Chapter 3 Decomposition Strategy

Three-step process

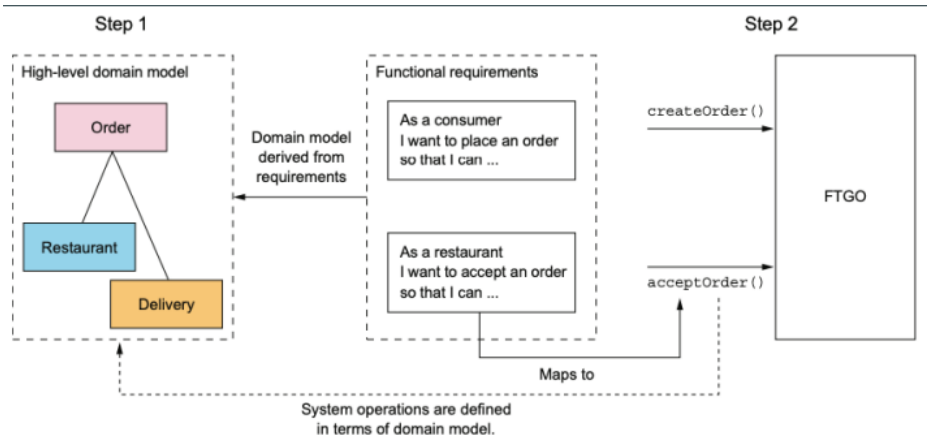
Step 1: Identify System Operations

ต้องการระบุว่าระบบทำอะไรบ้าง

- **Functional Requirement** >> user stories
- **System Operation** >> function

Steps

1. Create a **high-level domain model** >> นำ functional requirements มาเขียนเป็น high-level domain model
2. Define the **system operations** >> นำแต่ละ functional requirements มาแปลงเป็น system operation (function)



Step 2: Identify Services

ตัดสินใจ ว่าจะแตก FR ออกเป็น service ยังไง จะแตกออกเป็นกี่ก้อน (มี strategy ที่ช่วยให้แบ่งได้หลายอย่าง)

> 💡 จะแบ่งออกเป็น Service **ควรแบ่งด้วย Business capabilities (Top-down)** ไม่ใช่ **Technical capabilities** (Bottom-up)

(2.1) Business Capabilities

Business Capabilities เป็น strategy ที่แบ่งโดยคำนึงถึง Business process เช่น app ร้านค้าออนไลน์มี order/inventory management, shipping และอื่น ๆ

- **Capability** สามารแบ่งออกเป็น **Sub-capability** ได้ เช่น Claim management รวม Claim information management และ Claim review
- ข้อดี
  - Architecture ที่ออกแบบมาจะค่อนข้าง stable เพราะ business ของ application นั้น ๆ จะเหมือนเดิม
- ข้อเสีย
  - มี communication ระหว่าง service ค่อนข้างเยอะ (Excessive)
  - Service มีความซับซ้อน ในอนาคตจะแยกยาก **(God class)**

(2.2) Domain-Driven Design

DDD เป็น strategy ที่แบ่งโดยใช้ **object-oriented domain model** เป็นศูนย์กลาง

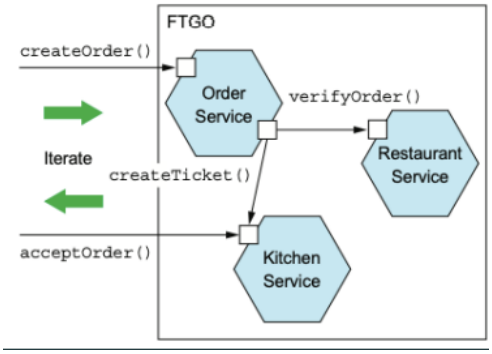
- มี 2 concepts ที่สำคัญ
  1. **Sub-domain** >> แต่ละ domain แตกเป็น sub-domain ได้ (ต้องใช้ Business domain หรือคนที่เชี่ยวชาญมา identify)
  2. **Bounded context** >> scope of domain model

Obstacles to decomposing an application into services

การเปลี่ยนไปใช้ Microservices ต้องเจอข้อเสียเหล่านี้แน่นอน

1. **Network Latency** >> request ระหว่าง service ทำให้เกิด delay
2. **Reduced Availability** (due to synchronous communication) >> เพราะ synchronous อาจทำให้เกิด deadlock ได้
3. **Maintain data consistency** (across services) >> เห็นภาพง่าย ๆ เช่น transaction ระหว่างธนาคาร ที่ธนาคารนึงโอนแล้วแต่อีกธนาคารล่มอยู่
4. **Obtaining a consistent view of the data**
5. **God classes preventing decomposition** >> class ที่แตกได้ยากมาก ๆ อาจเป็นเพราะเป็นจุดศูนย์กลางของธุรกิจ (แต่ละ business มักมี god class อย่างน้อย 1 อันเสมอ)

Step 3: Define service APIs and collaboration



ดูว่า **Subsystem** แต่ละกันมีความสัมพันธ์กันอย่างไร ส่ง message อย่างไร (API & Collaboration)

Steps

Service	Operations	Collaborators
Consumer Service	verifyConsumerDetails()	—
Order Service	createOrder()	▪ Consumer Service verifyConsumerDetails() ▪ Restaurant Service verifyOrderDetails() ▪ Kitchen Service createTicket() ▪ Accounting Service authorizeCard()
Restaurant Service	▪ findAvailableRestaurants() ▪ verifyOrderDetails()	—
Kitchen Service	▪ createTicket() ▪ acceptOrder() ▪ noteOrderReadyForPickup()	▪ Delivery Service scheduleDelivery()
Delivery Service	▪ scheduleDelivery() ▪ noteUpdatedLocation() ▪ noteDeliveryPickedUp() ▪ noteDeliveryDelivered()	—
Accounting Service	▪ authorizeCard()	—

1. **Map** แต่ละ service operation กับ service
2. **Determine API** สำหรับ collaboration กับ service อื่น