ตำราบางเล่ม

เรียก Tactical Design

Model-Driven Design

Parinya Ekparinya

Parinya.Ek@kmitl.ac.th

Model to Code!?

1 model เขียนได้ model หลายแบบ

1 model → many code

- Any domain can be expressed with many models, and any model can be expressed in various ways in code.
- For each particular problem there can be more than one solution. Which one do we choose?
- It is important to choose a model which can be easily and accurately put into code.
- The basic question here is: how do we approach the transition from model to code?
- One of the recommended design techniques is the so called ***analysis model***, which is seen as separate from code design and is usually done by different people.

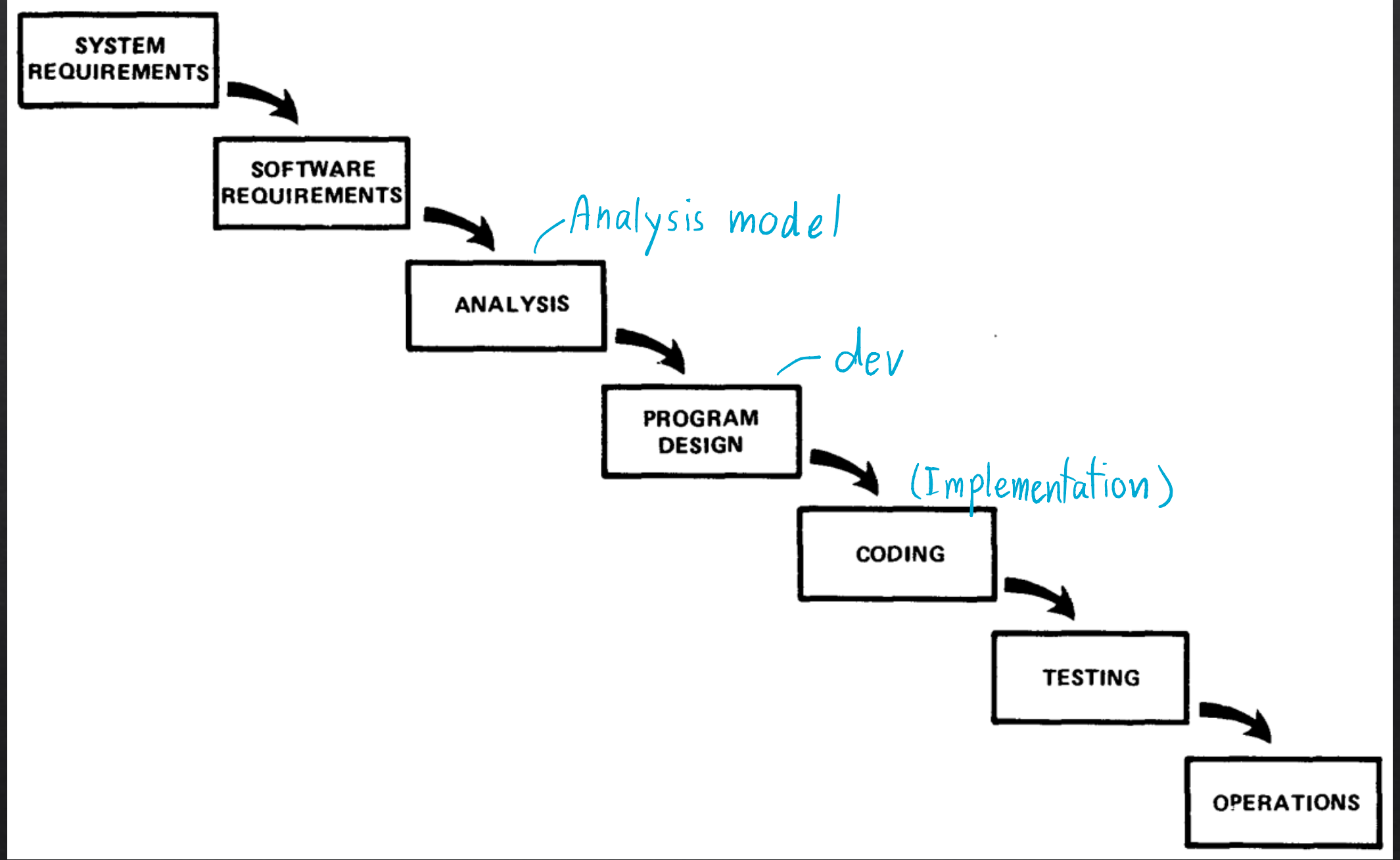
↳ สร้าง model for คุยกับคนให้ req. (ไม่ได้คำนึงเรื่อง code)

Marinescu, F., & Avram, A. (2007). *Domain-driven design Quickly*. Lulu. com.

Criticisms of Analysis Model

- The analysis model is the result of business domain analysis, resulting in a model which has no consideration for the software used for implementation. → ไม่ได้คิดเรื่อง code
- Such a model is used to understand the domain. A certain level of knowledge is built, and the model resulting may be analytically correct.
- This model reaches the developers which are supposed to do the design. Since the model was not built with design principles in mind, developers will be forced to make some decisions on their own, and will make design changes.
- Some of the analysts' knowledge about the domain and the model is lost and there is no longer a mapping between the model (ของ domain model) and the code.
- The result is that analysis models are soon abandoned after coding starts.

Marinescu, F., & Avram, A. (2007). *Domain-driven design Quickly*. Lulu. com.



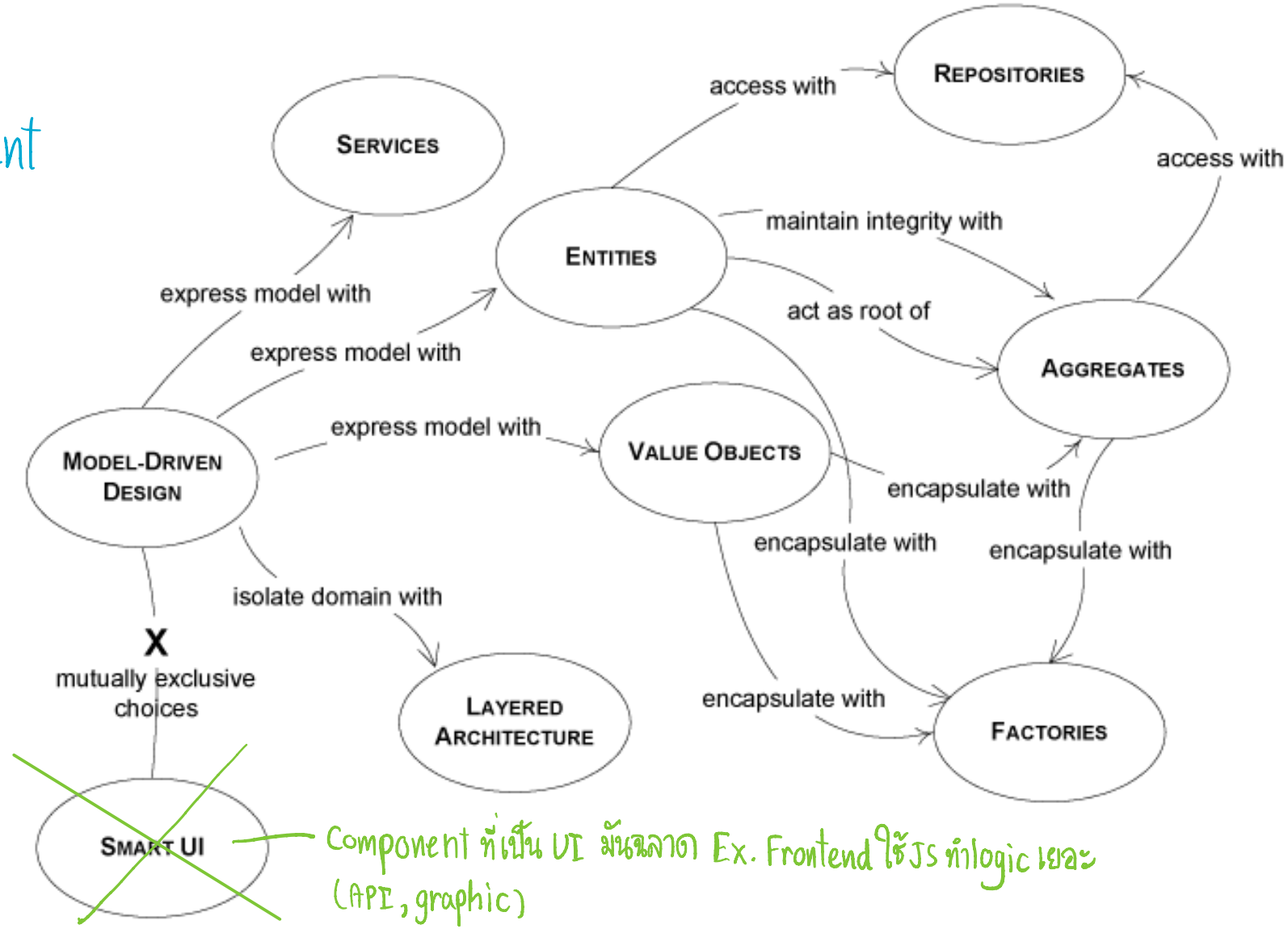
Royce, W. W. (1987, March). Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering* (pp. 328-338).

Model-Driven Design

- A better approach is to closely relate domain modeling and design. The model should be constructed with an eye open to the software and design considerations.
- Developers should be included in the modeling process.
- The main idea is to choose a model which can be appropriately expressed in software, so that the design process is straightforward and based on the model.
- Tightly relating the code to an underlying model gives the code meaning and makes the model relevant.
- To tightly tie the implementation to a model usually requires software development tools and languages that support a modeling paradigm, such as object-oriented programming.

Marinescu, F., & Avram, A. (2007). *Domain-driven design Quickly*. Lulu. com.

Domain event



Ex-student ชื่อ-สกุล, อายุ, วันเกิด, ส่วนสูง, นน.
same 2 คนนี้ไม่ใช่คนเดียวกัน
- obj. in same class, arttribute ก็ยังเป็นต่าง obj.

Entities (1/2)

ส่งผ่านไปต่าง context

- Many objects represent a thread of continuity and identity, going through a lifecycle, though their attributes may change.
  - ระบุได้ว่ามันคนละอัน
- Some objects are not defined primarily by their attributes. They represent a thread of identity that runs through time and often across distinct representations. - ไปที่อื่นก็ยังอยู่
- Sometimes such an object must be matched with another object even though attributes differ. → change attribute ก็ยังอันเดียวกัน
- An object must be distinguished from other objects even though they might have the same attributes. Mistaken identity can lead to data corruption.
  - บ่งชี้ ข้อมูลเพี้ยน

Ex นศ.จบแล้วไปเป็น พนง. บริษัท ก็ยังเป็นคนเดิม

Entities (2/2)

ID, hash

primary key

Identity

Therefore:

ระบุได้ด้วย identity

- **When an object is distinguished by its identity, rather than its attributes, make this primary to its definition in the model.**

สำคัญ

- **Keep the class definition simple and focused on life cycle continuity and identity.**
- **Define a means of distinguishing each object regardless of its form or history. Be alert to requirements that call for matching objects by attributes.** → ไปต่างที่ก็ระบุได้
- **Define an operation** (method) **that is guaranteed to produce a unique result for each object, possibly by attaching a symbol that is guaranteed unique. This means of identification may come from the outside, or it may be an arbitrary identifier created by and for the system, but it must correspond to the identity distinctions in the model.**
- **The model must define what it means to be the same thing.**

Ex พัสดุที่ addr. same
ก็ไม่ต้องแยกพัสดุว่าอันเป็นอันไหน
ส่งตามที่อยู่ก็พอ

Value Object (1/2)

(VO)

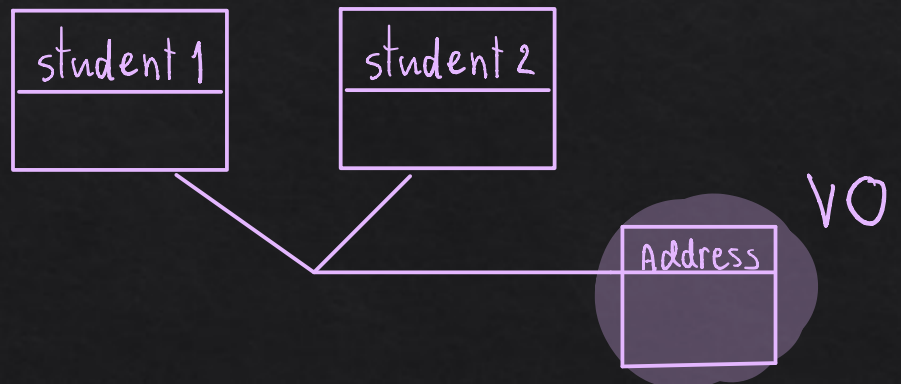
→ สนแค่ attribute ตอนใช้

→ obj ที่ไม่มี identity

คุณลักษณะ

- Some objects describe or compute some characteristic of a thing.
- Many objects have no conceptual identity.
- 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. Software design is a constant battle with complexity. We must make distinctions so that special handling is applied only where necessary.
- However, if we think of this category of object as just the absence of identity, we haven't added much to our toolbox or vocabulary. In fact, these objects have characteristics of their own, and their own significance to the model. These are the objects that describe things.

Ex



Value Object (2/2)

Therefore:

- **When you care only about the attributes and logic of an element of the model, classify it as a value object. Make it express the meaning of the attributes it conveys and give it related functionality.**
- **Treat the value object as immutable. Make all operations Side-effect-free Functions that don't depend on any mutable state.**
- **Don't give a value object any identity and avoid the design complexities necessary to maintain entities.**

Ex เที่ยวรอบก.เดินเรือ มีปัญหา

entity, aggregate สร้าง

Domain Events (1/3)

- Something happened that domain experts care about.
- An entity is responsible for tracking its state and the rules regulating its lifecycle. But if you need to know the actual causes of the state changes, this is typically not explicit, and it may be difficult to explain how the system got the way it is.

เก็บ state แต่บอกอะไรไม่ได้

- Change histories of entities can allow access to previous states, but ignores the meaning of those changes, so that any manipulation of the information is procedural, and often pushed out of the domain layer.
- A distinct, though related set of issues arises in distributed systems. The state of a distributed system cannot be kept completely consistent at all times. We keep the aggregates internally consistent at all times, while making other changes asynchronously.

แบบกระจาย เก็บ state ไม่ 100%

- As changes propagate across nodes of a network, it can be difficult to resolve multiple updates arriving out of order or from distinct sources.

Domain Events (2/3)

↳ เป็น class

Therefore:

- **Model information about activity in the domain as a series of discrete events.**
- **Represent each event as a domain object.** These are distinct from system events that reflect activity within the software itself, although often a system event is associated with a domain event, either as part of a response to the domain event or as a way of carrying information about the domain event into the system.

  ↗ แยกจาก system event

- A domain event is a full-fledged part of the domain model, a representation of something that happened in the domain.

  → แสดงถึงสิ่งที่เกิดกับ domain

- Ignore irrelevant domain activity while making explicit the events that the domain experts want to track or be notified of, or which are associated with state change in the other model objects.

Domain Events (3/3)

- In a distributed system, the state of an entity can be inferred from the domain events currently known to a particular node, allowing a coherent model in the absence of full information about the system as a whole.
- Domain events are ordinarily immutable, as they are a record of something in the past.
- In addition to a description of the event, a domain event typically contains a timestamp for the time the event occurred and the identity of entities involved in the event.
- Also, a domain event often has a separate timestamp indicating when the event was entered into the system and the identity of the person who entered it. When useful, an identity for the domain event can be based on some set of these properties. So, for example, if two instances of the same event arrive at a node they can be recognized as the same.

Services บริการ

- Sometimes, it just isn't a thing.
- Some concepts from the domain aren't natural to model as objects. Forcing the required domain functionality to be the responsibility of an entity or value either distorts the definition of a model-based object or adds meaningless artificial objects. → ให้ entity

Therefore:

- **When a significant process or transformation in the domain is not a natural responsibility of an entity or value object, add an operation to the model as a standalone interface declared as a service.**
  ↳ class เดี่ยวๆ
  ติดต่อยังไง (for เรียกใช้) API
- **Define a service contract, a set of assertions about interactions with the service. (See assertions.) State these assertions in the ubiquitous language of a specific bounded context.**
- **Give the service a name, which also becomes part of the ubiquitous language.**

service ใน domain layer → domain service

- container ของ class

Modules (1/2)

- Everyone uses modules, but few treat them as a full-fledged part of the model. Code gets broken down into all sorts of categories, from aspects of the technical architecture to developers' work assignments. Even developers who refactor a lot tend to content themselves with modules conceived early in the project. for แบ่งงาน
- Explanations of coupling and cohesion tend to make them sound like technical metrics, to be judged mechanically based on the distributions of associations and interactions. Yet it isn't just code being divided into modules, but also concepts.
- There is a limit to how many things a person can think about at once (hence low coupling). Incoherent (ไม่เกี่ยวกัน) fragments of ideas are as hard to understand as an undifferentiated soup of ideas (hence high cohesion).

Modules (2/2)

Therefore:

- Choose modules that tell the story of the system and contain a cohesive set of concepts.
- Give the modules names that become part of the ubiquitous language. Modules are part of the model and their names should reflect insight into the domain.
- This often yields low coupling between modules, but if it doesn't look for a way to change the model to disentangle the concepts, or an overlooked concept that might be the basis of a module that would bring the elements together in a meaningful way.
- Seek low coupling in the sense of concepts that can be understood and reasoned about independently.
- **Refine the model until it partitions according to high-level domain concepts and the corresponding code is decoupled as well.**

ใน module เกี่ยวข้องกัน

domain exper เข้าใจ

สอดคล้อง

Ex กรุ๊ปเลือดคนไข้ (A B AB O) วันนึงเปลี่ยน AB → C ก็change ทั้งdatabase ∴ไม่ควรมีข้อมูลAB ถ้ามีนางอันฟินAB = inconsistency ก็เลยlock database จนแก้ข้อมูลเสร็จ (ไม่ให้คนอื่นใช้)

** Aggregates (1/3) **

- It is difficult to guarantee the consistency of changes to objects in a model with complex associations. ค.สัมพันธ์
- Objects are supposed to maintain their own internal consistent state, but they can be blindsided by changes in other objects that are conceptually constituent parts.
- Cautious database locking schemes cause multiple users to interfere pointlessly with each other and can make a system unusable.
- Similar issues arise when distributing objects among multiple servers, or designing asynchronous transactions.

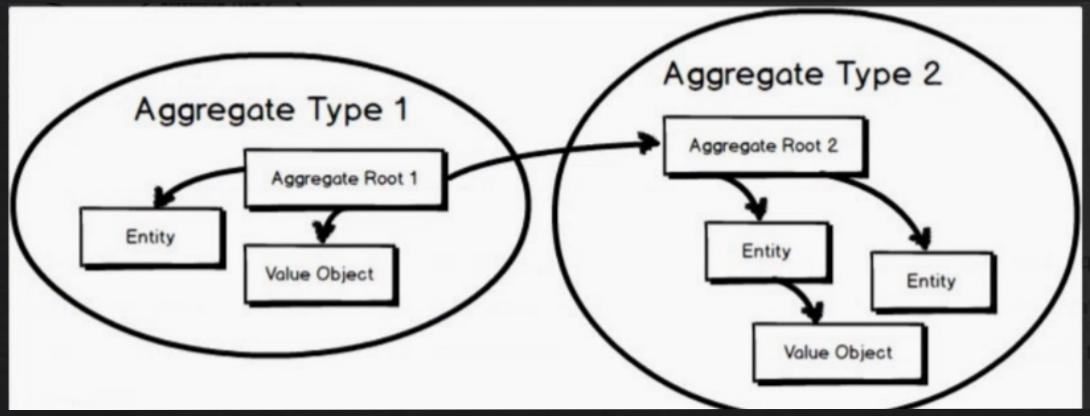
Aggregates (2/3)

กลุ่ม class เป็น tree
มี class root
ต้องอยู่ใน bounded context
no service
สร้าง domain event

Therefore:

- Cluster the entities and value objects into aggregates and define boundaries around each.
- Choose one entity to be the root of each aggregate, and allow external objects to hold references to the root only (references to internal members passed out for use within a single operation only).
- Define properties and invariants for the aggregate as a whole and give enforcement responsibility to the root or some designated framework mechanism.

กฎ



เพื่อรักษาconsistency

Aggregates (3/3)

สื่อสารข้าม bounded context ส่ง domain expert

- Use the same aggregate boundaries to govern transactions and distribution.
- Within an aggregate boundary, apply consistency rules synchronously. Across boundaries, handle updates asynchronously.

bounded context same

- Keep an aggregate together on one server. Allow different aggregates to be distributed among nodes.
- When these design decisions are not being guided well by the aggregate boundaries, reconsider the model. Is the domain scenario hinting at an important new insight?
- Such changes often improve the model's expressiveness and flexibility as well as resolving the transactional and distributional issues.

Repositories (1/3)

เส้น bet. Obj เยอะ ↑ ก.acess ข้อมูลก็ยากขึ้น

- Query access to aggregates expressed in the ubiquitous language.
- Proliferation of traversable associations used only for finding things muddles the model. In mature models, queries often express domain concepts. Yet queries can cause problems.

High growth rate

- The sheer technical complexity of applying most database access infrastructure quickly swamps the client code, which leads developers to dumb-down the domain layer, which makes the model irrelevant.

หลายtable เชื่อมกัน เพื่อให้สอดคล้องกับ Business req.

Ex หาว่าส่ง Bill ให้ลูกค้ากี่คน, ลูกค้าจ่ายเท่าไหร่, คิดเป็น % ต้องใช้ข้อมูลจากหลายส่วน

ถ้าให้เข้าถึง obj, database โดยตรง
เวลาเปลี่ยนก็ยาก

Repositories (2/3)

- A query framework may encapsulate most of that technical complexity, enabling developers to pull the exact data they need from the database in a more automated or declarative way, but that only solves part of the problem.
- Unconstrained queries may pull specific fields from objects, breaching encapsulation, or instantiate a few specific objects from the interior of an aggregate, blindsiding the aggregate root and making it impossible for these objects to enforce the rules of the domain model.
- Domain logic moves into queries and application layer code, and the entities and value objects become mere data containers.

Repositories (3/3)

ให้คนอื่นมาทำสิ่งต่างๆ ผ่านตรงนี้
เพิ่ม, ลด, ฯลฯ

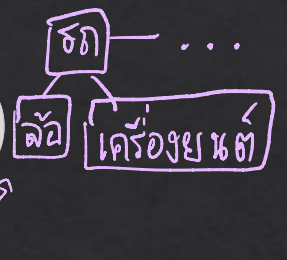
Therefore:

- **For each type of aggregate that needs global access, create a service that can provide the illusion of an in-memory collection of all objects of that aggregate's root type.**
- **Set up access through a well-known global interface.**
- **Provide methods to add and remove objects, which will encapsulate the actual insertion or removal of data in the data store.**
- **Provide methods that select objects based on criteria meaningful to domain experts.**
- **Return fully instantiated objects or collections of objects whose attribute values meet the criteria, thereby encapsulating the actual storage and query technology, or return proxies that give the illusion of fully instantiated aggregates in a lazy way.**
- **Provide repositories only for aggregate roots that actually need direct access.**
- **Keep application logic focused on the model, delegating all object storage and access to the repositories.**

Ex แทนที่จะมี class ล้อ, ส่วนอื่นๆ แล้วประกอบ เป็น class รถ เอง ก็มี ตัวสร้าง class รถ เลย

class, method

Factories (1/2)



- When creation of an entire, internally consistent aggregate, or a large value object, becomes complicated or reveals too much of the internal structure, factories provide encapsulation.
- Creation of an object can be a major operation in itself, but complex assembly operations do not fit the responsibility of the created objects.
- Combining such responsibilities can produce ungainly designs that are hard to understand.
- Making the client direct construction muddies the design of the client, breaches encapsulation of the assembled object or aggregate, and overly couples the client to the implementation of the created object.

Factories (2/2)

-ห่อหุ้มส่วนสร้างที่ซับซ้อน

Therefore:

- **Shift the responsibility for creating instances of complex objects and aggregates to a separate object, which may itself have no responsibility in the domain model but is still part of the domain design.**
- **Provide an interface that encapsulates all complex assembly and that does not require the client to reference the concrete classes of the objects being instantiated.**
- **Create an entire aggregate as a piece, enforcing its invariants.**
- **Create a complex value object as a piece, possibly after assembling the elements with a builder.**

Recommended Resources

- Domain Driven Design | DevIQ
- Domain-Driven Design: Entities, Value Objects, and How To Distinguish Them (jannikwempe.com)
- DDD Sample Application - Screencast (sourceforge.net)
- GitHub - citerus/dddsample-core: This is the new home of the original DDD Sample app (previously hosted at sf.net)..
- DDD Tutorial: Domain-Driven Design in Practice | Pluralsight
- GitHub - ddd-by-examples/library: A comprehensive Domain-Driven Design example with problem space strategic analysis and various tactical patterns.
- Designing a microservice domain model | Microsoft Docs