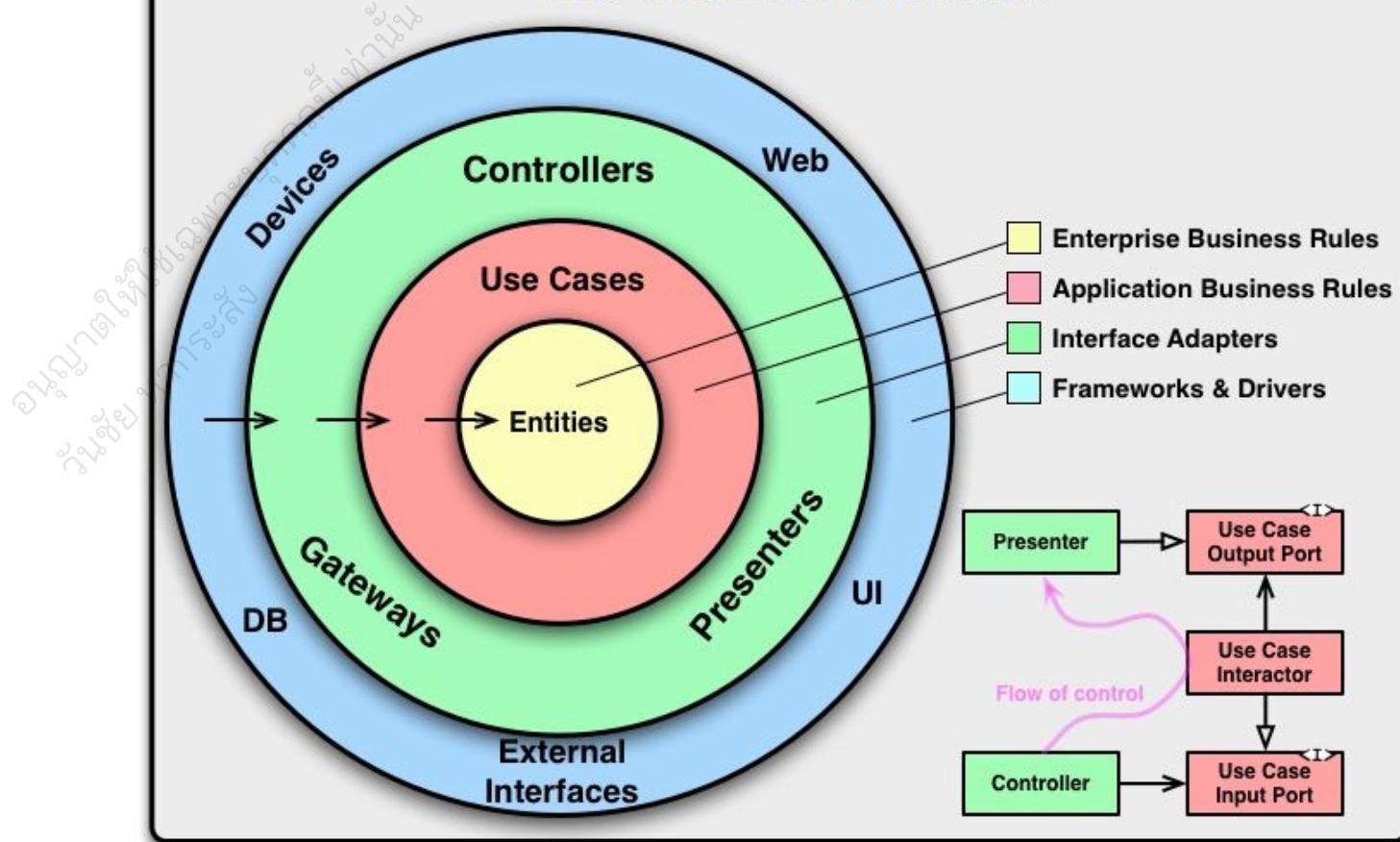
public class CoursesUseCases {
    private PaymentAdapter paymentAdapter;
    private ELearningAdapter eLearningAdapter;

    public CoursesUseCases(PaymentAdapter paymentAdapter, ELearningAdapter eLearningAdapter) {
        this.eLearningAdapter = eLearningAdapter;
        this.paymentAdapter = paymentAdapter;
    }

    public void BuyACourse(Course course, Student student, Payment payment) {
        if (!course.IsActive()) {
            throw new CourseNotActiveException();
        }
        if (!student.Blacklisted()) {
            throw new CannotBuyException();
        }
        if (this.paymentAdapter.Pay(payment) == PaymentResult.Success) {
            this.eLearningAdapter.RegisterCourse(course, student);
        }
    }
}
```

Use-cases does not know implementation, only interfaces of adapters

# The Clean Architecture



That's it. We have a Clean Architecture

# Key concept

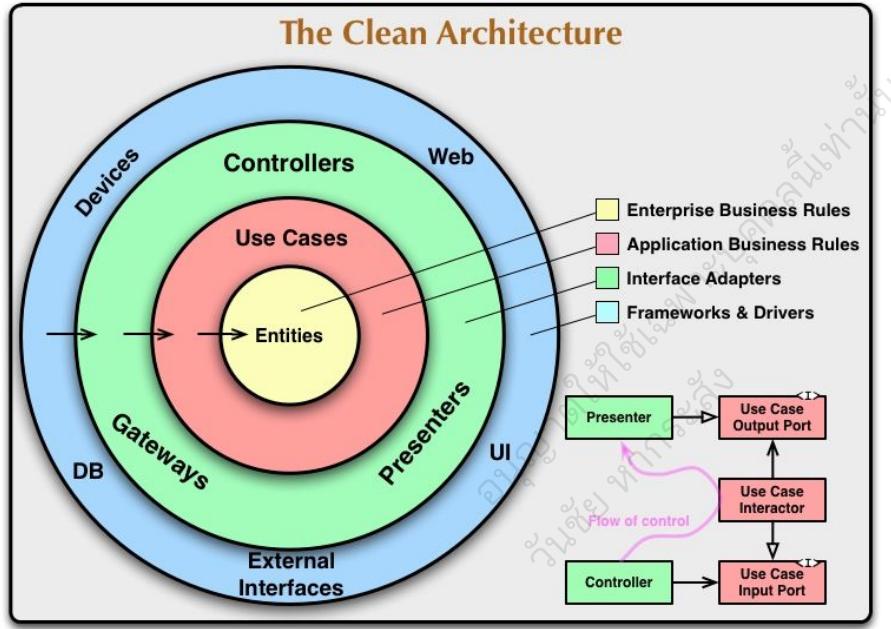
- Every layer need to understand domain entities and use-cases
- As a result, every programmer need to understand domain and use-cases
- But not every programmer need to understand 3rd party



# Key concept

- And yes, the idea is simply to emphasize domain knowledge
- Tech expert (SQL, etc.) who knows nothing about requirement
- General programmer who understand deeply about what we are building
- Who do we design for? Who do we push out?





Clean architecture emphasize domain knowledge by enforcing dependency inwards



มนต์มนต์มนต์มนต์มนต์  
มนต์มนต์มนต์มนต์มนต์  
มนต์มนต์มนต์มนต์มนต์  
มนต์มนต์มนต์มนต์มนต์

# When to go Clean vs Basic



© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.

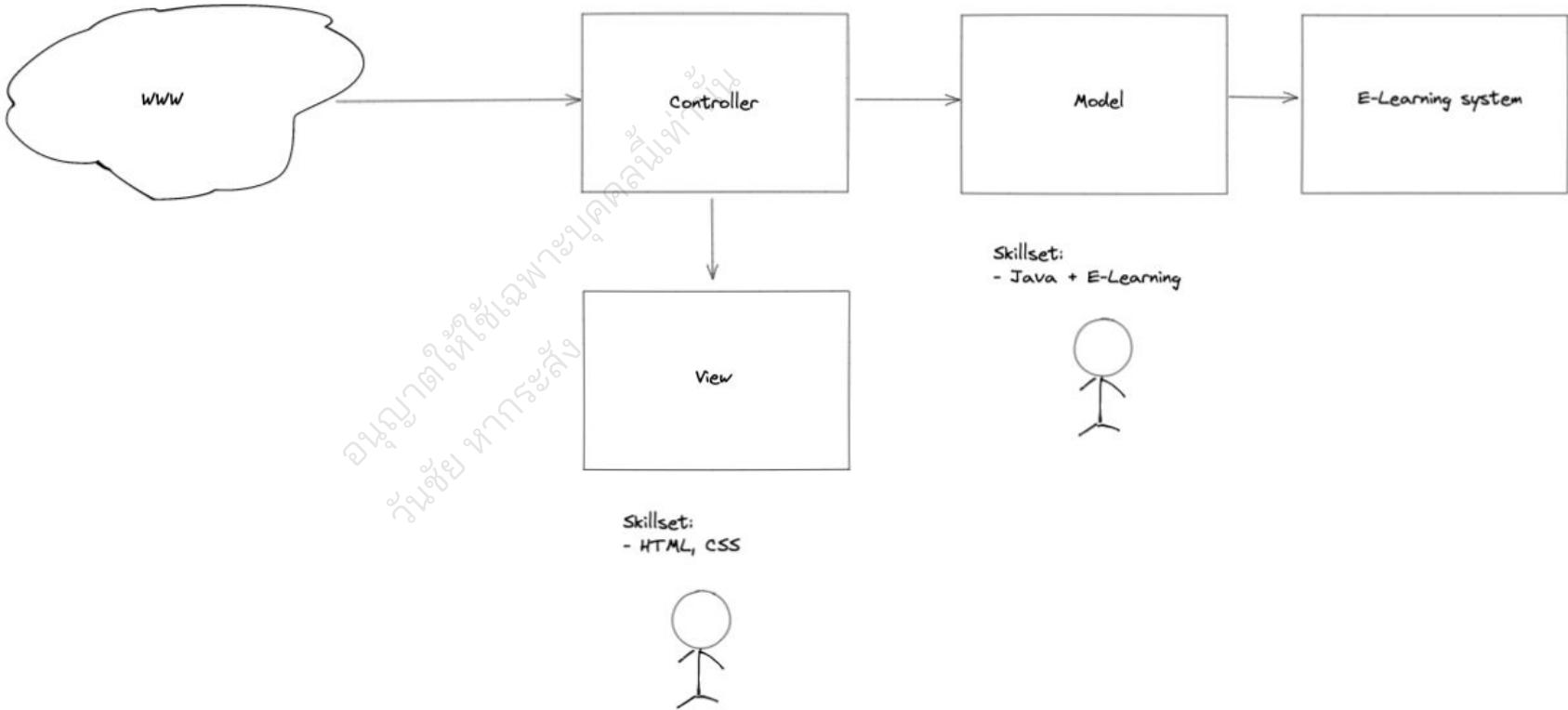
# Inspiration

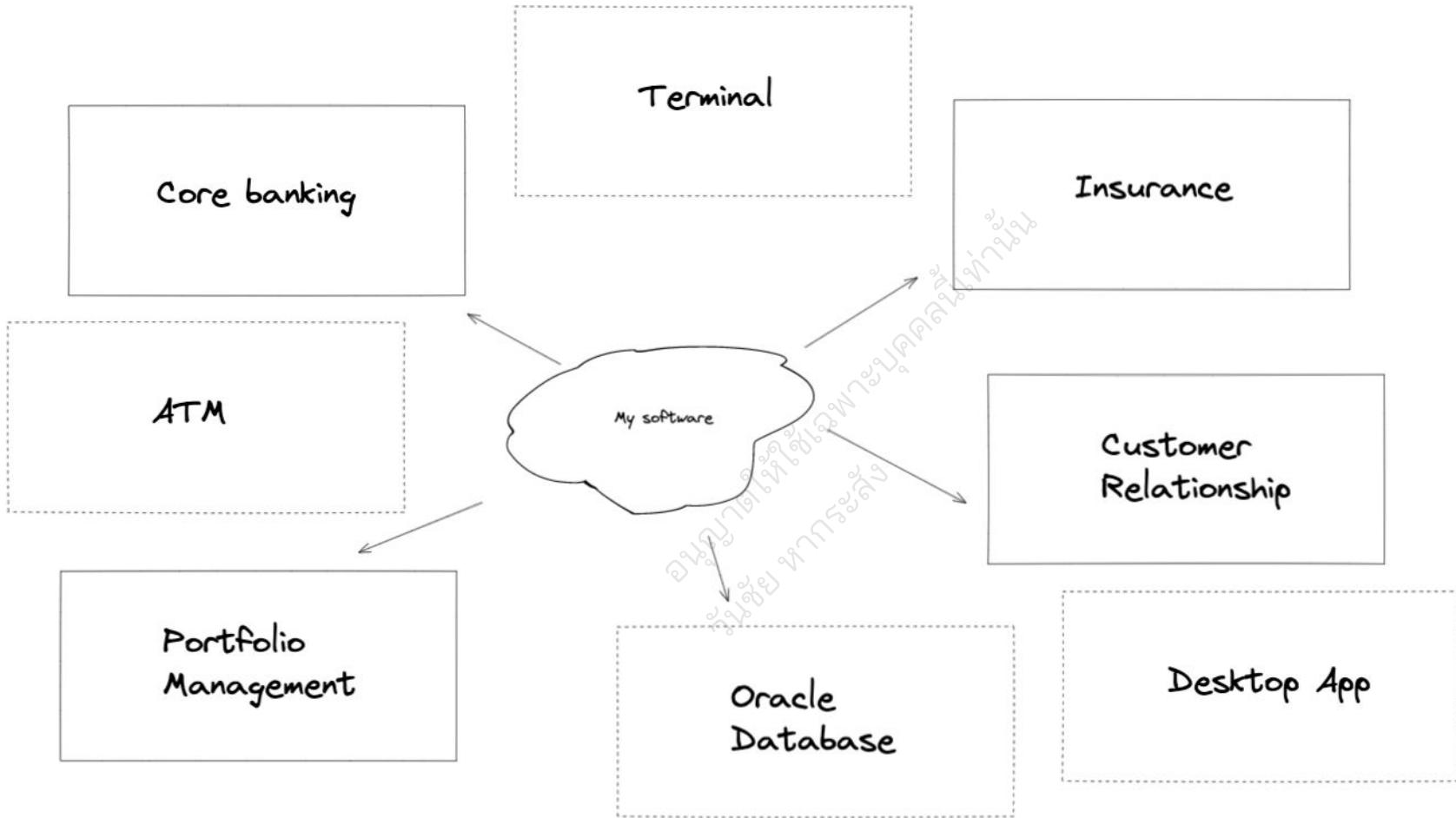
When people say “complex larger system need Clean Architecture”

What is the definition of “complex larger system”?

Is it line-of-code count?









## Upside & Downside

Clean architecture allow developer to contribute to the codebase by just knowing the basics of domain

- You don't need to know SQL
- You don't need to know integrations

This can be viewed as either good or bad.



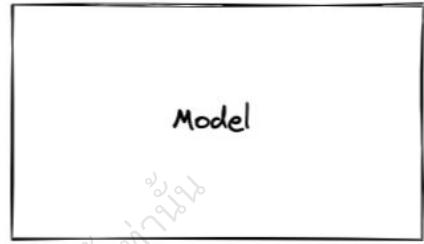
# Indicator: Skill Matrix

ຄອນຫຼາຍເຕີເກີ່ມຂົ້າແຂງພະບຸຄຄລ່ານໍາ  
ວິນສະຍໍ ພາກຮະລັບ

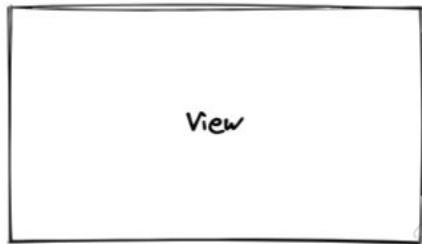
**Ask yourselves, how much do team member need to know?**



HTTP, Java, CSS

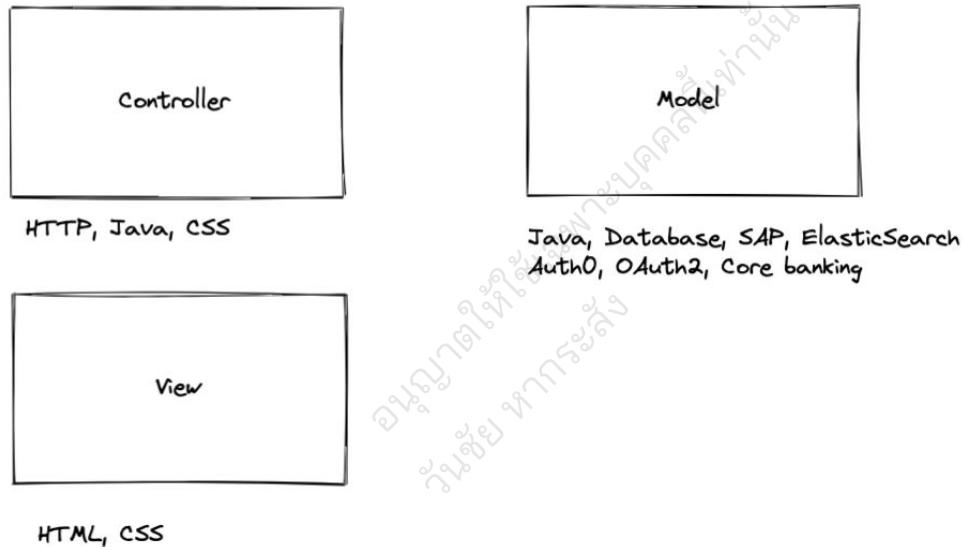


Java



HTML, CSS

This is ok



You might do something wrong here



Skill required: Web, Java



Skill required: Java



Tier3

Skill required: Java, SAP



Skill required: Java,  
Oracle



Skill required: Java,  
Core bank

การบริหารจัดการห้องแม่ข่ายและระบบสนับสนุนการดำเนินงาน

Architecture, hiring, team, organization  
structure.

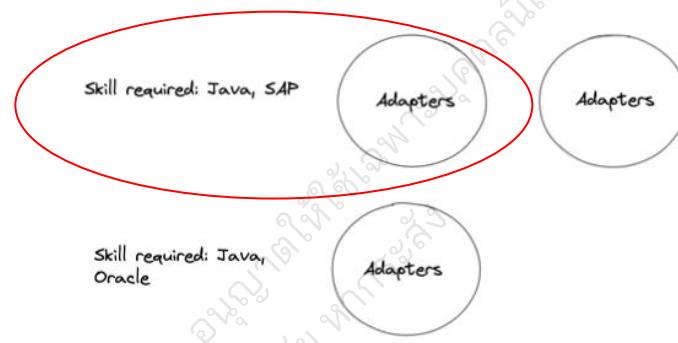
All must be aligned.



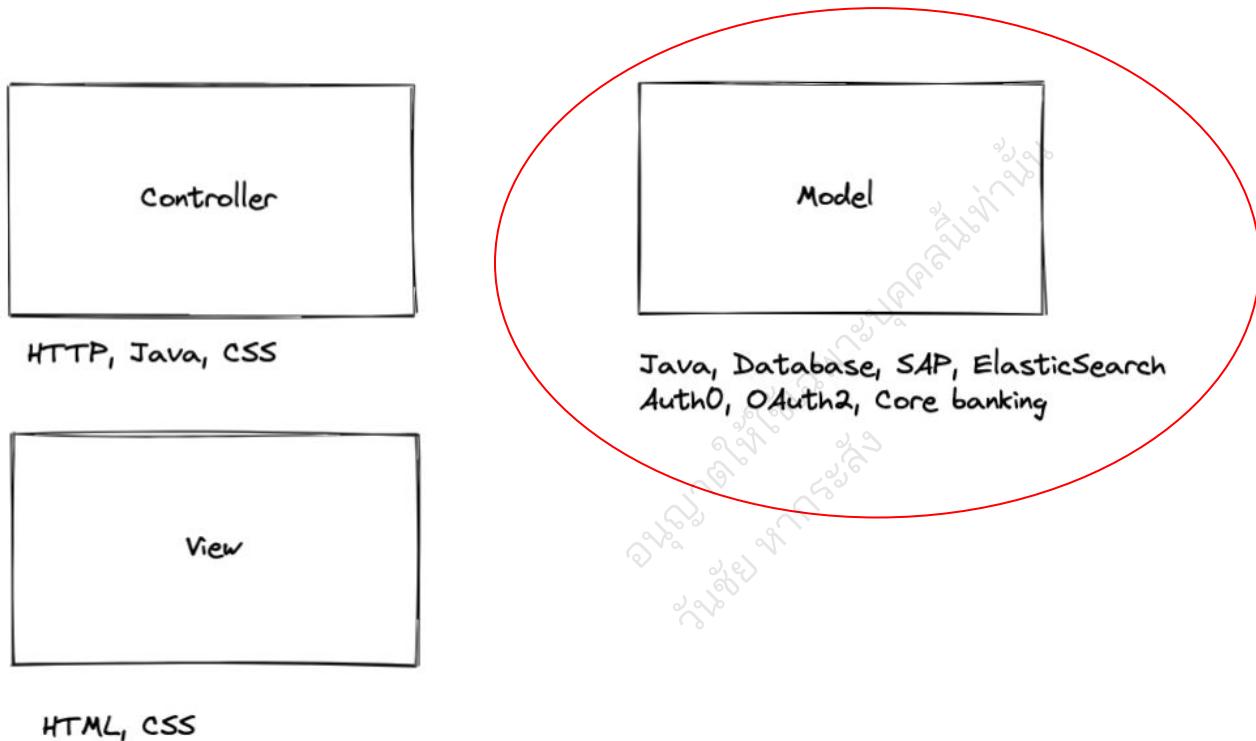
Skill required: Web, Java

Skill required: Java

Tier3



Maybe for the next candidate, we need SAP developer?



Maybe for the next candidate, we need a superman (I guess)?

# Select your tiers

## Good tiering

- helps you organize team
- helps you recruiting
- helps you create a training plan

## Bad tiering symptom

- “I need a good software engineer with a lot of knowledge”



## Complex-large system:

A system which require understanding  
of many domains



# **Sum up: Specialist Collaboration**

ร่องรอยการคุยที่สำคัญที่สุด

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.



---

# **Problem statement#1**

- Current state: We cannot navigate a code with a lot of classes
- Desired state: We can navigate it

Solution: Group them up!

- But a group without rules or heuristic does not help
- We need heuristic!



---

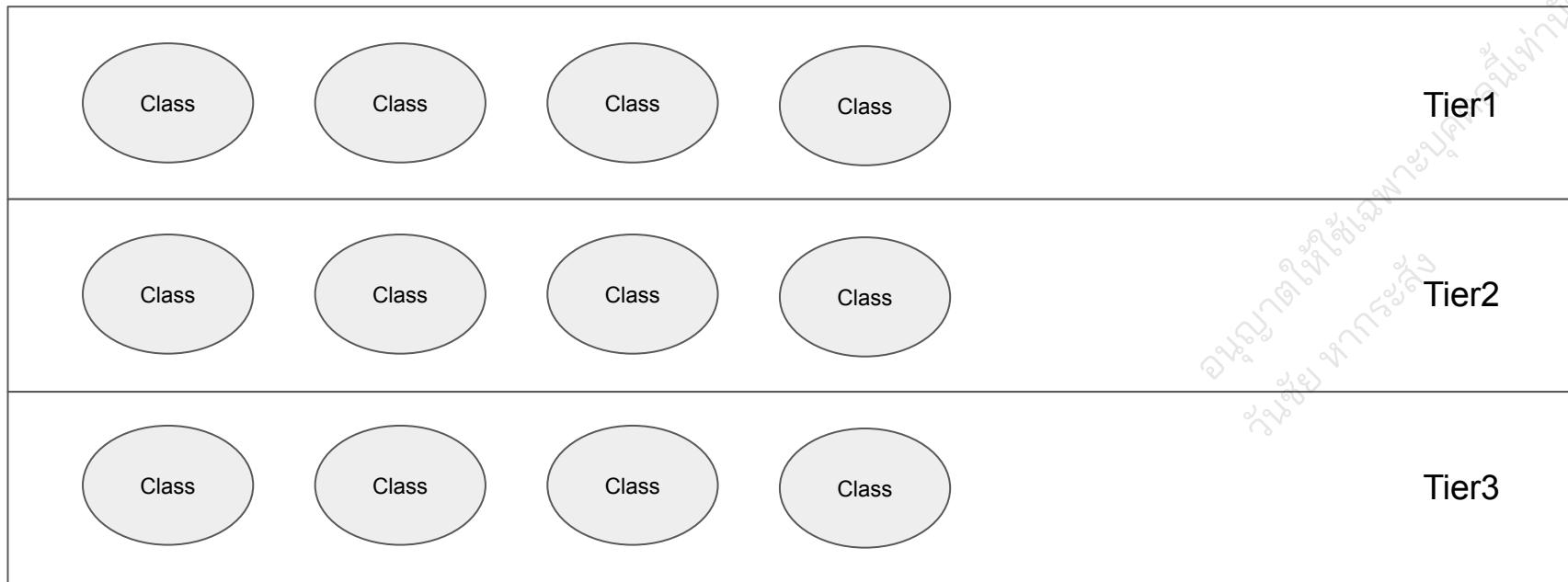
# N-Tier

A way to group classes

នៃការរចនា  
គម្រោង និង ផលិតផល  
ក្នុងបច្ចេកទេស



# N-Tier architecture



## N-Tier architecture

- N-Tier rules: Internal layer does not depends on external layer
- So we can reason about effect of change

ມັນຫຍຸງ  
ວ່ານີ້ສະໜອງ  
ອຳນວຍຕູກ  
ທີ່ຂໍ້ວິພາບ  
ຄວາມສໍາເລັດ



---

## Problem statement#2

- Current state: Every part of code require many do
  - Web server
  - Security
  - Legacy system
  - Frontend
- Expecting every developer to be an expert of every domain is hard
- Even if that is possible, it is far easier to know that these classes relate to this topic



---

## **Problem statement#2**

Desired state: A programmer who know only some technical domain still can contribute to the code



# MVC

---

## Model-View-Controller

- Expand input-output of the system
- Controller: Input-Output handling
- View: Layering
- Model: Everything else (3rd-party, Core)

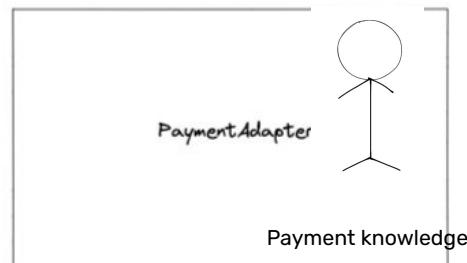
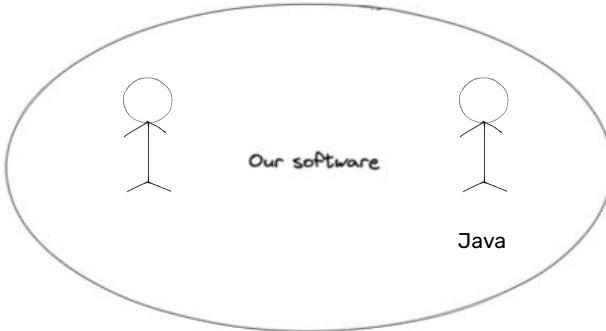
In modern web (Spring, Django, Rails): Model is a layer which connect to database (and that can be considered core for CRUD app)



# Clean architecture

- Invented from people who work on Enterprise solution
- Many external system and dependency
- So they find a way to organize and draw the line between “this is 3rd-party” and “this is us”
- You separated all 3rd-party out using dependency rules
- You have a core codebase that is independent of any third-party
- It's all about dependency rules





Everyone need to know domain knowledge

**Our specialists can focus in their area  
with our architecture**

Last note: Beware of knowledge leak

รัฐวิสาหกิจที่ให้เช่าความรู้  
กับนักศึกษา

```
● ● ●
```

```
@GetMapping("/")
@ResponseStatus(org.springframework.http.HttpStatus.FORBIDDEN)
public String index(@RequestParam String username) throws AccessDeniedException {
    if (username == "Madonna") {
        throw new AccessDeniedException("No access");
    }
    return "";
}

// =====

@GetMapping("/")
@ResponseStatus(org.springframework.http.HttpStatus.FORBIDDEN)
public String index(@RequestParam String username) throws AccessDeniedException {
    User user = new User(username);
    if (!user.CanAccess()) {
        throw new AccessDeniedException("No access");
    }
    return "";
}
```

**Our specialists can focus in their area  
with our architecture**



# Introduction to Business collaboration

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.



```
class UserSelectedCourses {  
    public void Pay () {  
        // จ่ายเงิน  
    }  
}
```

# Scenario

อนุญาตให้ใช้เพื่อพำนุบคลิปเท่านั้น  
วันชัย หการส์

BA: เวลาลูกค้าเอาซื้อคอร์สพากนี้ เราจะมีโปรโมชั่นให้  
นะครับ ยังไงตอนเอารเข้าตະกร้าลงเซ็คด้วยนะครับ

PG: ตະกร้าคืออะไรนะครับ

BA: อันนี้ใน เวลาที่ลูกค้าเลือกคอร์สอ่า เขายจะหยิบลง  
ตະกร้า

PG: (อ้อ คอร์สที่ลูกค้าเลือก)

PG: แล้วโปรโมชั่นแปลว่าอะไรนะครับ

BA: ส่วนลดได้ ส่วนลด



```
class UserSelectedCourses {  
    public void Pay () {  
        // จ่ายเงิน  
    }  
  
    public Price Discount () {  
        // ทำการจัดการส่วนลด  
    }  
}
```

# Scenario

อนุญาตให้ใช้สิ่งพะบุคคลนี้ทำหน้าที่  
ร่วมช่วย ทางระบบ

BA: อ่านน้องพึงเข้ามาใหม่เหรอ

New Junior PG : ใช้ครับ พึงเข้ามาได้ 1 เดือนครับ

BA : อ่ออ้อ แล้วพี่คริสไปไหนอ่ะ

New Junior PG : อ่อ พี่คริสติดประชุมครับ มีอะไรฝาก  
ผมไว้ได้ครับ

BA: ลูกค้าเข้าบอกว่าเขามีปัญหาในตั้งกร้า เวลา  
เช็คเอาท์ โปรโมชั่นมันไม่ทำงาน ช่วยดูให้หน่อยนะครับ

BA: แล้วก็ฝากช่วยดูเพิ่มให้นิดนึงว่าเวลาลูกค้าใช้โปรโม  
ชั่นพวกลนี้ช่วยส่งข้อความไปบอกเจ้าของคอร์สด้วยนะ

New Junior PG: ...





```
class UserSelectedCourses {  
    public void Pay () {  
        // จ่ายเงิน  
    }  
  
    public Price Discount () {  
        // ทำการจัดการส่วนลด  
    }  
}
```



โปรแกรมเมอร์พูดจาไม่รู้เรื่องเลย

อนุญาตให้เผยแพร่ตามบุคคลที่ท่าน  
วันชัย ห้ารัลส์



BA พูดจาไม่รู้เรื่องเลย



**Still, that not works well. Until...**

อนุญาตให้ใช้สิ่งที่บุคคลน้ำหนึ่ง  
วันซึ่ง ทำการลับ

**Eric Evans: Let's build a system  
according to the domain**

Copyright © 2004

# Domain-Driven DESIGN

Tackling Complexity in the Heart of Software



Eric Evans

Foreword by Martin Fowler

Copyright © 2004



# What is Domain Driven Design?

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.



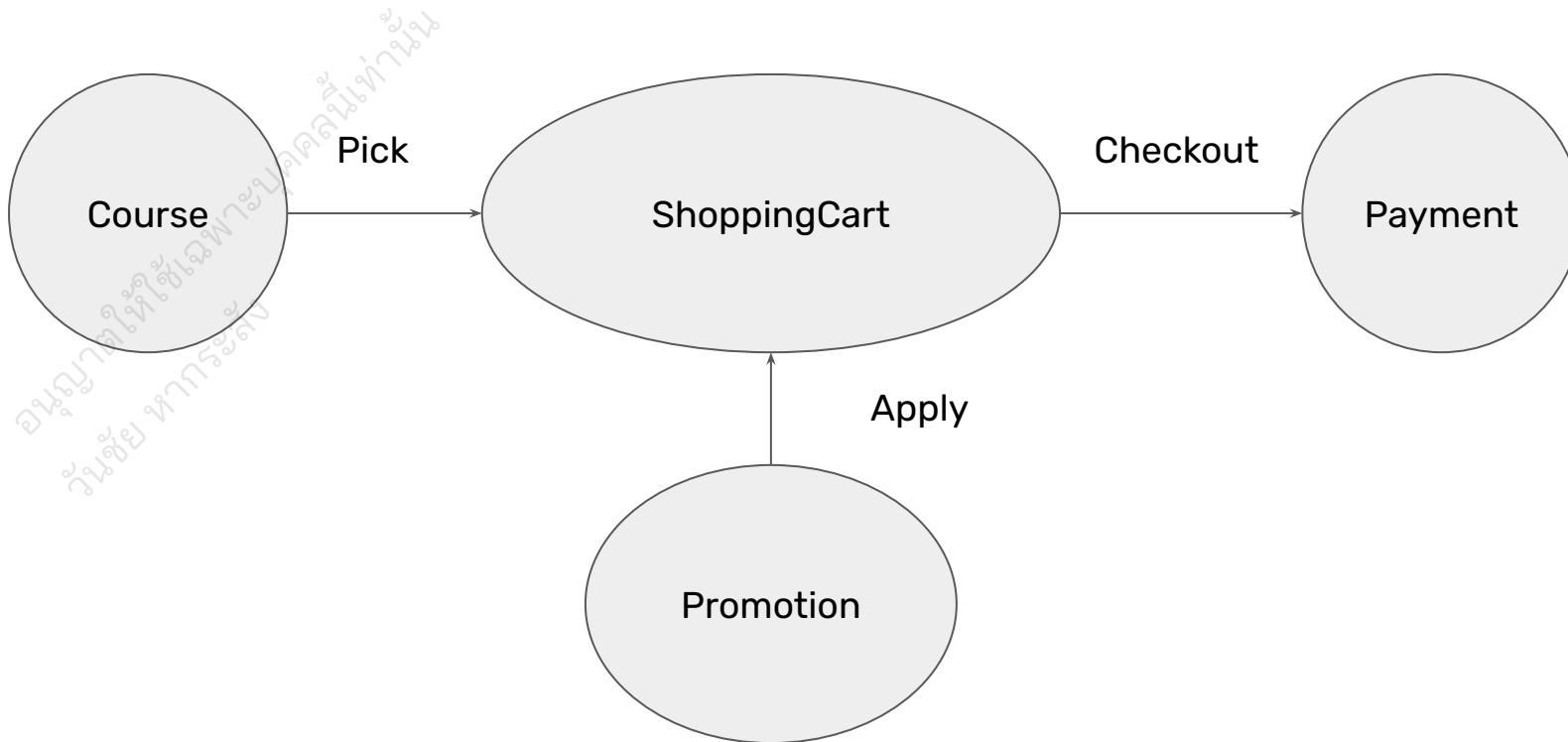
# What is Domain Driven Design?

Essentially, design a software according to the model.

This should allow developer and business domain expert to have a same perspective of how software works.

Now requirement change should be easier to be implemented





Business expert mental model



# Codebase



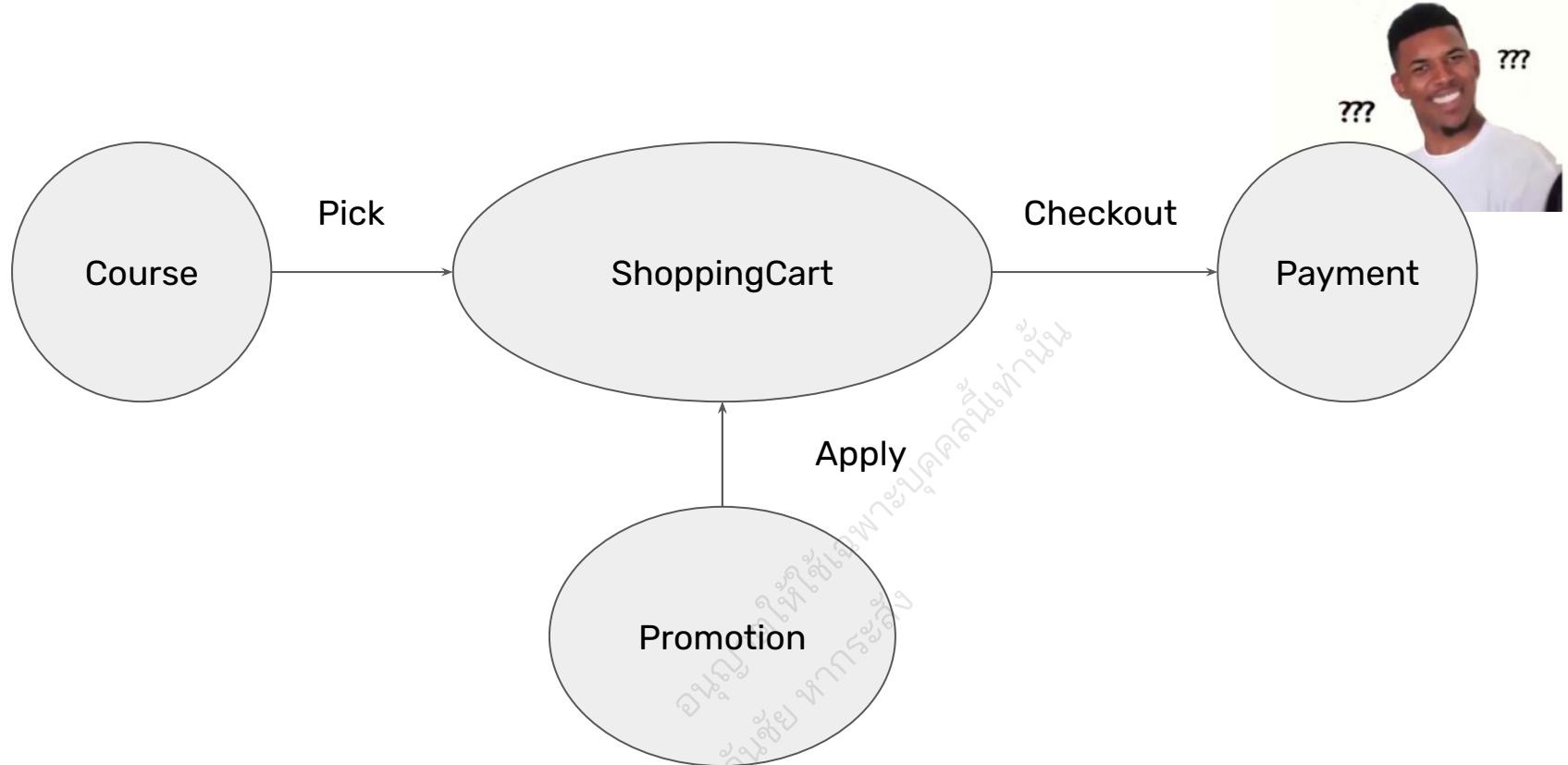
Can you validate checkout process?



Next quarter: We want to have a special promotion



Can I get a payment info?



How do we handle discount?



# What is Domain Driven Design?

Let's model according to business expert





```
class ShoppingCart {  
    public Payment Checkout() {  
        // Pay logic here  
        Promotion promotion = promtionRepo.findApplicable(this.courses)  
        if (promotion != null) {  
            promotion.Apply(this.price);  
        }  
        // Continue  
    }  
}  
  
class Promotion {  
}
```

```
class ShoppingCart {  
    public Payment Checkout() {  
        // Pay logic here  
        Promotion promotion = promtionRepo.findApplicable(this.courses)  
        if (promotion != null) {  
            promotion.Apply(this.price);  
        }  
        // Continue  
    }  
}  
  
class Promotion {  
}
```



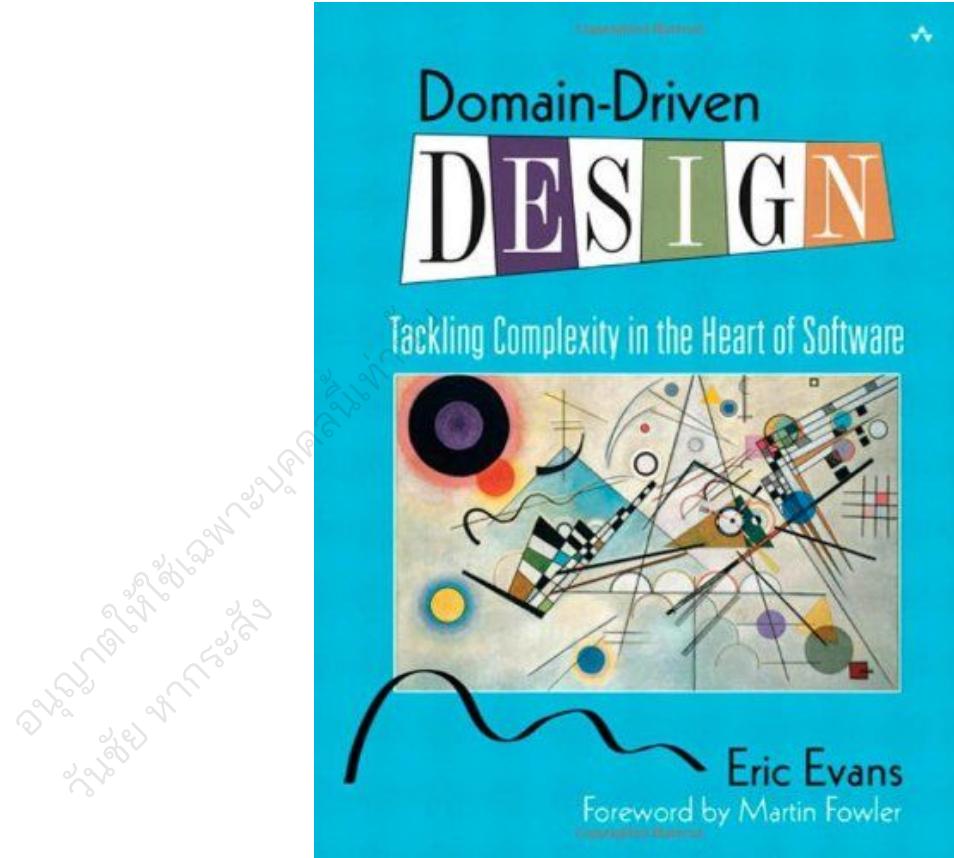
Can you validate checkout process?



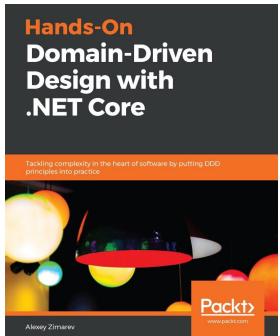
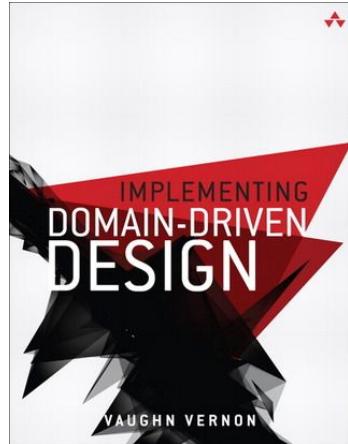
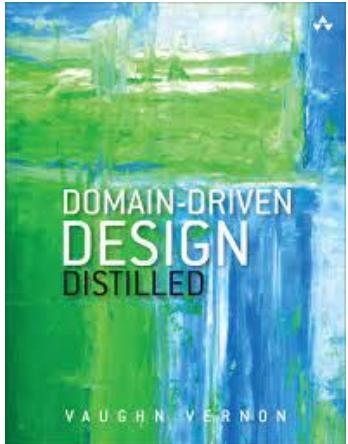
อนุญาตให้ใช้สิ่งพิมพ์ใน  
ชั้นเรียน ห้องเรียน

**This is not an easy tasks**

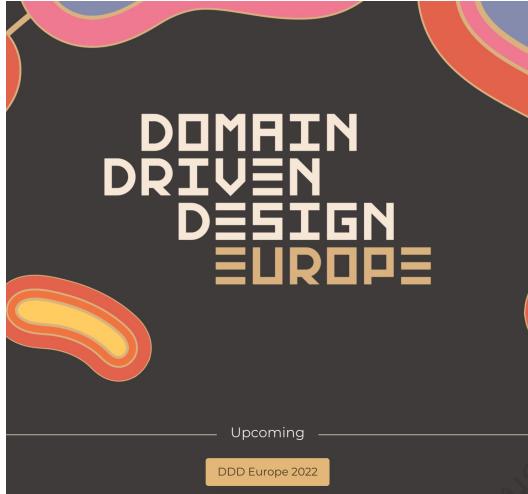
รุ่นที่ ๑  
ภาระสังคมไทยในปัจจุบัน



Original Book by  
Eric Evans: 2003



More books are  
still in published



Many events are  
still ongoing

# **DDD is still evolving**

มนต์เสน่ห์แห่งการเขียนโค้ด  
อนุญาตให้ใช้เฉพาะบุคคล

# Domain Driven Design

Domain driven design is a technique that allow us to model the system according to the domain expert

Both terminology and relationship must be matched

So we can collaborate better

---

Ideally, you can even fix the code in the meeting room



We will take you to a crash course of  
domain driven design



# Three level of Domain Driven Design

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.





# **Many meaning of Domain Driven Design**

There are many aspects to Domain Driven Design

And after you learn from a course or a book, you might find out that another book talk about totally different idea

Let's look at the overall landscape

อนุญาตให้เขียน  
วันชัย ห้ามระลัง



# Domain-Driven Design On A Page

DDD is a philosophy for developing software systems that encourages Domain Thinking at each step of the Software Development Lifecycle

## Domain Discovery

Exploratory DDD

- Model as-is, to-be, and could-be states of the domain
- Model collaboratively and visually so that the whole team\* learns the domain and contributes to solution ideas
- Example: Big Picture EventStorming

## Software Architecture

Strategic DDD

- Bounded Contexts: Split a large software system into specialised models aligned to areas of the domain
- Integrate bounded contexts using domain events\*\*
- Identify strategically-significant core domains

## Software Design

Tactical DDD

- Create models in code which align to the team's shared understanding of the domain
- Use appropriate patterns in each context: entities, aggregates, event sourcing etc.

\* whole team: all roles involved in product development including business experts

\*\* domain events: business-relevant happenings communicated via (technical) events or commands

# DDD Doctrine

Principles and practices that are almost universally applicable in DDD

## **Make The Implicit Explicit**

Avoid ambiguity and improve communication by making all relevant details visible.

## **Explore Multiple Models**

When the benefits of a better model are significant, don't stop at the first idea

## **Cultivate a Shared Language**

Create a common language within each bounded context to foster improved collaboration.

## **Focus on Concrete Scenarios**

Avoid creating beautiful designs that fail to solve the problem by challenging designs with concrete scenarios.

## **Learning Never Stops**

There is always more to learn about the domain so a learning mindset is essential

## **Design is Evolutionary**

Because learning never stops there is a constant stream of feedback which can be used to improve the design.

# What we will cover?

Domain Driven Design itself is a very big area

- Basic of strategic domain driven design
- Some part of Tactical domain driven design



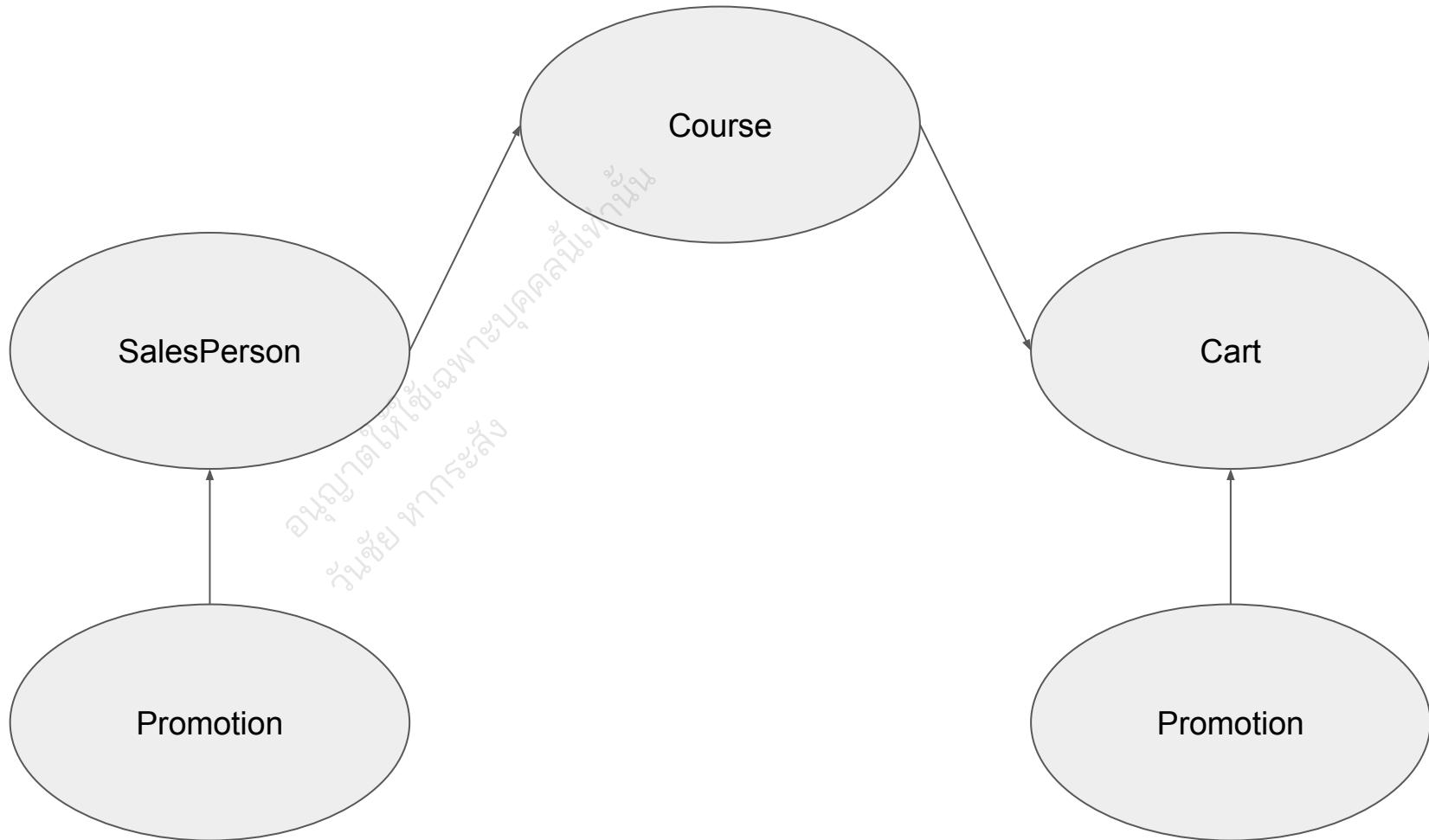


# Bounded Context & Ubiquitous Language

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.





# How to solve this?

อนุญาติให้ใช้สิ่งที่จำเป็น  
วันนี้ชัย ภาระสำคัญ

Human Resource

Course

E-Commerce

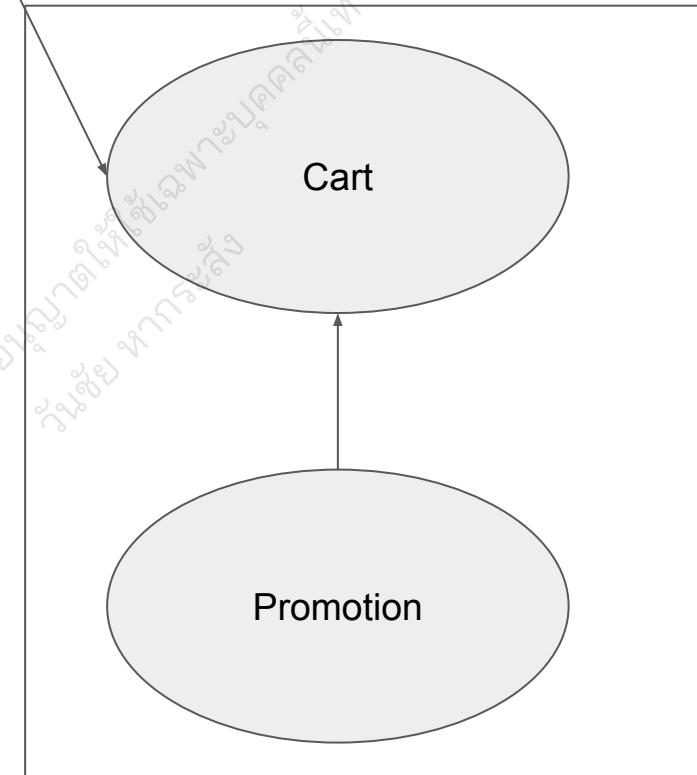
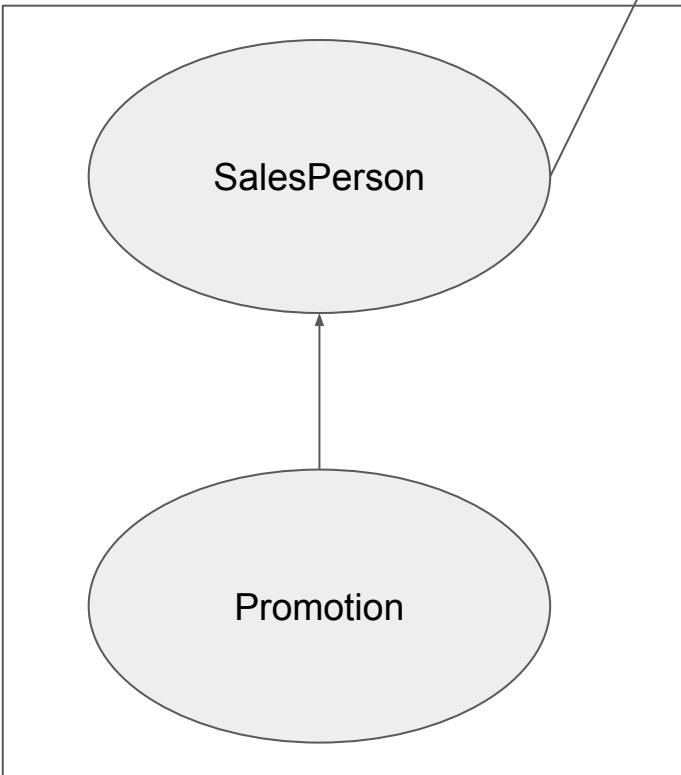
SalesPerson

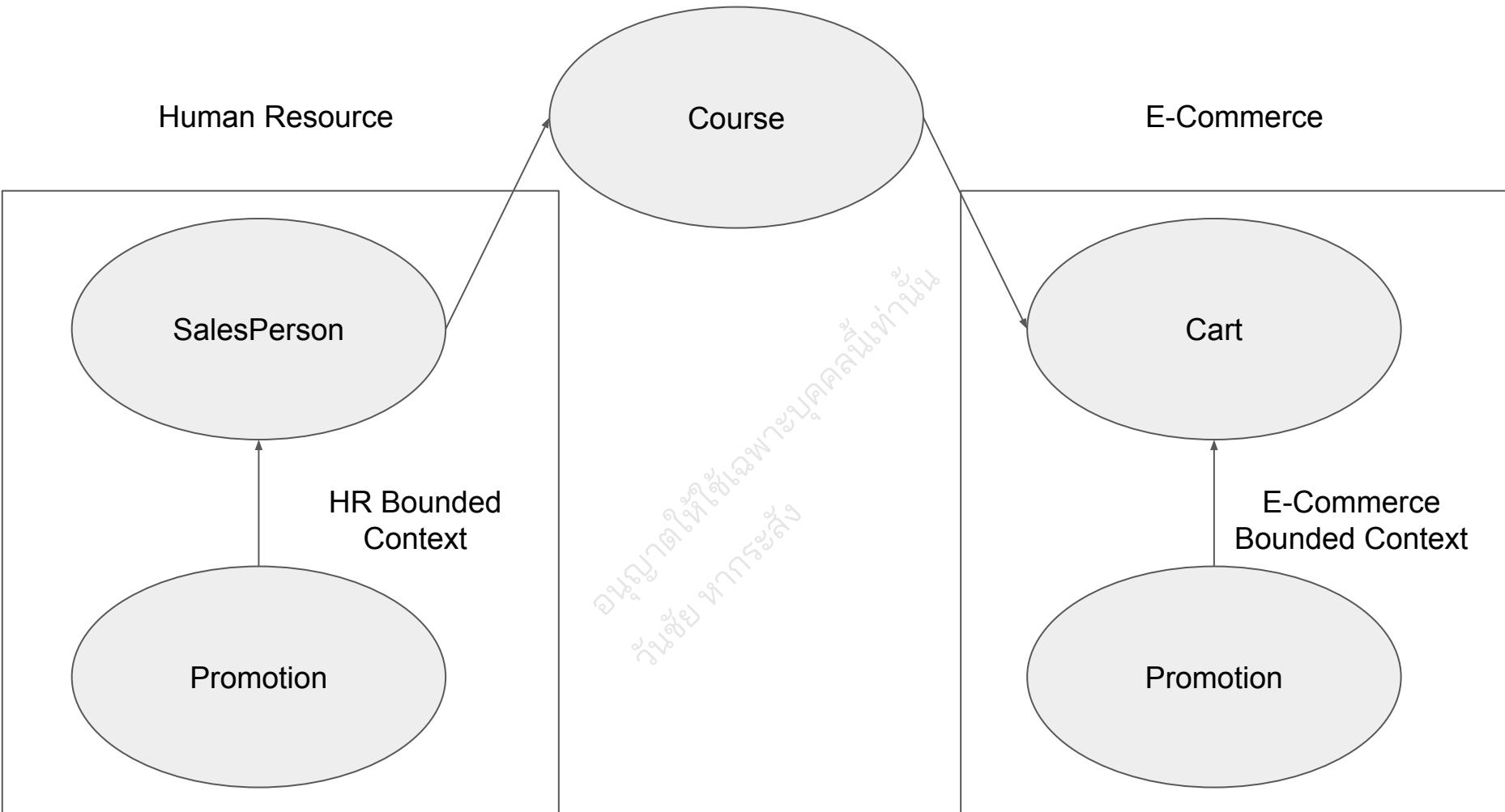
Cart

Promotion

Promotion

องค์กรตัวอย่าง  
รุ่นแรก ห้ารุ่นสุดท้าย





# Bounded Context

We draw conceptual boundary called Bounded Context.

In a bounded context, we agreed to use an ubiquitous language, which mean a single language.



# **Ubiquitous Language**

อนุญาตให้ใช้เฉพาะบุคคลเท่านั้น  
วันนี้จะยังคงเป็นอย่างเดิม

Ubiquitous Language is the term Eric Evans uses in Domain Driven Design for the practice of building up a common, rigorous language between developers and users.



# Bounded Context

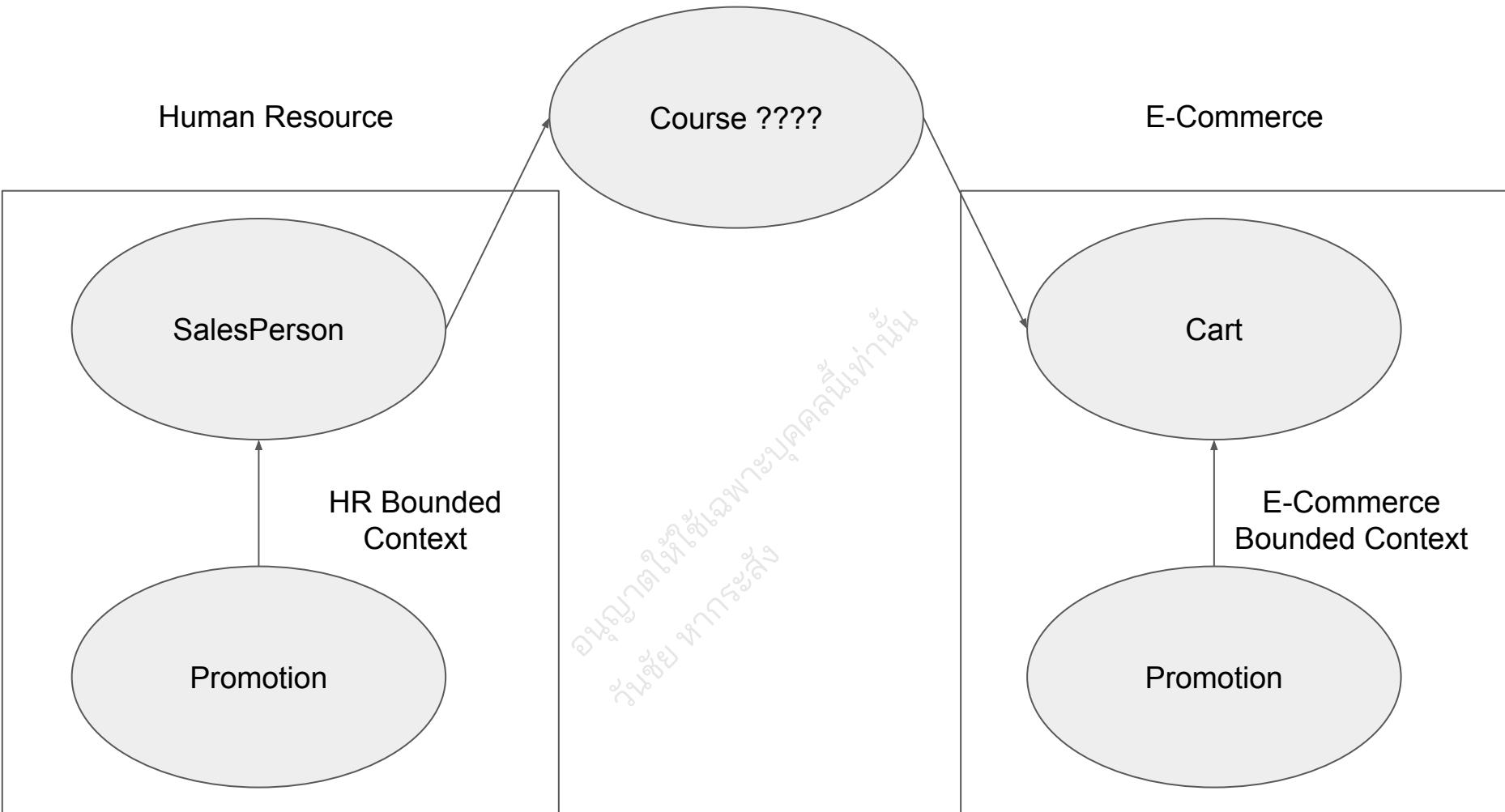
We can implemented bounded context using

- Namespace
- Package
- Library
- Multi-service

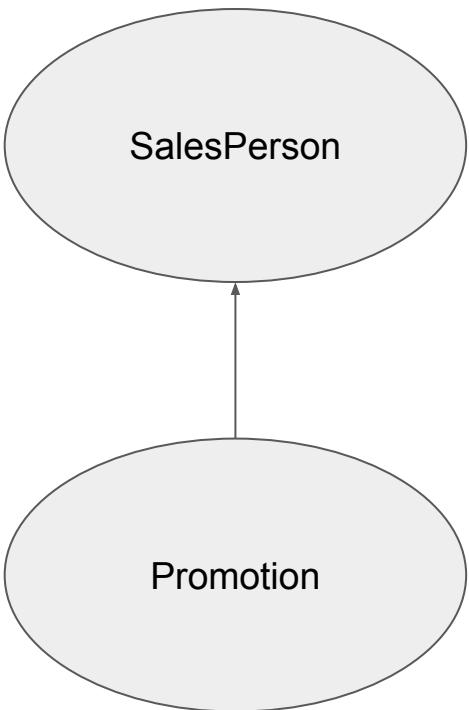
อนุญาตให้ใช้เฉพาะบุคคลนี้เท่านั้น  
วันนี้จะมีการร้องขอ



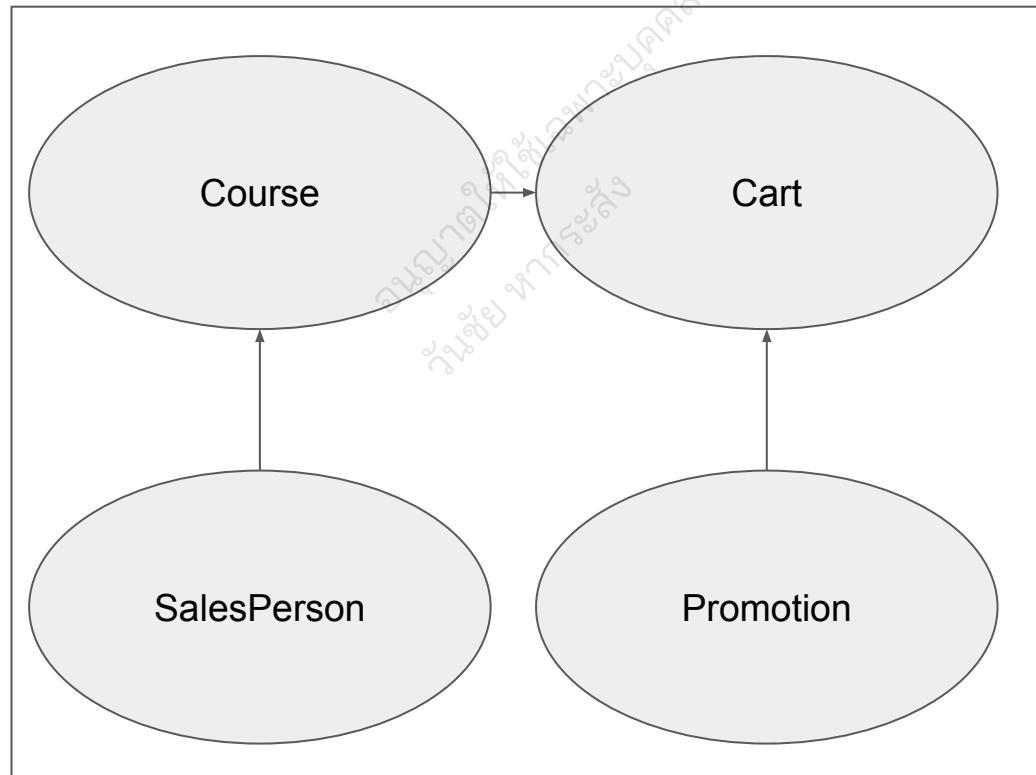
**Actually, we drawn the model wrong**



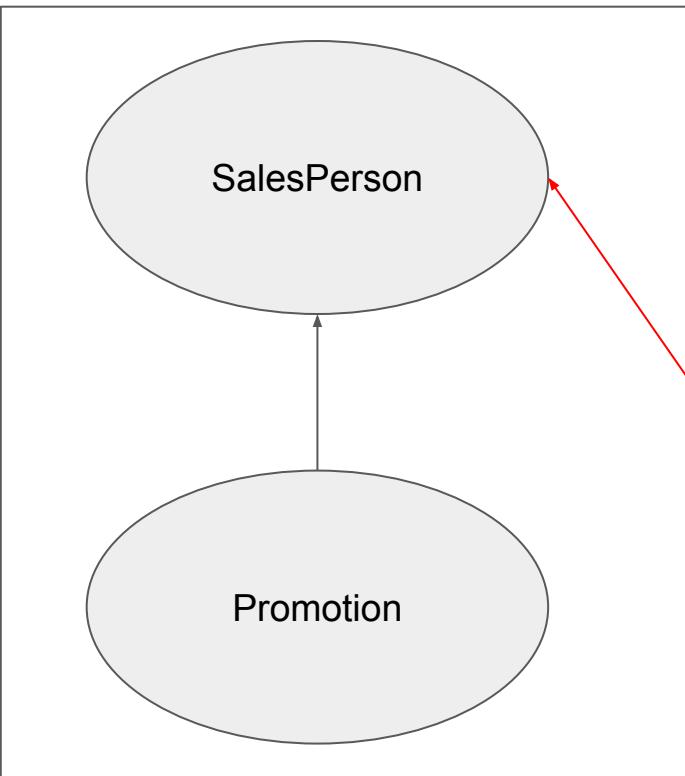
Human Resource



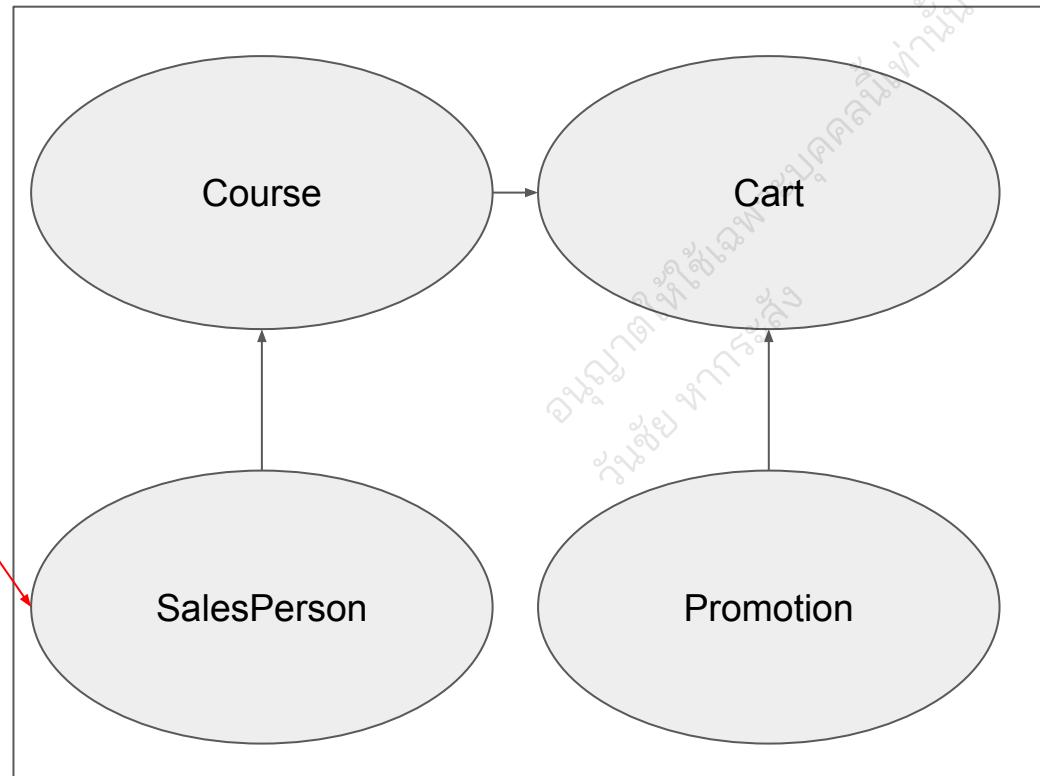
E-Commerce



Human Resource



E-Commerce



# **Context Mapping**

- A mapping between bounded context
- Communication and Integration Pattern
- Use same ID?
- Use Event?
- Use REST API Call?

อนุญาตให้ใช้เฉพาะบุคคล  
ในสิ่งที่ทรงสร้าง



**For today, let be mindful about this**

อนุญาตให้เข้าเฉพาะผู้มีอำนาจ  
วันชัยพากะสัง

# Sum up

**Bounded context:** Boundary of Ubiquitous language

**Ubiquitous language:** Consistent single language shared between domain expert and developer

**Context mapping:** Integration and Communication between context





# Drawing Bounded Context

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.

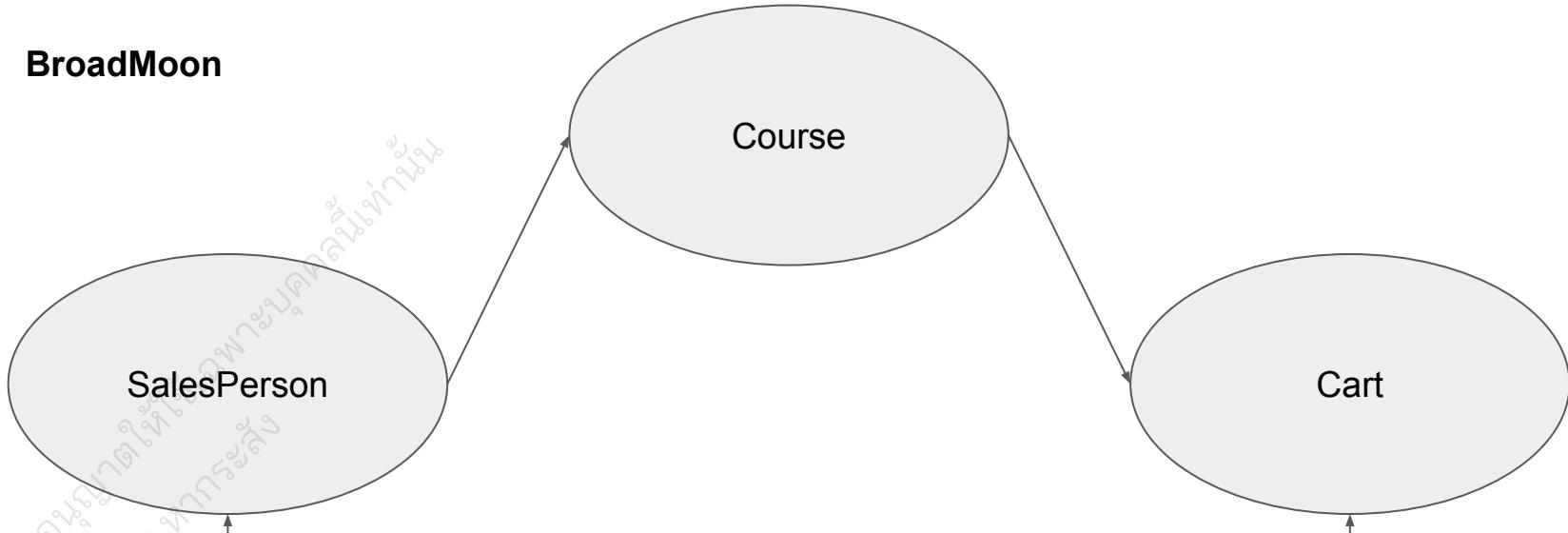


# Rules#1

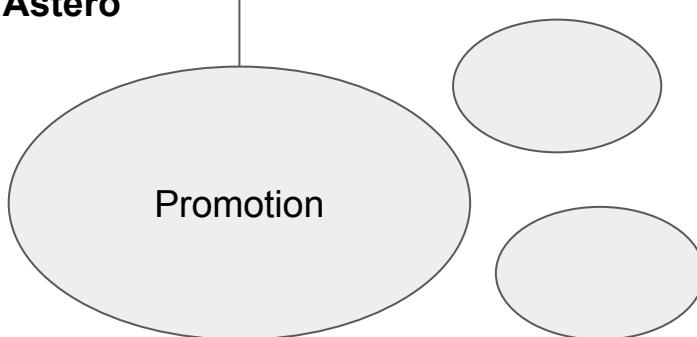
Bounded context must have a valid name



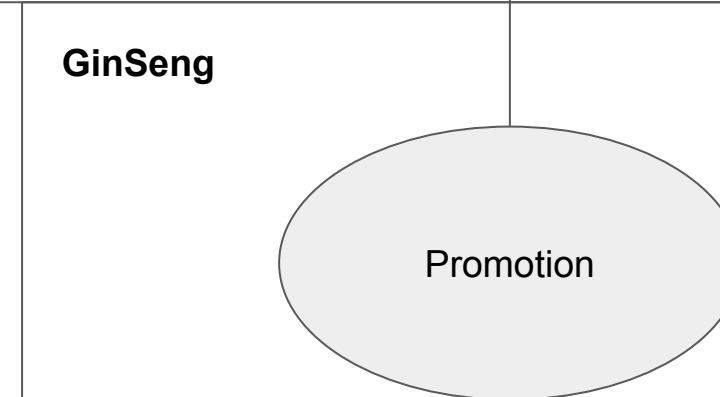
**BroadMoon**



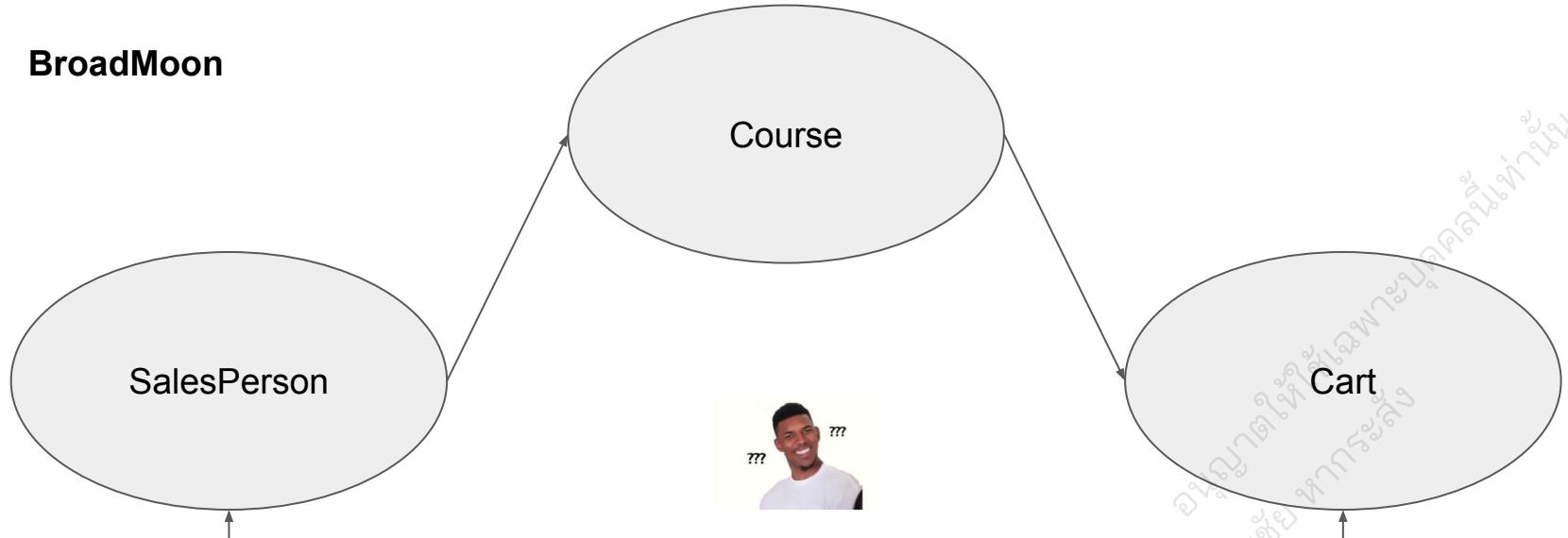
**Astero**



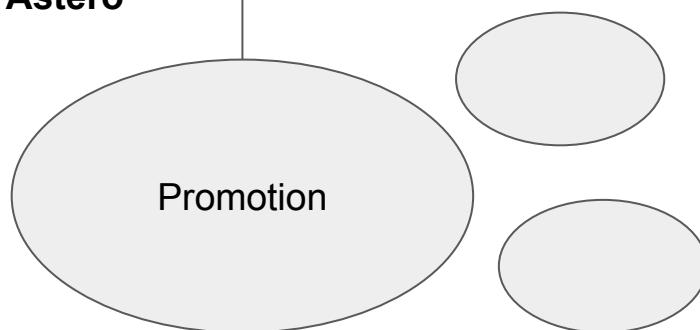
**GinSeng**



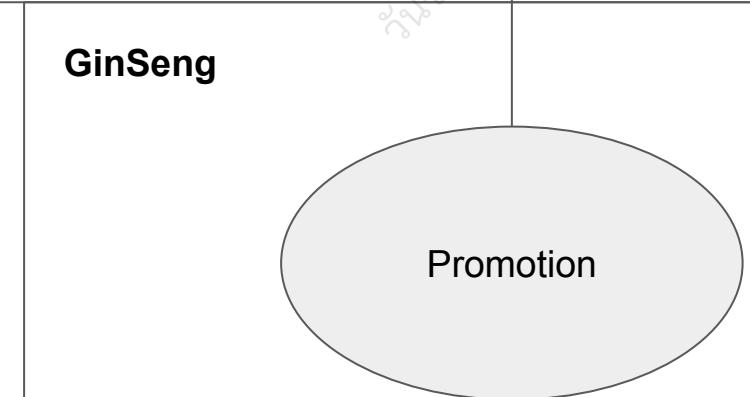
## BroadMoon



## Astero



## GinSeng



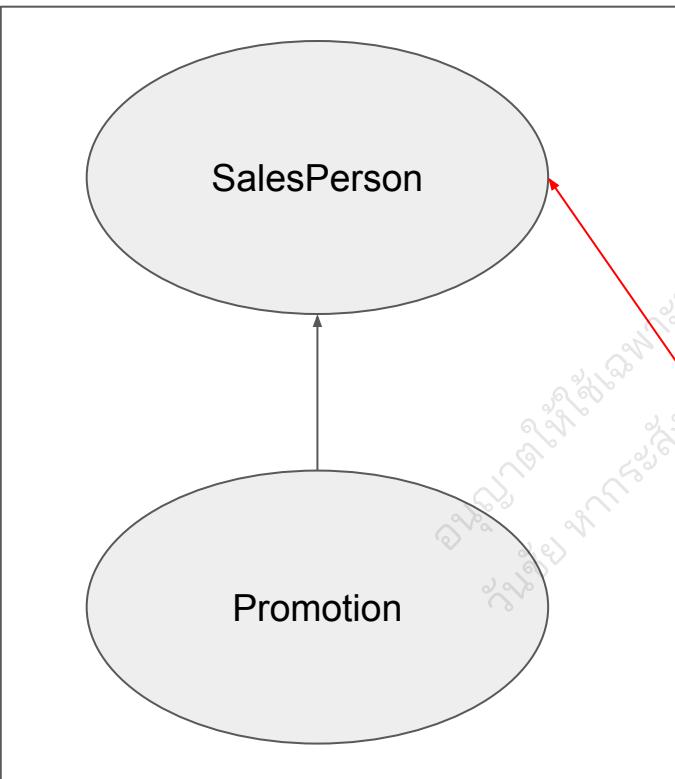
## Rules#2

Bounded context required Ubiquitous language

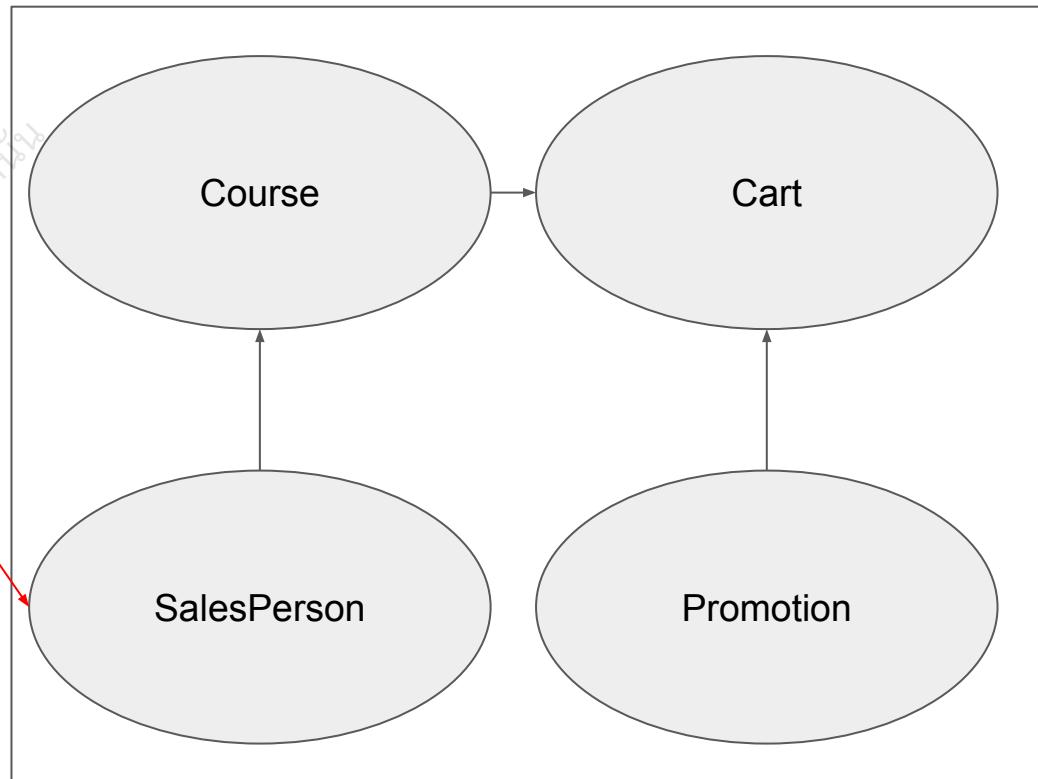
Ubiquitous language required agreement from  
Domain Expert and Programmer



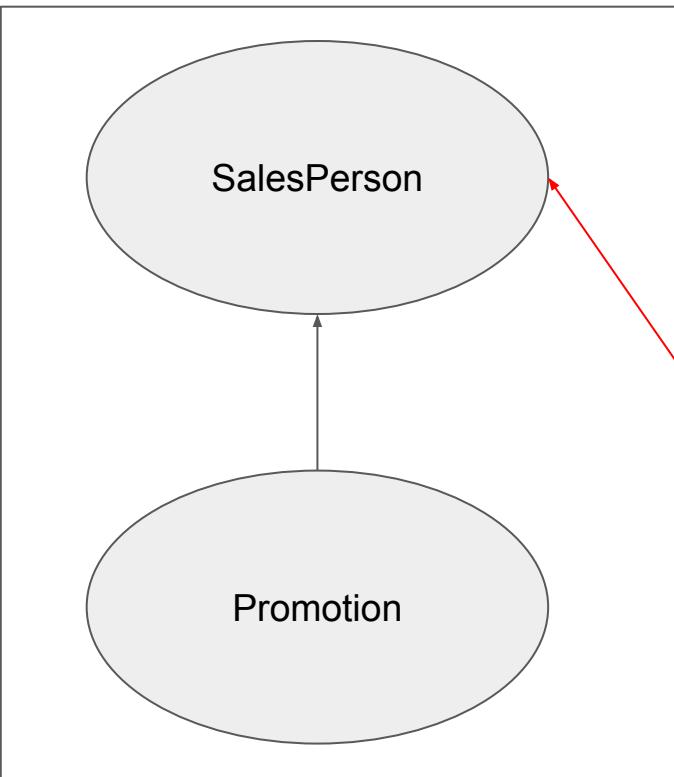
Human Resource



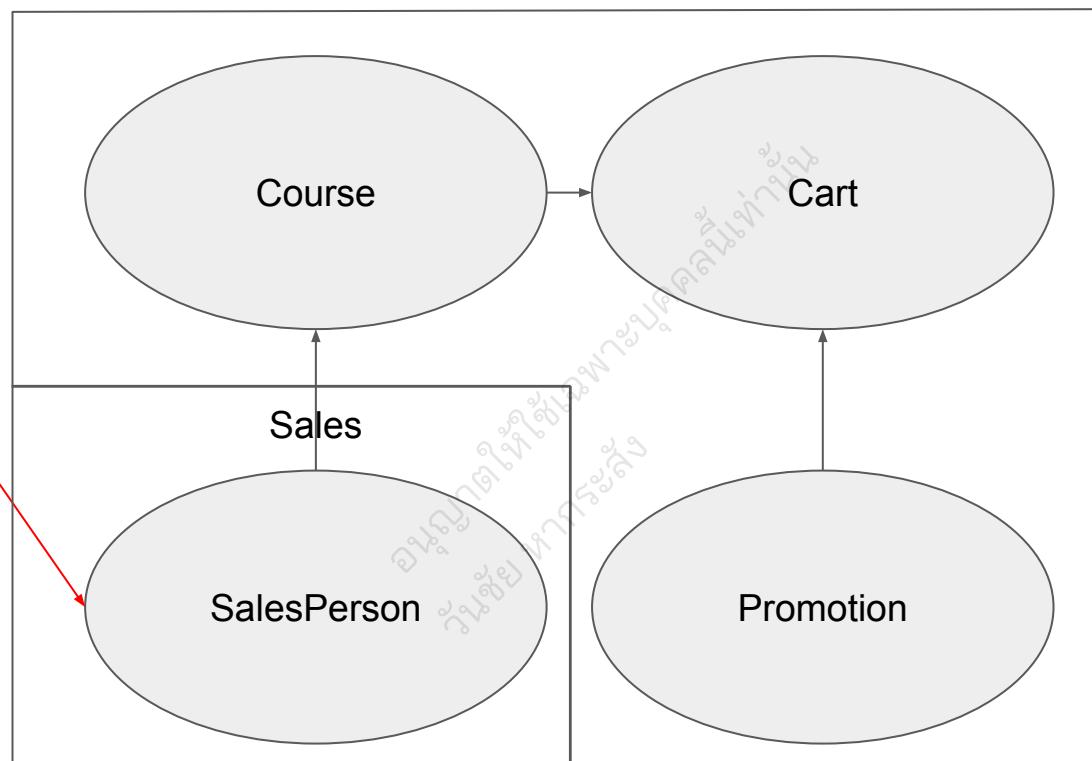
E-Commerce



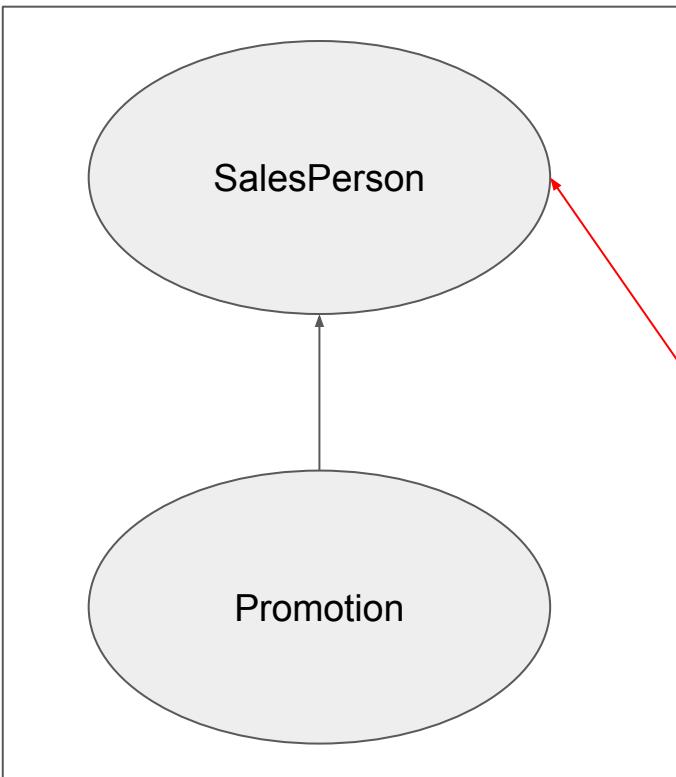
Human Resource



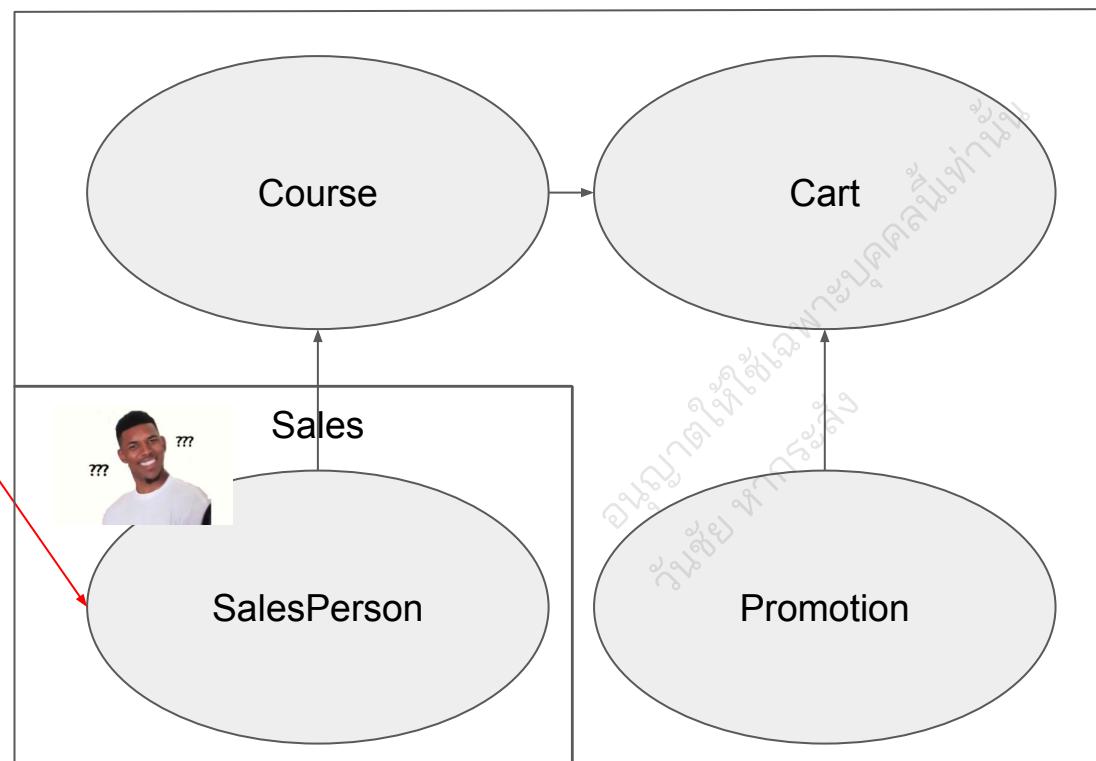
E-Commerce



Human Resource



E-Commerce



## Rules#2

Bounded context is not about technical limitation  
It's about communication with business expert



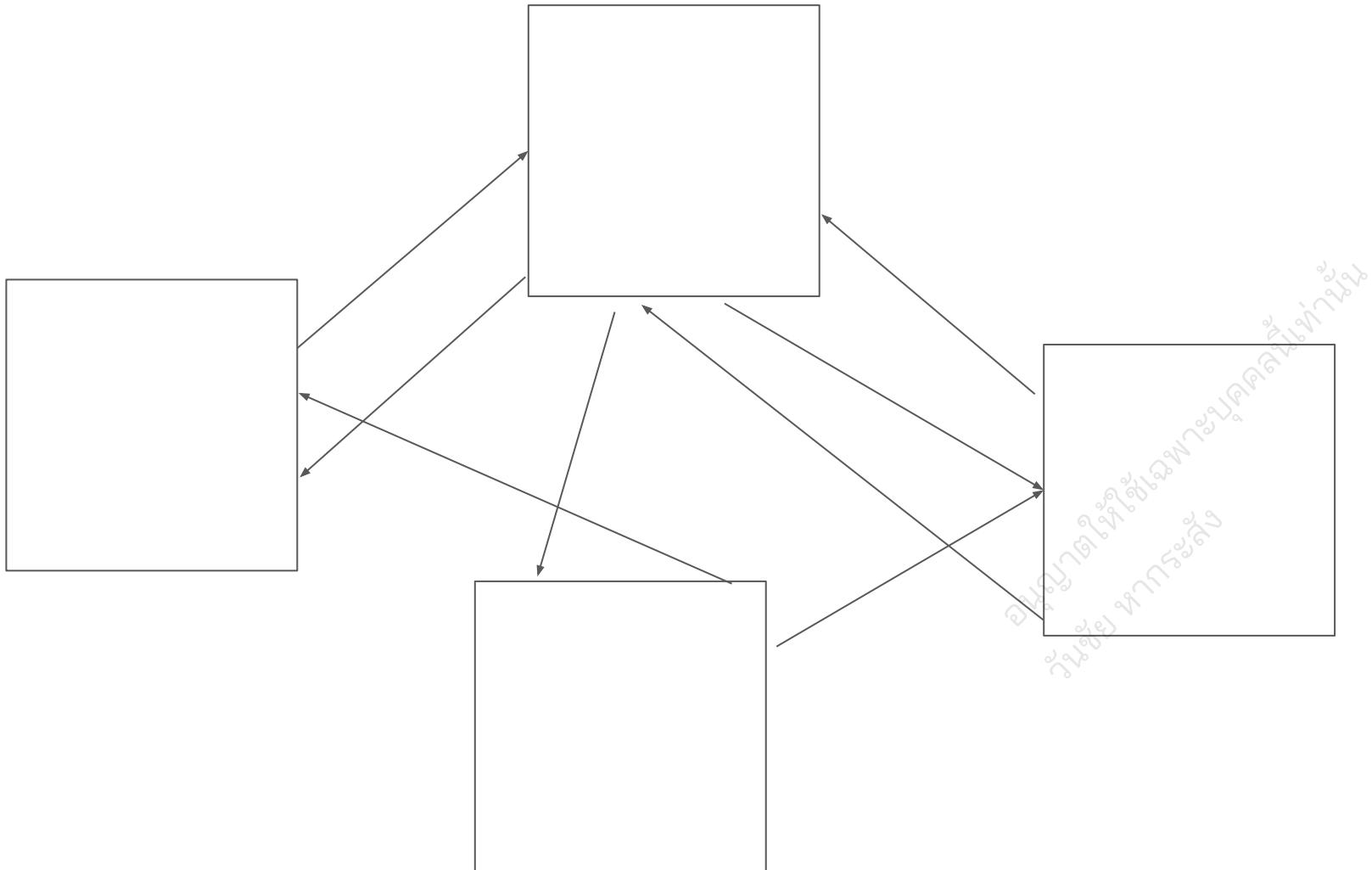
Draw bounded context according to  
business expert and organization chart

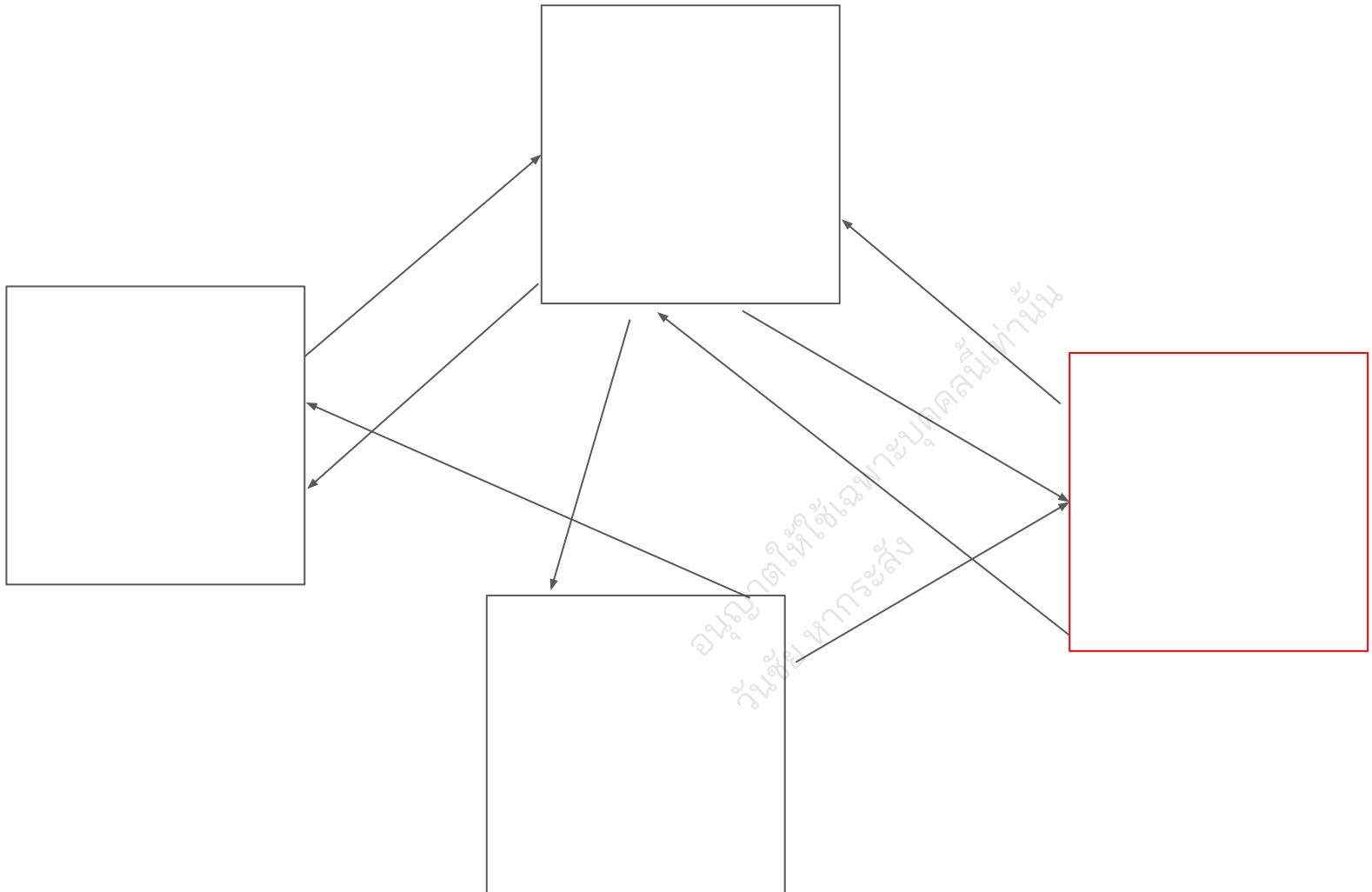
# Rules#3

Avoid complex dependencies

อนุญาตให้ใช้เฉพาะบุคคลซึ่งทำนั้น  
วันซึ่ง ห้ามระบุ







# Rules#3

Simplify context mapping

เรื่องการนำร่อง  
การดำเนินการตามมาตรฐานสากลที่กำหนด





# Tactical Domain Driven Design: Entity & Value object

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.



# Entity & Value object

จักรยานที่ใช้ในระบบ  
วันนี้ ทางเราจะ



# Entity

“An object defined primarily by its identity is called an ENTITY”

Eric, Evans. Domain-Driven Design (p. 91).  
Pearson Education. Kindle Edition.

อนุญาตให้ใช้เพื่อการ  
วิเคราะห์ การสอน



# Entity

It's an object with lifecycle. Its value can be mutated through timeline

- User
- Course
- ShoppingCart
- Invoice

อนุญาตให้เข้าชมเฉพาะบุคคลนี้  
วันนี้ ห้ามคราวน์



# **Value object**

An object which an identity are defined by its value are called value object

- DateTime
- Price with Currency
- Color

อนุญาตให้ใช้สิ่งของบุคคลนี้ทำ  
กิจกรรมทางการศึกษา



# Entity

“Tracking the identity of ENTITIES is essential, but attaching identity to other objects can hurt system performance, add analytical work, and muddle the model by making all objects look the same”.

Eric, Evans. Domain-Driven Design (p. 97)



Dealing with Entity and Value object  
required different mindset and approach

This is why the categorization is useful

```
● ● ●
```

```
class SomeService {
    public void DoSomething() {
        User currentUser = this.userRepository.getById(userId);
        // Do something

        User currentUser = this.userRepository.getById(userId); // <--- Some property of user might already
        change at this point
    }

    public void DoAnother() {
        Price myPrice = Price.create(200, Currency.USD);
        // Do something

        Price myPrice = Price.create(200, Currency.USD); // <--- 200 USD is always 200 USD
    }
}
```

# Dealing with Entity

The benefit of defining entity

- Equality by identity
- Ask yourselves “Is this up-to-date?”
- Ask yourselves “Will I create a conflict?”
- Transaction consideration
- Multi-thread consideration



# Dealing with Value object

- Race condition free
- Thread-safe
- Equality in value
- Cannot represent timeline



```
class SomeService {
    public void IncreaseViewCount() {
        User currentUser = this.userRepository.getById(userId);
        currentUser.viewCount = currentUser.viewCount + 1;
        // The above might breaks in a case of parallel processing, such as web server
    }
}
```



```
class SomeService {
    public void DiscountPrice() {
        Price myPrice = this.getCoursePrice(courseId);
        myPrice.discountBy(0.2); // No concern at all!!
    }
}
```

# Entity & Value object

By identifying what is Entity vs. Value object

We can put our thought and concern on what matter.

We can reduce our mental workload by having this categorization.



We don't want to do a “paranoid  
programming” or “overthinking  
programming”

It's not a thing. Don't strive for it.



# Tactical Domain Driven Design: Entity & Value object #2

อนุญาตใช้งานบนคลัง  
วัสดุ การรับส่ง

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.



# Requirement

“User must be able to put courses into a shopping cart”

“Each course have a description and info, which is a complex text with styling”

“Then, user will checkout the cart and make a payment”



# Dissecting

First step of modeling: Dissecting Verbs & Nouns



## Noun

- User
- Course
- CourseDescription
- ShoppingCart
- Payment

## Verbs

- Put
- Checkout

อนุญาตให้เข้าชมพากบุคคลนี้  
ก่อนซึ่งทางรัฐ



## Noun

- User
- Course
- CourseDescription
- ShoppingCart
- Payment

Noun become an object

## Verbs

- Put
- Checkout

Verb become a method



## Noun

- User ← Entity
- Course ← Entity
- CourseDescription ← Value object
- ShoppingCart ← Entity
- Payment ← Entity

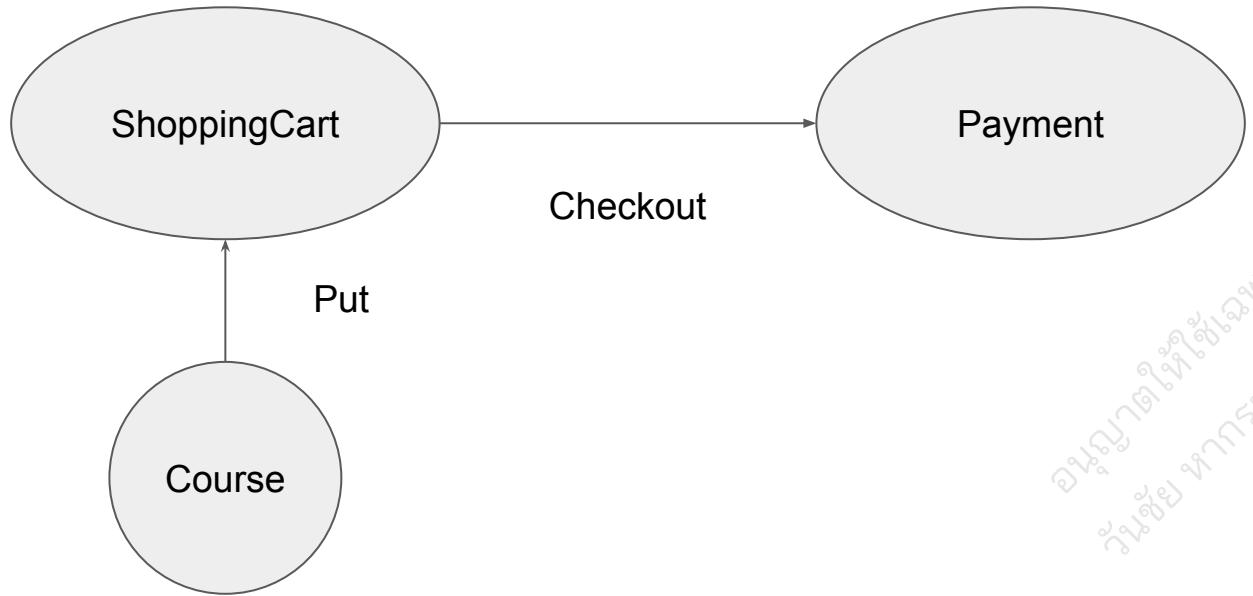
Noun become an object

## Verbs

- Put
- Checkout

Verb become a method





ឧប្បត្តមន៍  
រាយការណ៍  
ធនធានកម្ពុជា



```
package fordemo.entities;

import java.util.UUID;

import fordemo.valueobjects.CourseDescription;

class Course {
    private UUID id;
    private String courseName;
    private CourseDescription description;
}
```



```
package fordemo.valueobjects;  
  
public class CourseDescription {  
  
}
```



```
package fordemo.entities;

import java.util.List;

public class ShoppingCart {
    private List<Course> courses;

    public void Put(Course course) {

    }

    public Payment Checkout() {
    }
}
```

# Sum up

อนุญาตให้เชื่อมทางบุคคลนี้  
วันนี้ ห้ารรลส์



# Sum up

อนุญาตให้ใช้เฉพาะบุคคลนี้เท่านั้น  
วันที่ ๙ มกราคม พ.ศ.๒๕๖๘

- Modeling start by extracting noun and verbs
- A noun become an object
- A verb **mostly** become a method
- We introduce two types of object, entity and value object
- Entity ← Change through time
- Value object ← Equality by value
- Entity is time-sensitive. Value object is not.
- Categorization help eliminate type of concern
  - Don't strive for overthinking programming
  - Minimize mental workload





# Tactical Domain Driven Design: Before we begin

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.



# Before we begin

Tactical Domain Driven Design is about how to model business domain into a codebase.

The technique is based on classic OOP language such as (Java, C#)



# Before we begin

We can just say “Go model it”

But...

We found a lot of pattern on modeling the real-world and we create a toolbox around these pattern



# Types of classes

Just like MVC architecture, Clean Architecture, Domain driven design have **types of classes**.

Controller ← Handle input-output

Repository ← Handle persistent datastore (aka. Database)

We also have types of classes to help categorize the intention and specific property for the first glance



# Before we begin

So what we do is:

1. We will go through requirement
2. We will introduce DDD toolbox, bit by bit.
3. We will apply to the requirement





# Tactical Domain Driven Design: Aggregate

อบรมอาชีวศึกษาเพื่อความคุ้มครอง  
วัฒนธรรมไทย



© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.

# Requirement

“Enterprise customer can put a course for specific amount, so later they can distribute to their employee”

“Enterprise customer can apply their promotion code into the shopping cart and get discount”

“Our Promotion: Total amount of course exceeds 10 course, you get 3000 THB discount”

We will focus on ShoppingCart



Skooldio

← → X ↻ ⌂

## Shopping Cart

[+]

Course	Amount	Price
OOP The right way	5	7450
Basic Architecture	5	10000
Advance React	3	30000

Promo code: SkooldioRocks555

Total	47,450
Discount	3,000
Net	44,450

〃

# Dissecting

Design a class which can be used to render this page

อนุญาตให้ใช้เฉพาะบุคคลนี้เท่านั้น  
วันชัย ห้ามระลัง





```
public class ShoppingCart {  
    private UUID id;  
    private List<ShoppingCartItem> items;  
    private PromotionCode promotion;  
    private double netPrice;  
}  
  
public class ShoppingCartItem {  
    public UUID id;  
    public Course course;  
    public int Amount;  
}  
  
public class PromotionCode {  
    private UUID id;  
    private String code;  
  
    public String getCode() {  
        return code;  
    }  
}
```

# Dissecting

We have three entities:

- ShoppingCart
- ShoppingCartItem
- Promotion

อนุญาตให้ใช้เฉพาะบุคคลนี้เท่านั้น  
ก่อนรับสิ่งของ



**What if user add one more of Basic  
Architecture course?**



```
public class SomeClass {  
    public void SomeMethod() {  
        ShoppingCart cart = getShoppingCart();  
        cart.getItems().get(1).Amount++;  
    }  
}
```

Does PromoCode still valid?

What about NetPrice, do we need  
recalculation?

# Dissecting

There are some entities, which consist of multiple domain objects.

But the consistent boundary of their children must be persist.

So we create a new term to call these entities:  
“Aggregate”



# Aggregate

A DDD aggregate is a cluster of domain objects that can be treated as a single unit.

An aggregate will have one of its component objects be the aggregate root. Any references from outside the aggregate should only go to the aggregate root. The root can thus ensure the integrity of the aggregate as a whole.

([https://martinfowler.com/bliki/DDD\\_Aggregate.html](https://martinfowler.com/bliki/DDD_Aggregate.html))





```
public class SomeClass {  
    public void SomeMethod() {  
        ShoppingCart cart = getShoppingCart();  
        cart.getItems().get(1).Amount++;  
    }  
}
```

In another word: Don't do this



```
public class ShoppingCart {  
    public void AddItemToCart(Course course) {  
        if (this.totalAmount() > this.promotion.ReceivedDiscountAmount) {  
            throw new UnsupportedOperationException("Cannot add item for a cart with this promotion code");  
        }  
        ShoppingCartItem courseItem;  
        for (ShoppingCartItem item : items) {  
            if (item.course.id == course.id) {  
                courseItem = item;  
            }  
        }  
        if (courseItem == null) {  
            courseItem = new ShoppingCartItem(course);  
        }  
  
        courseItem.Amount++;  
  
        // Recalculate Net  
    }  
}  
  
public class SomeClass {  
    public void SomeMethod() {  
        ShoppingCart cart = getShoppingCart();  
        cart.AddItemToCart(course);  
    }  
}
```

# Aggregate

Aggregate is a consistent boundary

Whenever we found aggregate, we only modify aggregate via **Aggregate Root**, and nothing else

Note: Technically, every aggregate root is an entity, but not every entity is aggregate root.



Common confusion: What's different  
between aggregate and entity?

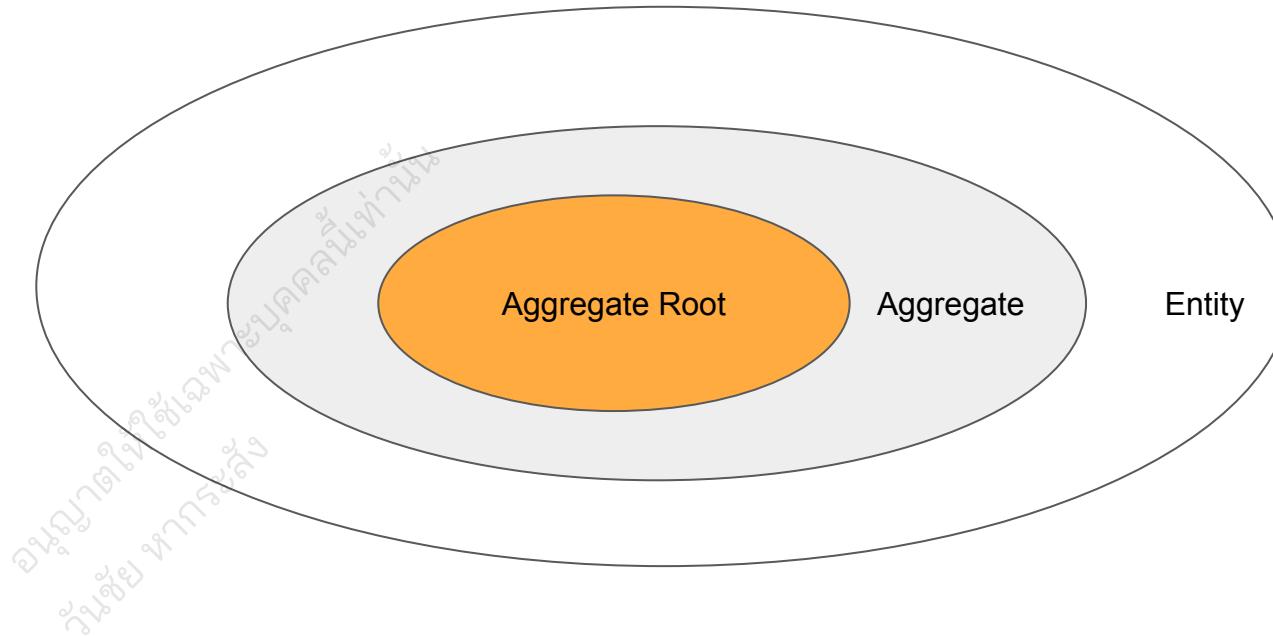
# Aggregate

Beware. Many developer (in my experience) misunderstand that Aggregate != Entity.

Kinda true and not true.

It's more like next image.





Aggregate and Aggregate root is specialized subsets of Entity.

They are entities which has some extra distinctive characteristic.

**Aggregate is not a relationship**

อนุญาตให้เขียน  
วันนี้ ยกเว้น  
การเขียนในห้องเรียน



# Tactical Domain Driven Design: Service

อนุญาตให้ใช้เพื่อพัฒนา  
วัสดุทางการศึกษา

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.

# Requirement

“User can received a promotion code from Skooldio”

“User can use only their code”

“Some promotion code can be distributed to many users”



# Dissecting

Noun:

- **User**
- **PromotionCode**
- Skooldio

Verb

- **Receive**
- **Distribute**
- Can use
- Can be





```
public class User {  
    public void Receive(Promotion promotion) {  
        // ???  
    }  
}  
  
public class Promotion {  
    public void Distribute(User user) {  
        // ???  
    }  
}
```

# Why?

In many cases, there are some operations which involved multiple entities.

And it is not clear which entity should it belong too.

Should we put it in user?

Should we put it in promotion?

Contrast to: Checkout



# Introducing Service Object

นี่คือวิธีการรับ  
ค่าตอบแทนที่ดีที่สุด

# Service Object

Services are **stateless objects** that perform some logic that do not fit with an operation on an Entity or Value Object.

They perform domain-specific operations, which can involve multiple domain objects.

(<https://thedomaindrivendesign.io/what-is-tactical-design/>)





```
public class UserPromotionService {  
    public void Distribute(Promotion promotion, User user) {  
    }  
}
```

# Why stateless?

อนุญาตให้เข้ามาในบุคคลท่าน  
วันนี้ยังคงมีความสำคัญต่อไป

# Usefulness

Whenever we see service object, we know that it is a stateless coordinator

No transaction boundary concern (That's an aggregate)

No lifecycle concern (That's an entity)

Not even an equality (That's a value object)



# Beware

It's easy to implement entities coordination as service object

Service object works when you don't want to track the coordination.

If we want to keep track of the coordination, you need to create a new entity.



# New entity vs. Service object

Example: Display the date we start distribute these Promotion

อนุญาตให้เข้ามาในบุคคลนี้ท่าน  
วันนี้ยัง ทำการสั่ง





```
public class PromotionDistribution {
    private DateTime distributedDate;
    private UUID userId;
    private UUID promotionId;

    public void Distribute(Promotion promotion, User user) {
        this.userId = user.id;
        this.promotionId = promotion.id;
    }
}
```

# New entity vs. Service object

Be mindful of a business operation disguised in a verb form.

Tracking required concrete object and noun.

Example:

- Approve → Approval
- Pay → Payment
- Submit → Submission





# Tactical Domain Driven Design: Domain Event

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.



# Requirement

“Whenever a shopping cart with more than 10,000 THB checked out, send sales a notification”

“Whenever a new user registered with type enterprise, send sales a notification”

“Whenever the special promotion code is applied, send sales a notification”





```
public class User {
    public void Register() {
        // Register
        SalesTeam.SendNotification("User registered");
    }
}

public class ShoppingCart {
    public void Checkout() {
        // Checkout
        if (this.Amount > 10000) {
            SalesTeam.SendNotification("Big shopping cart purchased");
        }
    }
}

public class ApplyPromotionService {
    public void Apply(Promotion promotion, ShoppingCart cart) {
        // Apply
        if (promotion.type == PromotionType.Special)
            SalesTeam.SendNotification("Special promotion was applied");
    }
}
```

# Requirement

Let say we have two domain experts

- Sales domain expert
- Courses domain expert

Sales domain expert want to add more notification

Sales domain expert want to how many notification types sales got





```
public class User {
    public void Register() {
        // Register
        SalesTeam.SendNotification("User registered");
    }
}

public class ShoppingCart {
    public void Checkout() {
        // Checkout
        if (this.Amount > 10000) {
            SalesTeam.SendNotification("Big shopping cart purchased");
        }
    }
}

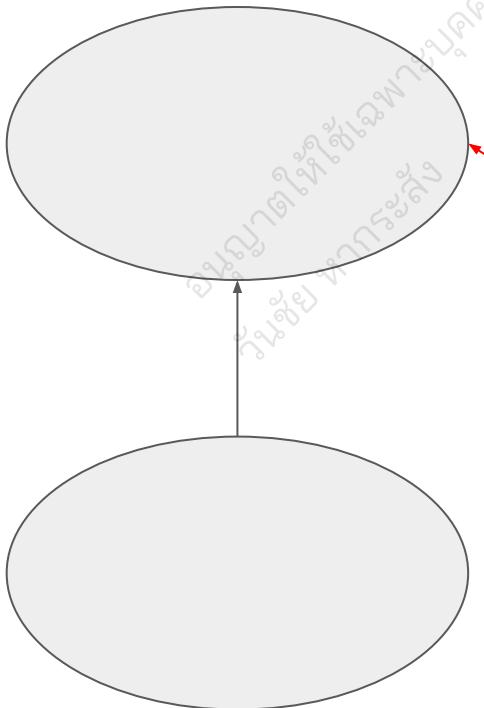
public class ApplyPromotionService {
    public void Apply(Promotion promotion, ShoppingCart cart) {
        // Apply
        if (promotion.type == PromotionType.Special)
            SalesTeam.SendNotification("Special promotion was applied");
    }
}
```

**What happen?**

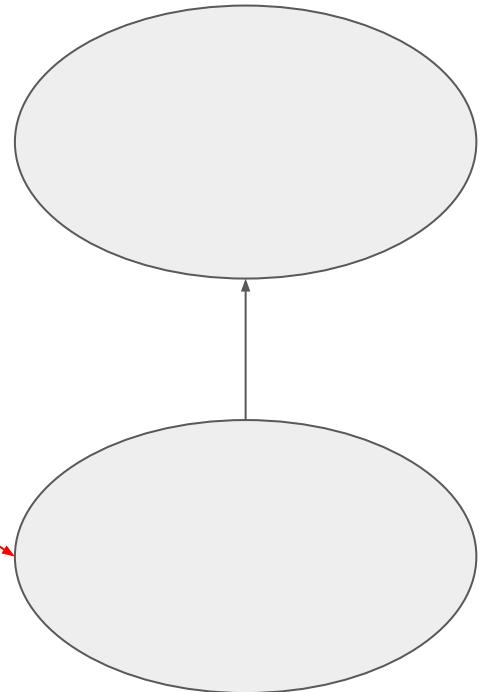
សំណើនៅក្នុងគម្រោងបានរាយការណ៍

Domain1

ข้อมูลใหม่เพิ่มเข้ามาในคลังข้อมูล



Domain2



# What happen

In case of A domain rely on something that happen in B domain, we can

- Ask B domain to call A whenever B is done
  - But then when A domain expert comes along... it's hard to tell what happen

So we create a new style





```
public class User {
    public void Register() {
        // Register
        Event.Publish(UserEvent.Register, "User registered");
    }
}

public class ShoppingCart {
    public void Checkout() {
        // Checkout
        if (this.Amount > 10000) {
            Event.Publish(ShoppingCartEvent.Checkout, "Big shopping cart purchased");
        }
    }
}

public class ApplyPromotionService {
    public void Apply(Promotion promotion, ShoppingCart cart) {
        // Apply
        Event.Publish(PromotionEvent.Apply, "Special promotion was applied");
    }
}
```



```
public class SalesNotification {  
    public void Subscribe() {  
        Event.Subscribe(UserEvent.Register, this);  
        Event.Subscribe(ShoppingCartEvent.Checkout, this);  
        Event.Subscribe(PromotionEvent.Apply, this);  
    }  
  
    public void OnEventTriggered(Event e) {  
        SalesNotifciation.Send(e.message);  
    }  
}
```

## Sales Domain

```
public class User {  
    public void Register() {  
        // Register  
        Event.Publish(UserEvent.Register, "User registered");  
    }  
}  
  
public class ShoppingCart {  
    public void Checkout() {  
        // Checkout  
        if (this.Amount > 10000) {  
            Event.Publish(ShoppingCartEvent.Checkout, "Big shopping cart purchased");  
        }  
    }  
}  
  
public class ApplyPromotionService {  
    public void Apply(Promotion promotion, ShoppingCart cart) {  
        // Apply  
        Event.Publish(PromotionEvent.Apply, "Special promotion was applied");  
    }  
}
```

# Domain Event

Whenever some important stuff happen in each event, a domain can choose to publish a domain event

Another domain can choose to subscribe to those events and act accordingly

Compared to naively trigger other domain methods, this allow domain to have more autonomy and less dependant on others



Keep track of notifications here

Add more notifications here

```
public class SalesNotification {  
    public void Subscribe() {  
        Event.Subscribe(UserEvent.Register, this);  
        Event.Subscribe(ShoppingCartEvent.Checkout, this);  
        Event.Subscribe(PromotionEvent.Apply, this);  
    }  
  
    public void OnEventTriggered(Event e) {  
        SalesNotifciation.Send(e.message);  
    }  
}
```

## Sales Domain

# Domain Event

Property:

- Event must be immutable
  - Anything that happens, happens
  - Fix by reconciliation (ie. Accounting)
- Event usually have a timestamp



# Common Example

Accounting: Paid invoice generate receipt

ERP: Confirmed order generate delivery but  
delivery type logic should not belong to order





# Sum up: Business Collaboration

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.



# Problem Statement

How to sync mental model of development team and business team?

- We want to be able to grow together
- We have a same understanding of complexity

อนุญาตให้ใช้สิ่งพิมพ์ในห้องเรียน  
วันศุกร์ ห้ารุ่งสุข



# Domain Driven Design

A set of practices started by Eric Evans

- Domain Discovery
- Strategic Domain Driven Design
- Tactical Domain Driven Design



# **Strategic Domain Driven Design**

อนุญาตให้ใช้สิ่งพกพาในห้องเรียน  
วันนี้ด้วย ห้ามรบกวน

Since a large system can consist of many different domains and experts, we should have

- Bounded context
- Ubiquitous language
- Context mapping



Human Resource

Course

E-Commerce

SalesPerson

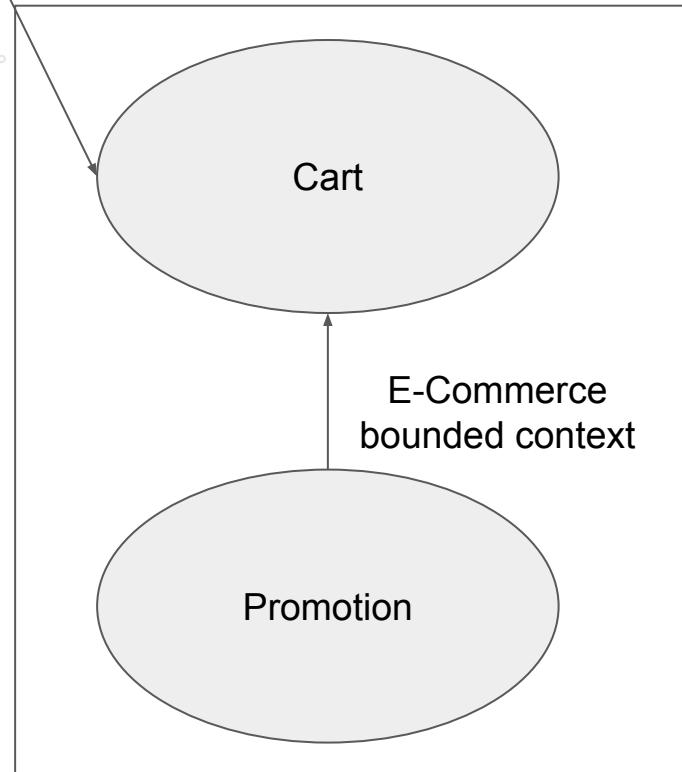
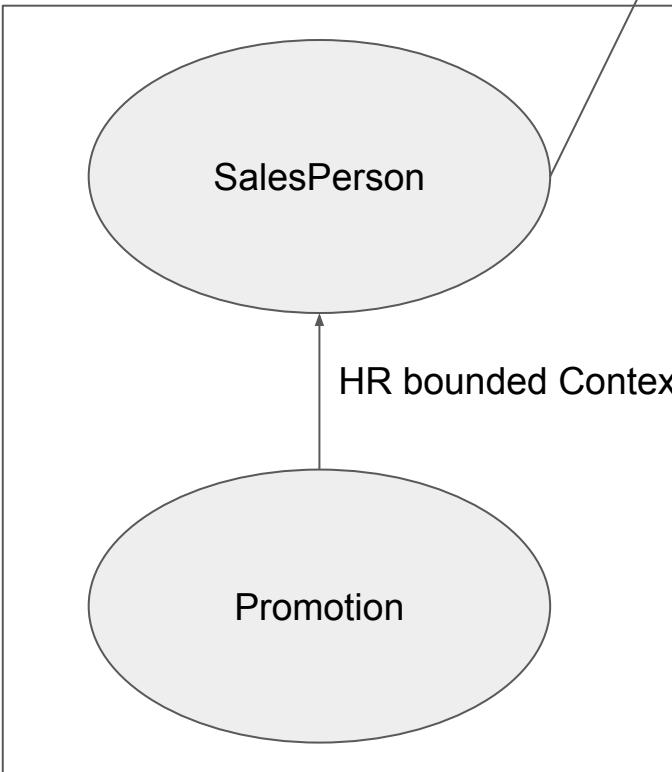
Cart

Promotion

HR bounded Context

E-Commerce  
bounded context

อนุญาตให้ข้อมูลของบุคคลนี้  
วันซื้อ ทำการซื้อ



# Tactical Domain Driven Design

How to design classes to align with Domain?

- Dissecting verbs & nouns
- Nouns become objects and verbs mostly become method

Organizing objects

- Entities
- Value objects
- Service objects
- Aggregates
- Domain events





# Conway's law

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.

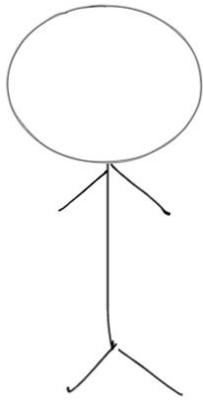




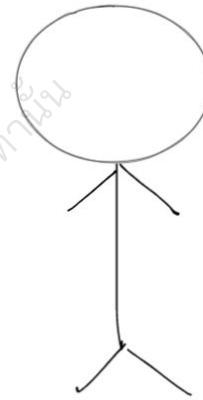
“Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure”

– Melvin E. Conway





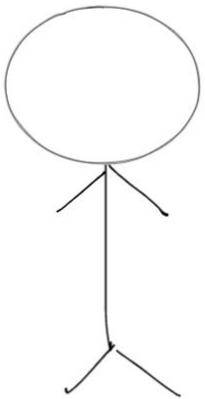
We don't want to talk



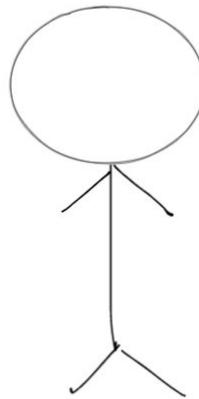
**What will happen?**



Here is my code



Here is my code

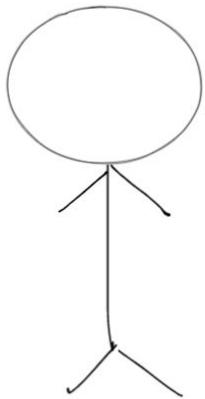


We don't want to talk



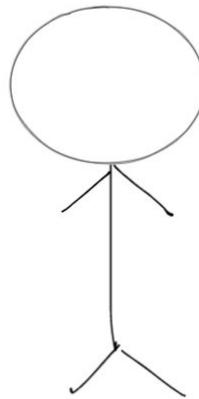
ຂໍ້ມູນກາຕໃຫ້ເພື່ອມວະບຸຄລຸນໜ້ານ໌  
ວິນຍໍ ລາກຮະສົງ

Use my API



We don't want to talk

Use my API



# Many reasons

Why don't we want to talk?

- We simply hate each other
- We are working in different team
- We have different boss
- We are aiming for different KPI
- We have different customer



# On the opposite

Why do we need to talk?

- We are aiming for the same goal
- We have a same boss
- Our KPI depends on that team

Simply put: When it's better to work together



## As a result

Our architecture must align with organization structure

- Force data analytic team to work with course team will result in a separate service & code anyway
  - MVC or MVxyzqwerC
- Do we have an awareness? Or are we kidding ourselves





# Cross-functional team

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.

# Traditional

Each requirement, we created 5 tickets

- Database: Prepare queries
- Security: Prepare for auditing
- Infrastructure: Nothing for now?
- Backend: Create an API
- Frontend : Create a webpage

อนุญาตให้ใช้สิ่งที่  
กวนชั้ย ห้ามคัดลอก



# Traditional

Ok. But every team should be busy

- Frontend: I can start next month
- Backend: I can start next week
- Database: I can start next two weeks
- Security: We are in a big audit. Wait for 3 months



# Traditional

- Frontend: I can start next month
- Backend: I can start next week
- Database: I can start next two weeks
- Security: We are in a big audit. Wait for 3 months ← **This is where we can start**

**How long does it takes to put this simple feature up?**

**Ans: At least next quarter.**



# Traditional

And they don't care

- Frontend: I want to produce many frontend stuff
- Backend: I want reliable backend
- Database: My database should not crash

Who care about the actual launch?

**Team structure does not align with organization value**



# Agile Methodology

อนุญาตให้ใช้เฉพาะบุคคลเท่านั้น  
วันนี้ การล็อป

# Agile

Because of this problem, Agile advocate for cross functional team.

Responsible for whole deployment

Team can do a work end-to-end



# Agile

As a result

- Success of team align with business
- Team members have an incentive to talk with each other
- Work get done, one-by-one



# Important Notes!

Both Agile and Traditional are not totally right

We might need multiple model

New theory: Team topologies



# Ok. Cross-functional team



# But..

- My course page need to know if user already paid for the course
- Our data analytic team must have a user behaviour data



# Approach

There are few approaches to this

- Let's work on a same codebase (Monolith)
- Separate services
  - SOA
  - Microservices





# Monolith

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.



© 2021 Skooldio Co., Ltd.  
All rights reserved.

# Collaboration Pattern

Let's work on a same codebase (Monolith)

Everyone working on the same codebase

Team own some part of code

- Folder
- Module
- Library
- Java Project



อบรมอาชีวศึกษาเพื่อพัฒนาคุณภาพชีวิต  
วันที่ ๙ แห่งการศึกษา

The screenshot shows a file tree structure for a Java application named "MONOLITH". The structure is as follows:

- MONOLITH
  - src
    - courses
      - CourseController.java
      - CourseService.java... 1
    - dataanalytics
      - DataAnalyticsBlah.java
      - DataAnalyticsService...

# Monolith

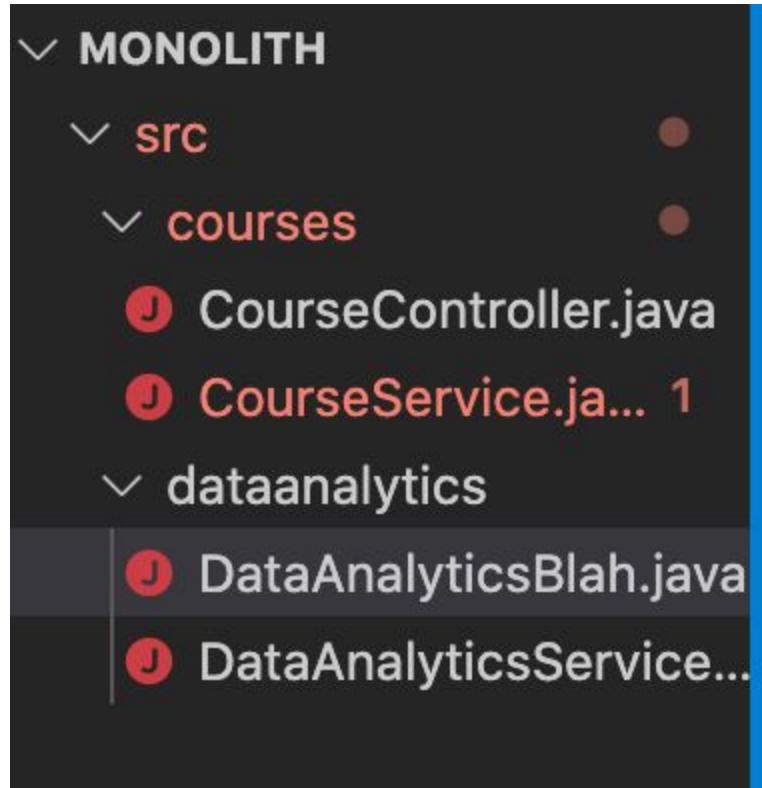
Monolith should be modular

- Interfaces between teams should be both thin and clear
- Minimize internal dependencies

แนวทางการจัดการความต้องการ  
และพัฒนาซอฟต์แวร์



อนุญาตให้ใช้สิ่งบุคคลที่สาม  
วันชัย ห้าร่องฟัน



# Define cross-team boundaries and APIs

อนุญาติให้ใช้ภาพประกอบ  
วัสดุที่ยังคงอยู่ในห้องน้ำ

# Monolith

อนุญาตให้มีเฉพาะบุคคลท่าน  
วันนี้ ทำการซื้อ

## Advantage

- Infra become simple
- No distributed system trick
- Enforce good coding standard practices across teams

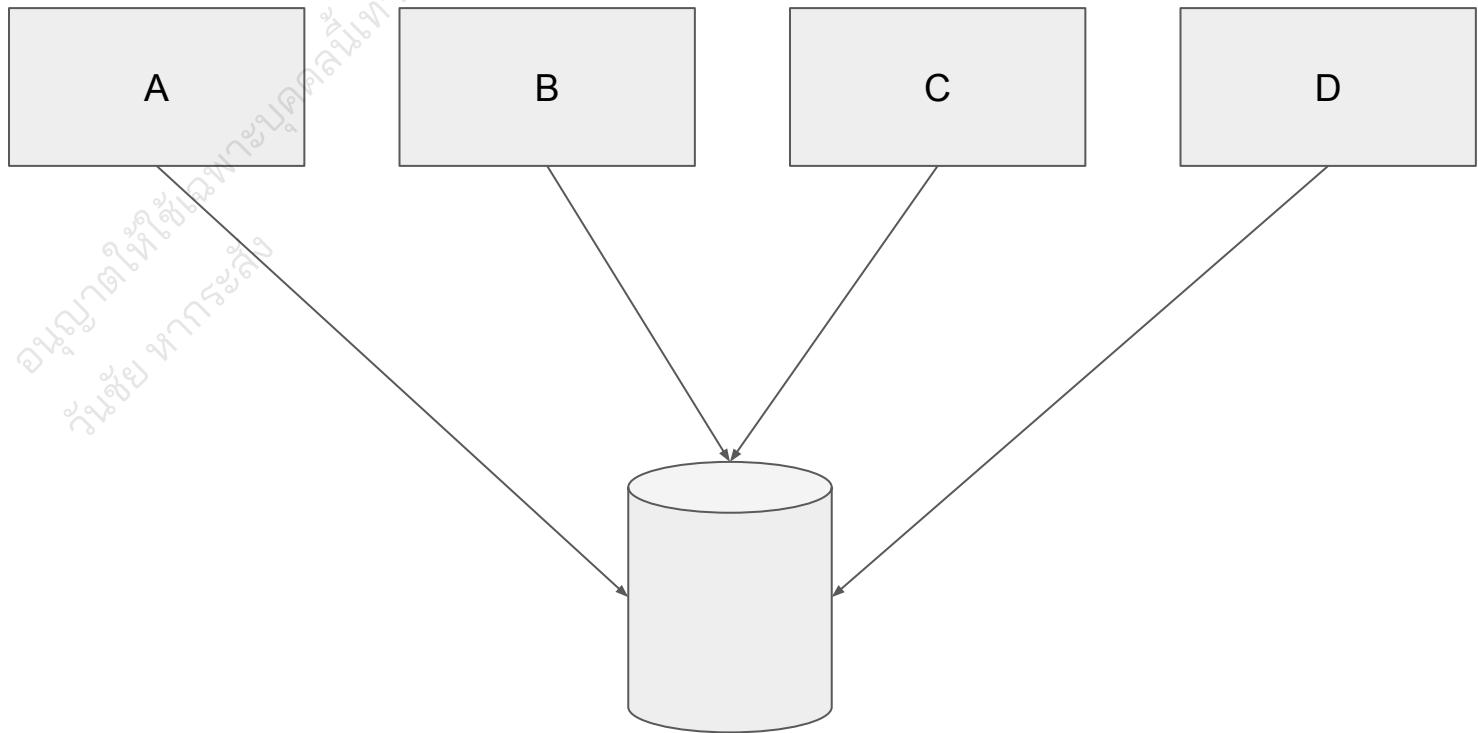


# Monolith

## Disadvantage

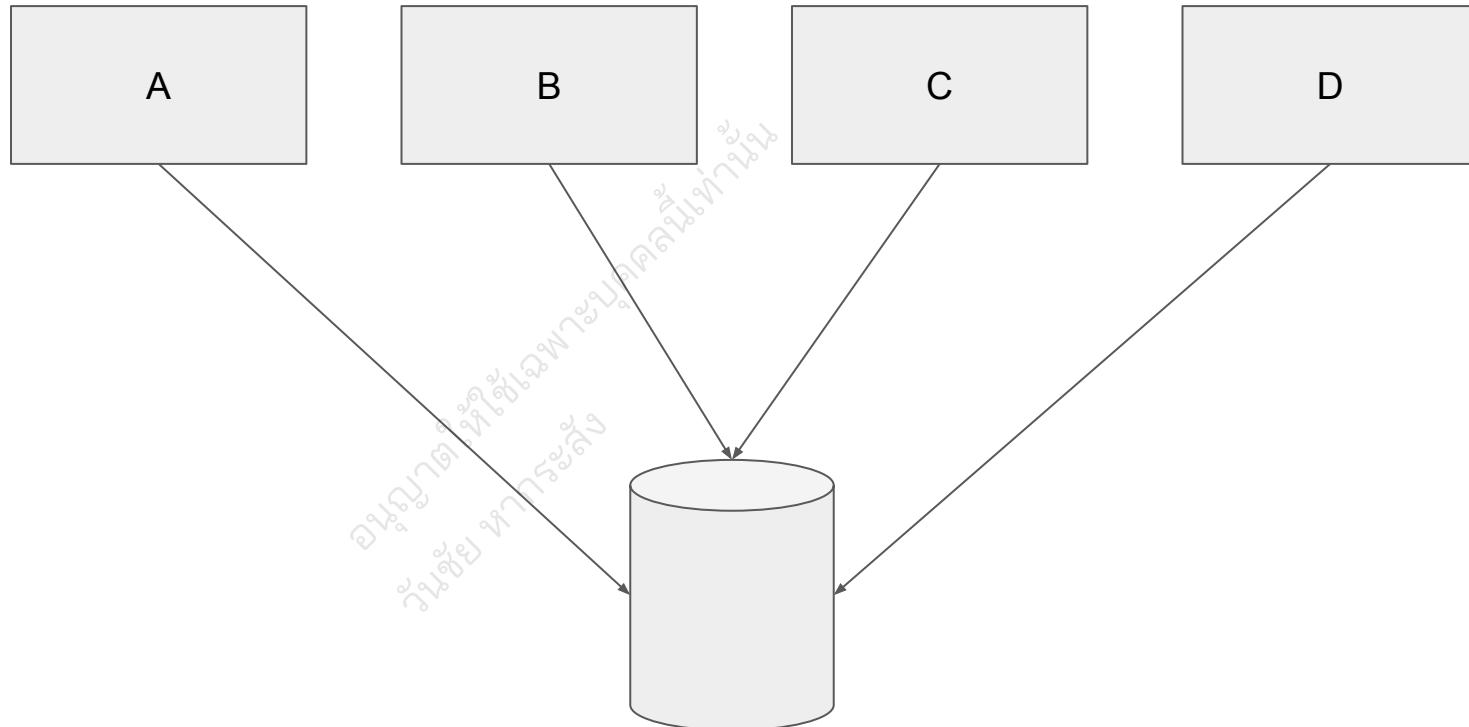
- Single technology
- Single style
- **Managing release!**



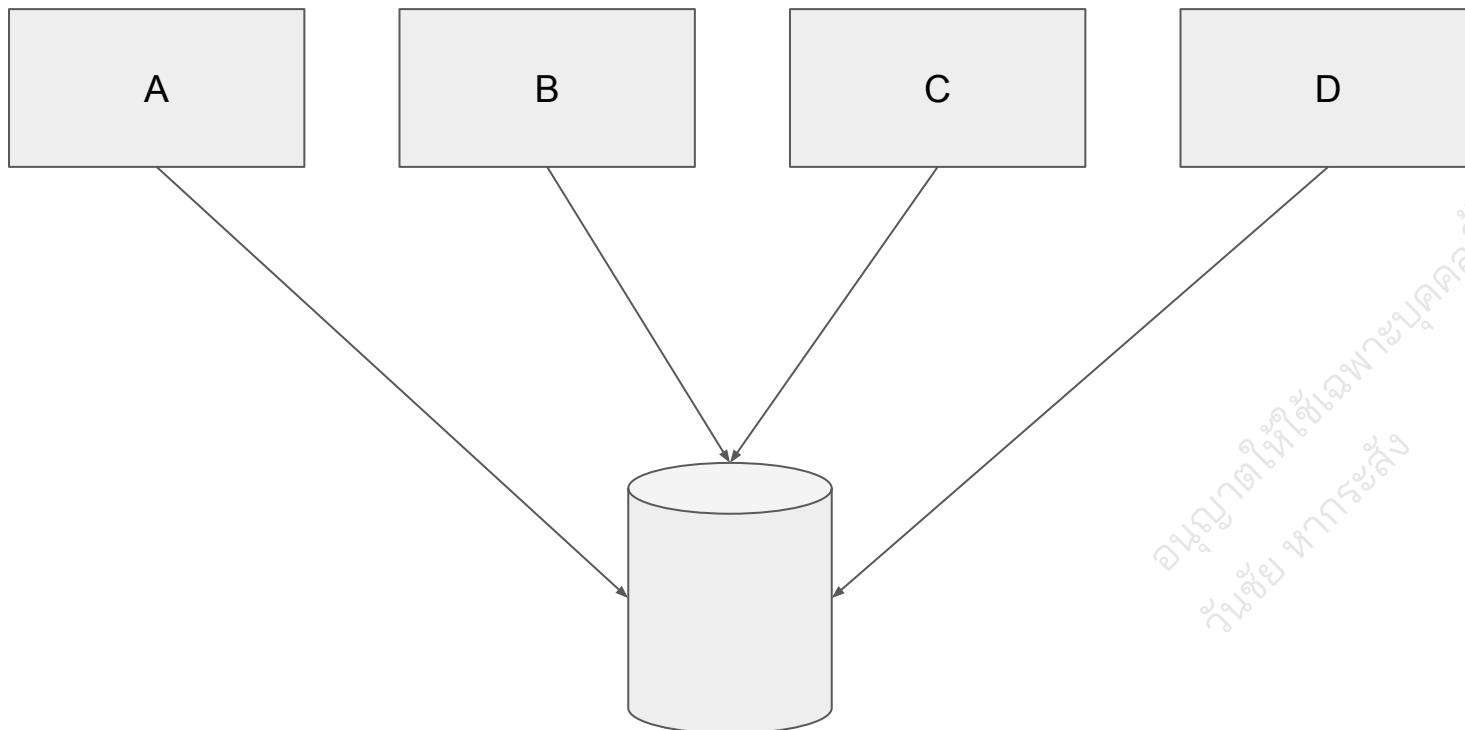


ຮ່ວມງານໃຫ້ສະບັບ  
ວິຊ່າການຂອງ

Breaking changes.  
Need to rollback and  
Freeze



I have deadline  
tomorrow with my  
client



នូវការណែនាំរបស់ខ្លួន  
នៃការងារដែលមិនអាចបញ្ចប់  
ឡើងទេ ហើយ ការស្វែងរក



# Monolith

- It's a good starter
- Avoid using Monolith when teams or team members start to have friction and conflict because of
  - Release cycle
  - Code conflict (might be solvable by better codebase engineering)
  - Technical decision according to domains
    - SQL vs Tableau





# **Multi-services architecture**

## **SOA & Microservices**

วิเคราะห์  
รุ่นชีวะ ห้องเรียน  
การศึกษา

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.

# What's monolith problem?

- Conflicting code standard
- Single technology
- Release cycle

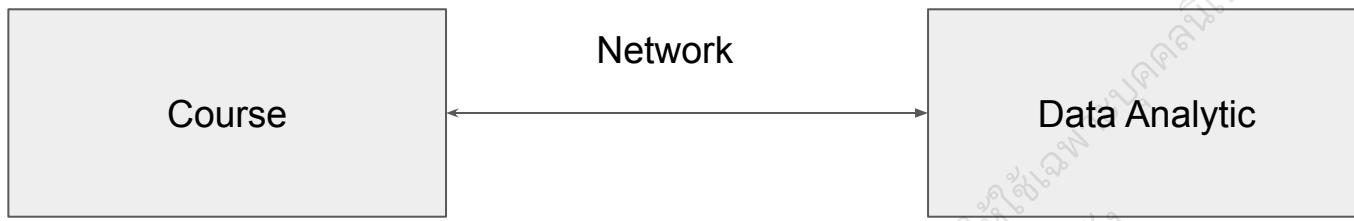
อนุญาตให้เข้าร่วมการนำเสนอ  
วันนี้เป็นการลับ



# The idea

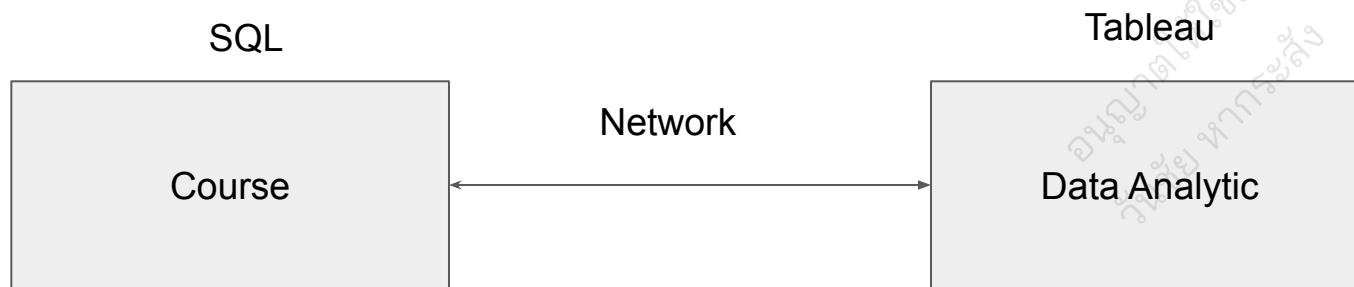
What if we separate system into multiple deployable services, and communicate via network?





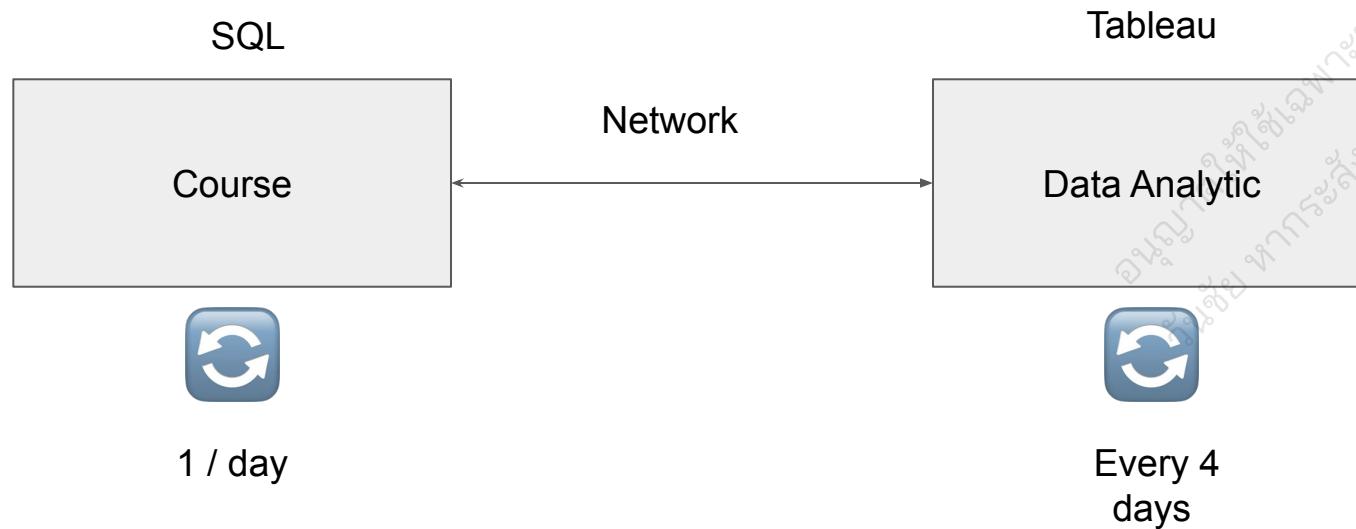
# SOA & Microservices

We can use multiple technology



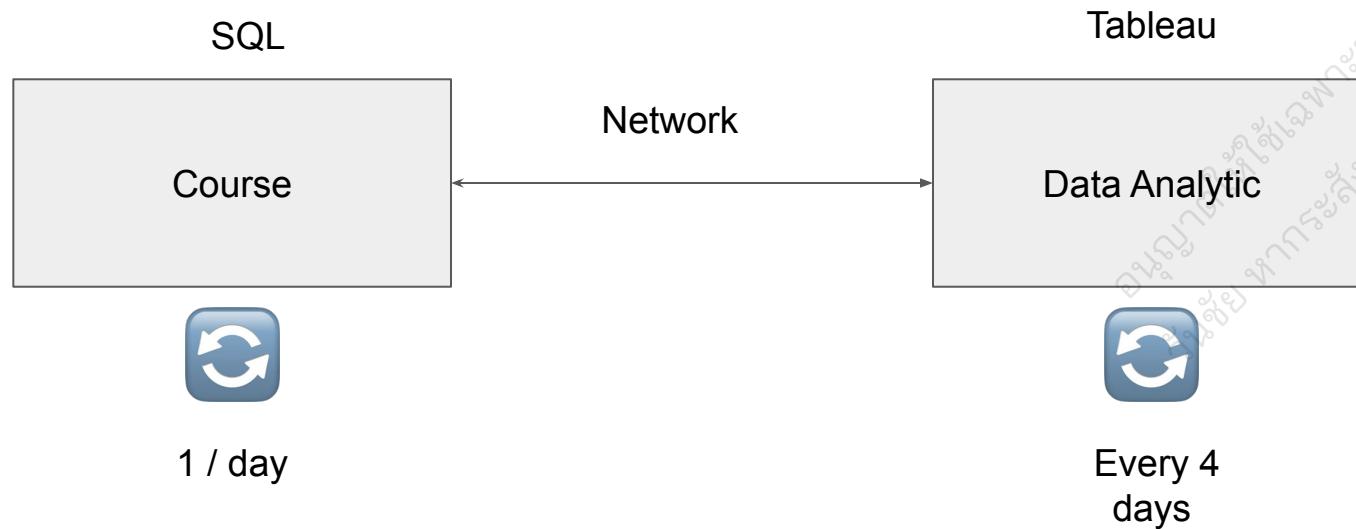
# SOA & Microservices

We can have multiple release cycle  
(since it is different server)



# SOA & Microservices

We can have multiple coding standard



อนุญาตให้เข้าเฉพาะบุคคล  
วัณชัย หกกระลัง

# Sounds good: What's bad?



# **Distributed system fallacies**

อนุญาตให้เข้ามาบุคลนท่าน  
ร่วมช่วย ห้ามระลิบ

- The network is reliable;
- Latency is zero;
- Bandwidth is infinite;
- The network is secure;



# What's the problem?

Distributed system is hard. Network is unreliable.

- Retry strategy
- Synchronization & Data loss
- Availability model
- SLA
- Monitoring
- Consistent boundaries
- CAP Theorem concern



# What's the problem?

Testing becomes harder

- Mock data
- Integrated environment

อนุญาตให้ใช้สิ่งพิมพ์โดยคลิกขวา  
วันนี้ ห้ามรีลัช



# What's the problem?

อนุญาตให้ใช้เฉพาะบุคคลเท่านั้น  
วันด้วย ห้ารูปถ่าย

## Versioning & Backward compatibility

- Data analytic use Course API
- API need changes
- v1/v2
- Every teams need to maintain backward compatibility. Increase whole organization-workload.



**Multi-services setup meant to solve  
organization problem**



# Difference between SOA & Microservices

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.



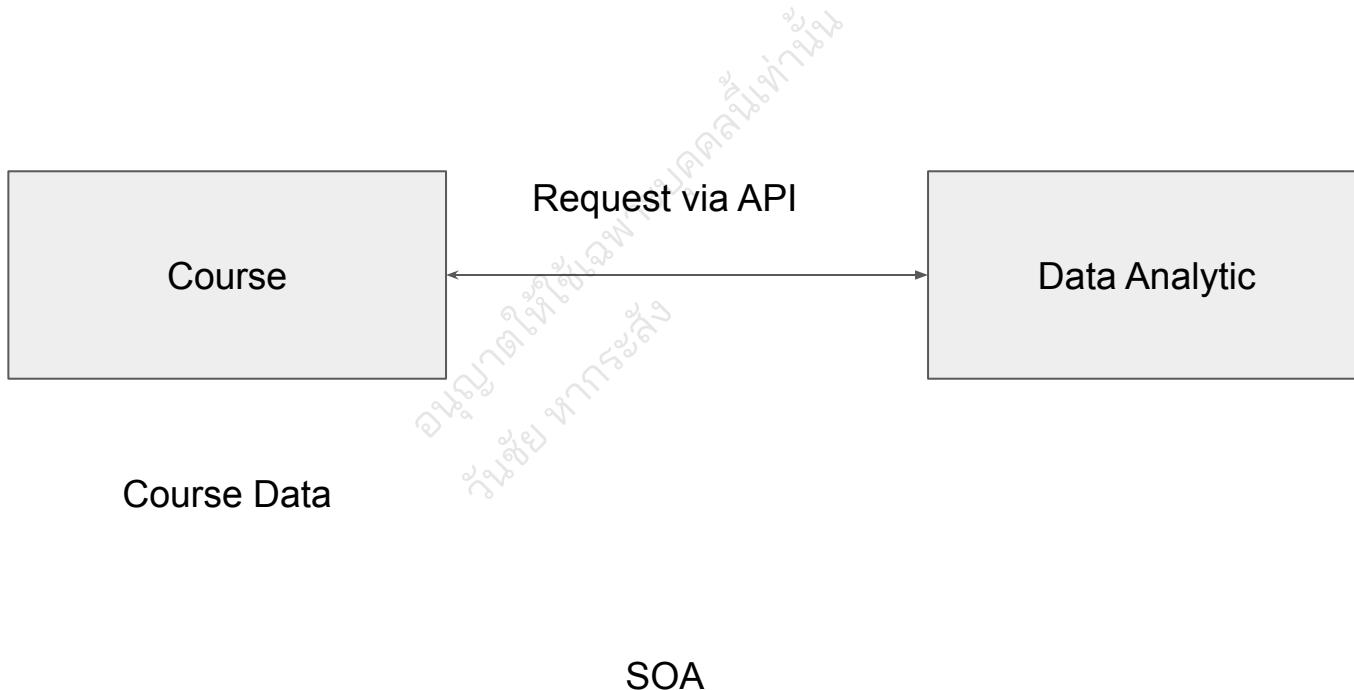
# **Textbook difference**

SOA encourage single source of truth

Microservices allow data duplication

อนุญาตให้มีเฉพาะบุคคลนั่นกัน  
วันซึ่ง ทำการซื้อ





ระบบการจัดการเรียนรู้แบบคลาวด์ที่มีชีวิต



Request via API and  
duplicate for future usage



Microservices

# That's it

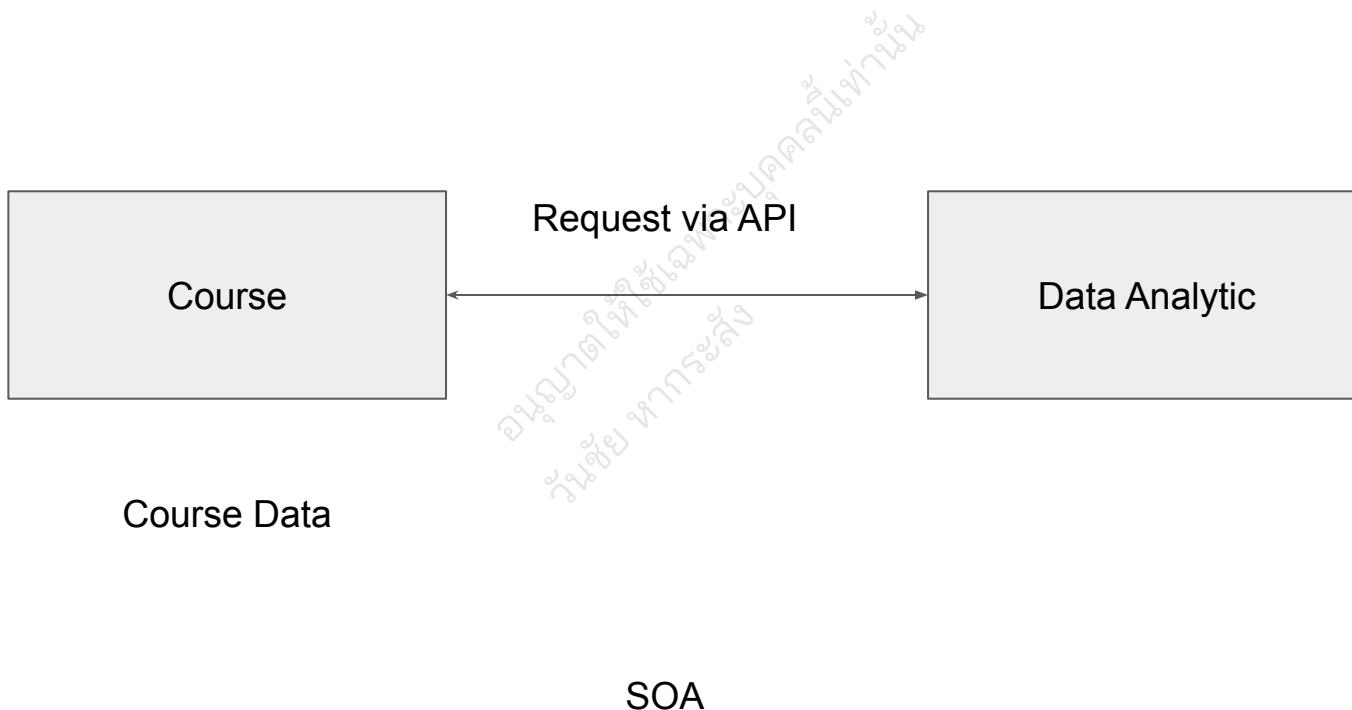
มนต์ที่ทำให้คุณเป็นคนดี  
มนต์ที่ทำให้คุณเป็นคนดี  
มนต์ที่ทำให้คุณเป็นคนดี

# Textbook difference

## Story time

- SOA is invented in the era where we believed that data duplication is the greatest sin
- SOA guarantee data consistency





# Textbook difference

## Story time

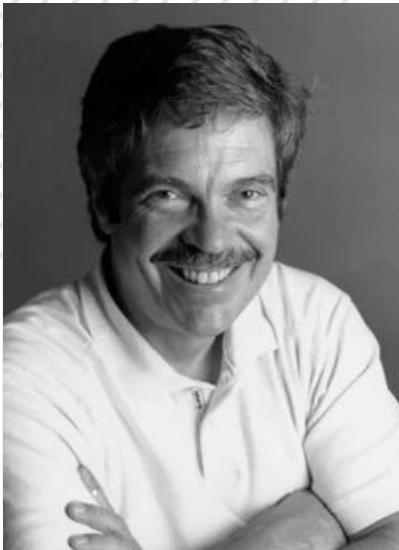
- Then we found out that data consistency is overrated
- Latency is a king
- Reliability is a king
- Let's do multiple services setup where we allow to duplicate data
- But wait, we already tell everyone out there that SOA must not duplicate data
- We can't take our word back!
- **Solution: new name**



**And that's how microservices born**

“

“Programming is a pop culture”



Alan Kay: Inventor of OOP

อนุญาตให้สื่อสารภาษาบุคลิกนี้  
วันซึ่ง ทางรัฐ



## Footnote

This is a story based on historical context that I know

If you asked Microservices inventor, they will not tell you that it is invented because of new name.

No-duplication dogma in SOA was real.

I believe the need for the new name is real.

Take this with a grain salt



## Footnote

Also, most people don't follow the definition.

Most of the time, we don't know the difference  
and it is unclear.

Don't be dogmatic about definition.

Because: Programming is a pop culture





ឧណ្ណាតហើង់ផែបាយបគ្គលូវការ  
វំជួយ ការរំសែ

# Design service boundary

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.

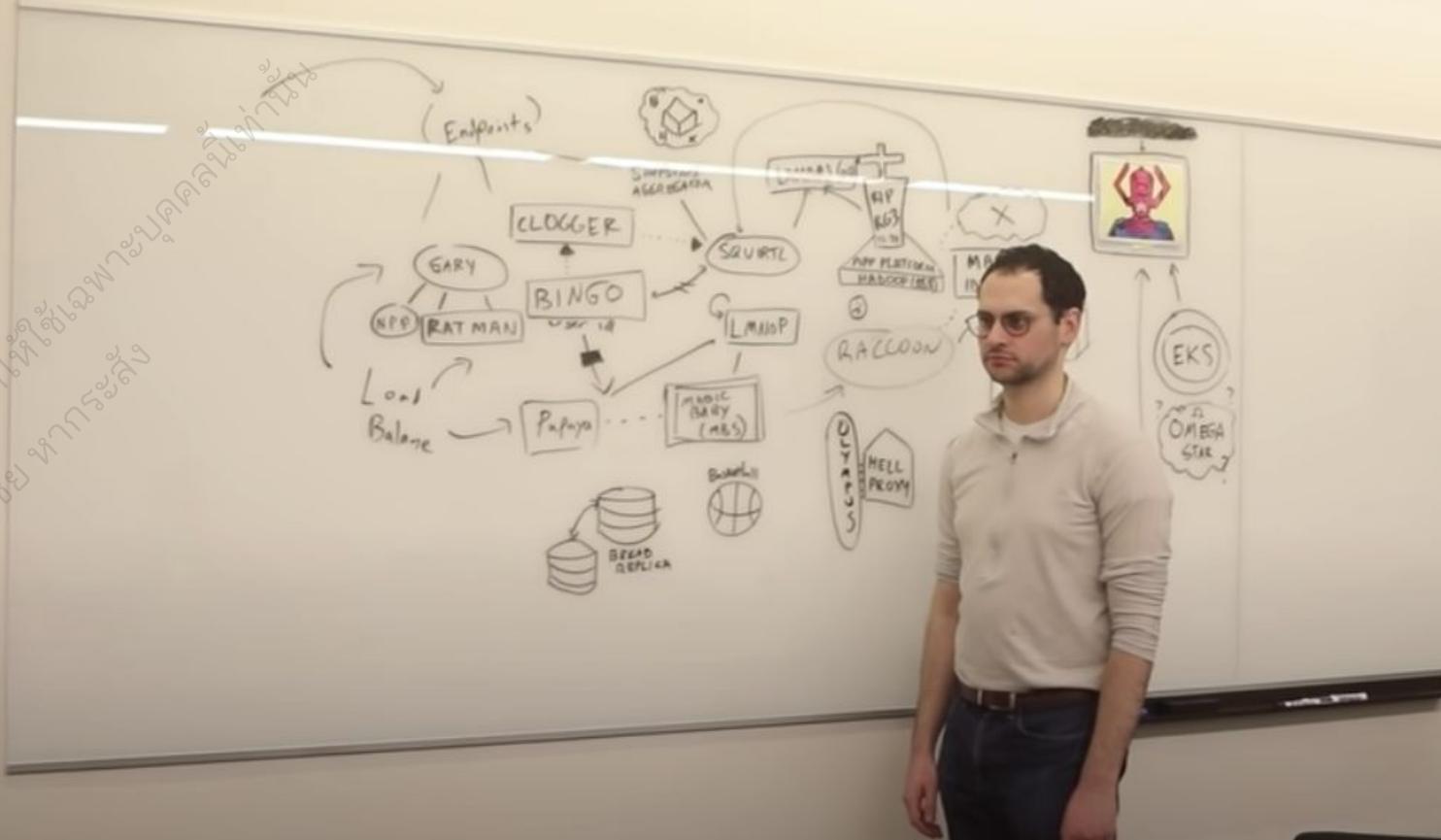
# Question

How many services should I have?

อนุญาตให้เชื่อมทางบุคคลนี้ท่าน  
วันนี้ยังคงดำเนินการล่วง



อนุญาตให้มีเฉพาะบุคคลที่งาน  
วิจัย ทำการระดับ



Source: <https://www.youtube.com/watch?v=y80noxKotPQ>

# One question: Why?

เรื่องการเมืองและการบุคคลที่น่าสนใจ  
วันนี้ทางเราจะล

# Designing service boundary

Respect two principles

- Bounded context
- Conway's law

อนุญาตให้เข้าออกแบบบุคคลที่รับ  
ภาระ ภาระลับ



# Designing service boundary

Bounded context determined business domain

อนุญาตให้ใช้เฉพาะบุคคลนี้  
วันนี้ยังคงรักษา



Context: Payment

Luna

Mirana

Batman

Captain

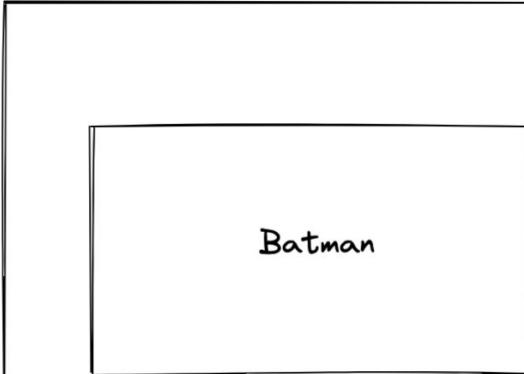
Context: Course

Context: Payment



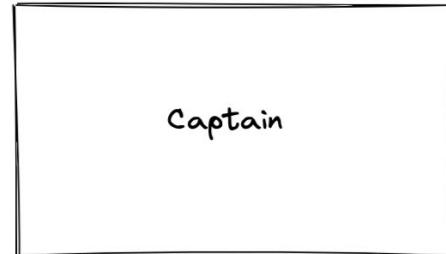
Payment failed  
Course need new feature

Mirana



Context: Course

Captain





# **Designing service boundary**

Conway's law determine team collaboration

บริษัทให้เชื่อมสู่คลังข้อมูลน้ำหนัก  
วันรุ่ง ทำการล็อค



Team: Han Solo

Luna

Mirana

Team: Delta

Batman

Captain

# Designing service boundary

Conway's law determine team collaboration

- Ownership of Batman?
- Conflict goal
- Conflict KPI
- Coding style and technology



# Designing service boundary

- Respect Bounded context: Ideally one bounded context per service
- Respect Conway's law: Ideally a service should have a single owner



# Design smell

## Success microservices

- Changes happened in few services own by same team

## Bad microservices

- This change require change in 6 services, coordinating release cycle and backward compatibility, big meeting across the organization

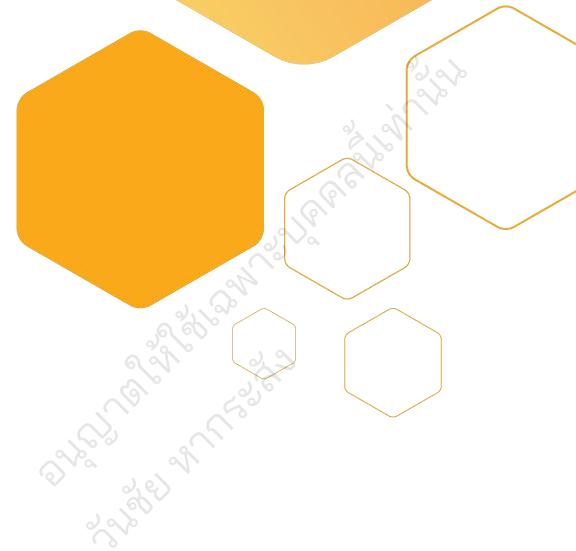




# **Sum up: Organization Collaboration**

© 2021 Skooldio Co., Ltd. This document contains proprietary information of Skooldio Co., Ltd. and shall not be reproduced, distributed, or transmitted, in whole or in part, without the prior written permission of Skooldio.

© 2021 Skooldio Co., Ltd.  
All rights reserved.



# Problem Statement

อนุญาตให้ใช้สิ่งที่  
ร่วมช่วย ห้ามลัง

How do we collaborate across organization?



# **Conway's law**

“Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure”

Consider your organization structure when you design organization level architecture



# Cross- Functional team

In modern tech company, we usually have a cross functional team responsible for end-to-end feature development (or business goal)



# Monolith

Basically a big codebase with a single deployment that everyone is working on, separate by namespaces, libraries or folder structure

Pros:

- Avoid distributed system challenge
- Quick connection between team boundary

Cons:

- Conflicting release cycle
- Single programming language



# Multi-services

Each team own codebase(s) and deployment(s)

Pros:

- Avoid release conflict
- Autonomy in tech stack
- Autonomy in managing release

Cons:

- Distributed system concerns
- Service boundary

Good when organization become large and working in monolith introduce conflict



# **SOA & Microservices**

อนุญาตให้ใช้สิ่งที่  
วันนี้ยังไม่สามารถทำได้

What's the difference between SOA and Microservices?

- Basically the same in community perspective
- SOA encourage single source of truth while Microservices allow data duplication



# Service boundary

Two principles of design services boundary

- Respect bounded context
- Respect Conway's law

อนุญาตให้รีดพากบุคลน์ที่น้ำ  
วันซึ่ง ทำการลับ



มูลนิธิเพื่อชีวภาพบุคคลมีชีวิต  
อนุรักษ์ ภาระสัตว์



**Skoolio**