領域驅動設計下的持續架構

講者：
Arthur

大綱

- 領域驅動設計的概念
- 架構
- 持續架構實務

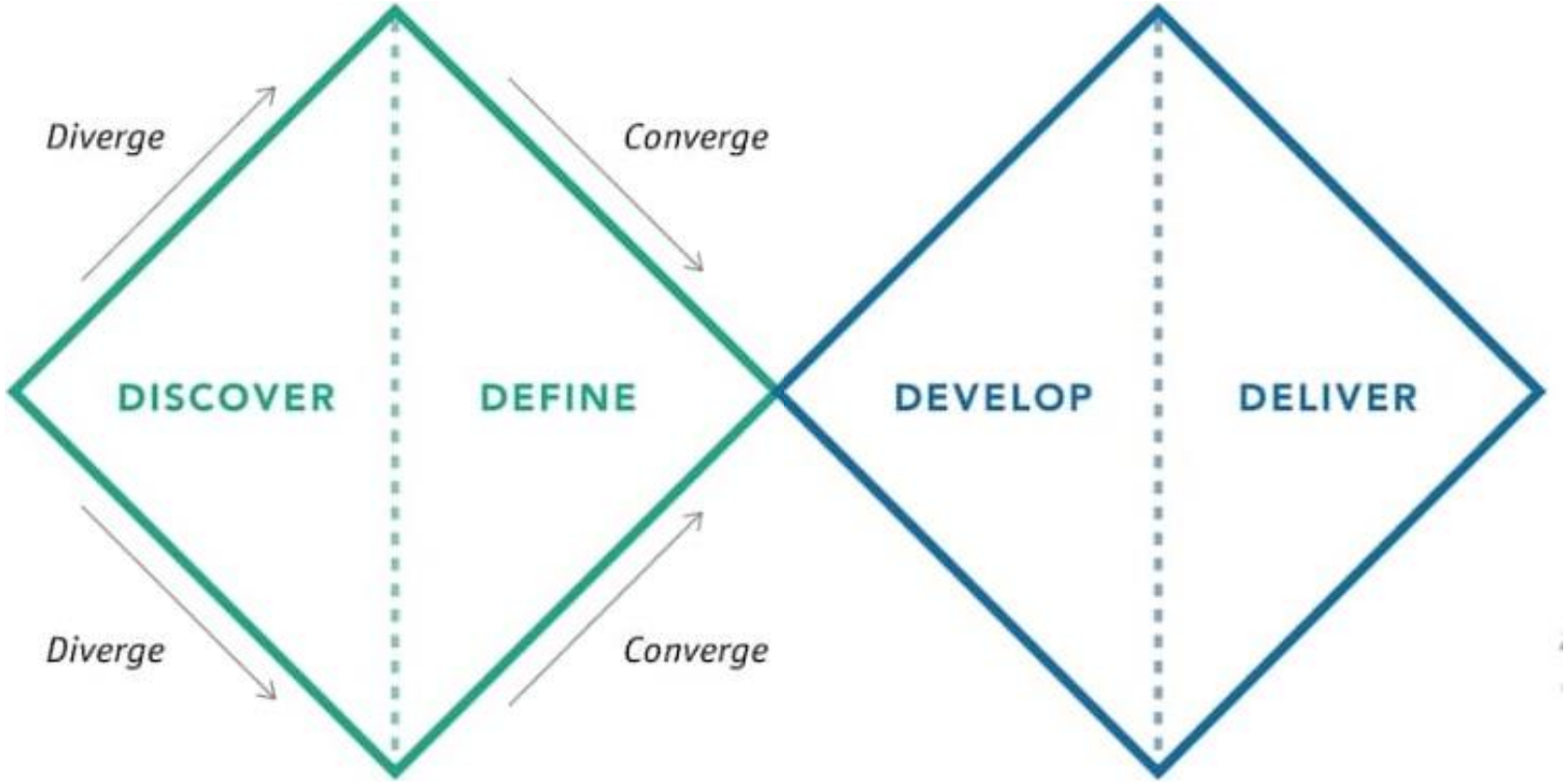
講師自介

Arthur

- 軟體業從業多年
- 歷經多種職位
- DDDTW社群 創辦人之一

FB粉絲團

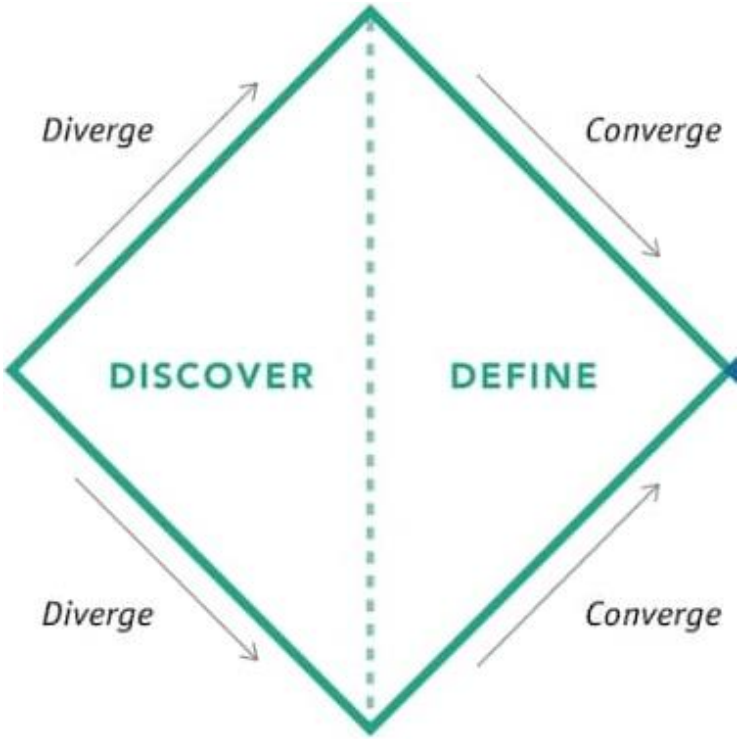
設計的運作模型 – 雙菱形模型



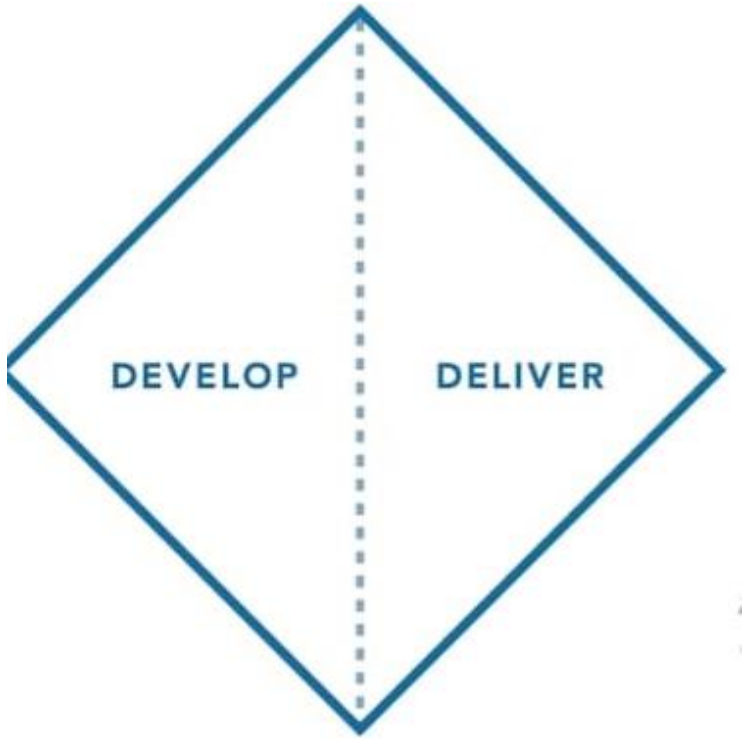
<圖源：nngroup>

雙菱形模型的結構

問題的探索+歸納

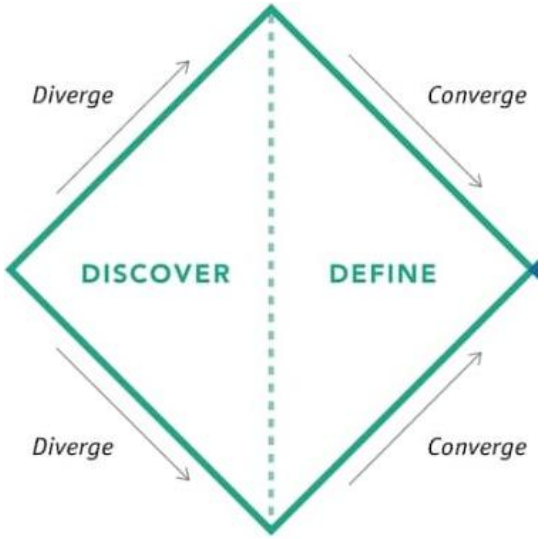


解方的發想+精煉



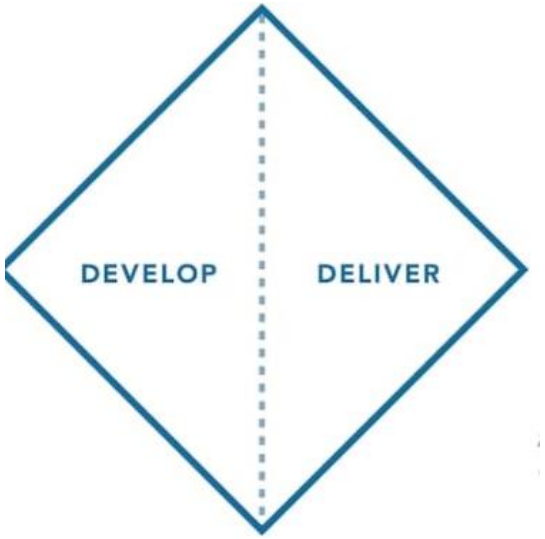
問題域與解決方案域

問題的探索+歸納



經歷問題的探索和歸納之後將成功視為一個集合

解方的發想+精煉



對應問題的集合,解決方案也需要有對應的解決方案集合

問題域與解決方案域

問題的**探索**+**歸納**

問題空間

連結兩個空間的**橋樑**

領域驅動設計

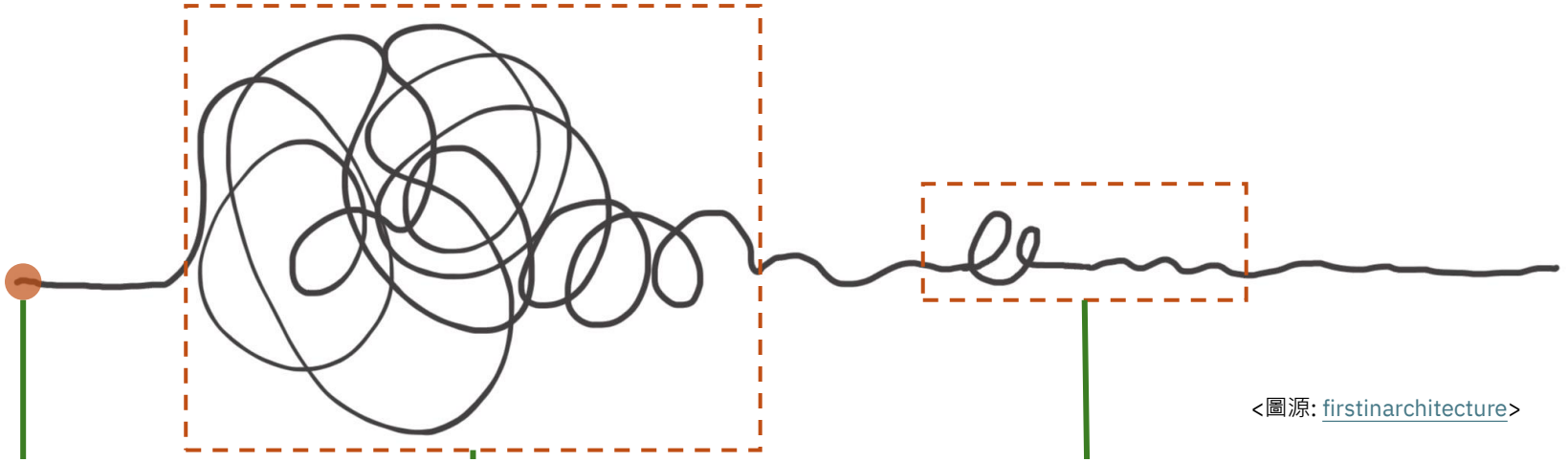
解方的**發想**+**精煉**

解決方案空間

經歷問題的探索和歸納之後將成功視為一個集合

對應問題的集合,解決方案也需要有對應的解決方案集合

產品/專案啟動



<圖源: firstinarchitecture>

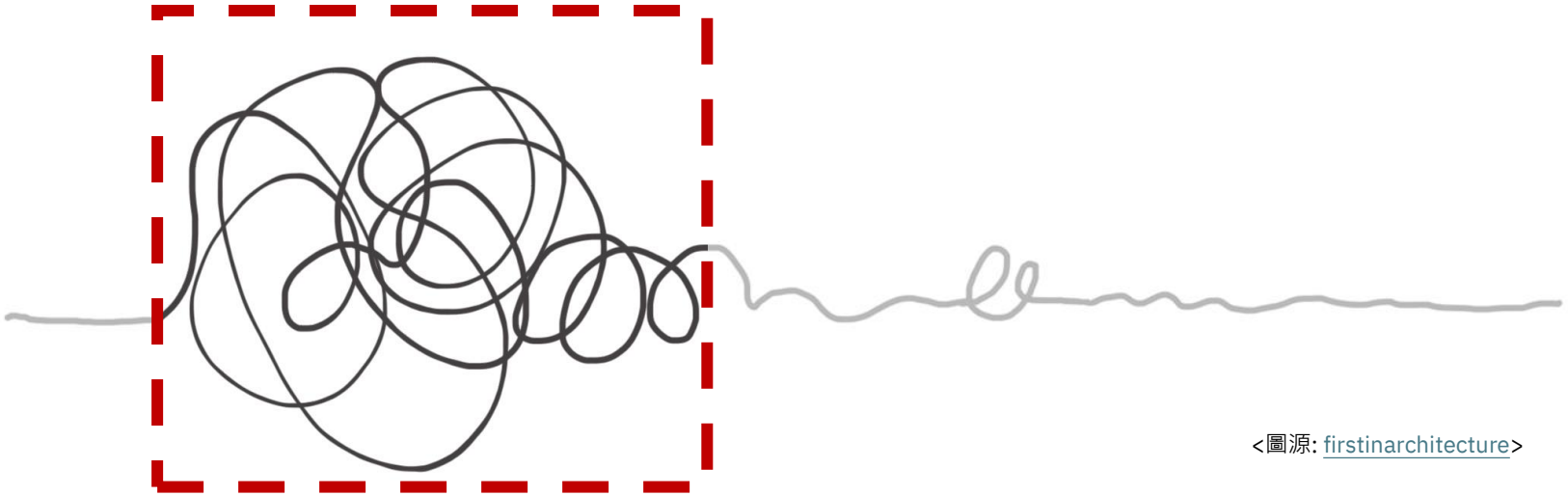
產品開發是任何產品驅動型公司的主要驅動力。從構思、設計到最終執行，一款出色的新產品可協助公司在競爭中領先一步，佔領新市場並增加利潤。

創意
起點

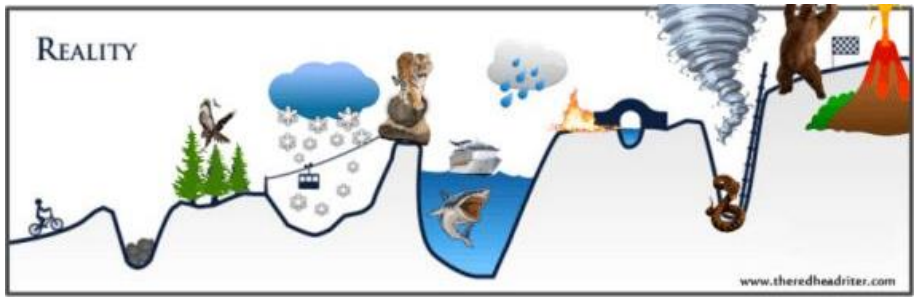
掙扎/
探索

團隊
對齊

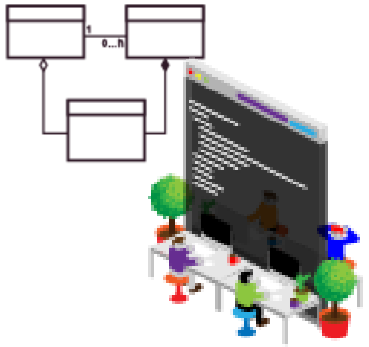
探索期間



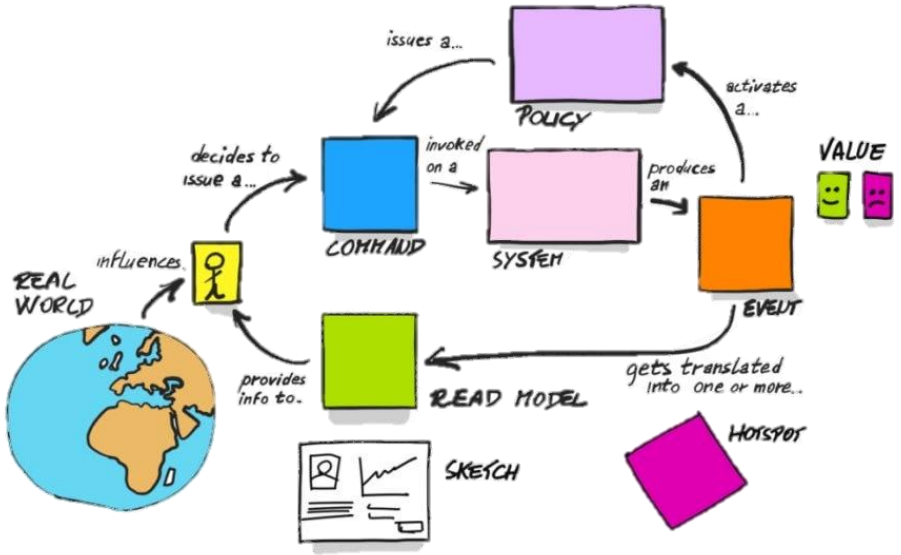
<圖源: firstinarchitecture>



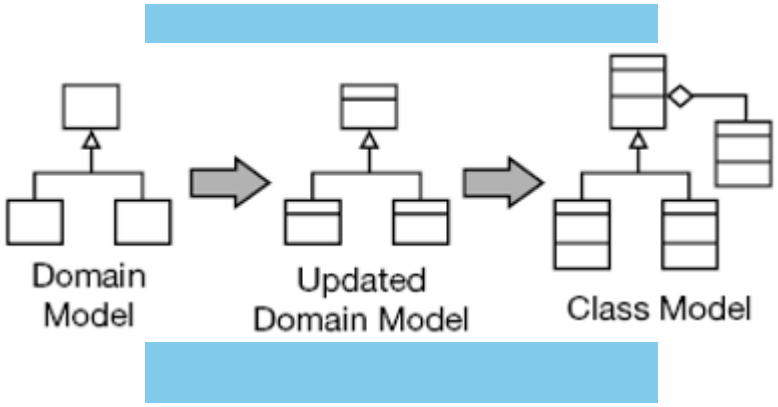
學習的方法 – 領域塑模



領域塑模



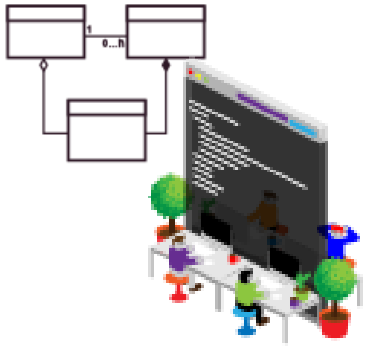
流程塑模



領域物件塑模

切割龐大的領域塑模

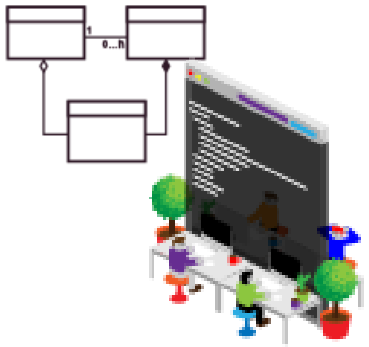
進行事件風暴的過程中，團隊會逐漸對齊(Alignment)領域知識。為了避免太過龐大的知識讓團隊難以消化，拆解為多個子領域有助於幫助團隊更加聚焦。



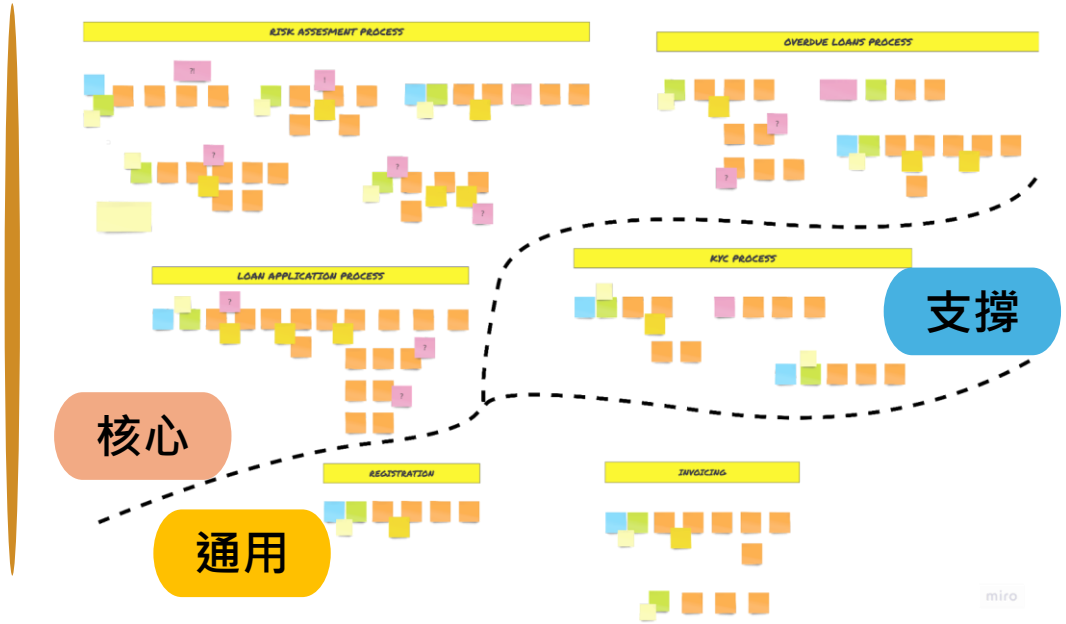
領域塑模

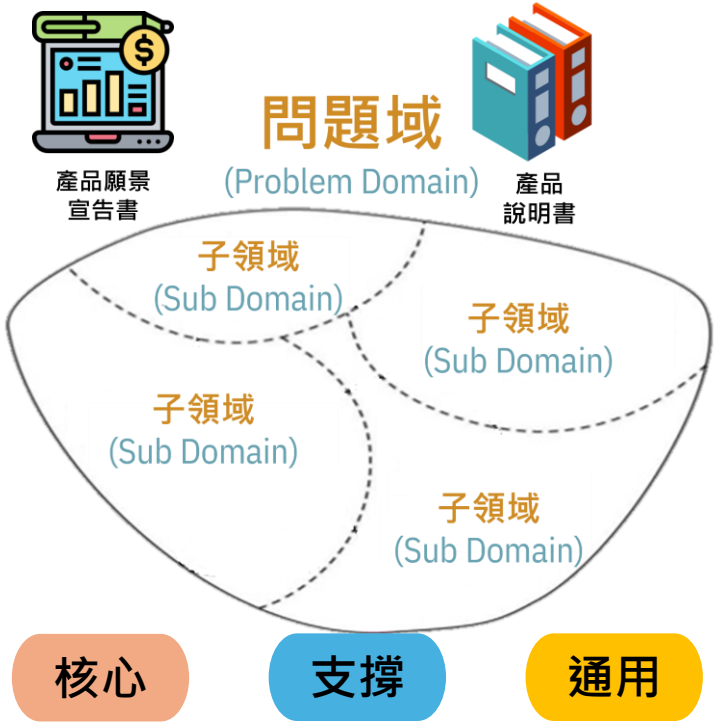
切割的成果 – 子領域

進行事件風暴的過程中，團隊會逐漸對齊(Alignment)領域知識。為了避免太過龐大的知識讓團隊難以消化，拆解為多個子領域有助於幫助團隊更加聚焦。



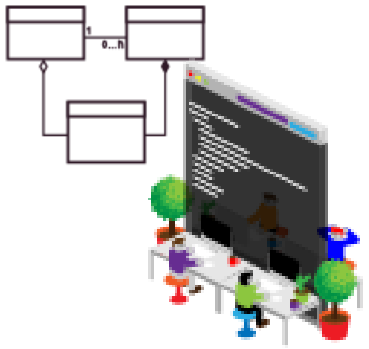
領域塑模



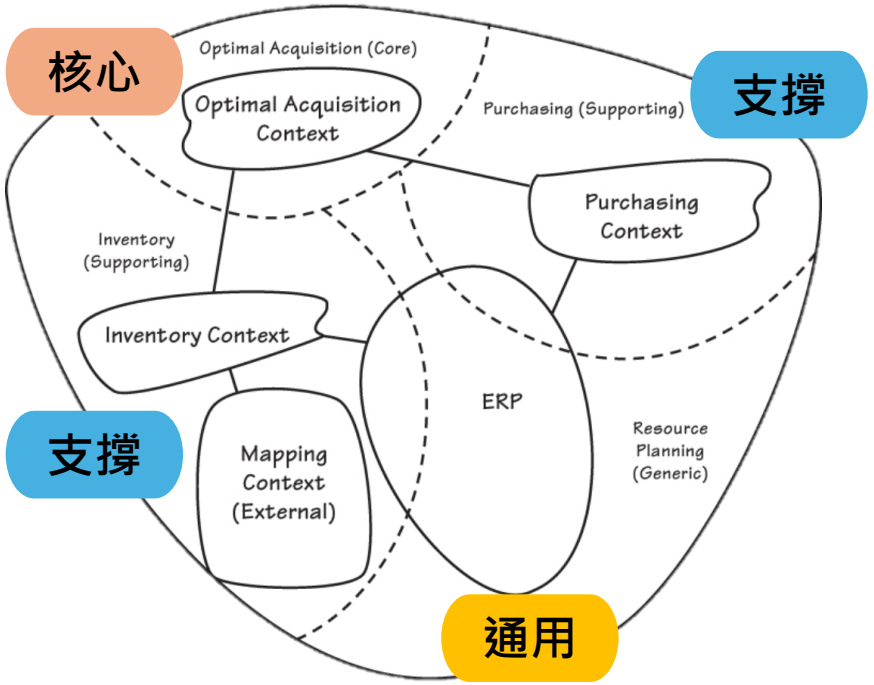


找出限界上下文

從核心子領域開始，收集和彙整領域故事就能夠發現，某些領域故事聚集再一起似乎是可以被歸類成某一個特定主題。將這個做法擴散到支撐子領域和通用子領域，就能夠逐漸讓所有限界上下文浮現出來。

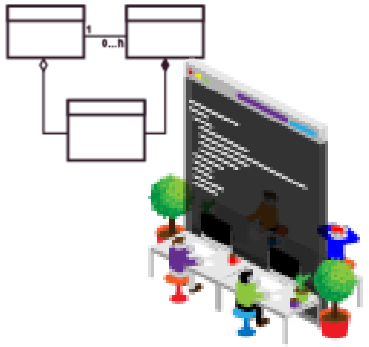


領域塑模

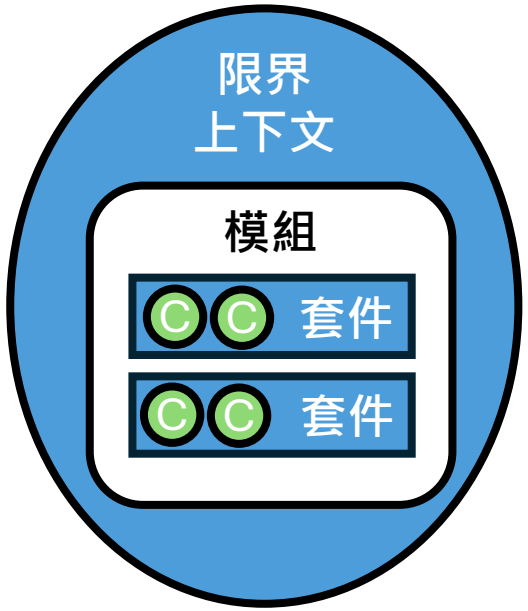


映射為模組

藉由找出的限界上下文，可以逐一映射成為一個個的模組。

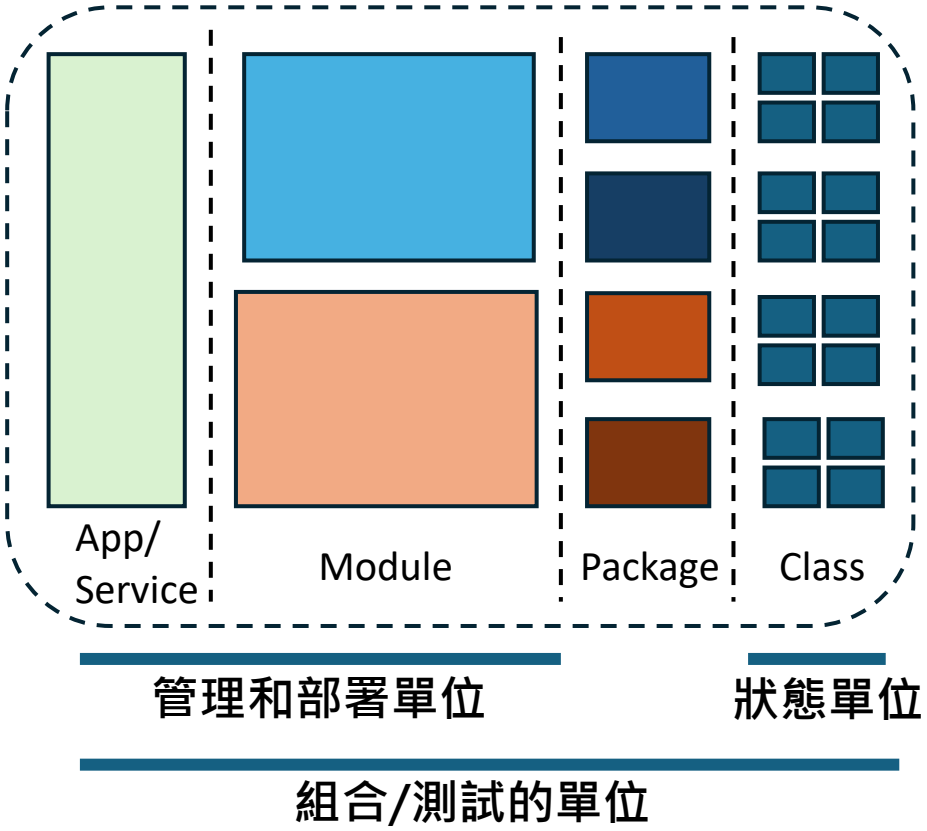


領域塑模



模組的特性

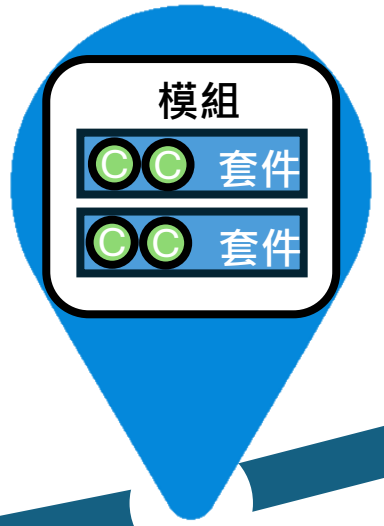
- 可部署
- 可管理
- 可測試
- 天然可重用
- 具可組合性
- 無狀態

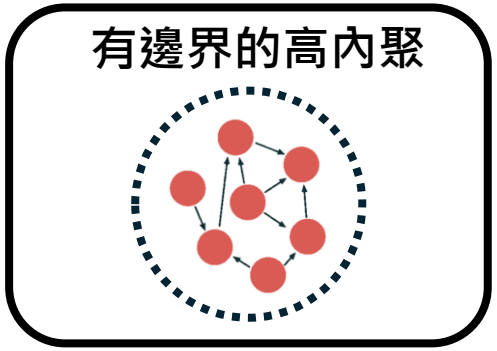


領域驅動的模組特質

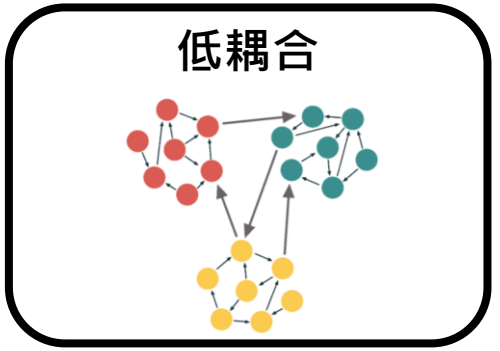
團隊之間使用相同的語言進行溝通，降低溝通的成本。

高內聚來自於清晰的語言脈絡，並且有明確的邊界。

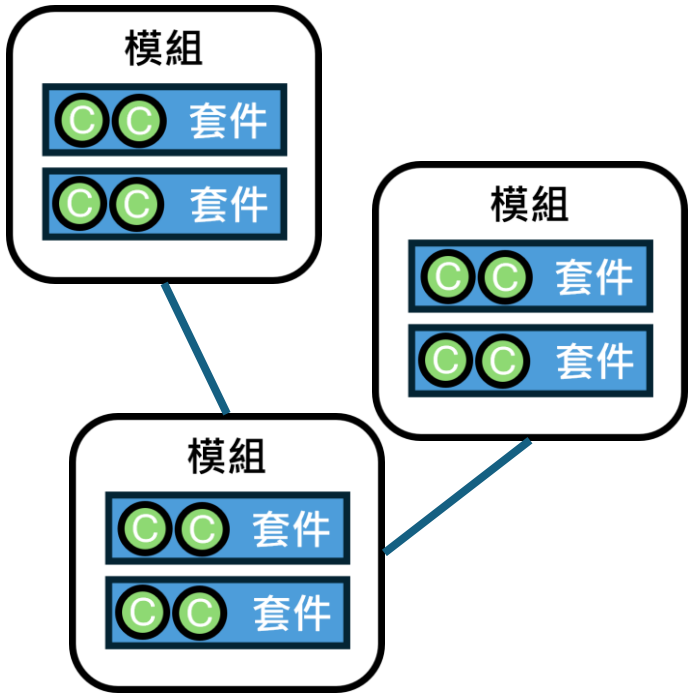




為避免誤解，每個模組內的概念都是獨立，不會有混淆的概念。

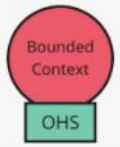


上下文地圖



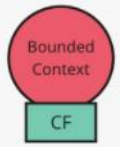
Open / Host Service

A Bounded Context offers a defined set of services that expose functionality for other systems. Any downstream system can then implement their own integration. This is especially useful for integration requirements with many other systems. Example: public APIs.



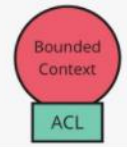
Conformist

The downstream team conforms to the model of the upstream team. There is no translation of models. Couples the Conformist's domain model to another bounded context's model.



Anticorruption Layer

The anticorruption layer is a layer that isolates a client's model from another system's model by translation. Only couples the integration layer (or adapter) to another bounded context's model but not the domain model itself.

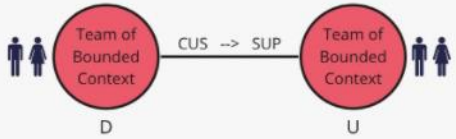


Shared Kernel

Two teams share a subset of the domain model including code and maybe the database. Typical examples: shared JARs, DLLs or a shared database schema. Teams with a Shared Kernel are often mutually dependent and should form a Partnership.

Customer / Supplier

There is a customer / supplier relationship between teams. The downstream team is considered to be the customer. Downstream requirements factor into upstream planning. Therefore, the downstream team gains some influence over the priorities and tasks of the upstream team.



Partnership

Partnership is a cooperative relationship between two teams. These teams establish a process for coordinated planning of development and joint management of integration.

Published Language

A Published Language is a well documented shared language between Bounded Contexts which can translate in and out from that language. Published Language is often combined with Open Host Service.Typical examples are iCalendar or vCard.

Separate Ways

Bounded Contexts and their corresponding teams have no connections because integration is sometimes too expensive or it takes very long to implement. The teams chose to go separate ways in order to focus on their specific solutions.

Big Ball Of Mud

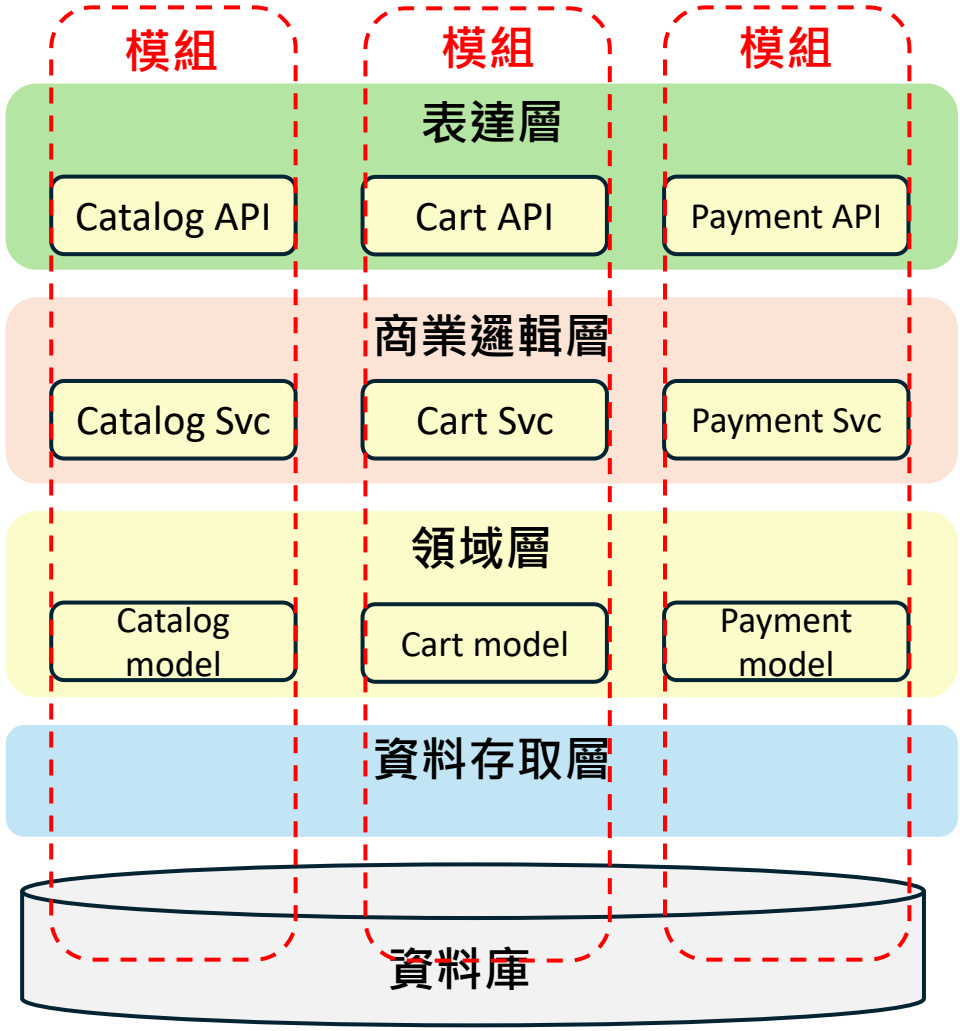
A (part of a) system which is a mess by having mixed models and inconsistent boundaries. Don't let this lousy model propagate into the other Bounded Contexts. Big Ball Of Mud is a demarcation of a bad model or system quality.

Recap

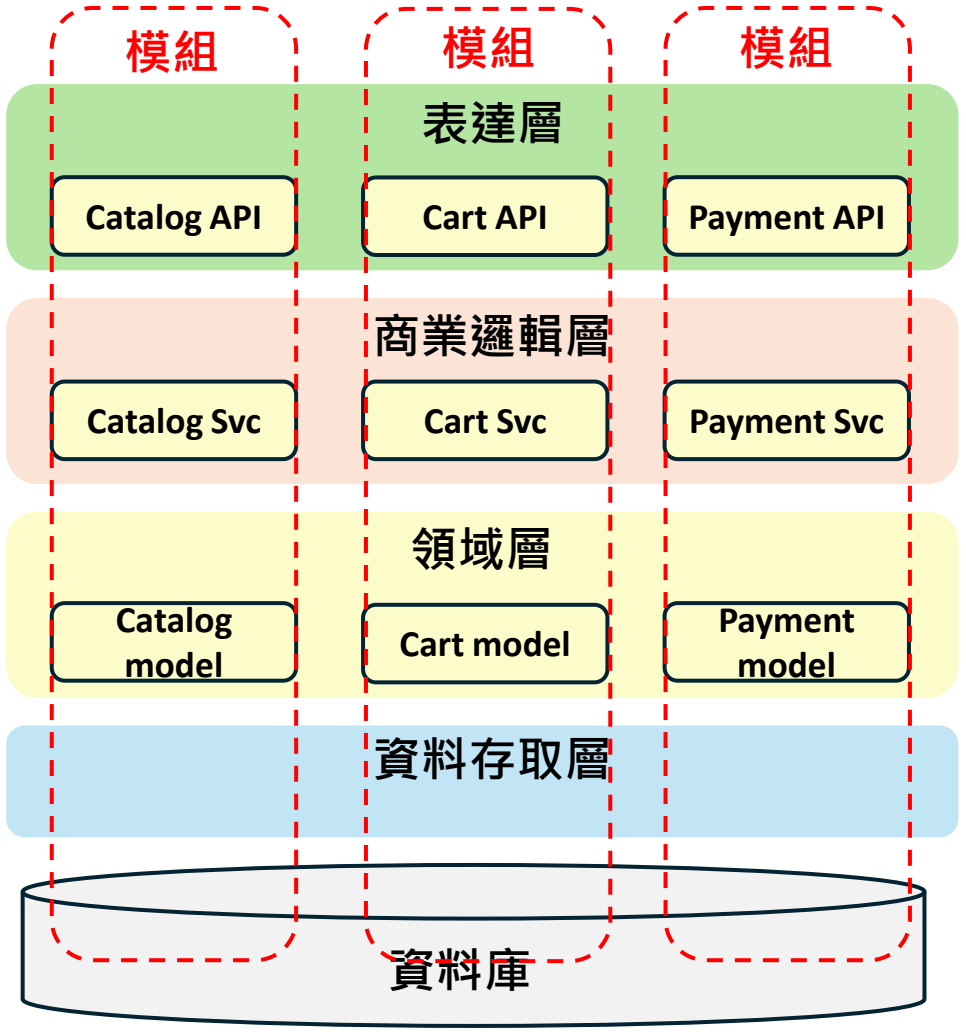
- 運用領域塑模降低開發團隊的學習負擔
  - 領域知識可以被切割成為多個子領域
  - 為了釐清概念，採用限界上下文捕捉概念的邊界和脈絡
- 限界上下文可以映射為模組，讓模組擁有限界上下文的優點
  - 溝通沒有歧異 – 統一語言
  - 有邊界的高內聚
  - 模組之間彼此低耦合
- 上下文地圖讓模組之間的關係更清晰，讓團隊具有系統全局觀

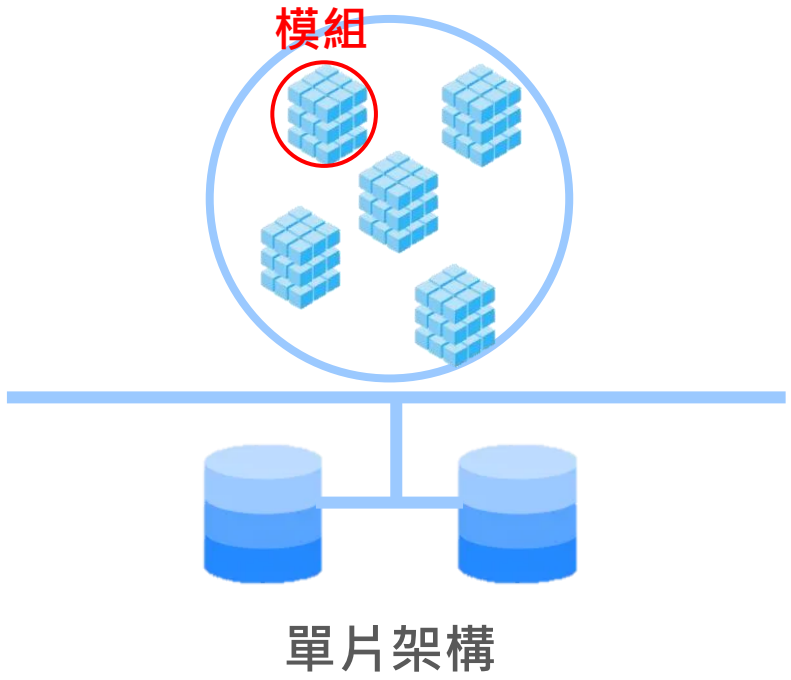
架構

分層架構與模組

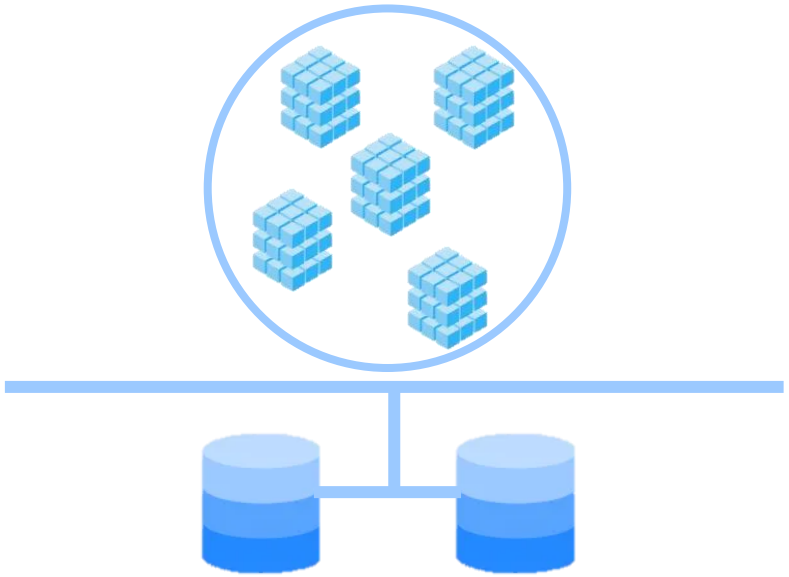

模組
模組
模組
表達層
Catalog API
Cart API
Payment API
商業邏輯層
Catalog Svc
Cart Svc
Payment Svc
領域層
Catalog model
Cart model
Payment model
資料存取層
資料庫

單片架構

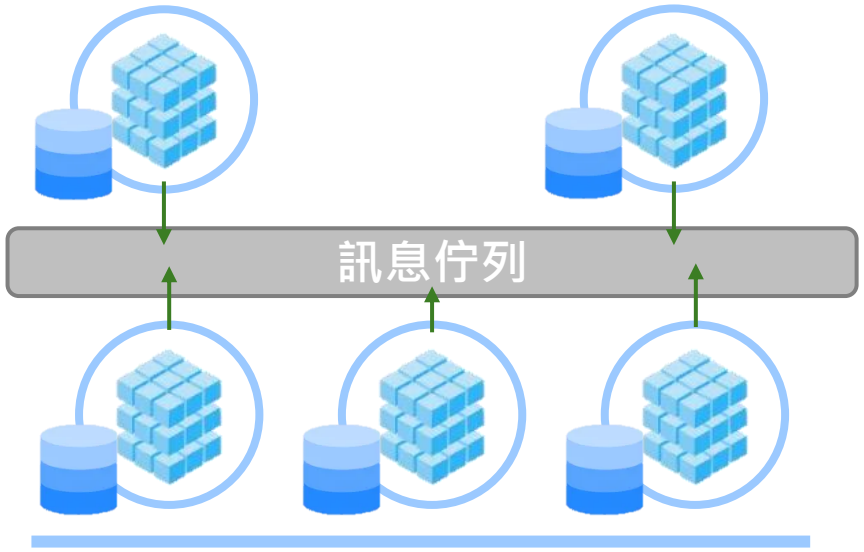

模組
模組
模組
表達層
Catalog API
Cart API
Payment API
商業邏輯層
Catalog Svc
Cart Svc
Payment Svc
領域層
Catalog model
Cart model
Payment model
資料存取層
資料庫


模組
單片架構

單片架構與微服務架構



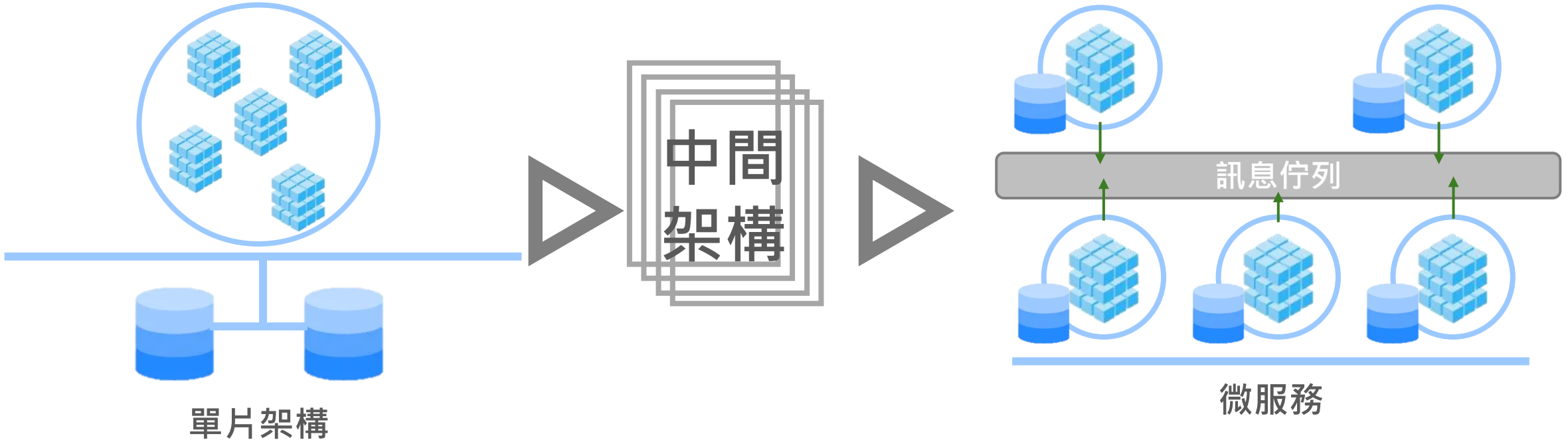
單片架構



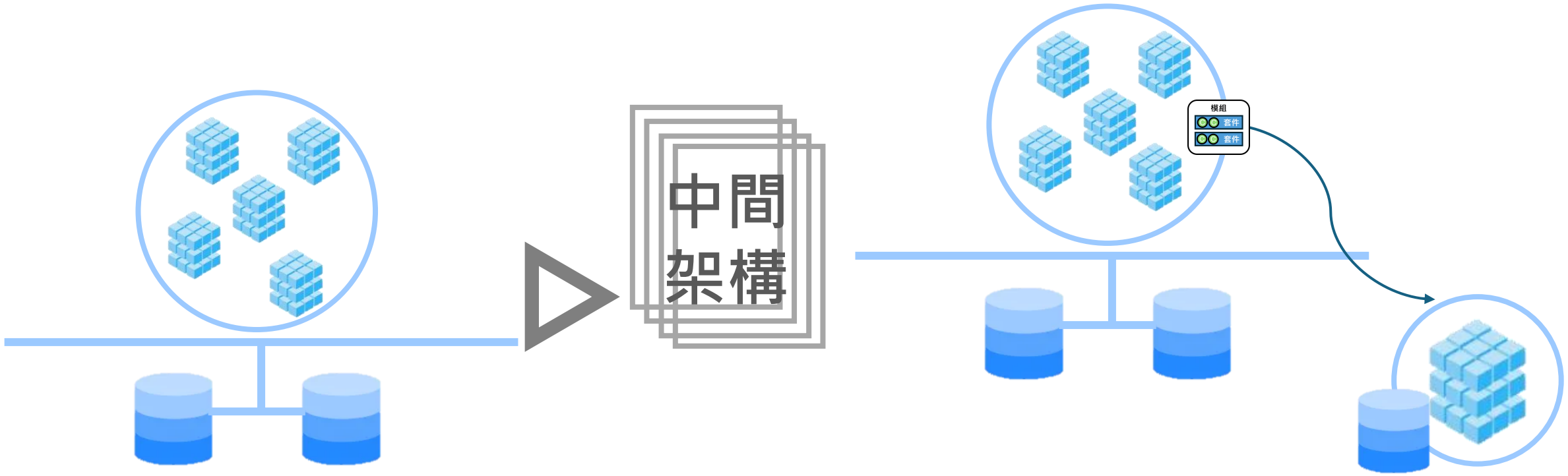
微服務

不同的部署思維

架構移轉的策略



架構移轉的策略-模組化單片架構



單片架構

單片架構+部分微服務


是什麼驅動著架構變化？

常見的驅動項目

商業環境

隨著商業環境的變化，系統不得不更新，讓系統可以幫助商業成功。

重工

對於領域知識的落差，造成理解的問題，導致需要重新開發部分系統。

系統缺陷

系統已知的缺陷已經造成商業或是開發方面損害，若不修改，損失持續。

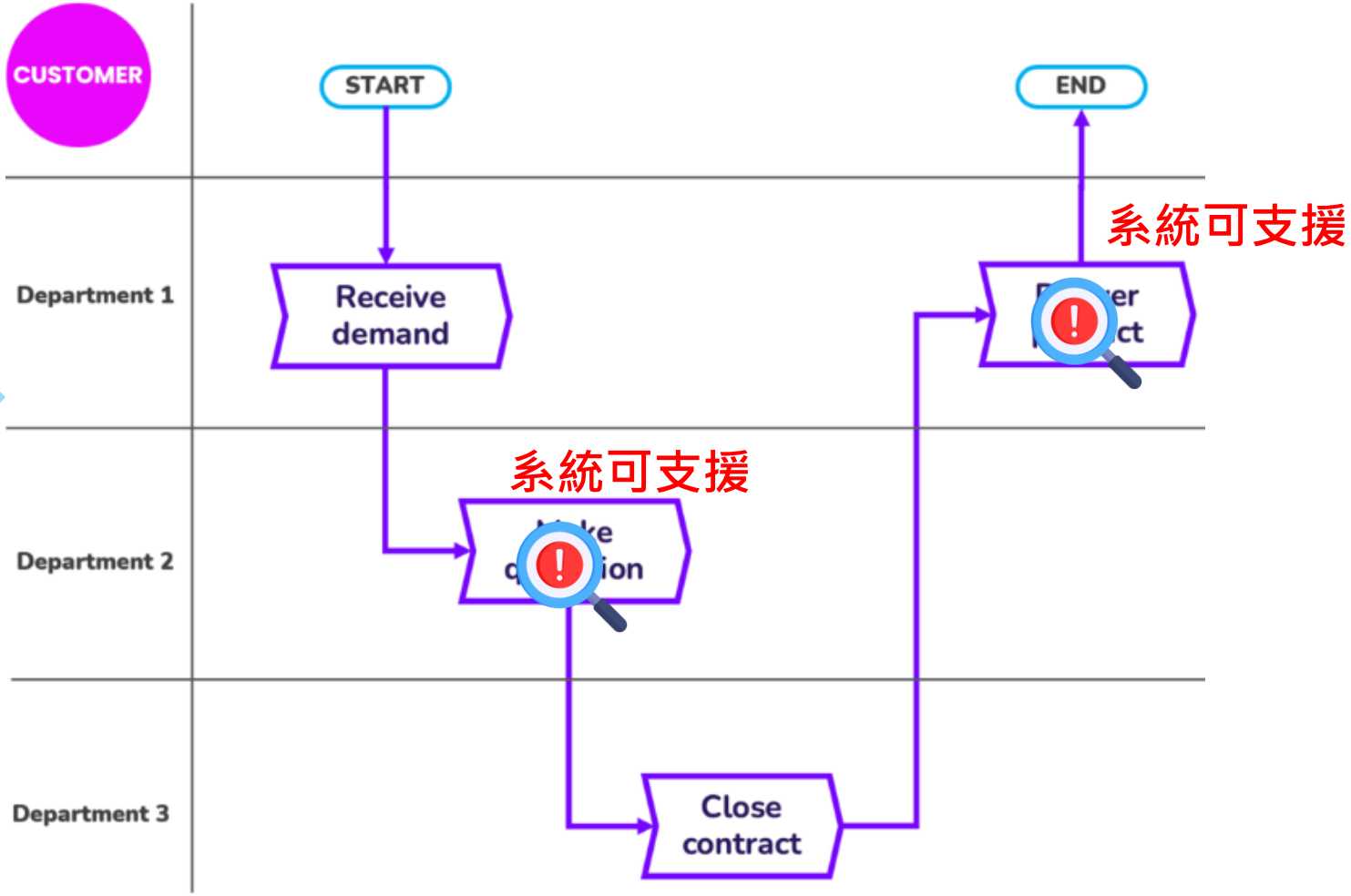
架構質量屬性


軟體產品質量屬性
ISO 25015
功能適應性
可靠性
高效能
易用性
安全性
相容性
可維護性
可移植性
商業環境
重工
系統缺陷

回顧架構設計-從商業流程開始

商業環境

從商業流程角度切入



商業軟體策略

As Is
現況

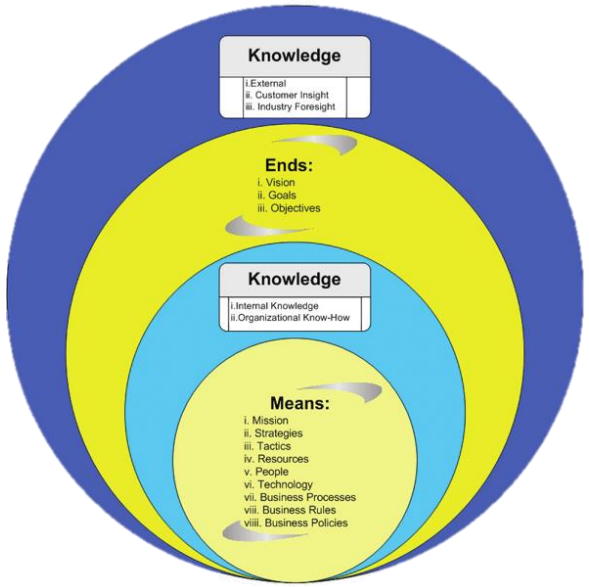
To Be
目標

目標 – 現況 = 差異

調整
商業能力

商業能力(Business Capability)

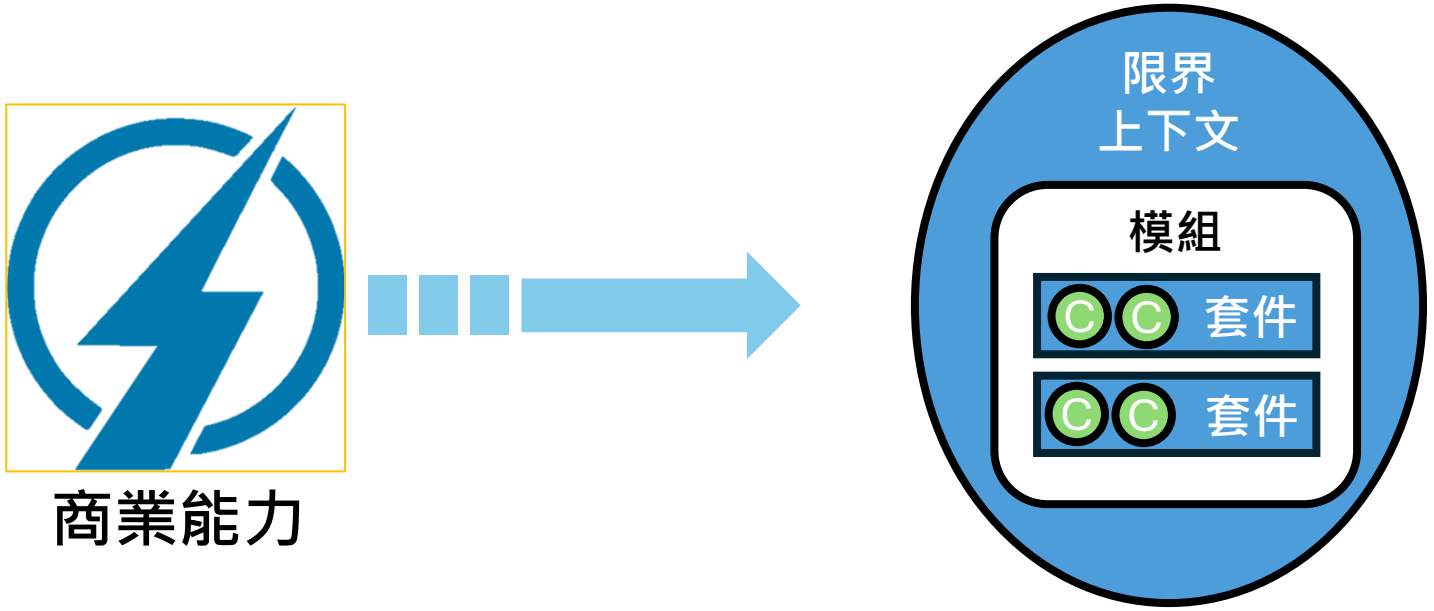
商業能力是一個組織在實現商業目標和策略時，所需的內在能力。



圖源：ResearchGate

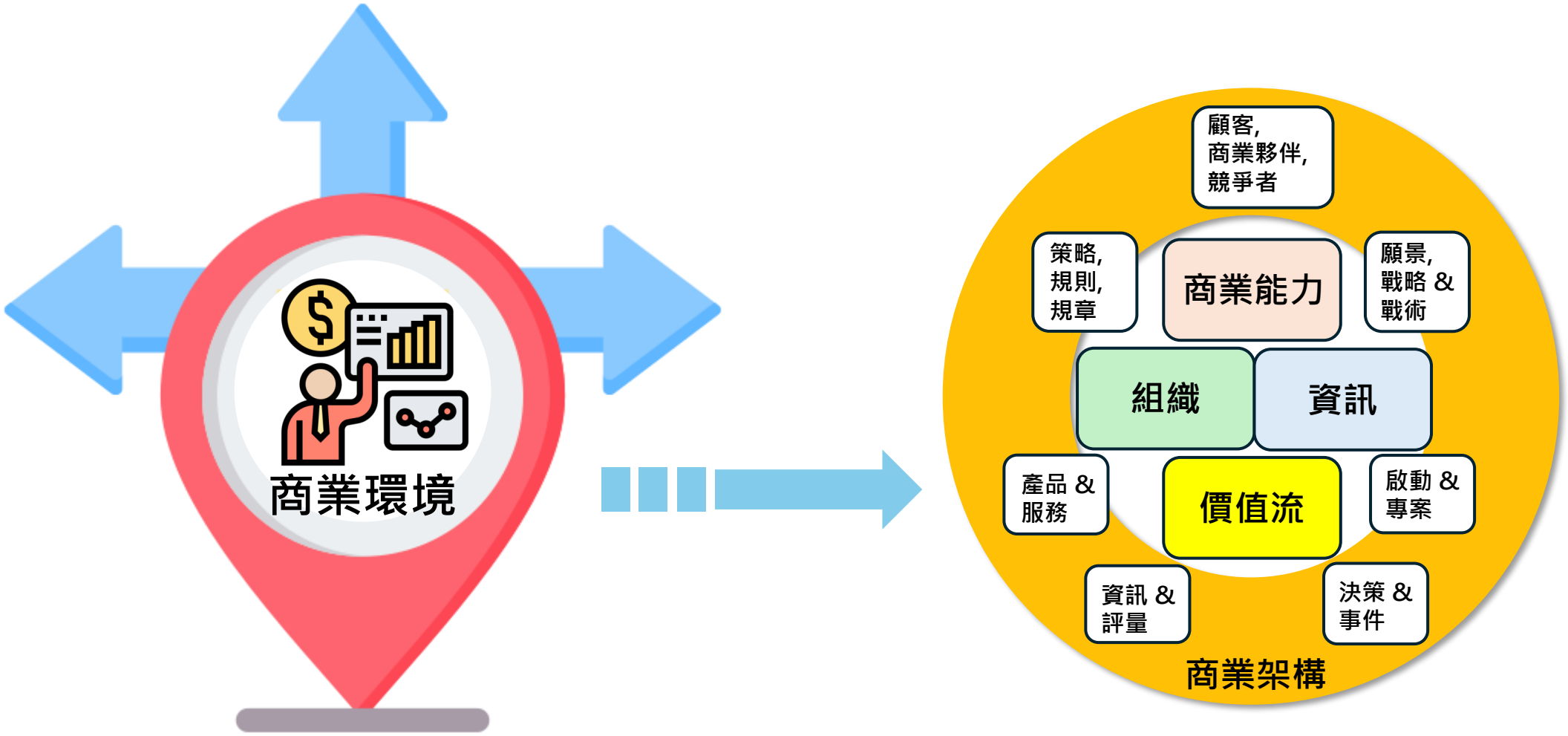
關鍵概念：「**做什麼**」而不是「**怎麼做**」

商業能力與限界上下文對應



商業能力可以比擬
為限界上下文

展開商業的架構

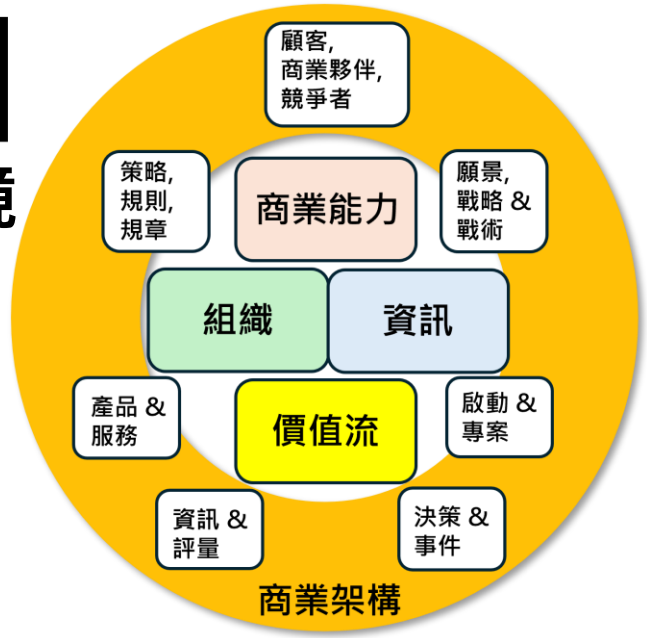


圖源：Business Architecture Guild, A Guide to the Business Architecture Body of Knowledge™

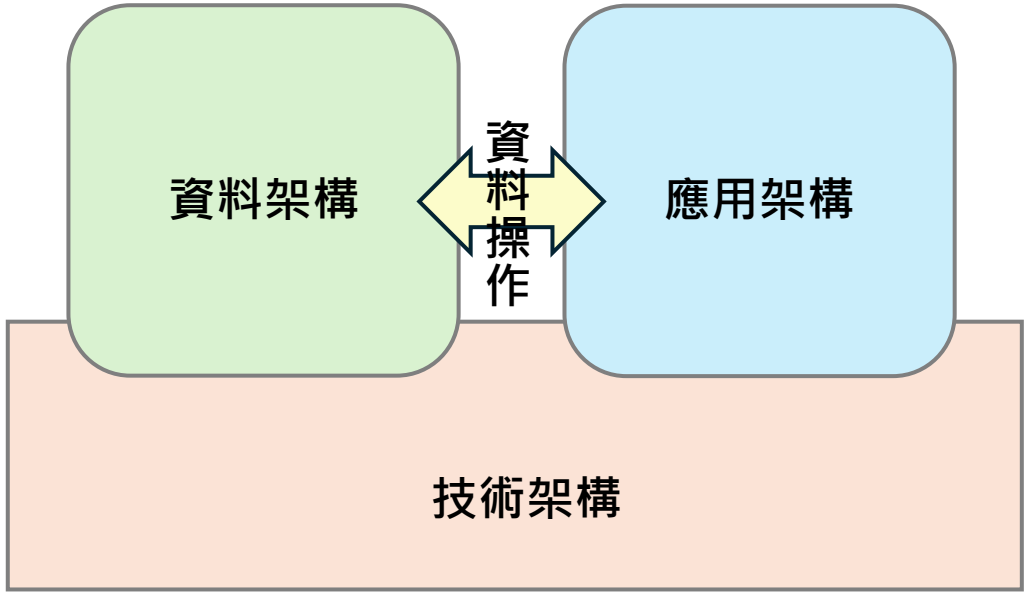
架構設計的驅動源

願景/環境

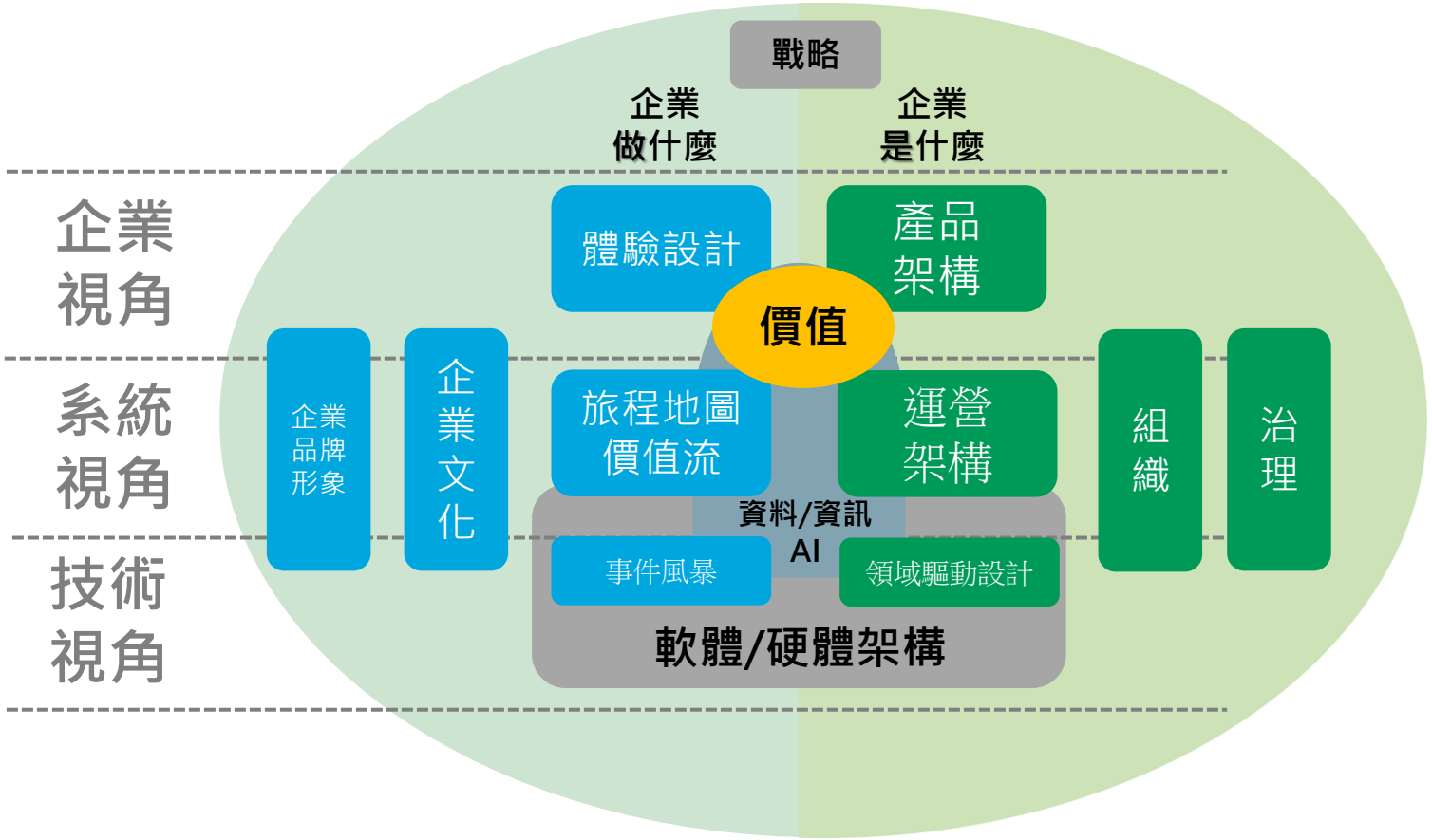
由商業驅動系統的變化，為了滿足系統發展
調整架構



驅動



架構開發概觀



圖源：TOGAF-架構開發

Recap

- 模組與分層架構，其貫穿多個層；逐漸形成一個單片架構
- 單片架構與微服務架構兩者都可以將模組視為服務單位
  - 單片架構可以採用中間架構逐步移轉成為微服務架構
- 架構質量屬性驅動著架構變化
- 商業需求的變化促使架構質量屬性的調整，以適應當前商業環境

持續架構

持續架構

持續架構準則

1. 用產品思維，而非專案思維來設計架構
2. 聚焦質量屬性，而不僅僅是功能性需求
3. 在絕對必要的時候再做設計決策
4. 利用”微小的力量”，以變化導向來設計架構
5. 為建構、測試、部署和營運來設計架構
6. 在完成系統設計後，開始為團隊做組織塑模

應用

系統場景

文化
規模

領域知識

開發方法
限制約束

挑選

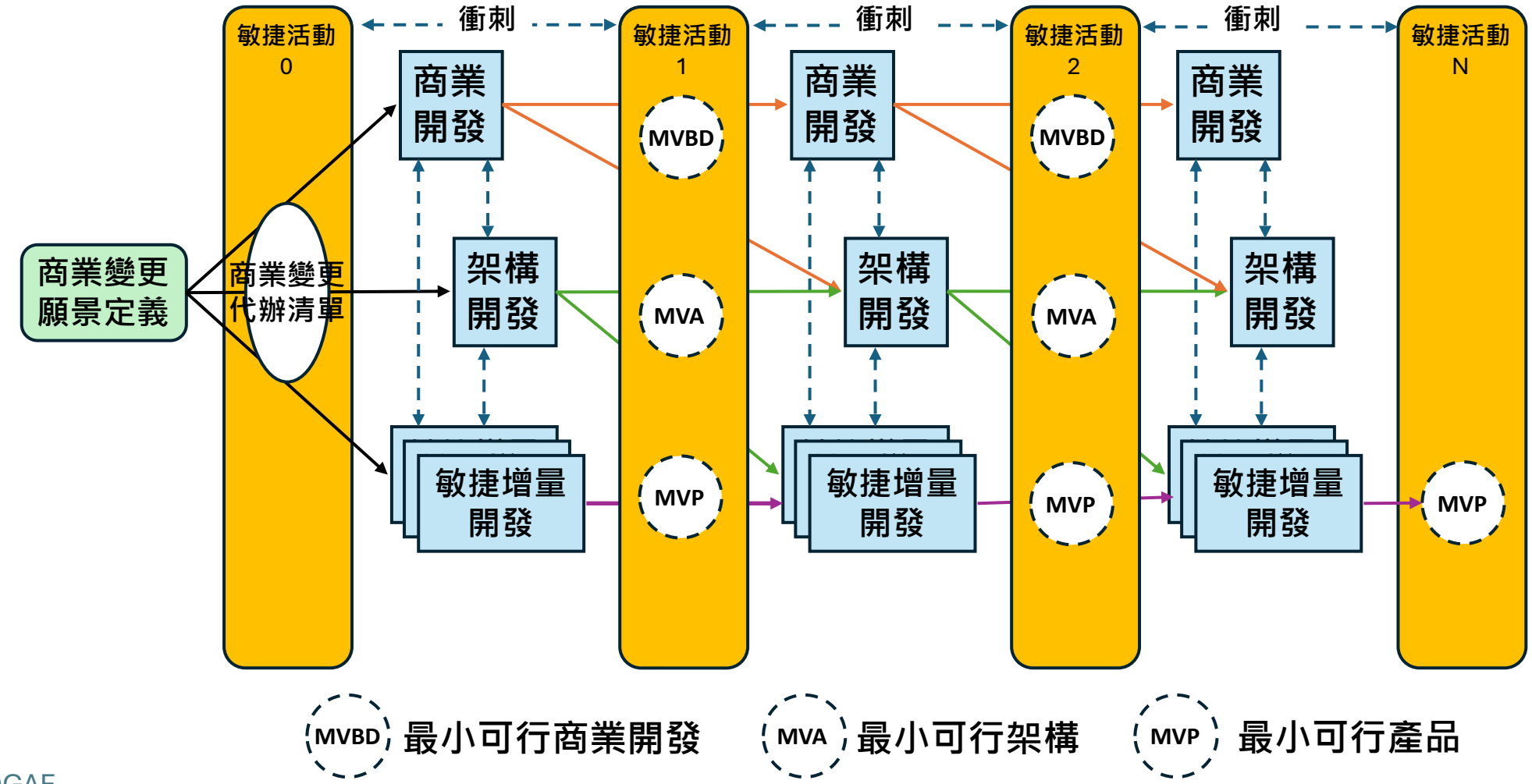
持續架構工具箱

說明

架構活動

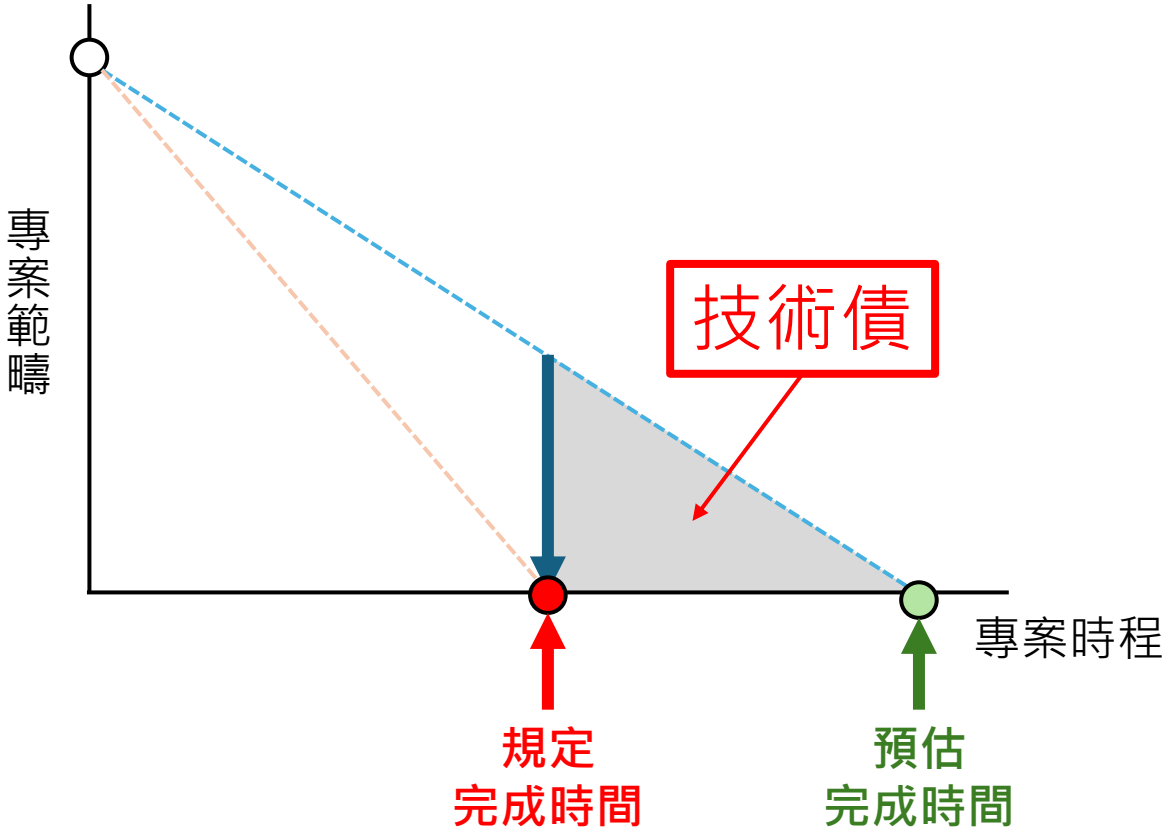
- 專注質量屬性
- 驅動架構決策
- 了解你的技術債
- 實施反饋循環

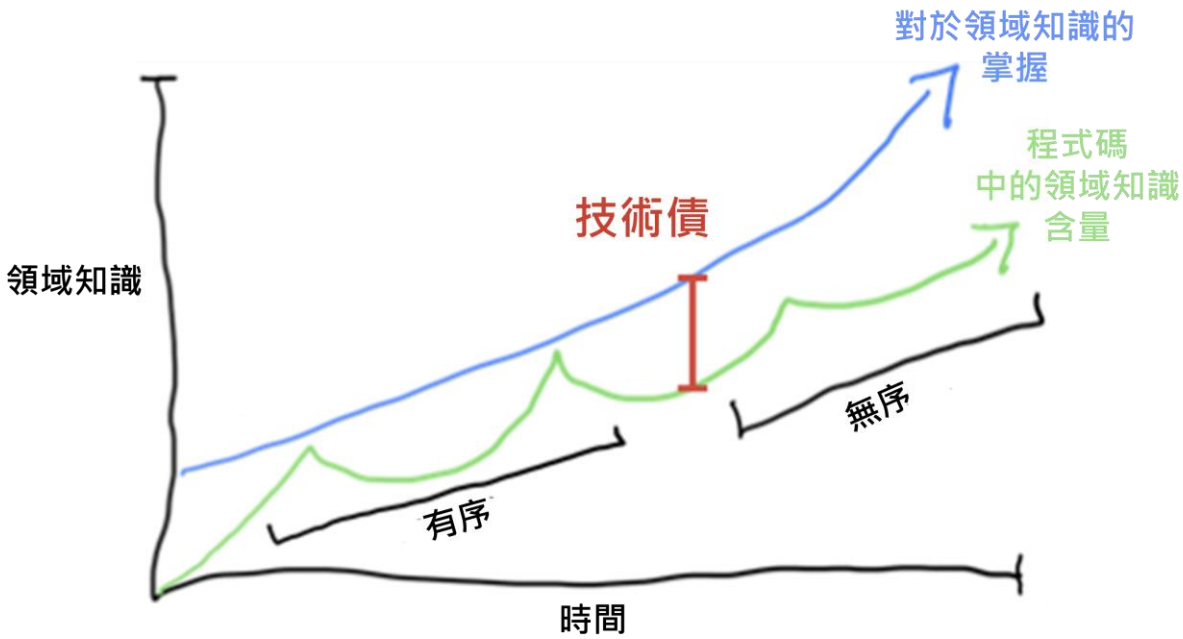
架構敏捷化



圖源：TOGAF

技術債


專案範疇
技術債
專案時程
規定
完成時間
預估
完成時間


對於領域知識的
掌握
程式碼
中的領域知識
含量
技術債
領域知識
有序
無序
時間

Recap

- 運用持續架構的準則，讓架構的演進具有可依循的指南
- 讓架構活動融入到日常開發中
- 留意技術債帶來的問題
- 建立回饋循環，永遠保持開放

感謝聆聽