Software Engineering

U-01 軟體工程概論

2024.04_V1.4

歡迎來到軟體工程世界

在此之前....

資料科學的學習地圖

Data Science Landscape

- Data Pre Processing
  - Feature Selection
  - Handling missing data
  - Obtaining data
  - Feature Engineering
  - Data Cleaning
- Software Engineering
  - Model Deployment
  - Soft development best practice
  - Web Development
  - Parallel Computing
  - Optimize Code
  - Tidy core
  - Data Structure
- Statistics
  - Experimental Design
  - Hypothesis Testing
  - Inferential Statistics
  - Descriptive Statistics
- Mathematics
  - Calculus
  - Optimization
  - Discrete Mathematics
  - Real Analysis
  - Geometry
  - Probability Theory
  - Linear Algebra
  - Matrices
- Programming
  - Spark
  - Bash
  - Julia
  - C/C++
  - Scala
  - SQL
  - Python
  - R
  - Java
  - Hadoop
- Machine Learning
  - Deep Learning (GAN, CNN)
  - Classification
  - Supervised Learning
  - Regression
  - Back Propagation
  - Neural Network
  - Algorithm
  - Support Vector Machine
  - Random Forest
  - Trees
  - X G Boast
  - Decision Trees
  - Unsupervised Learning
  - Clusterings
  - K-means
  - Principal Component Analysis
- Data Visualization
  - Exploratory Data Analysis
  - Types
    - Composition
    - Comparison
    - Relationship
    - Distribution
- Soft Skills
  - Critical Thinking
  - Communication
  - Presentation
  - Problem Solving
  - Lifetime Learning
  - Domain Knowledge
  - Creativity
  - Writing
  - Curiosity
  - Story Telling
  - Grit

Agenda

- 軟體工程概論
- 軟體工程與系統工程的差異
- 資訊系統的三種觀點:

1.企業經營觀點

2.系統架構觀點(3-Tier v.s N-Tier)

3.設計架構觀點(From MVC to Django MTV)

參考用書

- **軟體工程：軟體開發技術與軟體專案管理**
- Sommerville：Software Engineering 10/E
- 作者： 陳玄玲/譯
- 出版社：高立圖書
- 出版日期：2018/06/01
- 綱要：
- 第1單元　軟體工程概觀
  第2單元　可信賴度和保全性
  第3單元　軟體工程進階議題
  第4單元　軟體管理

詳細單元之1

- **第1單元　軟體工程概觀**

- 第 1 章　導論
- 第 2 章　軟體程序
- 第 3 章　敏捷式軟體開發
- 第 4 章　需求工程
- 第 5 章　系統塑模
- 第 6 章　架構設計
- 第 7 章　設計與實作
- 第 8 章　軟體測試
- 第 9 章　軟體演進

詳細單元之2

- **第2單元　可信賴度和保全性**

- 第10章　社會化技術系統
- 第11章　可信賴度與保全性
- 第12章　可信賴度與保全性規格
- 第13章　可信賴度工程
- 第14章　保全工程
- 第15章　可信賴度與保全性的保證

詳細單元之3

- **第3單元　軟體工程進階議題**

- 第16章　軟體再利用
- 第17章　元件式軟體工程
- 第18章　分散式軟體工程
- 第19章　服務導向架構
- 第20章　嵌入式軟體
- 第21章　觀念導向軟體工程

詳細單元之4

- **第4單元　軟體管理**

- 第22章　專案管理
- 第23章　專案規劃
- 第24章　品質管理
- 第25章　組態管理
- 第26章　程序改善

軟體工程概述

軟體工程

- 所有已開發國家的經濟都跟軟體有關
- 愈來愈多系統是受軟體控制
- 軟體工程是一門討論開發專業軟體的理論、方法與工具的學科
- 已開發國家對軟體工程的花費佔GNP非常大的比率
- 維基百科的說明：

「**軟體工程**是研究和應用如何以系統性的、規範化的、可定量的程序化方法去開發和維護軟體，以及如何把經過時間考驗而證明正確的管理技術和當前能夠得到的最好的技術方法結合起來的學科。它涉及到*程式語言*、*資料庫*、*軟體開發工具*、*系統平台*、*標準*、*設計模式*等方面。」(維基百科)

軟體工程

北大西洋公約組織（NATO）在1968年舉辦了首次軟體工程學術會議，並於會中提出「軟體工程」來界定軟體開發所需相關知識，並建議「軟體開發應該是類似工程的活動」。軟體工程自1968年正式提出至今，這段時間累積了大量的研究成果，廣泛地進行大量的技術實踐，藉由學術界和產業界的共同努力，軟體工程正逐漸發展成為一門專業學科。

[Ref]

NATO SCIENCE COMMITTEE的SOFTWARE ENGINEERING Report

➔**http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF**

軟體工程的核心知識(SWEBOK)

- ACM與IEEE Computer Society聯合修定的SWEBOK (Software Engineering Body of Knowledge)提到，軟體工程領域中的核心知識包括：

1. 軟體需求（Software requirements）
2. 軟體設計（Software design）
3. 軟體建構（Software construction）
4. 軟體測試（Software test）
5. 軟體維護與更新（Software maintenance）
6. 軟體構型管理（Software Configuration Management, SCM）
7. 軟體工程管理（Software Engineering Management）
8. 軟體開發過程（Software Development Process）
9. 軟體工程工具與方法（Software Engineering Tools and methods）
10. 軟體品質（Software Quality）

系統分析

- 維基百科的說明：

1. **系統分析**，旨在研究特定系統結構中各部分（各子系統）的相互作用，系統的對外介面與界面，以及該系統整體的行為、功能和局限，從而為系統未來的變遷與有關決策提供參考和依據。系統分析的經常目標之一，在於改善決策過程及系統性能，以期達到系統的整體最優。

2. 系統分析被看作是系統工程的一個重要程序和核心組成部分，以及系統理論的一項應用。 在系統開發生命周期中，系統分析階段先於系統設計，是系統開發前期不可或缺的工作。 系統分析大量借用數學模型、數學分析、計算機模擬等定量分析方法，試圖在具有不確定約束或邊界條件的情況下，對系統要素進行綜合分析、描述，得出較為準確或合理的結論。」(維基百科)

軟體成本

- 軟體的成本通常高於系統成本；而PC上的軟體成本通常高於硬體的成本
- 軟體成本中維護成本高於開發成本。對使用壽命很長的系統而言，維護成本將會是開發成本的數倍之多
- 軟體工程是討論如何達成最合乎經濟效益的軟體開發或演化

分辨

- 何謂「軟體」(Software)？
- 何謂「軟體工程」(Software Engineering)？
- 軟體工程與電腦科學(Computer Science)有何不同？
- 軟體工程與系統工程(System Engineering)有何不同？
- 何謂「軟體程序」(Software Process)？
- 何謂「軟體程序模型」(Software Process Model)？

軟體工程常見問答集

- 何謂軟體工程的成本？
- 何謂軟體工程方法？
- 何謂CASE (Computer-Aided Software Engineering)？
- 好的軟體有哪些特性？
- 軟體工程面臨的主要挑戰？

何謂軟體

- 電腦程式與相關文件
- 軟體產品專為特定客戶開發，或是針對一般市場進行開發
- 軟體產品分為兩類
  - 通用產品 (Generic) – 開發的產品可已販賣給各種不同客戶使用
  - 客製化產品 (custom) – 專為某一位客戶所定的規格而開發的產品

軟體工程與電腦科學之不同

- 電腦科學是與電腦和軟體系統基本理論和方法有關的學科；軟體工程則是與生產和交付有用軟體所面臨的實際問題有關
- 電腦科學的理論目前為止不足以扮演軟體工程完整的支撐

軟體工程與系統工程之不同

- 系統工程是跟電腦化系統開發各種層面有關的工程學科，包括硬體、軟體和流程；軟體工程則只是這個流程中的一部分。
- 系統工程師的工作包括系統規格制定、架構設計、整合與佈署等

軟體程序

- 進行軟體開發或演化的一組活動
- 所有軟體程序均有的通用活動：
  - 規格制定 – 定義系統應該做什麼事以及開發的限制
  - 開發 – 軟體系統的製作
  - 確認 – 檢查軟體是否為客戶所要的
  - 演化 – 更改軟體以回應變更的要求

何謂軟體程序模型

- 以某個特定觀點呈現的軟體程序簡化表示
- 程序觀點的範例有
  - 工作流程觀點 – 依序的活動
  - 資料流觀點 – 資訊流
  - 角色/動作觀點 – 誰應該做什麼事
- 通用的程序模型
  - 瀑布式(Waterfall)
  - 演化式開發(Evolutionary development)
  - 正規轉換(Formal transformation)
  - 以再利用元件整合(Integration from reusable components)

軟體工程的成本

- 大約 60% 的成本為開發成本，40%為測試成本。對客製化的軟體而言，演化成本通常會超過開發成本
- 成本會根據正在開發的系統類型以及各種系統屬性的需求而定，例如執行效能和系統可靠度
- 成本的分佈依據使用的開發模型而定

何謂軟體工程方法

- 軟體開發的結構化方法包括有系統模型、代表符號、規則、設計建議以及程序指引等
- 模型描述
  - 應該產生圖形化的模型描述
- 規則
  - 套用至系統模型的限制
- 建議
  - 良好的設計實務的建議
- 程序指引
  - 依循哪些活動

CASE

(Computer-Aided Software Engineering)

- 目的為提供軟體程序活動自動化支援的軟體系統。CASE系統通常用來做為方法的支援
- Upper-CASE
  - 支援如需求與設計等早期程序活動的工具
- Lower-CASE
  - 支援如程式設計、除錯與測試等後期活動的工具

好的軟體有哪些特性

- 軟體應該完成使用者所需的功能和執行效能，也應該能夠可維護、可信賴以及可使用。
- 可維護性(Maintainability)
  - 軟體必須能夠進行演化以符合變更的需求
- 可信賴度(Dependability)
  - 軟體必須能夠信任
- 效率(Efficiency)
  - 軟體不應該浪費系統資源的使用
- 可使用性(Usability)
  - 軟體必須可以讓使用者針對其設計來使用

軟體工程的主要挑戰

- 處理既有舊系統，處理逐漸增加的變化性以及處理減少交付時間的要求
- 既有舊系統(Legacy systems)
  - 舊的但有其存在價值的系統，因此必須加以維護與更新
- 異質性(Heterogeneity)
  - 系統為分散的且包含各種軟硬體的混合
- 完成與交付(Delivery)
  - 軟體快速完成與交付的壓力逐漸增加

道德規範 – 原則

- 1. PUBLIC
  - 軟體工程師應該固守公眾的利益。
- 2. CLIENT AND EMPLOYER
  - 軟體工程師應該以讓他的顧客和雇主得到最佳的利益，並且固守公眾的利益。
- 3. PRODUCT
  - 軟體工程師應該確保他的產品和相關的修改能夠儘可能符合最高的專業標準。

道德規範－原則

- 4.JUDGMENT
  - 軟體工程師在專業判斷上應該維護其正直與獨立。
- 5.　MANAGEMENT
  - 軟體工程師的經理和上司們應該在軟體開發與維護上支持與提倡合乎道德的管理方法。
- 6. PROFESSION
  - 軟體工程師應該提高專業的正直與聲譽，並與公眾的利益一致。

Code of ethics - principles

- 7.　COLLEAGUES
  - 軟體工程師應該公平的支援他們的同事。
- 8. SELF
  - 軟體工程師應該在他的專業實務上終身學習，並且應該在專業實務上提倡合乎道德的方法。

軟體工程與系統工程的差異

何謂「系統」？

- 系統是由一群相關的組成元件集合而成，為了達成某個目的而共同運作
- 系統可以包含軟體以及可讓人員操作的機械、電機與電子式的硬體
- 系統的組成元件必須依賴其他系統組成元件
- 系統各組成元件的性質與行為不可避免的必須相互混合

系統工程的問題

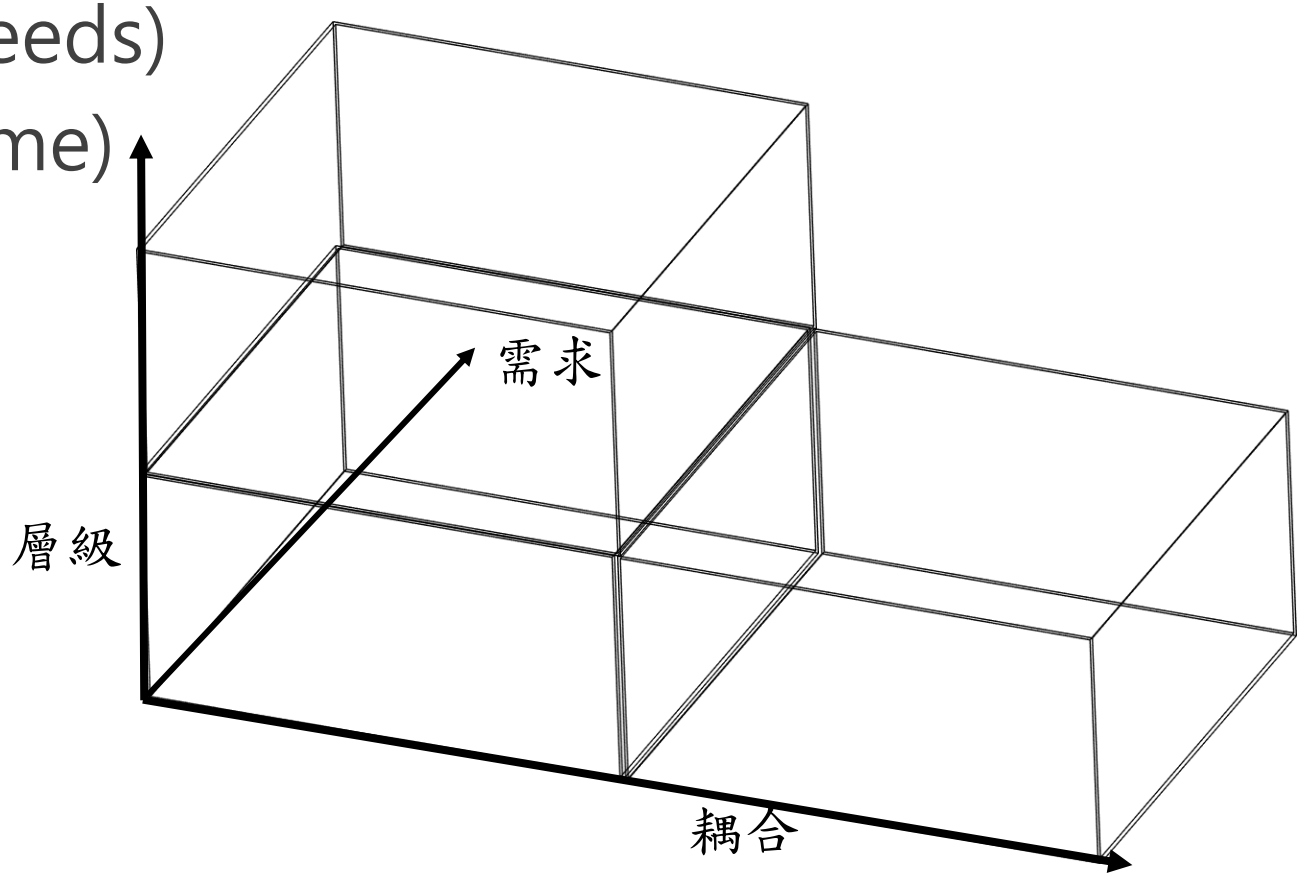
- 大型系統通常設計用來解決「複雜又難以處理的問題」
- 系統工程需要許多跨不同學科的協調
  - 幾乎不可避免的必須在不同的組成元件之間做取捨
  - 不同工程領域之間通常缺乏互信與瞭解
- 系統必須設計成可以在變更的環境中持續運作許多年

軟體與系統工程

- 系統中軟體部分的份量愈來愈重。以軟體驅動的通用電子事產品已漸漸取代特殊用途的系統
- 系統工程的問題與軟體工程的問題類似
- 不幸的是軟體在系統工程中被視為是問題。許多大型系統專案都是由於軟體的問題而延遲

系統工程四維構面

- 層級(hierarchical)
- 耦合(coupling)
- 需求(needs)
- 時間(time)



突顯性質(Emergent Properties)

- 突顯性質不是系統中某個組成元件的特性，而是當系統以整體來考量時所出現的性質
- 突顯性質是系統元件之間的關係所形成的結果
- 因此，這些性質只有在各元件整合成一個系統時才可以進行評估與度量

突顯性質的範例

- ***系統的整體價值***
  - 這個性質可以從個別組成元件的性質計算而來 。
- ***系統的可靠性***
  - 這個性質必須根據系統組成元件的可靠性以及各元件之間的關係而定 。
- ***系統的可使用性***
  - 這是一個非常複雜的性質，它不是直接從系統的軟硬體而來，而是根據系統的操作人員和使用環境而定。

突顯性質的類型

- **功能性的性質**
  - 當系統的所有組成部分一起運作而達成某個目標時所出現的性質。例如，以各種零件組合而成的自行車，在組合完成之後就具有一項能夠當成運輸工具的功能性質。
- **非功能性的性質**
  - 例如可靠性、執行效能、安全性和保全性。這些性質都跟操作環境中的系統行為有關。而且對電腦化的系統而言這些是非常重要的性質，因為只要系統無法達成定義的最小等級性質，系統就會被視為無法使用。

系統可靠度工程

- 由於元件相互之間的相依性，使得錯誤會在系統中擴散開來
- 系統故障通常是由於沒有預見到元件之間的相互關係所產生的
- 各元件的所有可能關係不太可能都預期得到
- 軟體可靠度的度量可能會誤導系統的可靠度

可靠度的影響

- ***硬體可靠度***
  - 指硬體元件發生故障的可能機率，以及修護該元件所需的時間。
- ***軟體可靠度***
  - 指軟體元件產生錯誤結果的可能性。軟體故障通常不同於硬體的故障，因為軟體不會被用壞。
- ***作業員可靠度***
  - 指系統操作人員造成失誤的可能性。

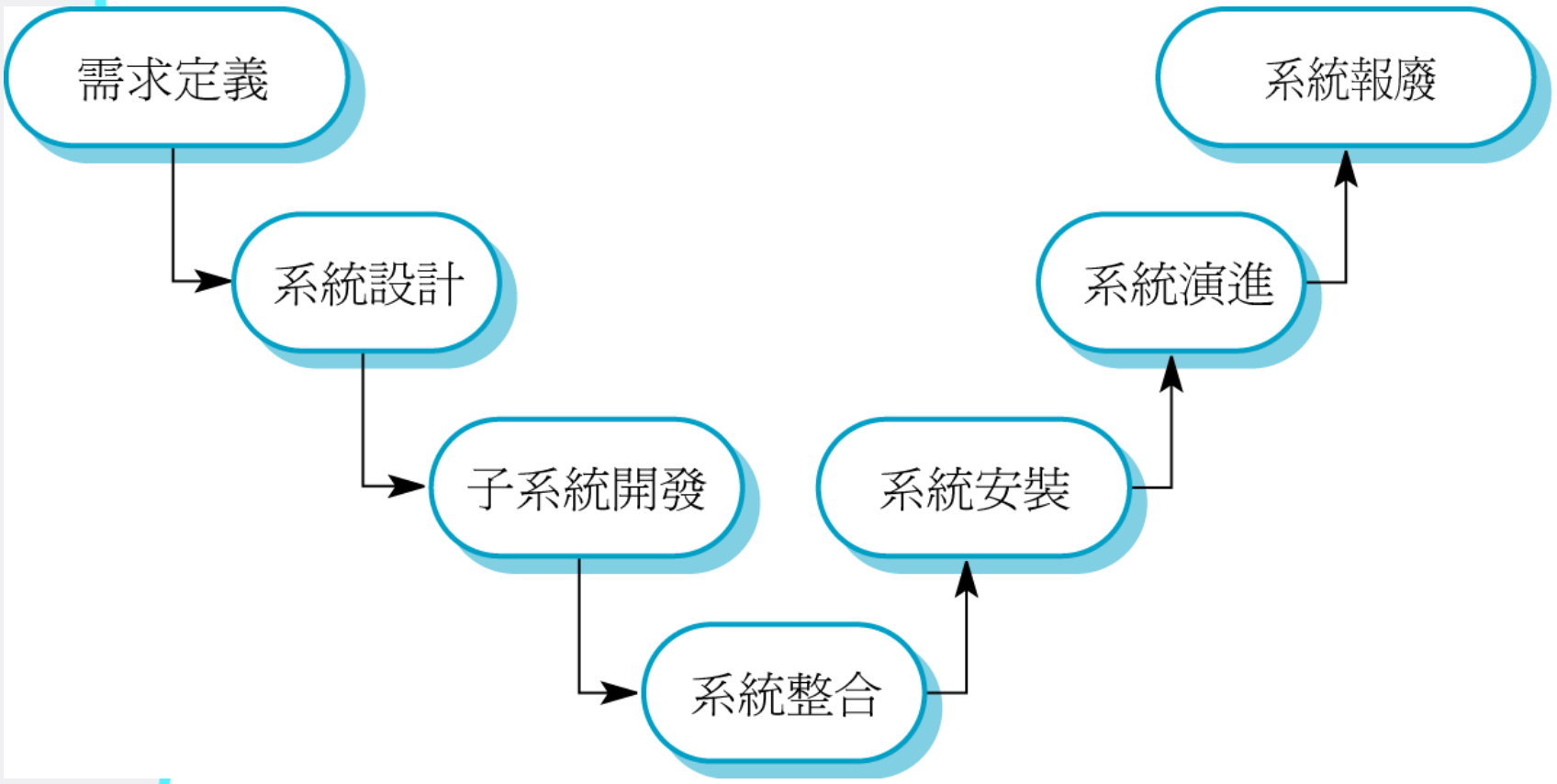
「不應該」的性質

- 執行效能和可靠度之類的性質是可以被度量的
- 然而，有些性質是以系統不應該出現的性質來表示，例如
  - **安全性(Safety)** – 系統不應該出現不安全的行為
  - **保全性(Security)** – 系統不應該允許未經授權的使用者使用
- 度量或評估這些性質非常困難

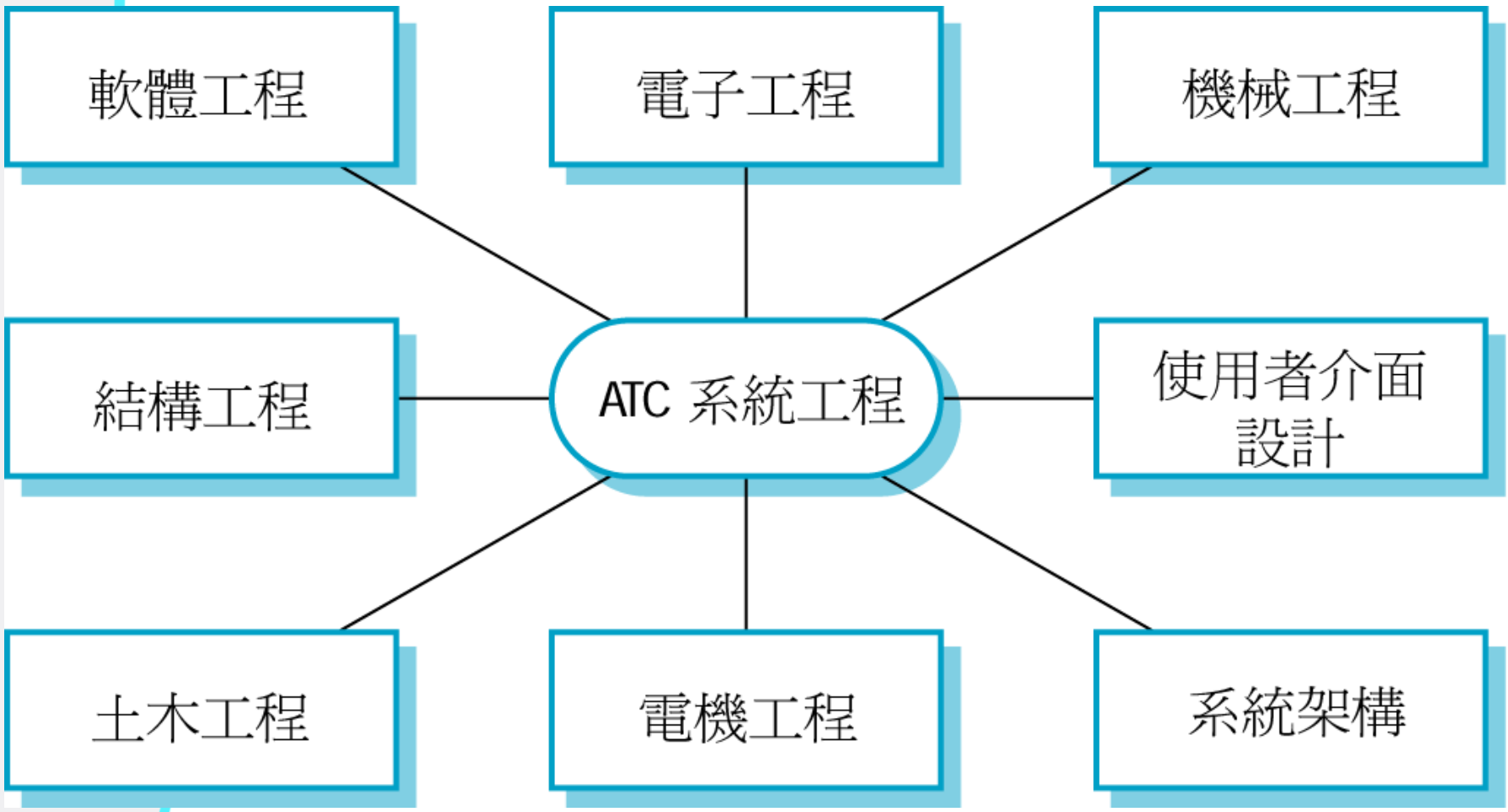
系統工程的程序

- **因為系統不同部分需要並行的開發，所以通常是以「瀑布式」模型為主**
  - 因為硬體的變更非常昂貴，所以不同階段之間的重複範圍很小。軟體可以必須補償硬體發生的問題
- **必須有不同領域的工程師參與一起合作**
  - 產生的誤解可能會變大，不同領域通常會使用不同的辭彙，因此需要大量的溝通與協調。工程師們可能也會有各自的工作需要完成

系統工程程序



不同學科領域的參與



系統需求定義

- 這個階段需要定義下列三種需求：
  - **抽象的功能需求**。以抽象方式定義系統功能
  - **系統的性質**。定義通用的系統非功能性需求
  - **系統不應該有的特性**。指定不可接受的系統行為
- 定義系統的整體組織目標

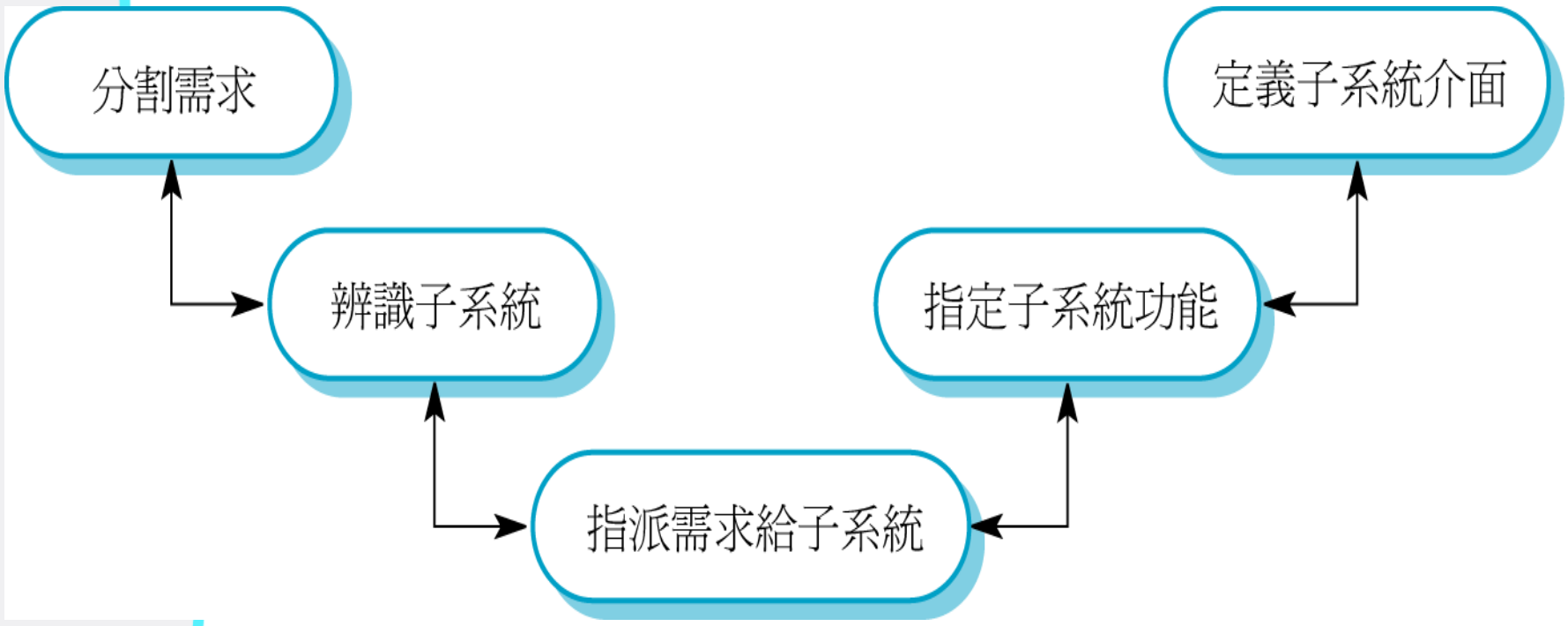
系統目標

- **功能性目標**
  - 例如:為辦公大樓提供一個防火和入侵者警報系統，以便提供內部和外部的火警警告或未經授權的侵入警告。
- **組織性目標**
  - 確保大樓內各項工作的正常運作，不受意外事件的侵擾，例如火警和未經授權的侵入行為。

系統設計程序

- **分解需求**
  - 將各需求組織成相關的群組
- **辨識子系統**
  - 辨識出一組子系統，將這些子系統集合起來可以符合系統的需求
- **指派需求給子系統**
  - 當需要整合 COTS 時會造成特殊的問題
- **指定子系統功能**
- **定義子系統介面**
  - 並行子系統開發的主要活動

系統設計程序



系統設計問題

- 分解成硬體、軟體和人員組成元件的需求可能牽涉到大量的溝通與協調
- 難以設計的問題通常會假設可以很容易的用軟體來解決
- 硬體平台可能不適用於軟體的需求，所以軟體必須對此做些補償

子系統開發

- 典型的平行開發專案，分別進行硬體、軟體與通訊的開發
- 可能牽涉到某些現成商用系統(COTS, Commercial Off-the-Shelf)的採購
- 各個實作小組之間可能缺乏溝通
- 若系統的變更需要經過官僚機制的拖延，開發時程可能就會由於需要重新修訂而延遲

系統整合

- 將硬體、軟體和人員集合在一起組成一個系統的過程
- 應該以遞增的方式來處理，以便可以一次整合一個子系統
- 子系統之間的介面問題通常會在這個階段發現
- 未經協調的系統元件交付成果可能就會出現問題

系統安裝

- 環境的假設可能不正確
- 人員可能會抗拒新系統的引入
- 系統可能必須與其他替代系統並存一段時間
- 可能會有實體的安裝問題發生，例如佈線問題
- 必須辨識出作業員所需的訓練

系統運作

- 將會揭露一些未發現的需求
- 使用者可能會以系統設計者意料之外的方式來使用系統
- 可能會在與其他系統的互動中發現一些問題
  - **實體不相容的問題**
  - **資料轉換問題**
  - **因介面不一致產生的作業員錯誤率增加**

系統演化

- 大型系統有較長的使用壽命，為了符合需求的變更，所以必須進行演化
- 演化非常耗費成本
  - 必須從技術與商業的觀點來分析變更
  - 子系統間的互動會產生一些非預期的問題
  - 原始設計的決策通常很少有記錄
  - 系統進行變更後可能會毀損系統的結構
- 需要進行維護的現存系統有時候稱為「**既有舊系統**」(**legacy systems**)

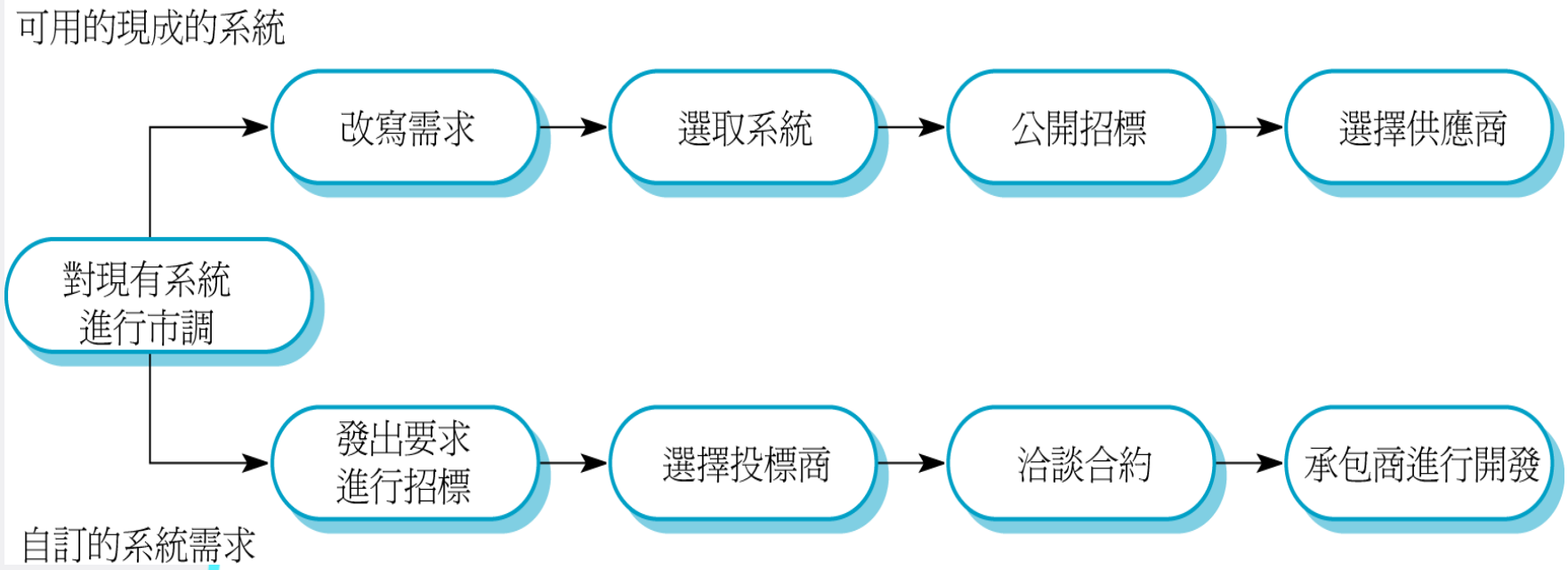
系統報廢

- 在系統使用壽命到期後停止系統的服務
- 可能需要一併移除污染環境的一些物質，例如危險的化學物質
  - **在系統設計時應該規劃使用封裝的方式**
- 資料可能必須重新建構，或轉換成其他系統可以使用的格式

系統採購

- 為組織取得符合某些需求的系統
- 在進行採購之前通常必須有系統規格和架構設計
  - 必須有規格才能讓承包商進行系統的開發
  - 規格中可能允許購買現成的商用系統(COTS)，這個方式通常比從頭開發系統來得便宜

系統採購程序


可用的現成的系統
改寫需求
選取系統
公開招標
選擇供應商
對現有系統
進行市調
發出要求
進行招標
選擇投標商
洽談合約
承包商進行開發
自訂的系統需求

資訊系統的三種觀點

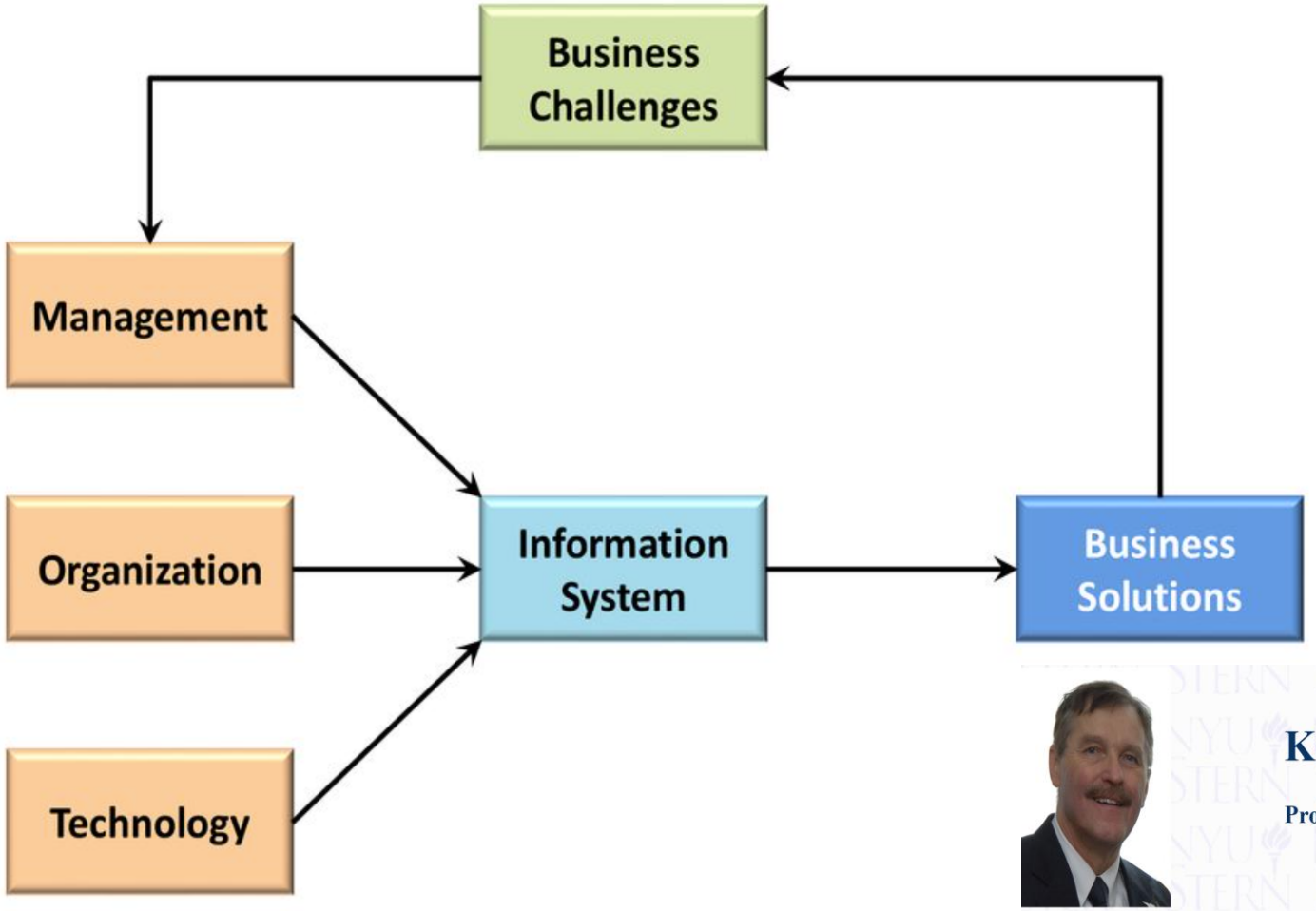
資訊系統的三種觀點




- 企業經營觀點(MIS View)
- 系統架構觀點(3-Tier v.s N-Tier)
- 設計架構觀點(From MVC to Django MTV)

1.企業經營觀點

Prof. Laudon: MIS 架構基礎



Kenneth C. Laudon

Professor, Information Systems

CONTACT INFORMATION

Department of Information, Operations, and Management Sciences

NYU Stern School of Business

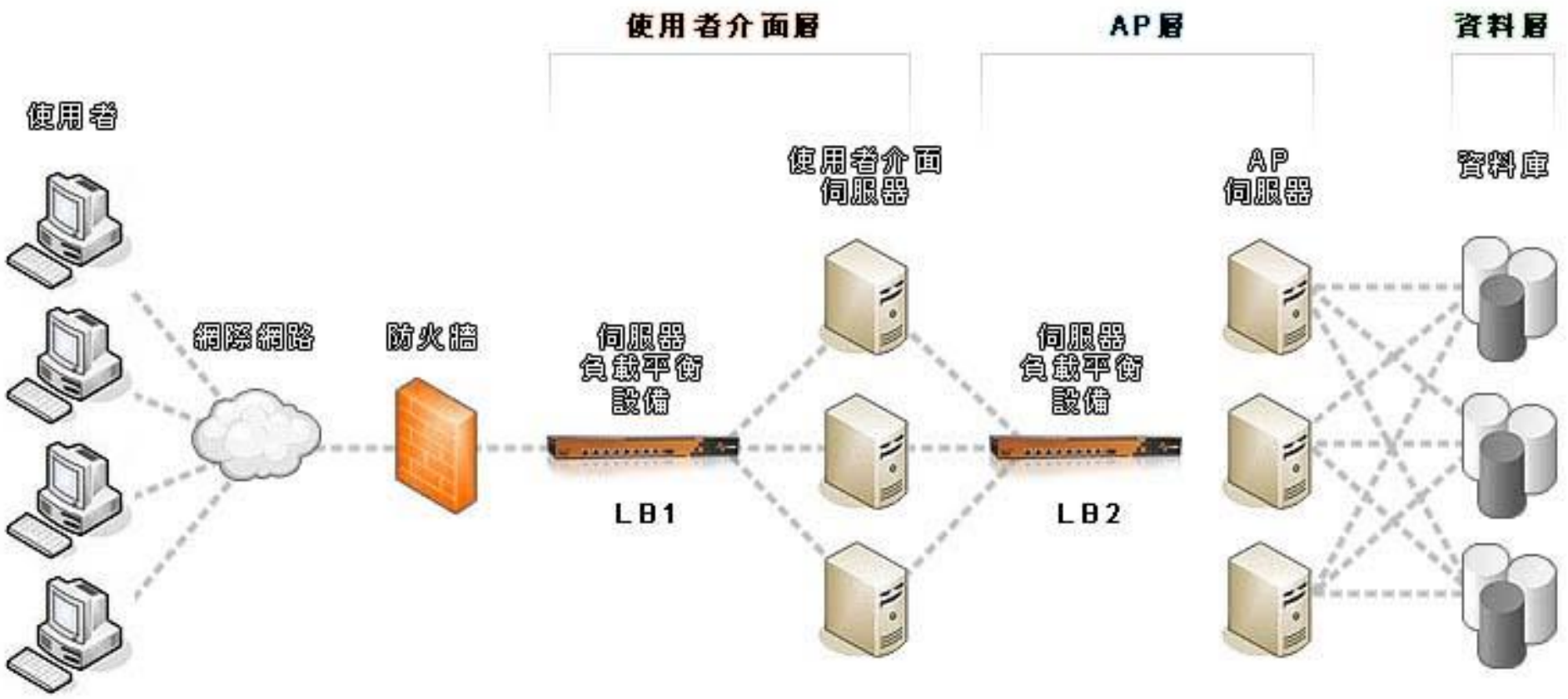
2.系統架構觀點

3-Tiers架構

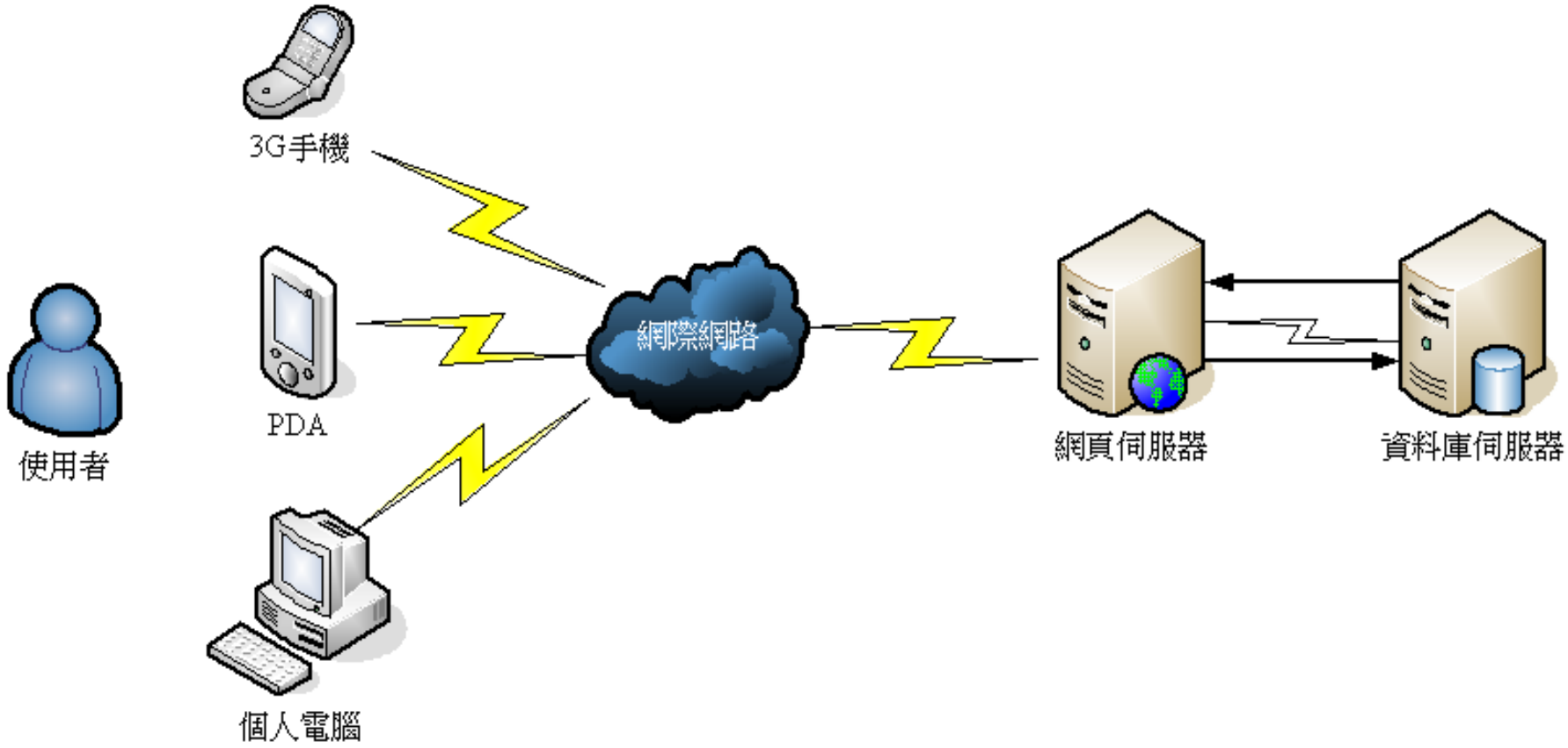
Thin Client v.s Heavy Client

3-Tiers架構(企業版)

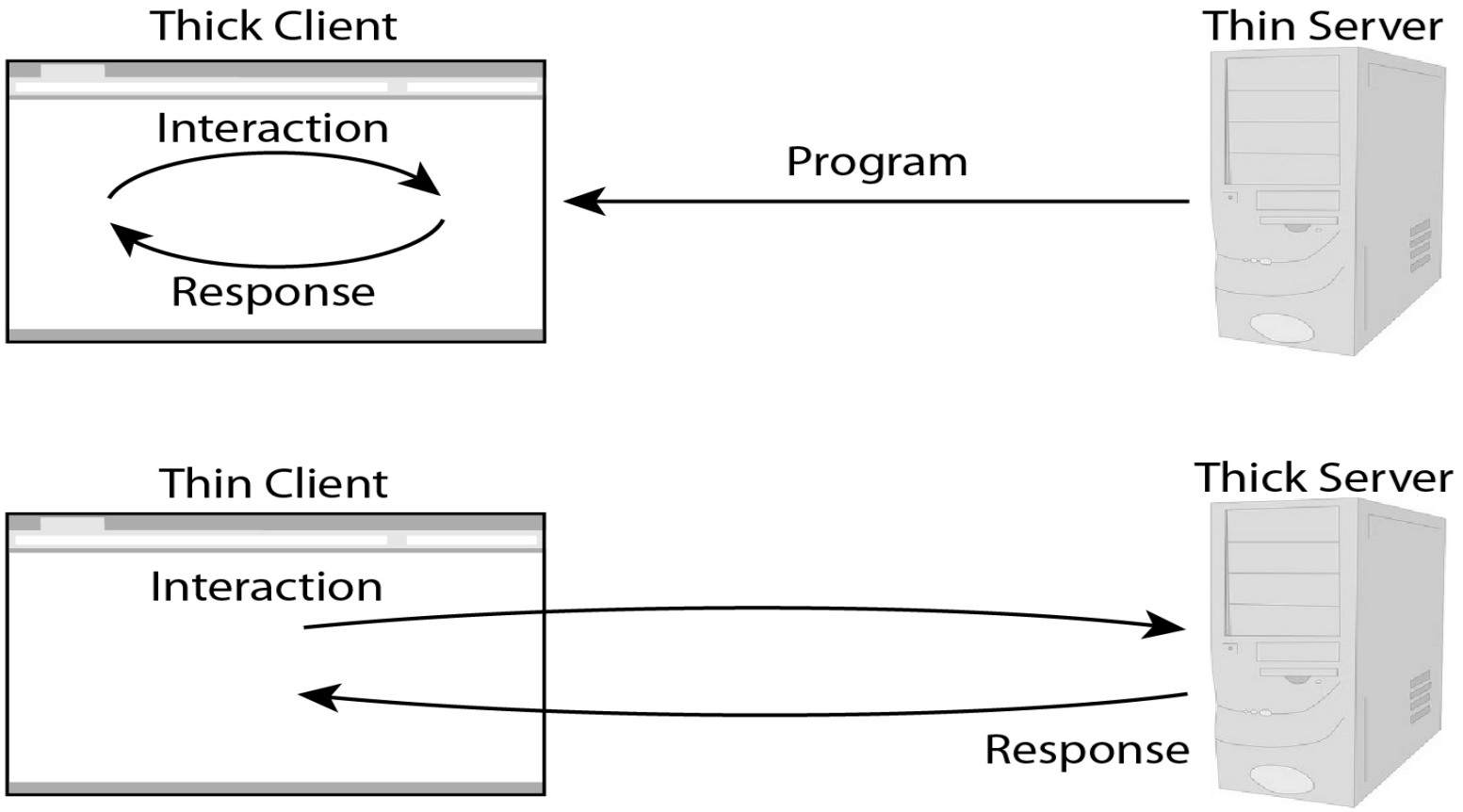
(圖1)



3-Tiers架構(專題版)



Thin Client v.s Heavy (Thick) Client



3.設計架構觀點

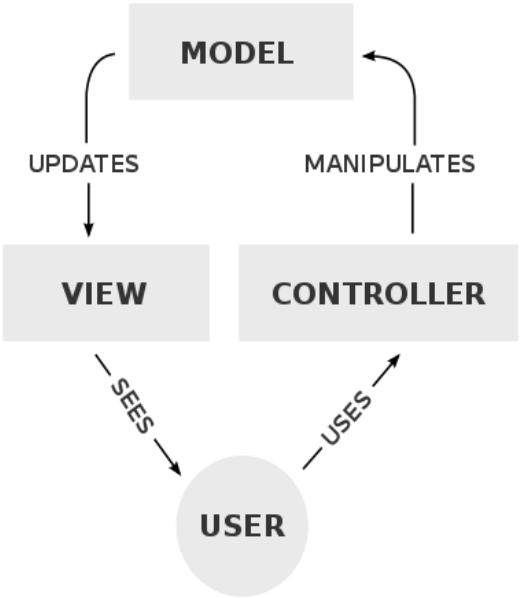
MVC> MVC/Model2 >MVP >
MVVM >MTV

MVC (集中式架構-中大型電腦)

MVC模式（Model–view–controller）是軟體工程中的一種軟體架構模式，把軟體系統分為三個基本部分：模型（Model）、視圖（View）和控制器（Controller）。最早由Trygve Reenskaug在1978年提出，是全錄帕羅奧多研究中心（Xerox PARC）在20世紀80年代為程式語言Smalltalk發明的一種軟體架構。

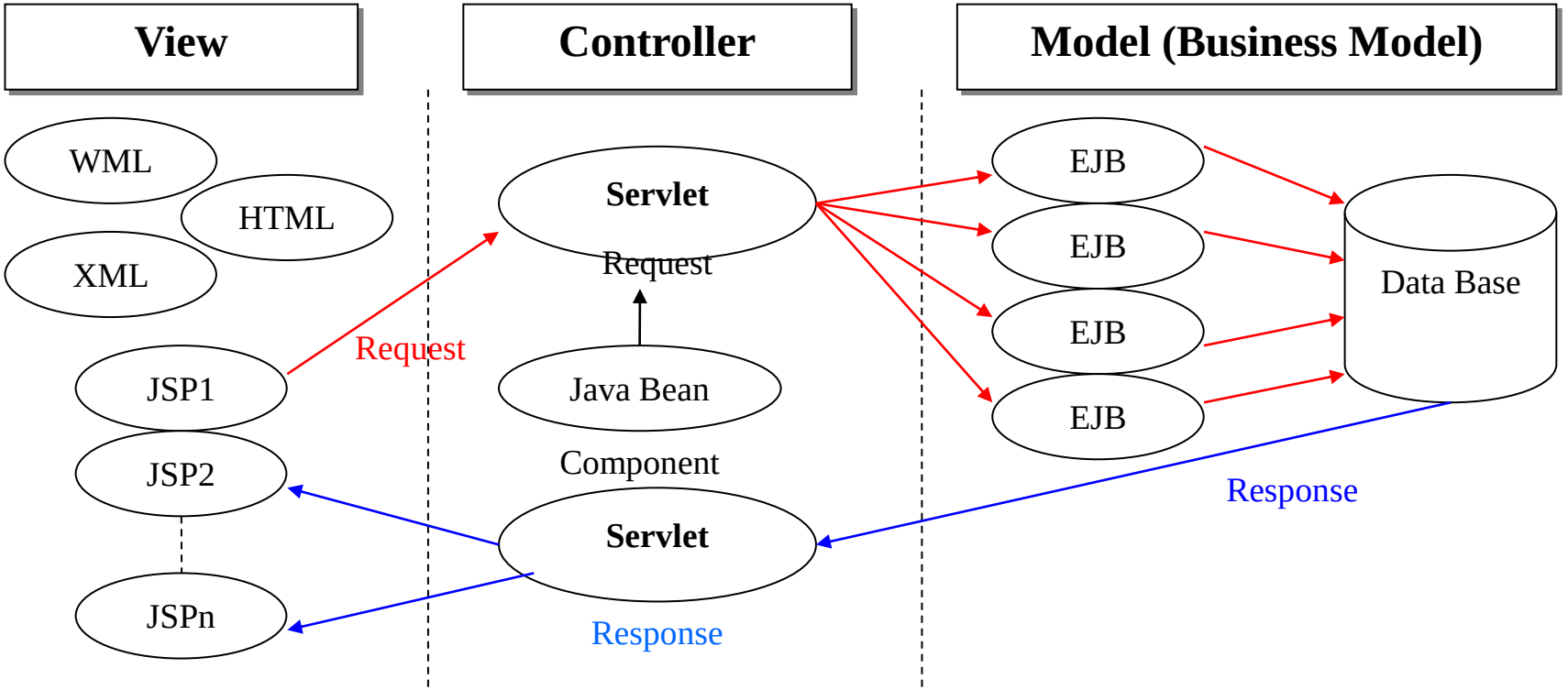
MVC模式的目的是實現一種動態的程式設計，使後續對程式的修改和擴充簡化，並且使程式某一部分的重複利用成為可能:

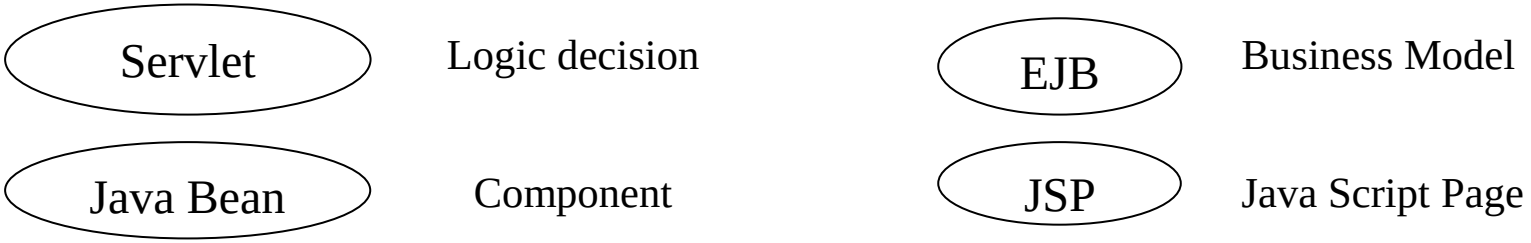
- **模型（Model）** - 程式設計師編寫程式應有的功能（實現演算法等等）、資料庫專家進行資料管理和資料庫設計(可以實現具體的功能)。
- **視圖（View）** - 介面設計人員進行圖形介面設計。
- **控制器（Controller）** - 負責轉發請求，對請求進行處理。



[REF]: https://zh.wikipedia.org/wiki/MVC

Model View Controller Architechure (M.V.C)(JAVA版)

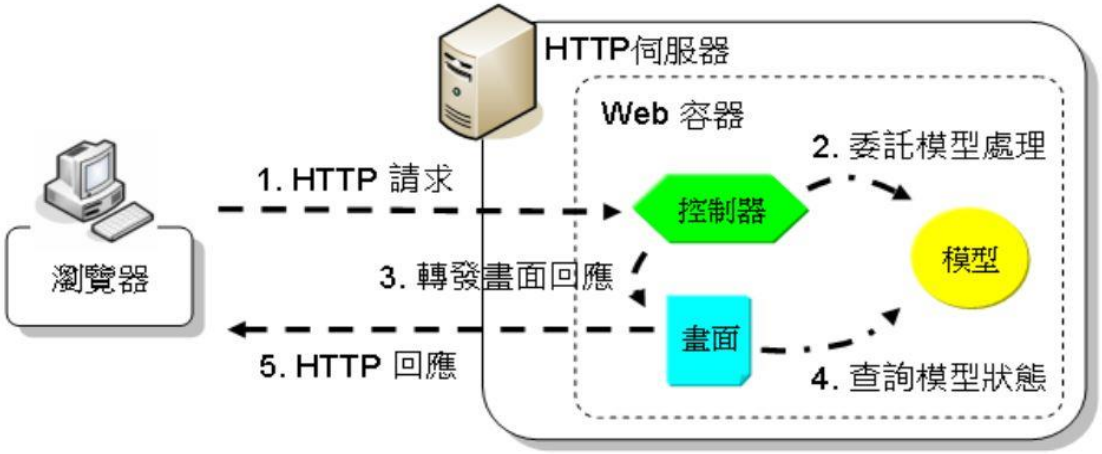




MVC/Model2 (分散式三層架構)

- 將 Web 應用程式的組成劃分為模型、畫面與控制器三個角色，Web 應用程式借鏡桌面應用程式 MVC 架構，取其 Model/View/Controller 的職責劃分，並修改流程為適用於 HTTP 請求/回應特性，這個修改後的架構為併稱為 MVC/Model 2。

- **控制器（Controller）的職責**
  - 接受請求
  - 驗證請求
  - 判斷要轉發請求給哪個模型
  - 判斷要轉發請求給哪個畫面
- **模型（Model）的職責**
  - 保存應用程式狀態
  - 執行應用程式商務邏輯（Business logic）
- **畫面（View）的職責**
  - 提取模型狀態
  - 執行呈現邏輯（Presentation logic）組織回應畫面

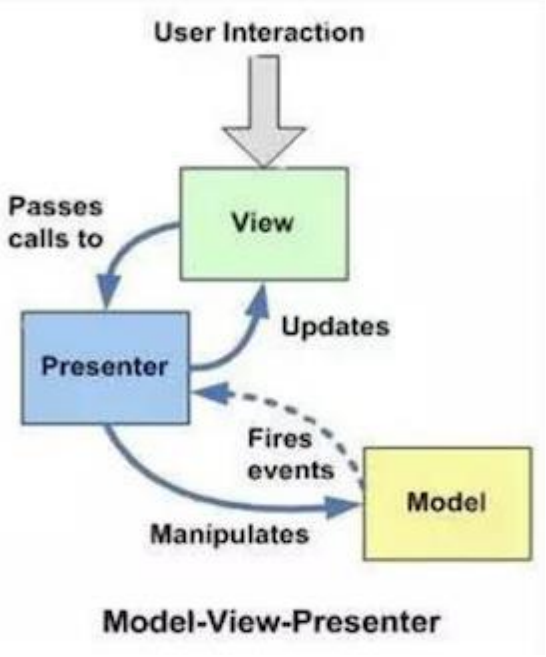


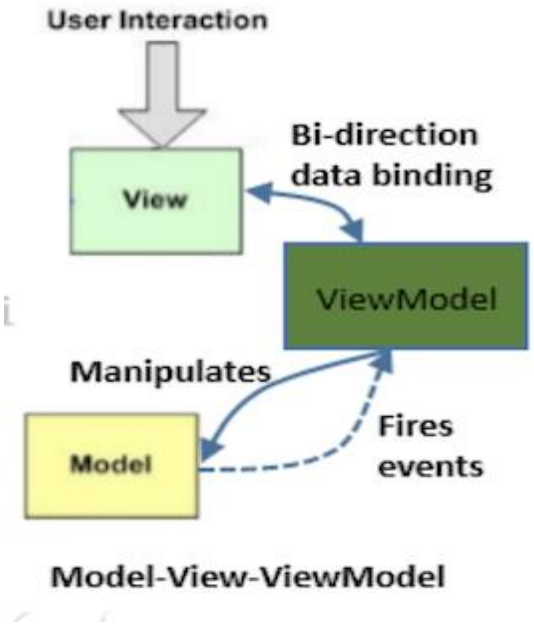
[REF]: https://openhome.cc/Gossip/ServletJSP/Model2.html

MVP與MVVM(手機App開發`

- MVP(Presenter)

和 MVC 不同的是，Model 層拿到數據後，並不直接傳給 View 更新，而是交還給 Presenter，Presenter 再把數據交給 View，並更新畫面。





- MVVM(Model-View-ViewModel)

透過觀察者模式將 View 和 Model 巧妙地連接在一起，一旦 Model 的數據發生變化，觀察者 View 就能夠感應到這個更動，並把數據更新到 UI 畫面上，ViewModel 甚至不需要持有 View的引用，更方便進行單元測試。

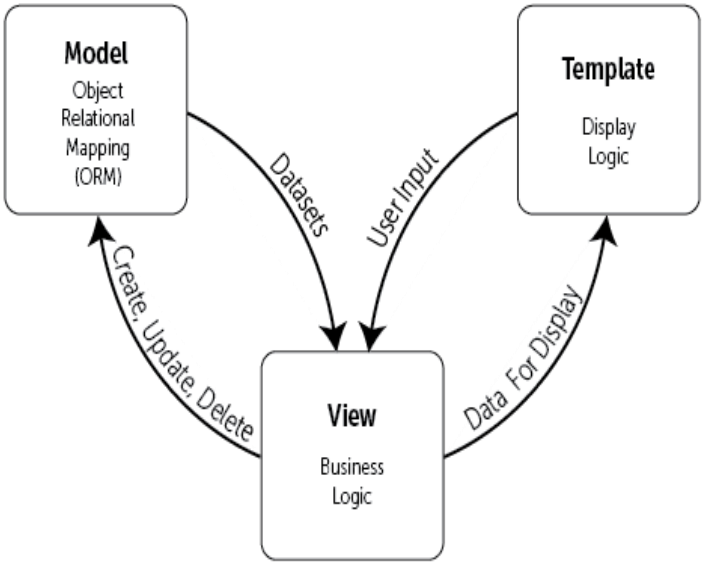
[REF]: https://ithelp.ithome.com.tw/articles/10218263

MTV（Model-Template-View）

Django 採用的是類似 MVC 的 MTV (Model-Template-View) 架構：

- **M（Model，模型）**： 同上 Model。
- **T（Template，模板）**：同上的 View （2）生成頁面展現給使用者的部分，也就是我們看見的前端 HTML、CSS、JavaScript 的部分。
- **V（Views，視圖）**：同上的 View （1）處理業務　　邏輯、封裝結果的部分，負責處理 URL 與 callback函式之間的關係，每一個 view都代表一個簡單的Python function。

- **C（Controller，控制器）**的部分如果要對應的話為 Django 本身。



[REF]: https://ithelp.ithome.com.tw/articles/10218263