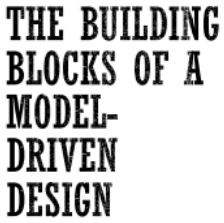
Chapter 03 Domain-Driven Design

Ubiquitous Language

ภาษาที่เป็นสากล >> ภาษาที่ developer และ user ใช้แล้วคุยกันรู้เรื่อง

Model-Driven Design



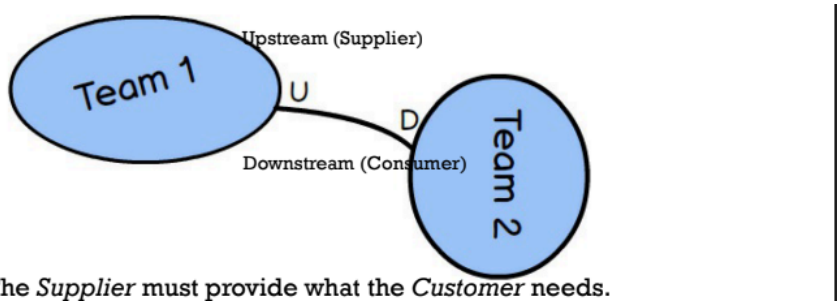
แบบจำลองที่ทำหน้าที่ represent แต่ละ domain ในระบบ

โดยในแต่ละ **Model** (แบบจำลอง) มีการแตกย่อยเป็นส่วน ๆ เรียกว่า **Building Blocks** โดยมี **Layer Architecture** ซึ่งเป็นสถาปัตยกรรมที่จะมาเชื่อมต่อ

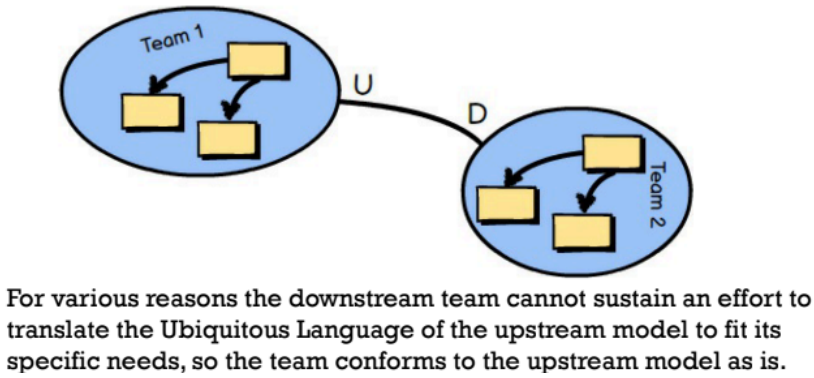
1. **Entity**
2. **Value Objects >>** ข้อมูลอื่น ๆ ที่อยู่ใน domain หรือระหว่าง domain นั้น ๆ ซึ่งเป็นข้อมูลที่ไม่จำเป็นต้องเป็น Entity หมายความว่าไม่จำเป็นต้องเก็บไว้ในระบบ เกิดมาแล้วก็หายไปเมื่อจบ transaction
3. **Aggregates** >> Entity ใด ๆ ที่เป็น superclass ของ Entities อื่น ๆ
4. **Services** >> สื่อถึง behavior ของ domain ทำหน้าที่ grouping และเป็น interface ให้กับ functions ต่าง ๆ
5. **Factories** >> ใช้ในการระบุ knowledge ทั้งหมดที่จำเป็นในการสร้าง object ใด ๆ
   - เช่น ตัวอย่างจะแสดง RouteFactory ซึ่งบอก function, parameter และ return ต่าง ๆ ที่เกิดขึ้นในการสร้าง Route object 1 ครั้ง
6. **Repositories** >> ตัวกลางระหว่าง Client และ Database ทำหน้าที่ในการ encapsulate logic และ transform เพื่อไปขอข้อมูลมาจาก database ให้กับ client
   - ทำหน้าที่เหมือน business logic (backend API)

Bounded Context

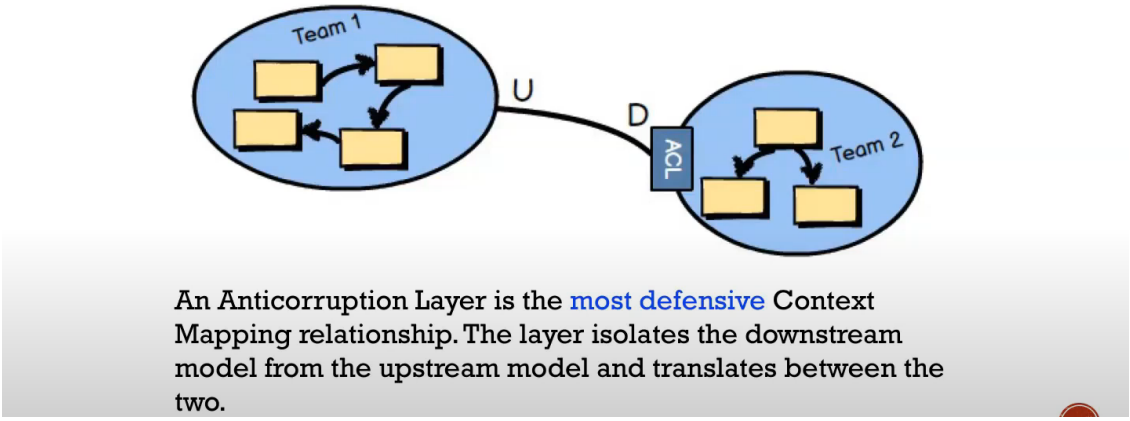
- **Bounded Context** >> boundary ของ domain นั้น ๆ (มีประโยชน์ในการออกแบบ architecture ของระบบ กล่าวคือ แต่ละทีมที่ทำงานคนละอย่างกัน จะได้ไม่ design ทับกัน)
- **Context Map** >> รูปแบบการพูดคุยที่เป็นไปได้ระหว่าง Bounded contexts ใด ๆ (ซึ่งมีหลายรูปแบบ)
  1. **Shared Kernel** >> มี attribute บางตัวทำหน้าที่เป็น key ทำให้สามารถแชร์ข้อมูลระหว่างกันได้
  2. **Customer - Supplier** >> domain นึงเป็น supplier ทำหน้าที่ส่งข้อมูลให้อีก domain (supplier จะต้องให้ข้อมูลที่ customer ต้องการ)



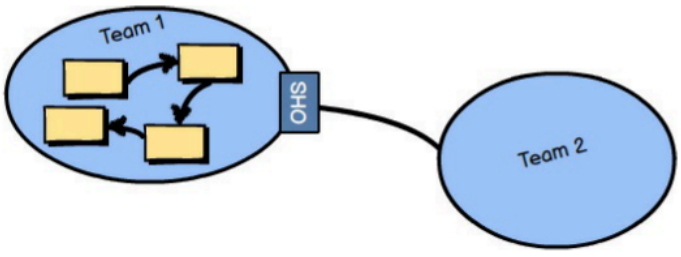
3. **Conformist** >> ข้อมูลที่ supplier (upstream) จะให้ consumer (downstream) นั้นมีรูปแบบที่ชัดเจน (consumer ต้องยอมรับ และนำไปหาวิธีใช้งานต่อเอง)



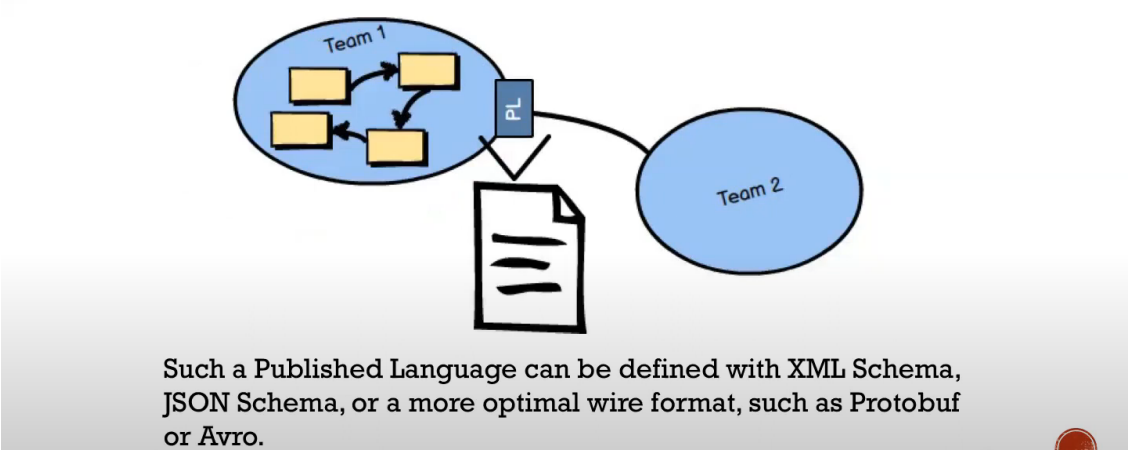
4. **Anticorruption Layer** >> มีการเพิ่ม **ACL** เข้าไปที่ consumer (downstream) เพื่อทำการป้องกันไม่ให้เกิด failure
   - ตัวอย่างที่เห็นภาพง่าย ๆ น่าจะเป็นเรื่องของการทำ data/datatype validation ก่อนจะเริ่มทำอะไรสักอย่าง



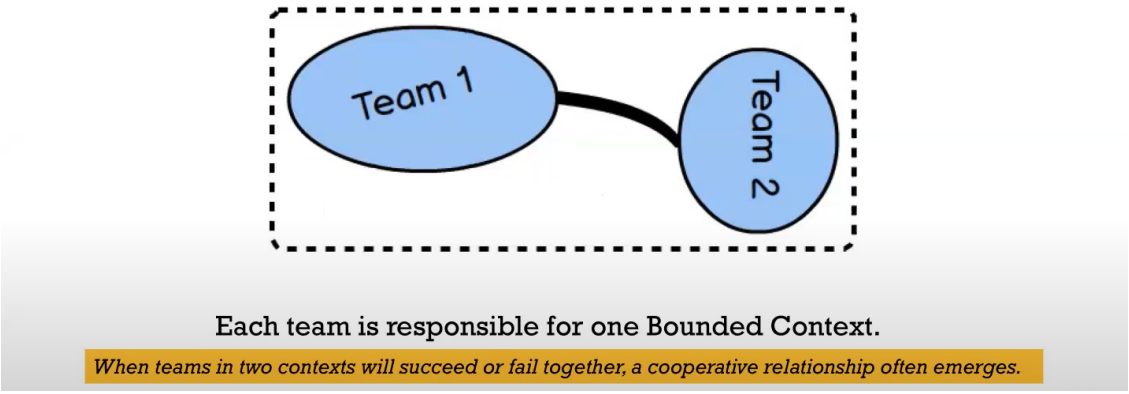
5. **Open Host Service** >> supplier (upstream) มีการทำ **OHS** ทำให้หาก consumer (downstream) ต้องการข้อมูลจาก supplier แล้วจะต้องส่ง request มาในรูปแบบตามที่ protocol หรือ interface ของ supplier กำหนดเท่านั้น (RESTful)



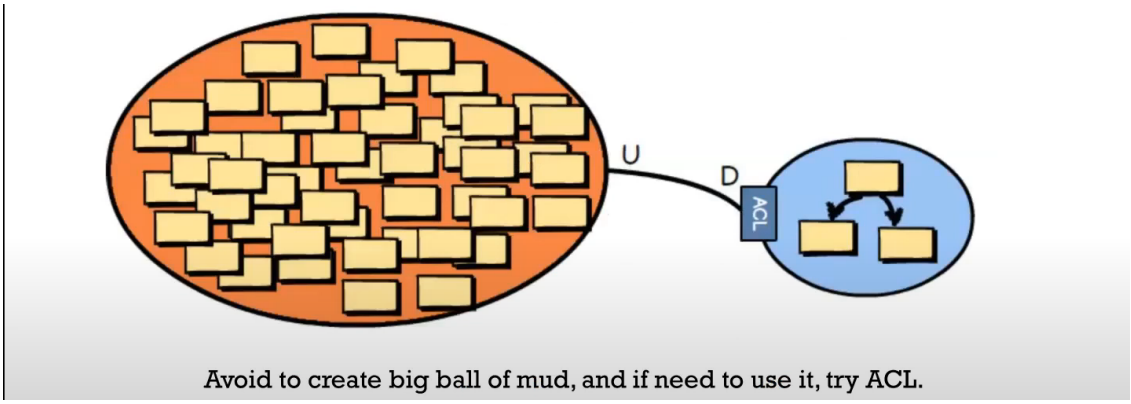
6. **Published Language** >> เป็นรูปแบบที่ more specific ของ OHS บอกว่า interface อาจอยู่ในรูปของ XML/JSON Schema (gRPC)



7. **Separate Ways** >> ทั้งสอง boundary context คุยกันไม่รู้เรื่อง
8. **Partnership** >> ไม่เชิง share kernel แต่ถ้าอันใดอันหนึ่งพัง อีกอันจะพังด้วย
    - เช่นกรณี input ของตัวนึงทำหน้าที่ triggered หรือเกี่ยวข้องกับการทำงานของอีกตัวนึง



9. **Big Ball of Mud** >> มีก้อน legacy application อยู่ แล้วอยากจะ implement เพิ่มให้ทำเป็นแบบ ACL



God Class

แบ่งเป็น microservices ด้วย **business capabilities** ทำให้เกิด God class กล่าวคือ class มีขนาดใหญ่และมี communication ระหว่าง domain เยอะมาก ๆ