軟體工程概論

授課教授：徐偉智 教授

何謂軟體工程

- 軟體工程是如何開發軟體的方法
- 資訊硬體日新月異，人們需要高品質且多功能性的軟體來輔硬體發揮效用。
- 軟體已從「單一程式」進而演變成為「複雜系統」。
- 傳統的單打獨鬥已無法應付此種變化。
- 軟體工程愈來愈受到重視。

軟體開發的處理

- 軟體的開發有一定的流程，並非想到哪做到哪。
- 軟體的開發大概分為以下幾個階段：

1 軟體規格建立
2 軟體的開發
3 軟體測試驗收
4 軟體更新

軟體規格的建立待續1

- 軟體系統開發之前需要先了解「需求」並界定功能。
- 事先未規劃好軟體的功能，會導致需求「無限擴張」。
- 影響整個開發時程、資源、資金與成功與否。
- 需求確定後就開始「分析」。

軟體規格的建立待續2

- 完成分析後，已確定軟體的功能。接著是「系統設計」。
- 對問題提出解決方案，同時設計軟體架構。

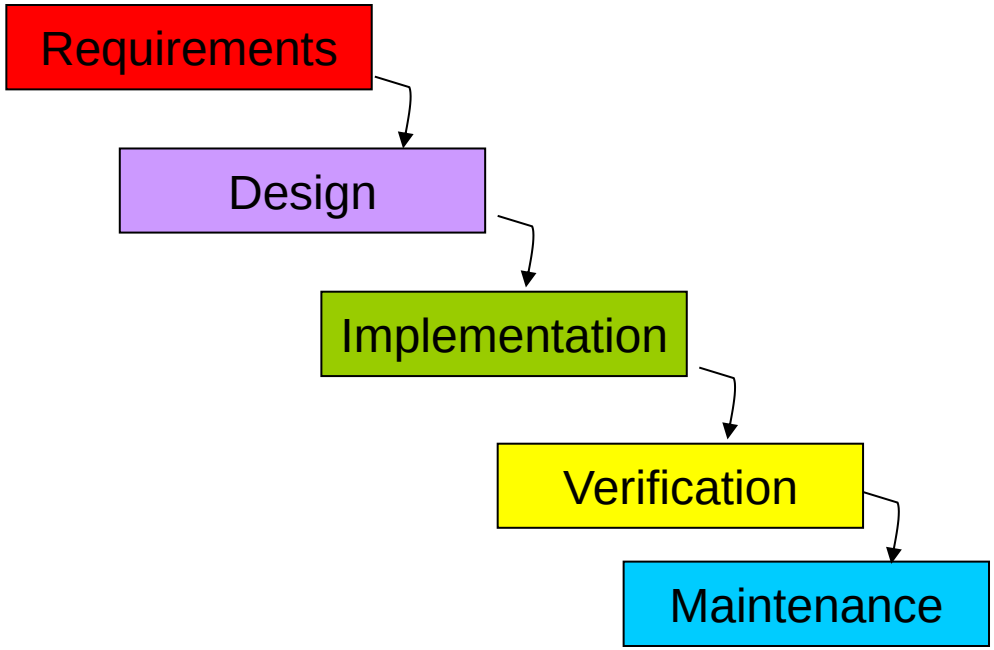
- 開發複雜的系統可以適當的切割成為多個子系統的方式進行開發。
- 同時由不同的開發者進行開發，最後在進行整合。
- 可縮短期程，避免在發生錯誤時影響整個系統。

軟體規格的建立

- 規格產出後仍需檢視其中各子系統的關連性與介面設計是否合適
- 模糊的規格再次定義。

軟體的開發待續1

■ **傳統的瀑布式(Waterfall model)的開發處理**



軟體的開發 待續2

- 瀑布模型的階段區隔清楚
- 然而實際開發時有許多無法控制的因訴，以致很難這麼清楚畫分階段。
- 所以發展出改良的開發處理

軟體的開發 待續3

■ 漸進式的軟體開發處理

- 分析初步需求後即進行系統設計開發、完成系統初版、測試與修改，直到最終版產出。
- 好像有很高的效率，但管理有問題存在
- 需視軟體特性選擇合適開發處理
- 高效率且便於管理。

軟體的開發待續4

■RUP(Rational Unified Process)

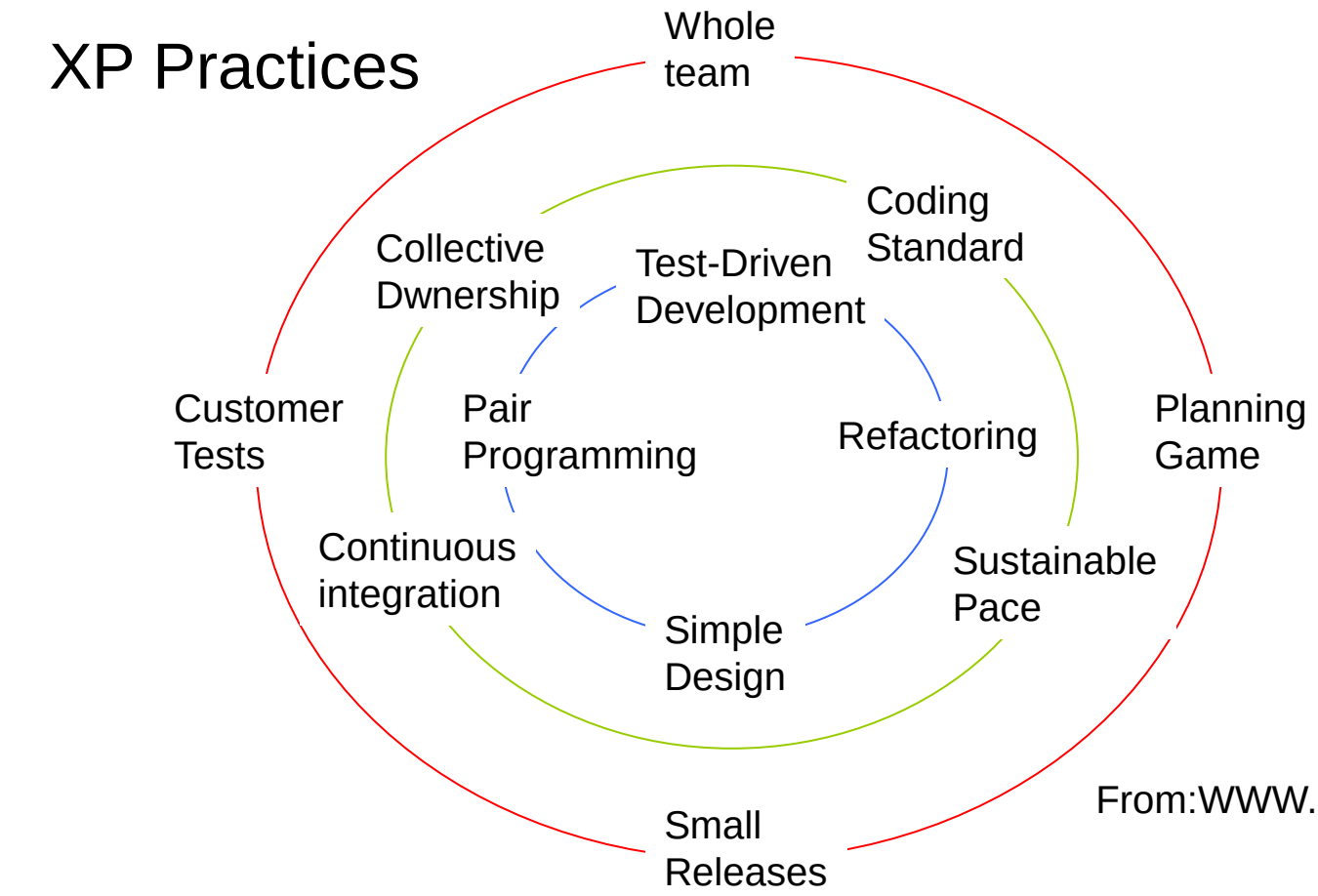
- ◆RUP是物件導向式的開發方法。
- ◆運用RUP需先確定開發模式與處理。
- ◆RUP專案有四個階段：
  1 開始階段(inception)：專案評估是否要進行下一階段。
  2 細化階段(elaboration)：發展USE CASES，並構思軟體系統架構。
  3 建置階段(construction phase)：建置，直至完成大部份功能。
  4 轉換階段(Transition phase)：進行一些不需反覆的工作。

軟體的開發 待續5

- **敏捷性的開發方式(agile development)**
  - XP(Extreme Programming) 、 FDD(Feature Driven Development) 、 DSDM(Dynamic Systems Development Method)等。
  - 特性:適調力大，對需求改變回應迅速。
- **XP(Extreme Programming)**
  - 特徵：成對(pair)編程與測試驅動。
  - 3個原則：持續測試、開發者開發、與使用者密切溝通。

軟體的開發

XP開發模型圖



From:WWW.XProgramming.com

軟體工程的重要性

- 軟體架構工程師與程式設計員有差異。
- 軟體架構工程師了解、設計系統而程式設計員撰寫程式。
- 系統開發勿採用「土法煉鋼」的方式，要有工法 。
- 實踐軟體工程要成本與人力， 但值得(在維護階段)。

軟體工程的應用與發展

- 軟體系統，參與的人數多，需要專案管理 (Project management)以掌握軟體開發過程與進度。
- 軟體工程與專案管理的關係?
- 軟體工程在技術上的發展?
- 如何建立實踐軟體工程的軟體開發環境?

專案發展（Project development）與專案管理

- 專案發展的過程通稱為專案生命週期發展（Project Life Cycle Develpment），以後簡稱為PLCD。
  - ◆PLCD定義軟體開發的過程，使軟體開發有跡可循。

循序專案開發處理（Sequential PLC, 即SPLC）

- SPLC軟體開發過程分為幾個階段:
  - 專案開始（Project Initiation）
  - 系統分析（Systems Analysis）
  - 系統設計（Systems Design）
  - 系統實作（Implementation）

專案開始及系統分析階段

- 專案開始：定義需求，初期的評估
  - 軟體開發需要成本，開發前要確定有開發的價值。
- 系統分析：開始軟體開發生命週期（SDLC, Software development life cycle）。
  - SDLC的目標是產出應用系統，先進行系統分析，找出系統需求、使用者介面初步設計。

系統設計與實作

- 系統分析定義系統需求，軟體工程師再根據系統需求把系統　設計出來。
- 系統設計建立嚴謹的系統規格（Specification）
  - 與電腦軟硬體環境有關係。
- 系統設計可分成幾個步驟：
  - ◆概念化設計（Conceptual Design）：系統功能的初步設計。
  - ◆系統架構設計（Architectural Design）：循序架構還是物件導向的架構。
- ◆系統實作：撰寫程式。

確定軟體架構與其他 (Software architecture)

- 系統架構設計包括下列工作：
  1. 軟體系統結構決定
  2. 系統組成模組（Module）的細部設計
  3. 使用者介面的設計及列印表格的型式之設計
- 其他
  1. 系統測試
  2. 系統啟動與運作（System Installation and Operation）
  3. 系統維護（System Maintenance）

  4系統的淘汰與更新（System Retirement and Renewal）

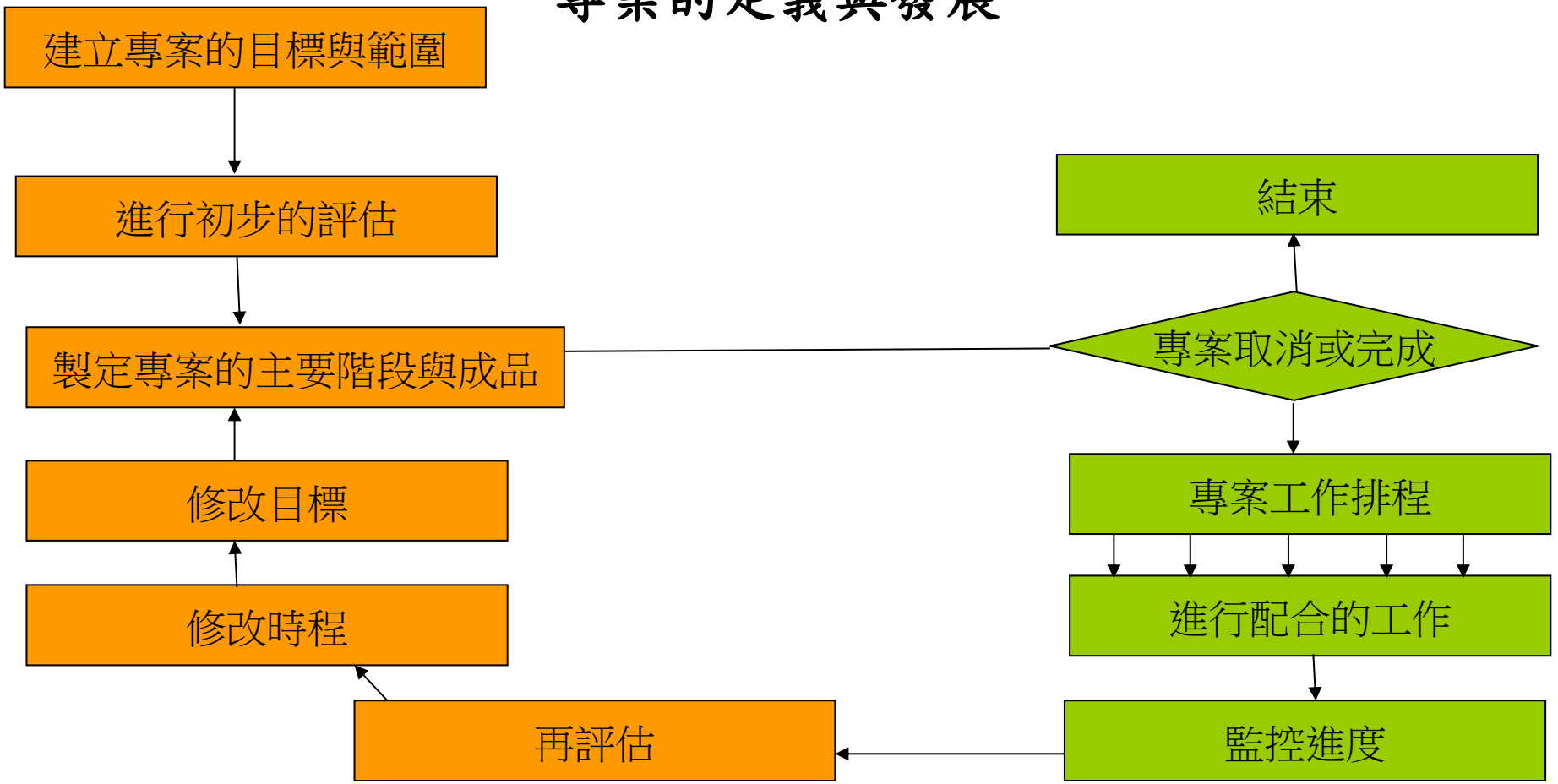
軟體專案管理

- 專案計畫書、成本預估、專案規劃、時程規劃、專案監控與評核、參與人員選擇等。

軟體專案規劃的流程

- 軟體專案經理（Software project manager）依照既定流程進行專案管理的工作。

專案的定義與發展



專案成敗的三個因素

- 系統開發者對問題的深入了解。
- 專案的規劃與管理。
- 系統效率、正確性、是否容易維護。

需求工程

- 了解應用領域與軟體系統的關聯。
- 如何描述應用領域的特性與需求。
- 如何表示軟體系統的功能。
- 了解軟體系統開發的規格化。
- 了解需求分析的方法。

資訊系統的規劃

- **3個主要的步驟 ：**
  1. 了解目前的狀況
  2. 規劃未來的藍圖
  3. 排定開發的時程
- **規劃的方式**
  1. 由上而下(top-down)的策略：從企業的資訊系統需求出發，整合各部門需求。
  2. 由下而上(bottom-up)的策略：從作業層次與部門的觀點出發。成效快成本低但容易忽略整合性。

軟體系統的規格

- 「**需求**」怎麼來？
  1. 讓客戶能隨時隨地下單(order)
  2. 廠房空間有限，調整產品的生產組合，使空間的使用最有效

需求工程
(Requirements engineering)

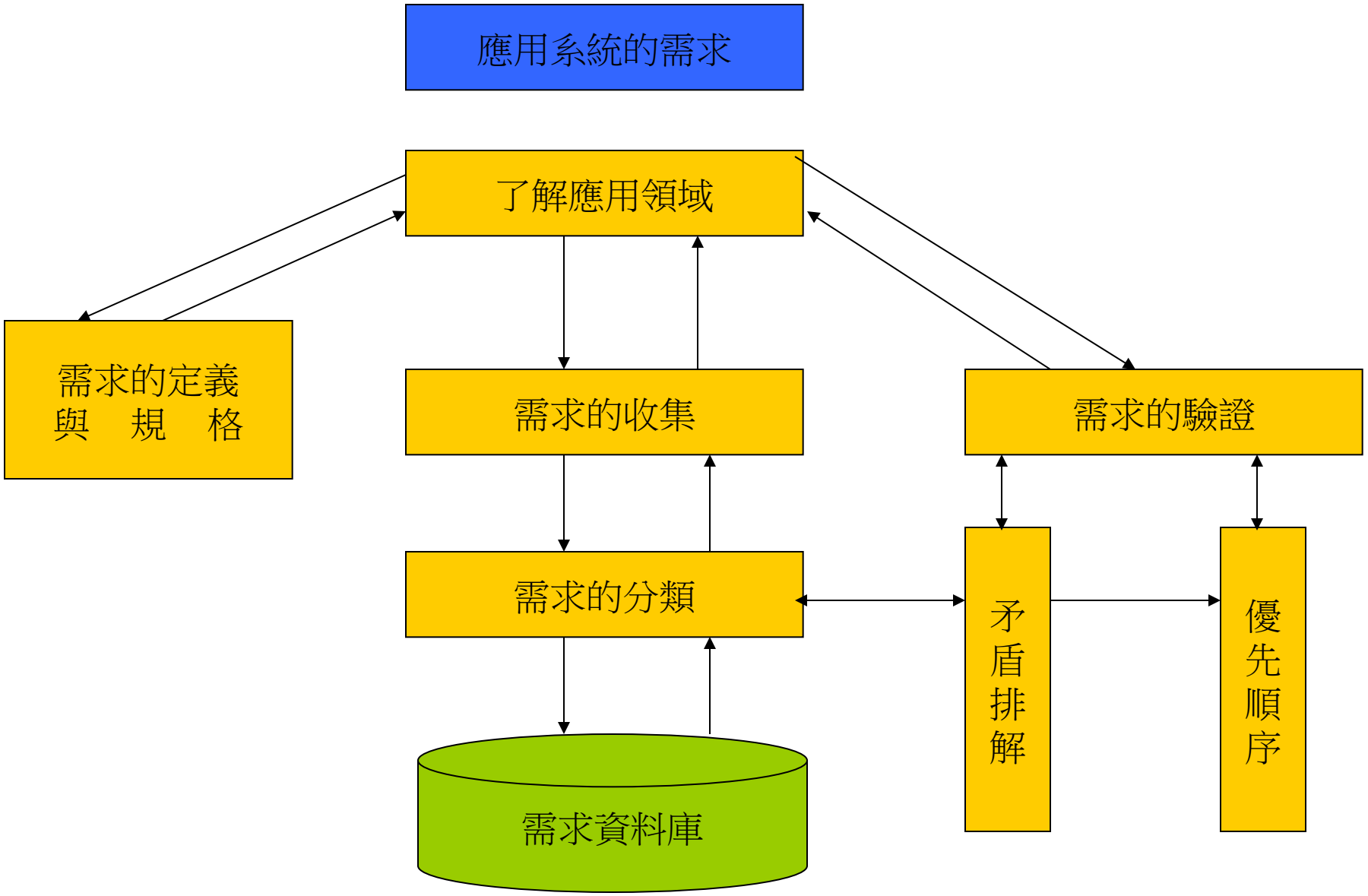
- 軟體開發的第一個步驟就是需求的建立。
- 這個階段得到應用系統的功能，以及使用上有哪些限制條件。
- 需求工程的產出就是軟體系統的規格：
  1. 需求即客戶的需求
  2. 需求規格就是系統的功能與性能規格
  3. 軟體系統的規格屬於技術性的規格，後續設計及製作的基礎。
  4. 軟體系統規格與需求規格有對應關係
  5. 軟體系統規格涵蓋大部分細節。

應用系統的需求

- 系統規格經過確證 (Validation)後才可定案。
- 應用系統的需求會隨時間或作業改變。
- 需求改變造成系統設計及製作上的變更。

需求分析的流程

- 一定要有領域的專家參與。
- 進行需求的收集，分析之後建立文件，可以資料庫管理。
- 消除互相衝突或類似的需求，得到完整的系統需求規格。


應用系統的需求
了解應用領域
需求的定義
與　規　格
需求的收集
需求的驗證
需求的分類
矛盾排解
優先順序
需求資料庫

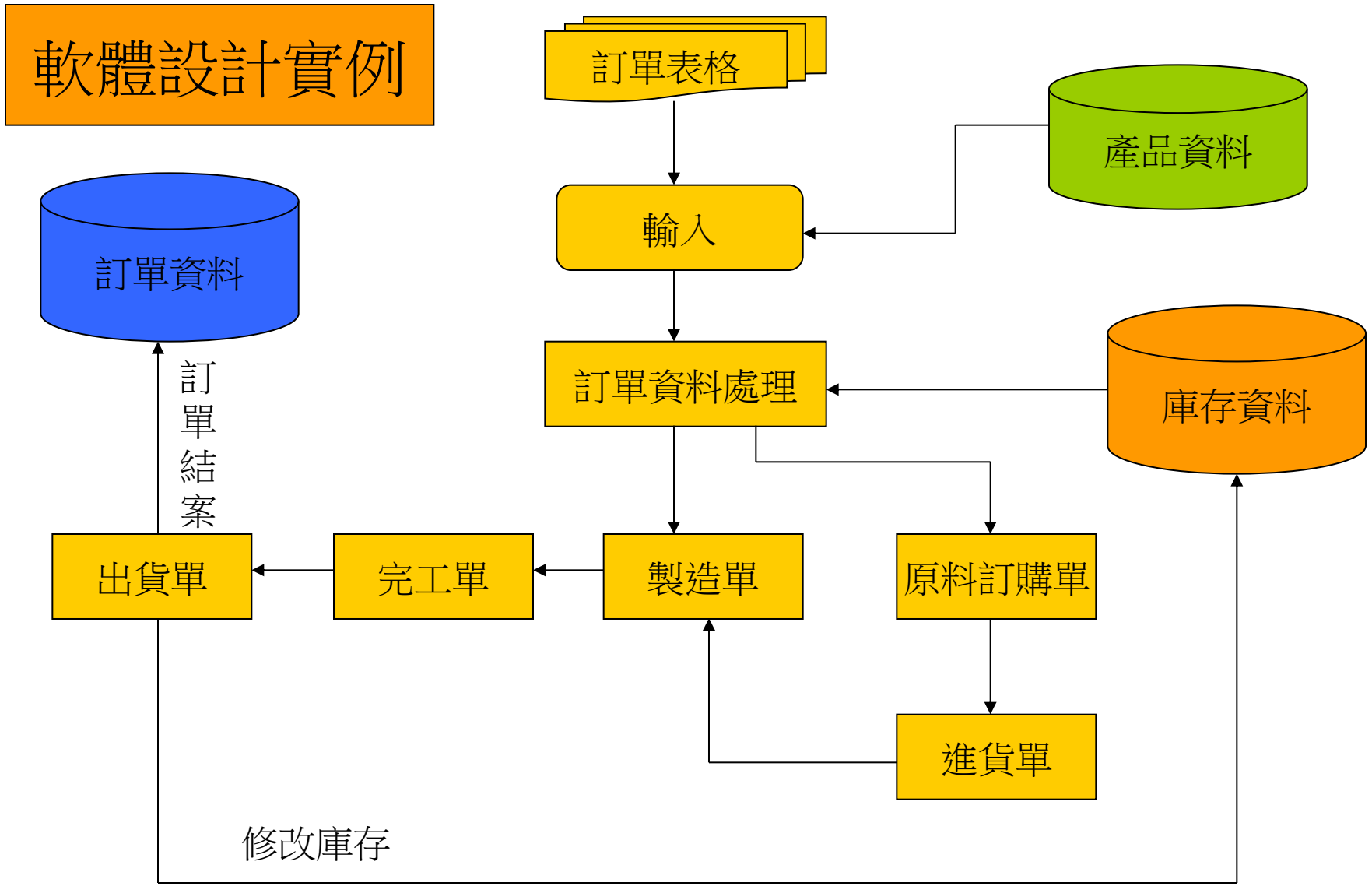
分析方法

- 分析方法：例如:資料流 (Data-flow analysis)的分析。
- 分析結果的表示：例如資料流程圖。稱為系統模型
- 系統模型的規範：系統模型有既定的規範，使系統開發人員統一的溝通標準。
- 分析方法防止不良分析設計，基於分析與設計的理論與過去的開發經驗。

二種模型表示法待續1

- 資料流模型 (Data-flow model)：資料流模型描述資料在軟體系統中被處理的過程。
- 訂單處理資料流的例子。
- 資料的流向是訂單作業的縮影，對系統開發者，相當於各軟體模組輸入、出的資料項目。

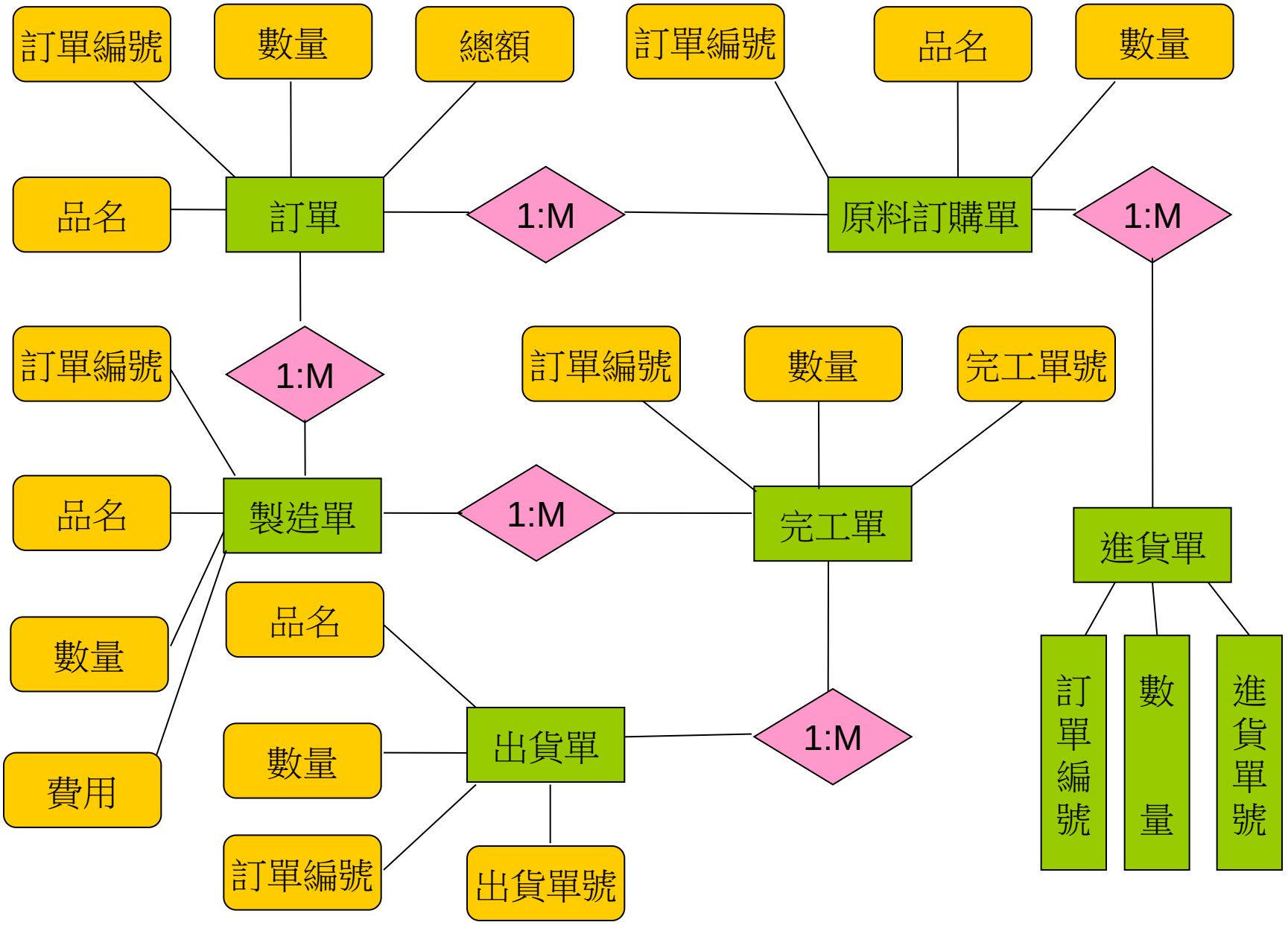
軟體設計實例



二種模型表示法

- 語意資料模型（Semantic data model）：描述資料的型態與資料的關係。
- 訂單處理的各種資料描述。
  - 矩形代表所描述的資料項目
  - 相連的橢圓形代表資料項目的資料屬性
  - 菱形代表所連接的資料項目的關係
  - 1:M代表訂單和製造單之間有一對多的關係，一張訂單可能會產生多筆製造單。

語意資料模型


訂單編號
數量
總額
訂單編號
品名
數量
品名
訂單
1:M
原料訂購單
1:M
訂單編號
1:M
訂單編號
數量
完工單號
品名
製造單
1:M
完工單
進貨單
數量
品名
訂單編號
數量
進貨單號
數量
出貨單
1:M
費用
訂單編號
出貨單號

需求的定義與規格

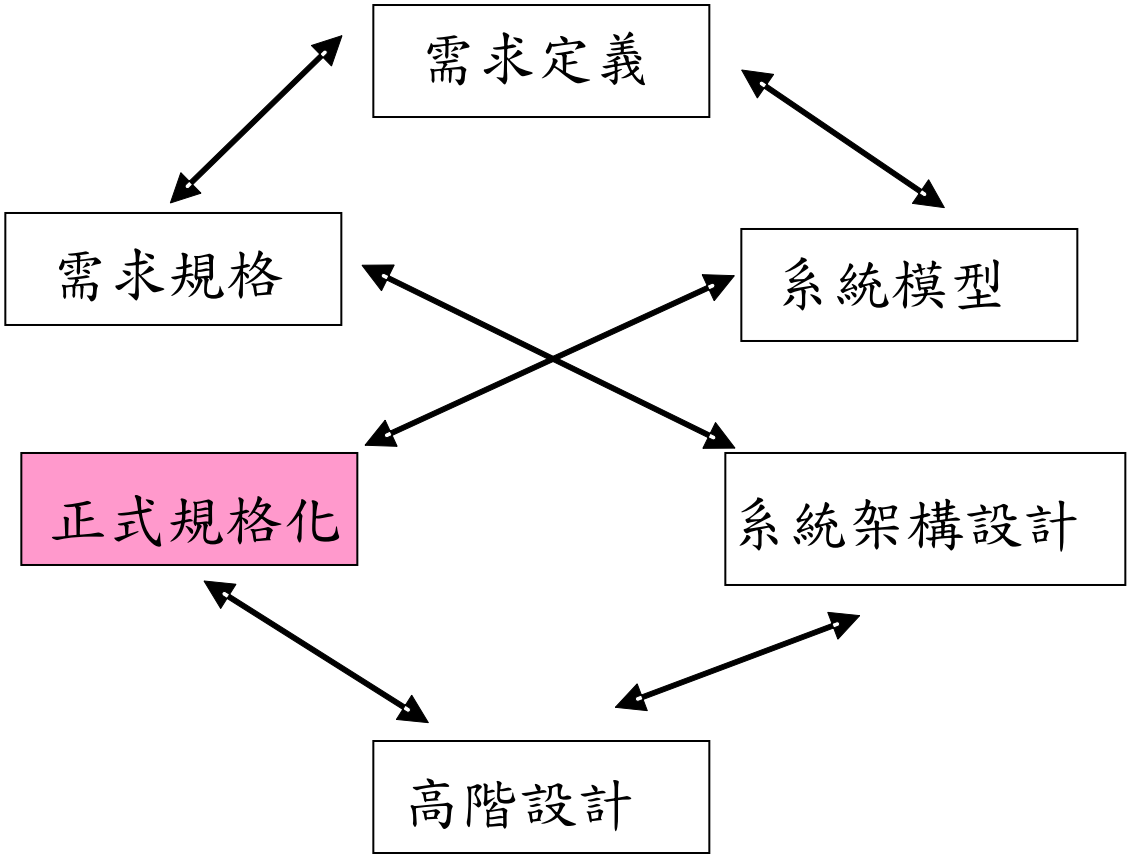
- 軟體系統的需求分功能性的需求（Functional requirements）與非功能性的需求（Non-functional requirements）。
- 功能性的需求與軟體系統必須提供的功能相關。
- 非功能性的需求涵蓋了與功能無關的其他需求。
- 需求的定義要有觀念性描述與技術的細節。
- 需求規格為了避免誤解，使用正規的方式描述。

描述需求規格常見的方法

- 需求規格語言：特定的語法及語意描述需求規格，配合工具的使用簡化描述的工作。
- 圖形表示法： 以圖形的方式來描述需求規格。
- 結構化的自然語言：結構化的定義加強自然語言的描述能力。
- 數學表示法：以正規化的數學表示法描述需求的規格。
- 類程式語言的表示法：使用類似於程式語言的語法與語意，以定義系統的作業方式為主要目的。類程式碼 (Pseudo-code)是典型的例子。

正式規格化

- 正式規格化（Formal specification)以數學理論為基礎，嚴謹規範系統規格
- 加入軟體開發處理:



近代需求決定方法

- 聯合應用程式設計(JAD，Joint Application Design)：專案的分析師、使用者、管理者都參與，由系統分析師主導。
- JAD參與的人多，平均每個人發言的機會較少，發言的可能傾向某一類意見，有人完全不發言，都是JAD潛在的問題 。
- 群組支援系統(Group support system)：適度克服JAD的缺點，例如讓JAD參與者匿名地輸入意見。
- CASE工具：運用在系統開發初期。包括規劃的工具、繪圖的工具與雛形化的工具。
- 雛形化(Prototyping)：簡易雛形的建立，早一點體驗系統的初步功能與輪廓。某些需求較無法確定，可透過雛形評估。

系統模型

- 系統模型的分析。
- 了解資料流程圖(DFD，data flow diagram)的繪製與使用。
- 認識資料塑模(data modeling) 。
- 了解資料塑模處理。
- 資料塑模的方法。

系統模型（System models）的定義

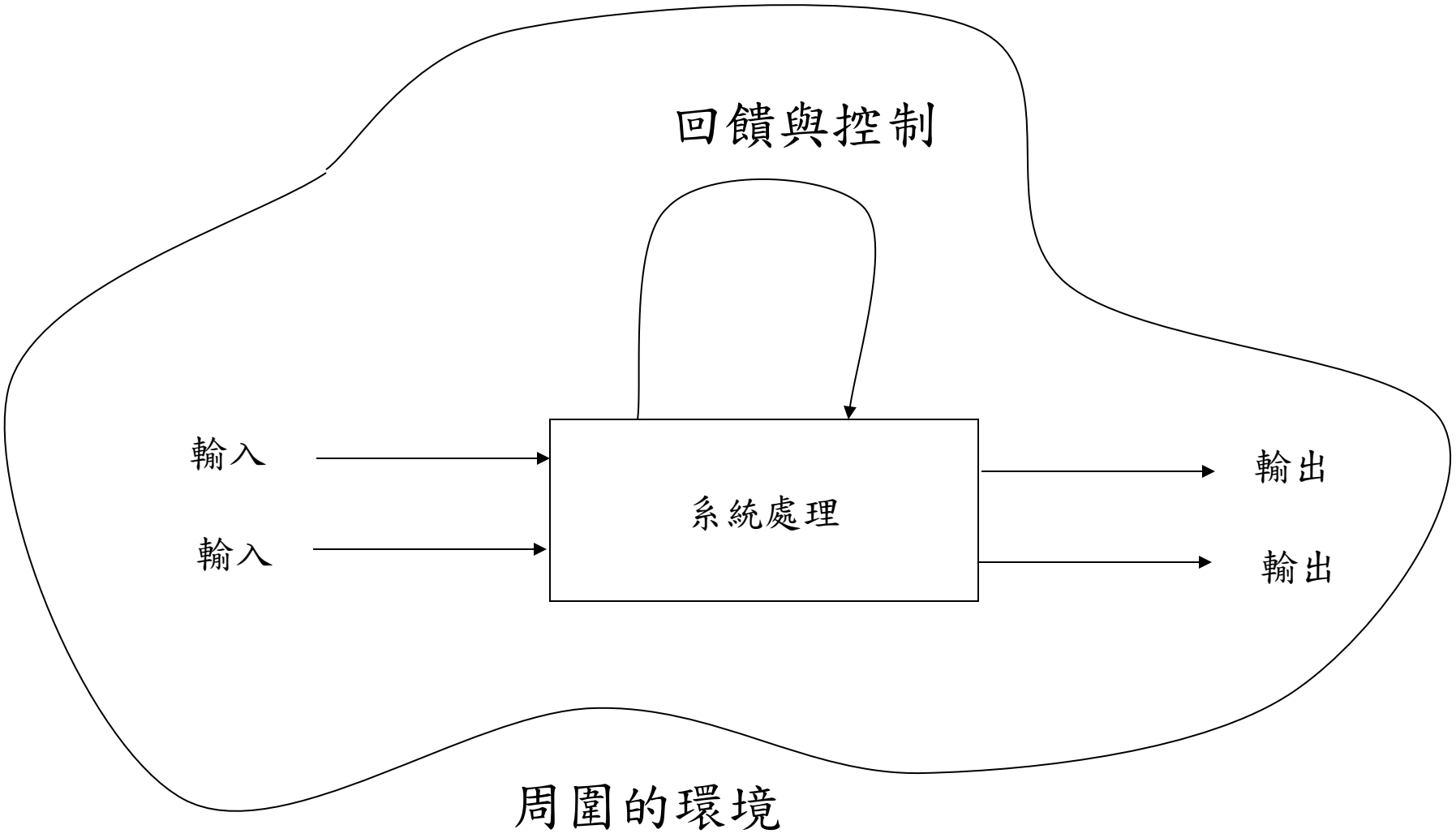
- 模型描述真實世界的一部分，模型建立過程稱為塑模（modeling）。
- 系統模型把需求資訊結構化，更清楚地描述一個軟體系統。

邏輯模型與實體模型

- 系統分析與設計的各種模型分為邏輯模型(Logical model)與實體模型(Physical model)。
- 邏輯模型描述系統本身及其功能，與系統如何實作出來無關的。
- 基於邏輯模型，系統用什麼方式實作都可。
- 邏輯模型也稱概念模型(Conceptual model) 或商業模型(Business model)
- 實體模型加上實作的方法與技術，進一步描述邏輯模型的內容。
- 實體模型必須考量技術限制，又稱為實作模型(Implementation model)或是技術模型(Technical model)。

處理（Process）的概念

- 處理是一個資訊系統的基本組成。把描述一個資訊系統的處理都找出就能把資訊系統拼湊出來。
- 處理描述業務事件(business events)，把資料處理成資訊。
- 在處理的塑模過程中，除了解其特性與功能外，還要確立處理與周圍環境、其他系統，以及其他處理的關係。
- 系統本身就是一個處理。以一個長方形框出系統的範圍，之外是系統的環境。
- 系統與環境透過輸入與輸出溝通，環境持續會變化，系統透過回饋與控制調整自己以適應環境。
- 處理代表輸入或發生了事件進而完成的一件工作。
- 處理塑模同樣有邏輯處理塑模(Logical process modeling)與實體處理塑模(Physical process modeling)之分。
- 邏輯處理說明完成什麼工作，實體處理則說明如何完成。

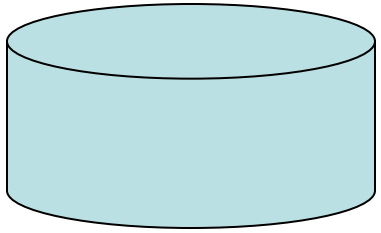

回饋與控制
輸入
輸入
系統處理
輸出
輸出
周圍的環境

資料流程圖
(Data flow diagram, DFD)

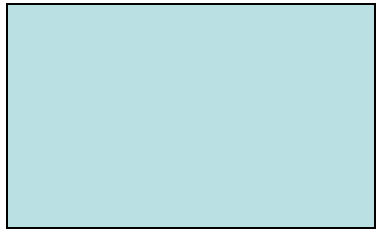
- 處理塑模是傳統的軟體工程。
- 資料流程圖描述系統資料的流動，及系統進行的工作。
- 資料流程圖使用以下的表示方式：
  1. 圓角的方形代表處理(process) 。
  2. 方形代表外部實體(external agents)，例如資料的來源(source)或是接受資料的地方(sink)。
  3. 開放的區域代表資料儲存(data store)，表示檔案與資料庫。
  4. 帶箭頭的線段表示資料流(data flow)、輸入，或輸出。

資料流程圖的表示法

- DFD表示法採用Gane & Sarson的圖示，DeMarco & Yourdon也提出類似的方式，略有不同，但大同小異。

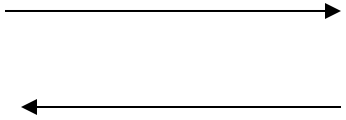


處理(process)



實體(entity)

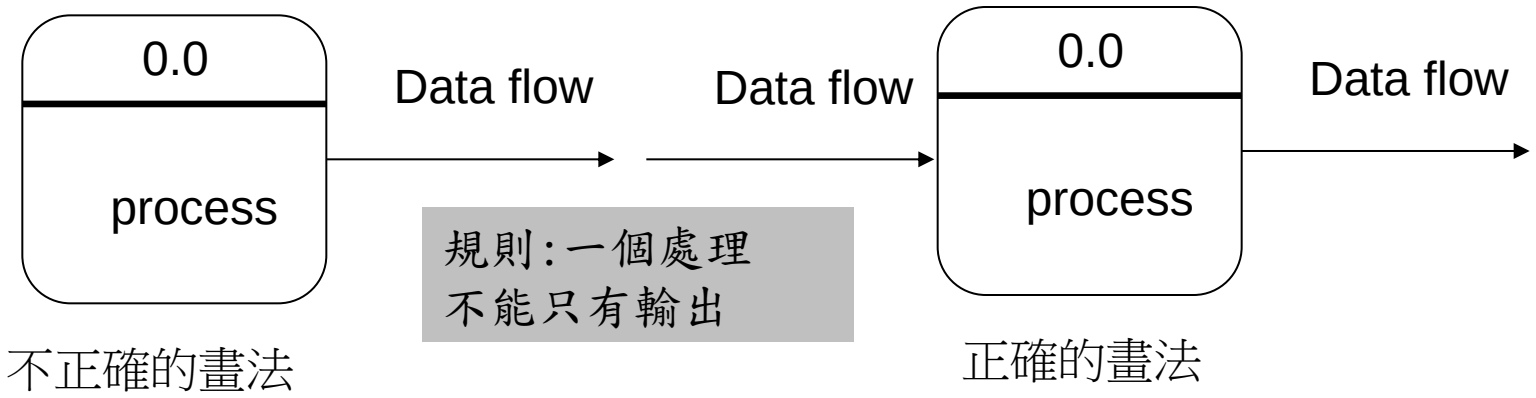
資料儲存區(data store)



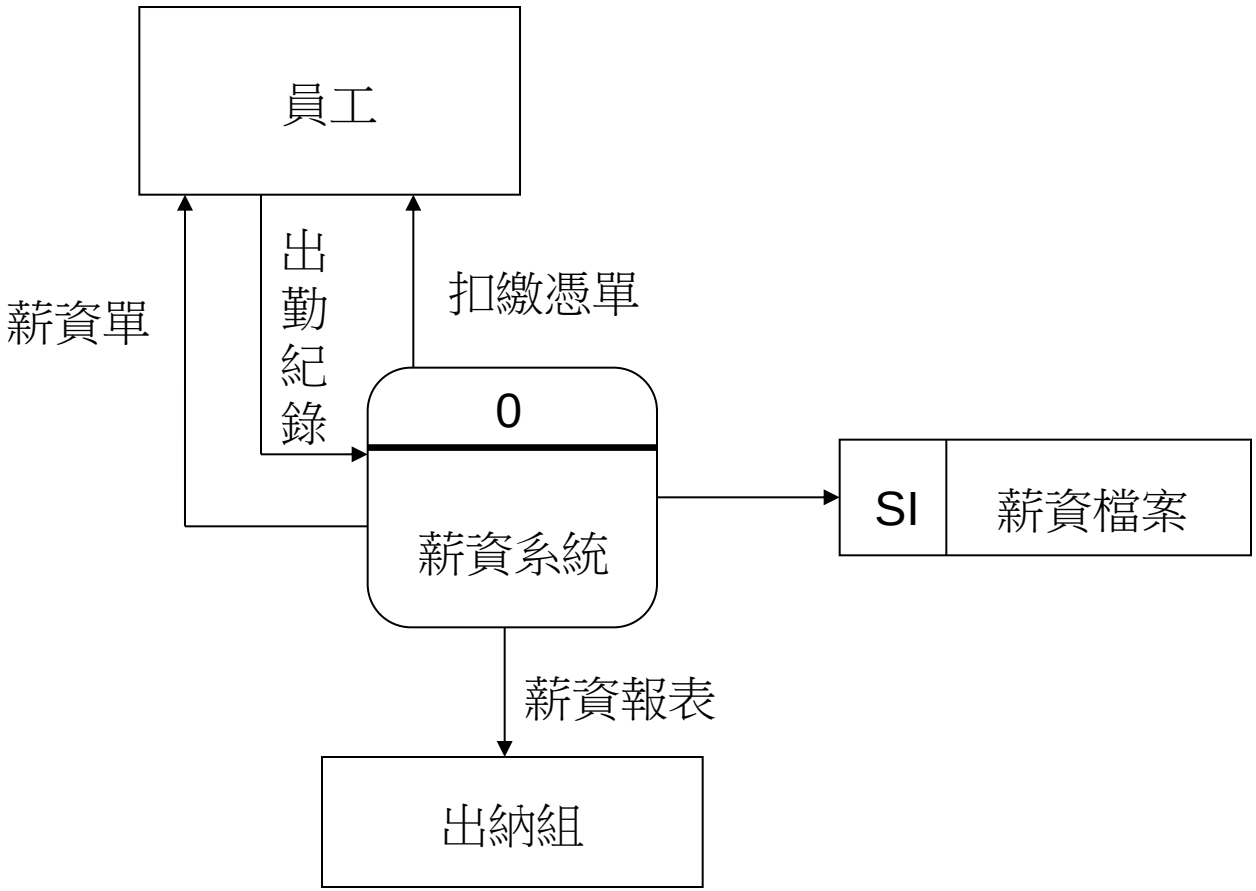
資料流處理(data folw)

資料流程圖繪製的規則

- 繪製資料流程圖必須遵循一些規則。



資料流程圖範例



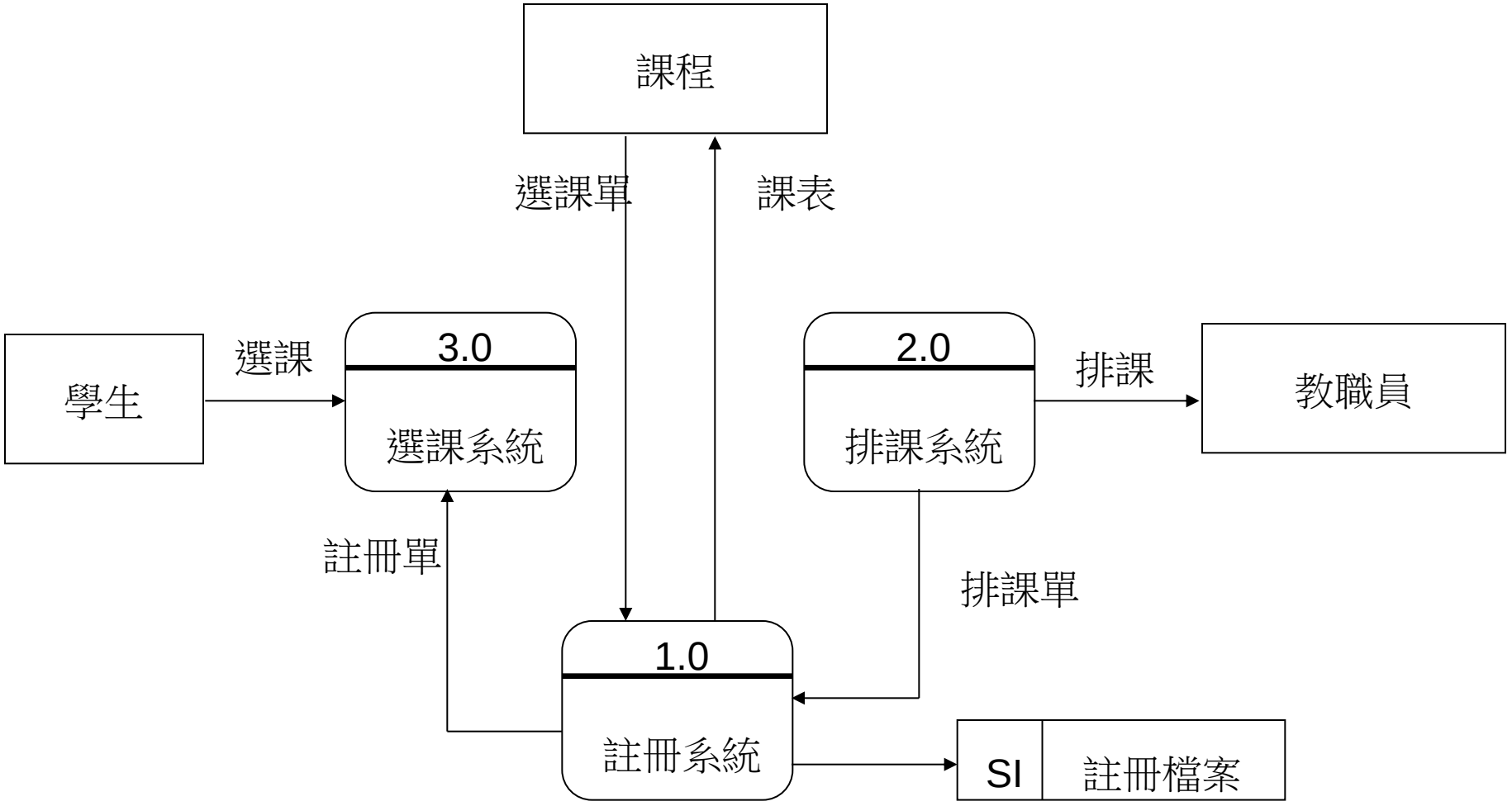
描述處理邏輯

- 資料流程圖找出系統的處理，並標示出處理之間關係。
- 每個處理內部詳細的運作邏輯並沒有記錄，要靠邏輯塑模(logic modeling)來說明。
- 有工具可以運用，不必跟程式語言有關。
- 工具包括純粹結構化的語言描述、決策表格(Decision table)、決策樹(Decision tree)與狀態變化圖(State-transition diagram)等。

Level-0 DFD

- Process 0由其他的處理組成，為第0層的資料流程圖 (Level-0 DFD)。
- 原來的source/sink都一樣，拆成多個處理完成process 0的功能。
- Level-0 DFD是系統最頂層的處理表示。
- 系統分析師可以進一步地了解資訊系統，並畫出Level-0 、Level-1、… 、 Level-n的DFD ，產生詳細的系統處理模型。
- 這是一種系統化與結構化的系統分析方法。
- DFD的資料流並沒有說明發生的時間、資料量或是資料流發生的頻率
- 很多細節沒有交待，因為DFD描述的是邏輯概念的資料流程。

第0層的資料流程圖（Level-0 DFD）



資料塑模（Data modeling）

- 資料塑模針對系統資料的特性進行組合與記錄。
- 資料模型在實作上通常採用資料庫的技術。
- 資料塑模是所有塑模技術中最重要的：
  1. 資料是資訊系統的核心，由多個處理共用，反映出系統的主要需求。
  2. 資料的定義比較穩定而少有改變，比起系統的處理容易確認多了，規模也比其他模型小。
  3. 資料塑模建立了後續溝通的共同術語與規則
  4. 完成資料模型後，系統分析者更容易了解系統的各種需求。

資料塑模的方法

- 不同的資料模型都會有特定的資料模型的表示法。
- 熟悉表示法可以看得懂別人畫的資料模型圖，或是自己也可以繪製。
- 資料塑模的關鍵在找出資料模型的內涵。
- 物件導向的分析與設計方法中，常透過情節(scenarios)與使用案例(use case)了解一個系統需求
- 試著從use case找出系統資料
- ER模型，實體(entity)是首先要找的
- 物件導向模型，則是先找出類別(class)。

資料塑模常用的方法

- 使用者面談，從溝通中找出關鍵詞彙，了解名稱與術語代表涵義。
- 面談中直接詢問使用者有那些資料需要使用、儲存或產生。
- 從現有的表單或報表中找出需要定義與使用的資料 。
- 透過找到的資料，運用技巧或方法確立實體或類別，例如CRC、腦力激盪等。
- 運用CASE工具，利用反向工程(Reverse engineering)技術從已存在的資料庫推演出資料庫定義。

從軟體系統的規格（specification）到設計

- 分析到設計的關鍵。
- 軟體系統設計的處理。
- 軟體設計的方法。
- 如何運用軟體系統設計的方法。

軟體系統設計

- 需求分析會得到軟體系統的具體規格文件。
- 軟體系統的設計則是系統實作前的步驟。
- 將規格化的需求轉換成具體的系統設計藍圖。
- 設計流程(Design process) 分成幾個不同階段的細部設計。
- 設計的優劣對於軟體系統影響很大，但設計流程並不像需求規格化那樣地嚴謹，常因開發者的技能、偏好與背景而異。

設計流程的步驟一

1. 架構設計 (Architectural design)：決定軟體系統的子系統之間的關係。
2. 抽象規格 (Abstract specification)：建立子系統的抽象規格。
3. 抽象指規格的表示不受限於工具或實作方式，是較高層次的設計。
4. 軟體規格 (Software specification) ：軟體規格確定系統各部分的功能。
5. 介面設計(Interface design) ：子系統間的介面設計，例如API的設計。

設計流程中的步驟二

6. 組件設計 (Component design)：子系統內組件的設計、及其交互作用。對於子系統的描述更詳細。
7. 組件規格描述組件設計的結果，為軟體規格的一部份。
8. 資料結構設計 (Data structure design)：資料結構定義系統的各種資料，是系統實作時需要的資料。
9. 資料結構規格是系統實作的基礎之一。
10. (Algorithm design) ：演算法描述系統的邏輯，也就是系統的執行過程
11. 演算法規格是系統實作的依據。

常見的系統模型

- 資料流程模型（Data flow model）：描述資料的流程，資料在各子系統的進出狀況。
- 實體關係模型（Entity-relationship model）：描述系統的實體 (entity)及實體間的關係。
- 實體的涵意很廣，系統的人、事、物，甚至抽象的觀念，都可以是實體。
- 結構化模型（Structural model）：描述系統內的所有組件，及之間的交互作用。

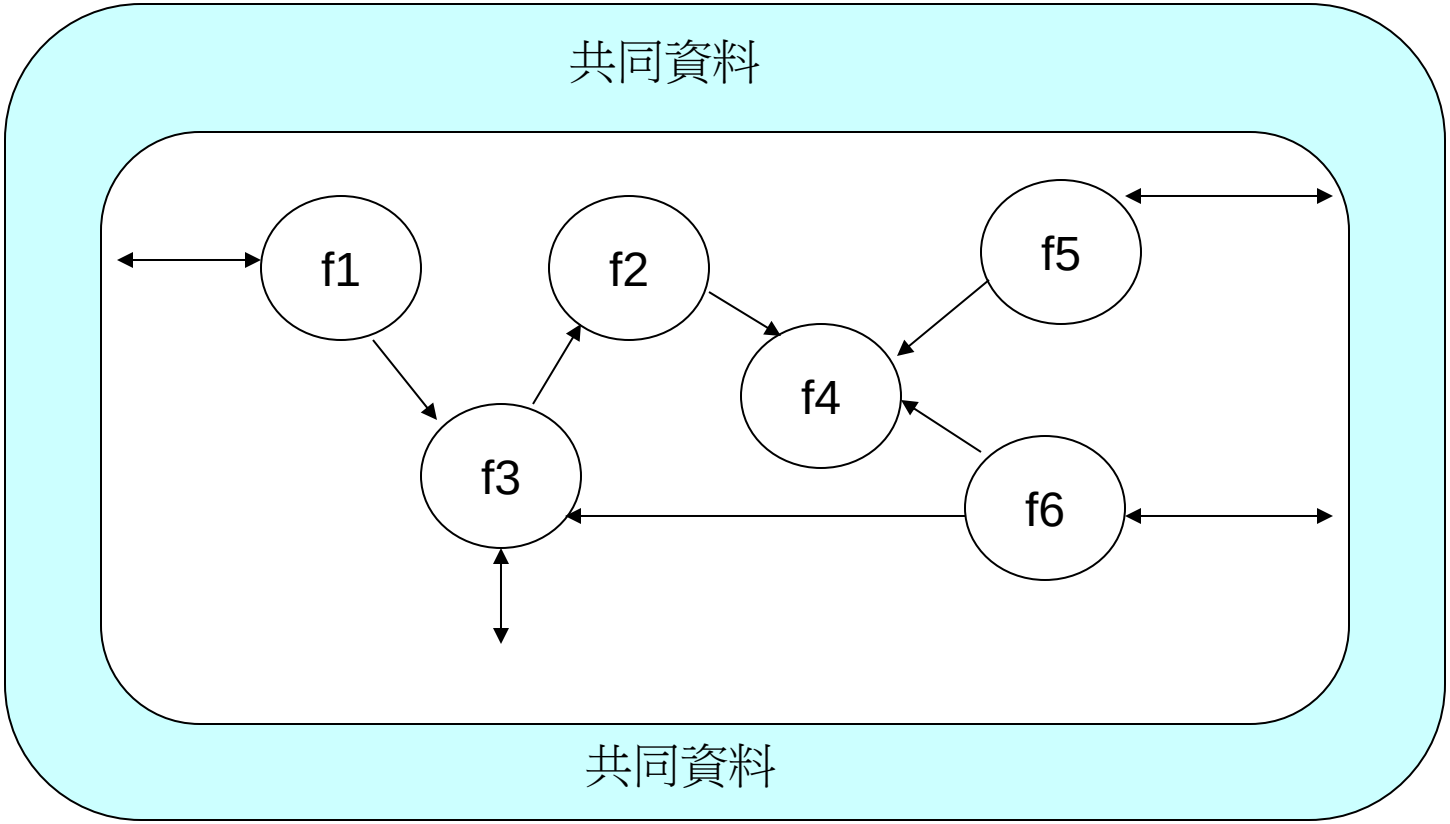
物件導向和軟體系統的設計

- 物件導向的設計方法成熟，物件導向簡化設計。
- 物件導向設計有很多選擇，有開發工具輔助。
- 軟體設計以產生品質好的軟體系統為目的。
- 良好的系統設計必須易懂、易維護、有彈性，子系統間要能密切合作，但相依性越少越佳。

函數導向的設計

- 函數導向的設計適用於共用資料少的系統，因不受之前所輸入資料的影響。
- 函數導向的設計，各函數內部演算方式是獨立的
- 假如共用資料多，軟體系統的維護會變得很複雜。

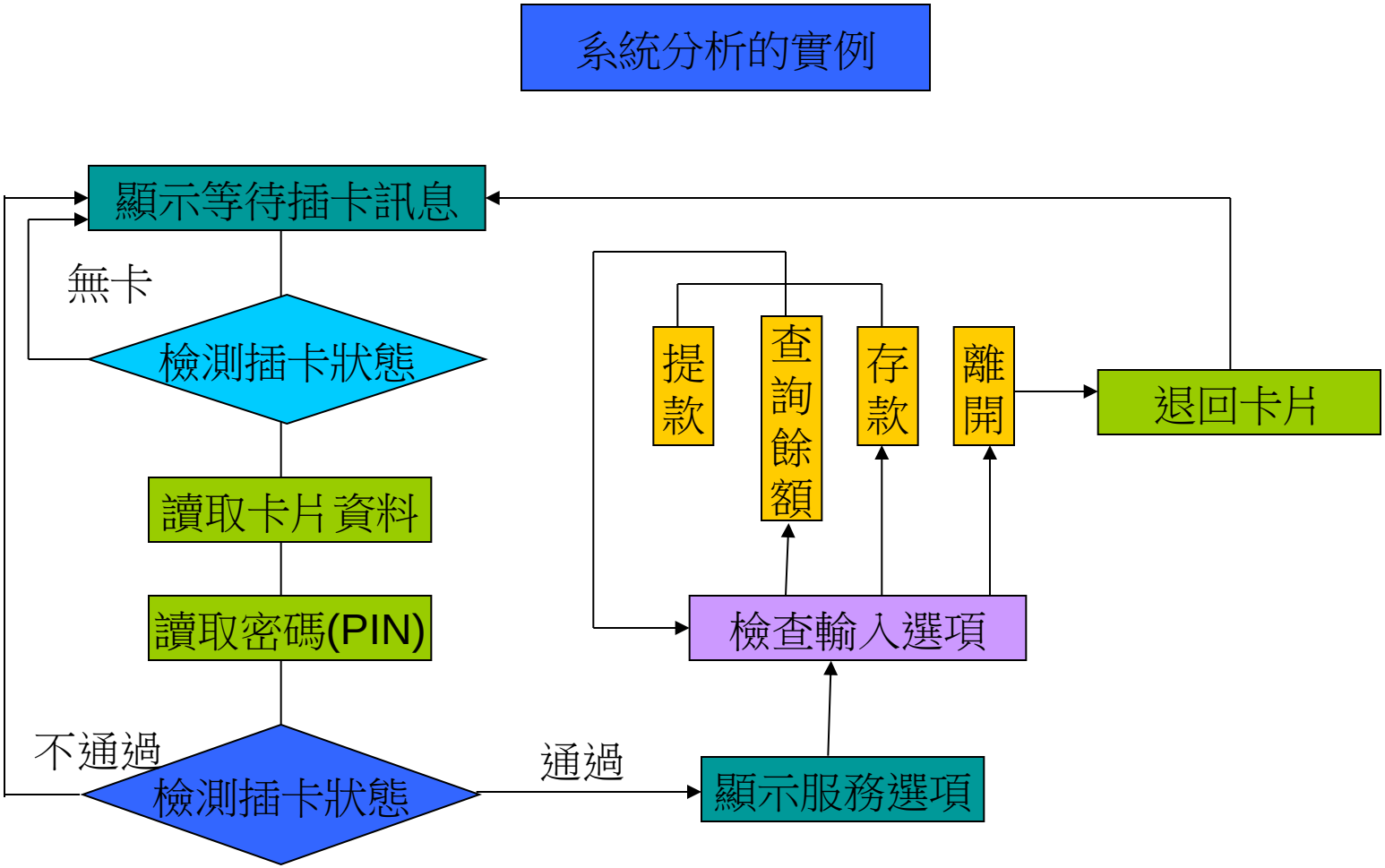
函數導向設計的基本觀念



函數導向設計的步驟

- 函數導向的設計有3個主要步驟：
  1. 資料流程(Data-flow)設計：利用資料流程圖描述資料在系統中被處理的過程，以找出負責處理的函數。
  2. 系統分析所得到的系統流程模型是資料流程設計的基礎。
  3. 結構化的分解(Structural decomposition)：描述函數如何分解成副函數(Sub-function)。
  4. 細部設計：描述細節，包括資料的定義與程式的控制結構(Program control structure)。

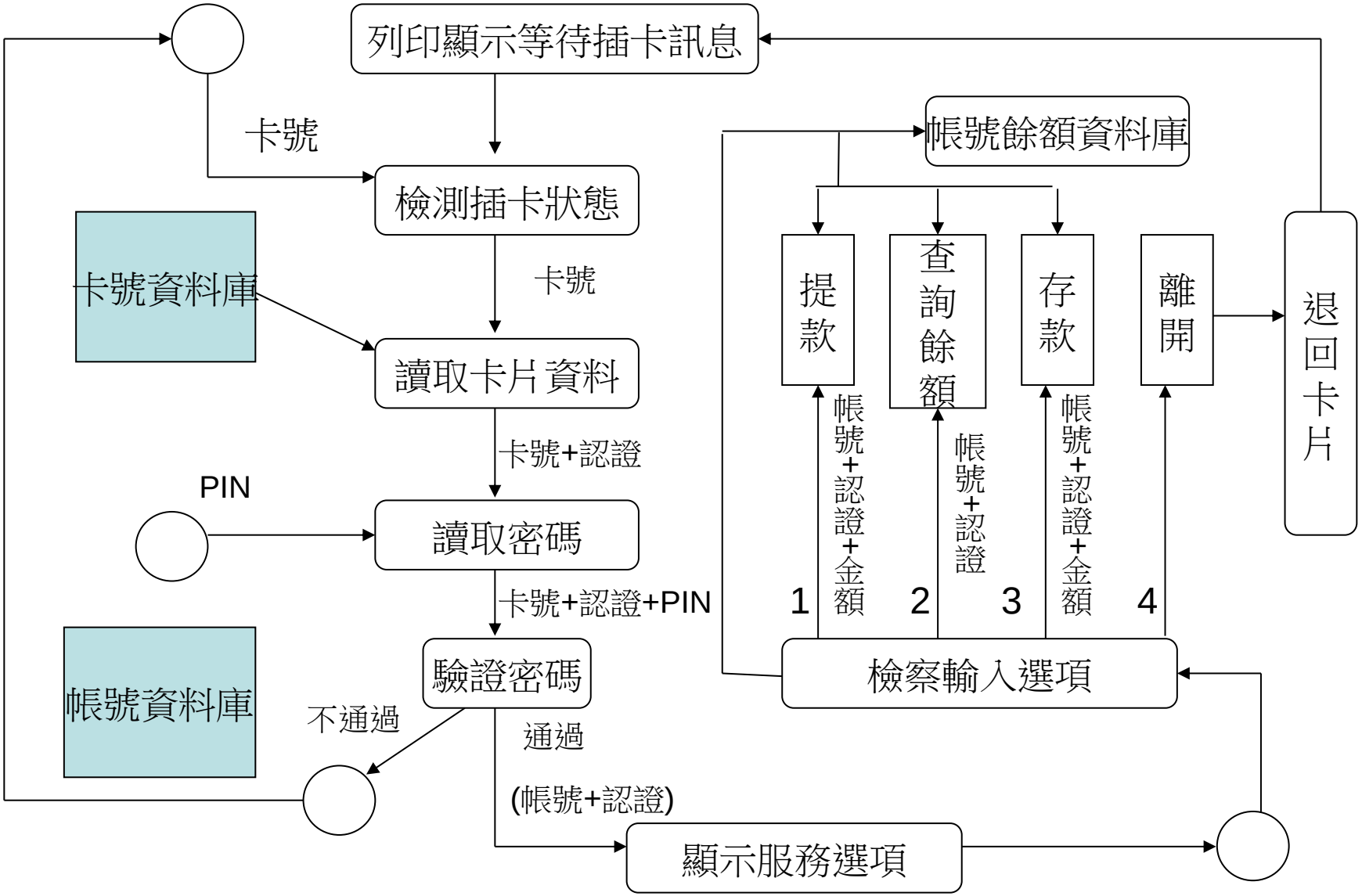
ATM系統的功能性設計



資料流程圖轉換成函數與副函數的結構圖

- 輸入資料的轉換：資料流程圖中的一個步驟分解成細部的副函數。
- 資料的轉換：輸出資料的格式化與準備等處理。
- 系統對於資料處理的轉換：資料輸入與輸出的處理之外的任何處理。

ATM系統的資料流程設計



軟體設計方法實務

- 開發工具提供的開發環境可了解軟體設計方法的用途。
- 整合性開發環境(IDE，integrated development environment)包括一般的功能選單，可容納多個工作視窗。
- 編譯（Compile）、連結（Link）及產生可執行碼的工作在實作及測試階段進行。

軟體設計方法的實務

- 物件導向設計透過類別之間的繼承關係提升程式碼與設計的複用性(reuse)。
- 有穩固的觀念與豐富的開發經驗，熟悉新的開發工具時所花的時間也會比較短
- 物件導向的設計方法，基本的觀念是大同小異的。

軟體系統設計的實務

- 處理設計的工作。
- 結構圖的繪製與使用。
- 資料流程圖如何轉換成結構圖。
- 使用者介面設計的內涵。
- 使用者介面的元件與設計的方法。
- 對話設計(dialogue design)。

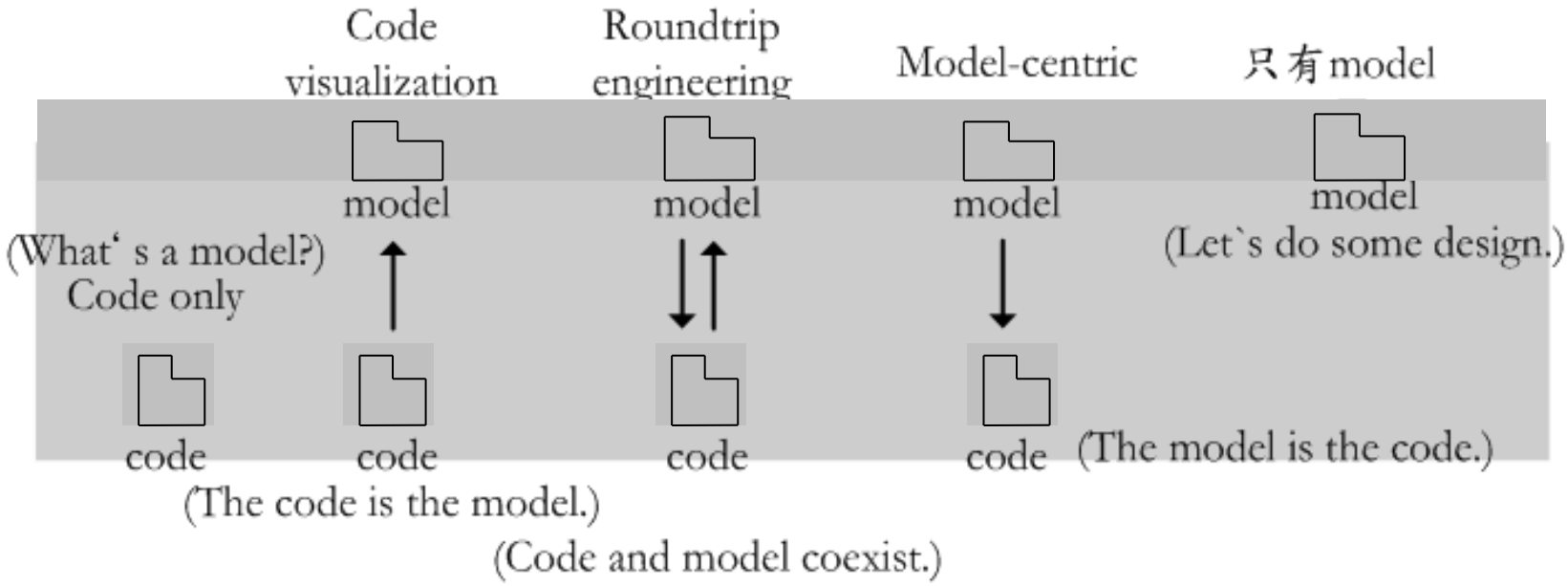
- 使用者介面(user interface)，不管軟體系統的功能多強大，若沒好用的使用者介面，無法發揮。
- 使用者透過使用者介面和資訊系統溝通、輸入資料、取得輸出，達到使用的目的。

處理設計

- 系統設計的過程，對輸入、輸出、使用者介面、資料庫、互動等系統的組成都進行設計。
- 系統內部的設計叫做系統的處理設計(process design)，得到系統的內涵(system internals)。
- 應詳細地描述，讓程式碼的撰寫有所依據。

建模與程式碼的兩極觀念

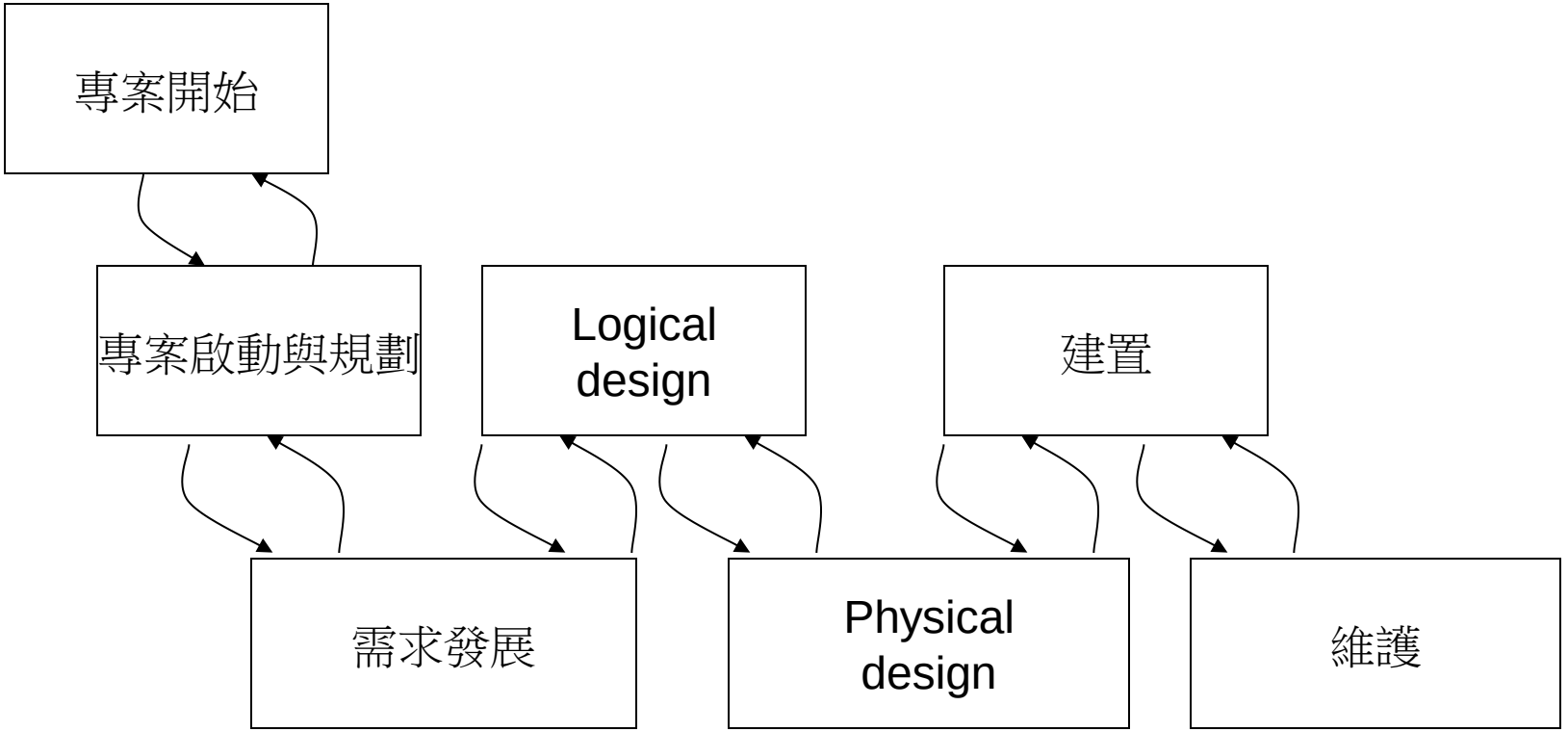
- 了解建模(modeling)與程式碼的兩極觀念。
- 正規的開發程序通常有分析與設計過程，為所要開發的軟體系統建立模型。



有模型再寫程式是開發軟體系統的正常流程

- 已有程式碼再試著去建立模型，是為了讓軟體系統好維護。
- 有些模型可以用來自動產生程式碼，節省軟體撰碼成本
- MDA技術的精神。

程式設計與開發流程



系統的處理設計（Process design）

- 系統分析的階段中繪製資料流程圖（data flow diagran）了解系統的處理(process)。
- 結構圖為系統結構的基礎，設計的結果影響整個系統的建置。

系統的處理設計

- 這個階段的設計會考量：
  1. 如何得到好的設計：結構化系統設計可以採用模組化方法（modularization）。
  2. 結合度(cohesion)：系統的每一個組件專注於一種功能。
  3. 連結性(coupling)：系統中不同組件間的相依性。

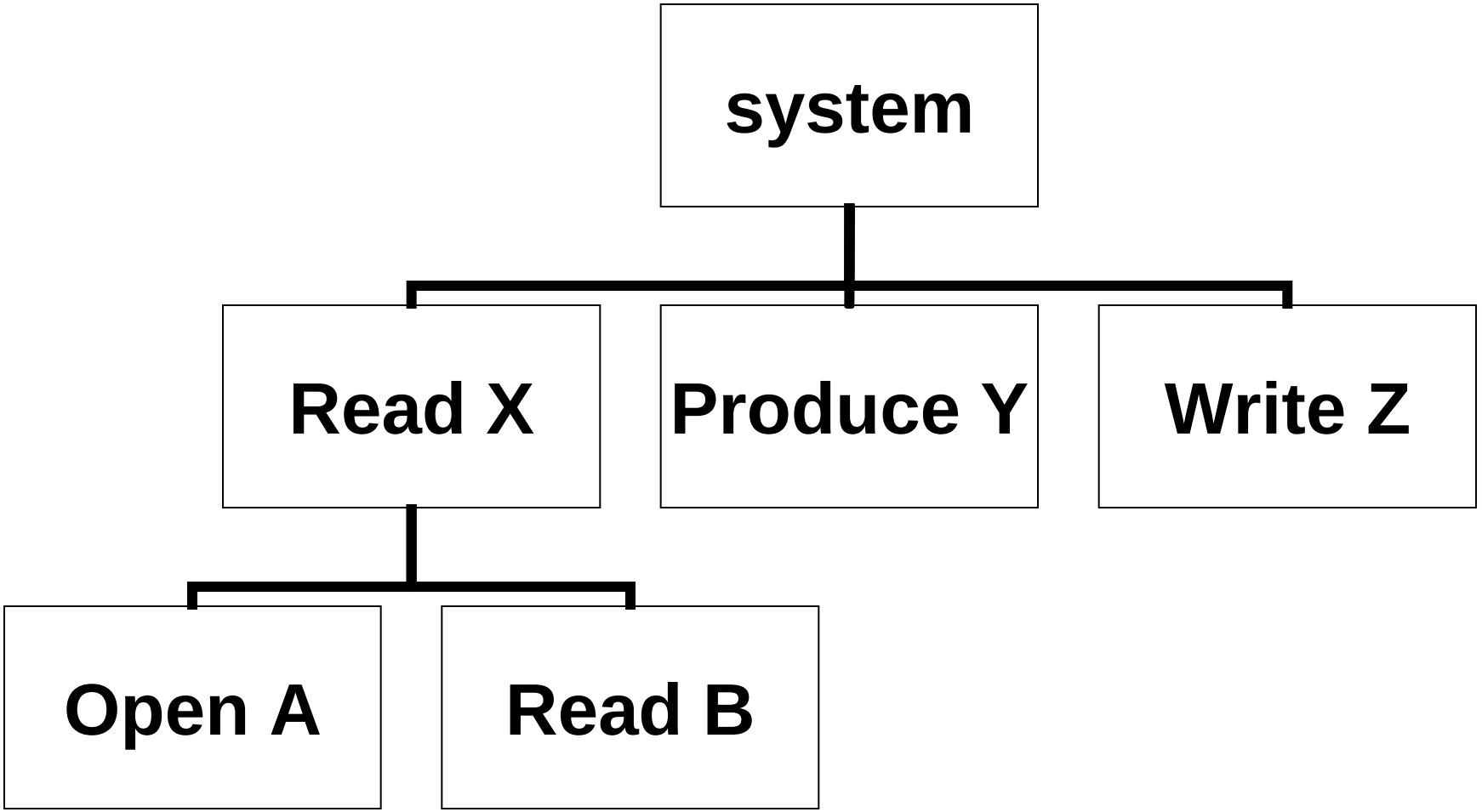
系統內部設計的產出結果

項目	細項	
Structure charts	含有細分的階層(fully factord)	
	包括data couples與flags的詳細描述	
Module specification	輸入(input)	
	資料庫的規格	
	其他輸入(input)	online/batch
		檔案(files)
		其他模組(modules)
	處理(Processing)	
	Pseudocode	
	Nassi-Shneiderman charts	
	輸出(Output)	
	檔案與資料庫的更新(Database and file update)	
	列印(print)	
	其他模組(modules)	

結構圖（Structure charts）

- 以階層(hierarchy)的方式
- 表示一個系統的組成。
- 描述系統及其組成之間的關係。
- 從結構圖裡看到系統如何分成多個組成的內部結構。
- 物件導向式的程式適合用狀態變化圖(state transition diagram)表示。

結構圖 (Structure charts)

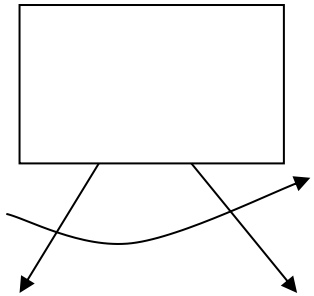


結構圖的圖示

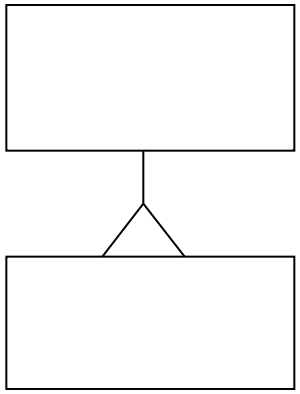
- 結構圖中的模組透過參數的傳遞溝通，參數具有資料的型式，以資料鍵(data couple)或旗標(flag)表示。
- 資料鍵代表兩個模組之間交換的資料，用有箭頭線段的空心小圓圈表示，箭頭的方向是資料的流向。
- 旗標代表兩個模組之間交換的控制類型資料或訊息，用帶有箭頭線段的實心小圓圈表示。
- 模組(module)是形成結構圖的基本單元
- 一個模組代表一個完整的系統組成。

結構圖的慣用表示圖案

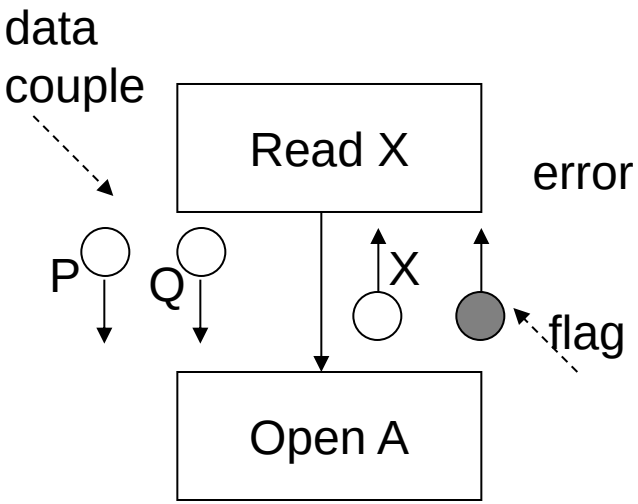
預先定義的模組

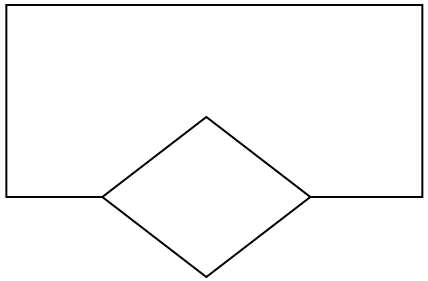


預先定義的模組



內嵌的模組





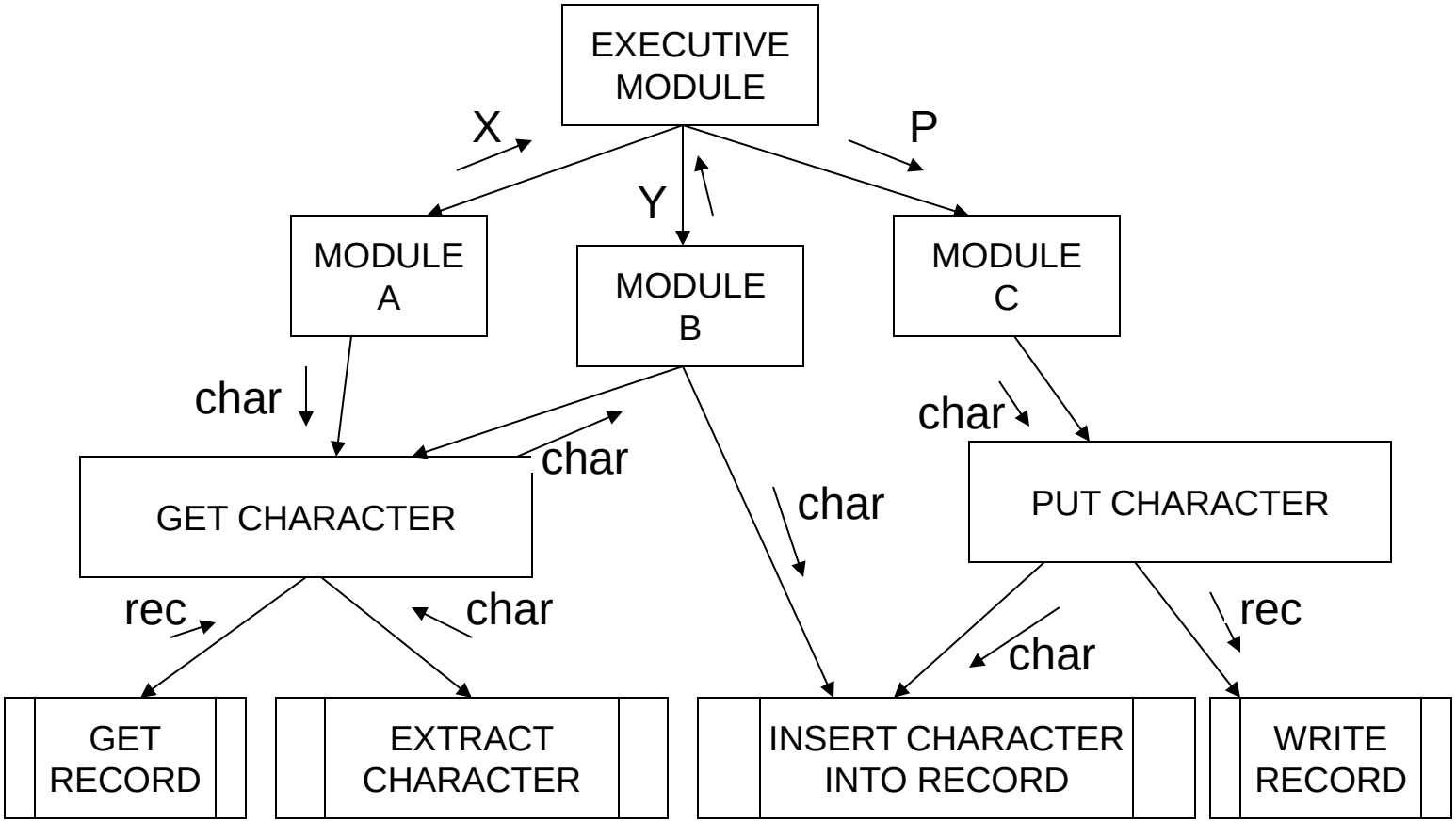
條件式的呼叫

從資料流程圖到結構圖

- 結構圖從資料流程圖轉換過來，方式有兩種：轉換分析(transform analysis)與交易分析(transaction analysis)。
- 跟資訊系統的類型有關，兩種資訊系統的類型：

1. 以交易為主的資訊系統(transaction-centered system)：系統的主要功能是將資料傳到適當的目的地。
2. 以轉換為主的資訊系統(transform-centered system)：系統的主要功能是從現有的資料產生新的資料。

結構圖
Structure Chart



處理設計的指引

- 怎樣才是好的處理設計呢？
- 進行系統處理設計時必須思考的問題。
- 基本要求:完成的系統容易了解、修改。
- 系統分析與設計的專業訓練中，一些處理設計的指引：
  1. 系統要有組織(factored)：系統應該要適度的分割，模組大小合宜，同時維持高度的結合性。
  2. 模組控制範圍(span of control)：父模組不要有太多的子模組，會增加系統的複雜度。
  3. 模組的大小：一個模組的程式行數目約在50~100行。
  4. 模組的結合性(cohesion)：模組的功能應該單純，不要包括多種功能。
  5. 模組的共用：子模組的功能盡量讓其他父模組共用。

連結（Coupling）的種類

- 連結性，系統模組之間的相關程度，相關性越低越好。
- 常見的連結：
  1. 資料連結(data coupling)：模組之間因交換資料產生連結，資料做為模組的溝通可使得模組的連動性不高。
  2. 戳記連結(stamp coupling)：模組之間因交換資料結構產生連結，壞處是讓系統複雜了。

3. 控制連結(control coupling)：模組之間因交換控制資訊而產生連結。
4. 共用連結(common coupling)：模組使用全域資料(global data)而產生連結，很多程式語言都支援這種用法。
5. 內容連結(content coupling)：模組可以直接引用其他模組內部的內容。

模組內涵的設定

- 結構圖記載了系統中模組的階層架構。
- 模組內部指令的執行細節可以採用下面二種方式來描述：
  1. 類程式碼(pseudocode)：跟程式碼類似，但沒有像程式語言那麼嚴謹的語法要求。
  2. 那西史奈德門圖(Nassi-Shneiderman diagram)：以圖示來表達程式的執行流程。

類程式碼

- 「類程式碼」(pseudocode)，它是一種介於程式語言與一般口語間的演算法敘述。

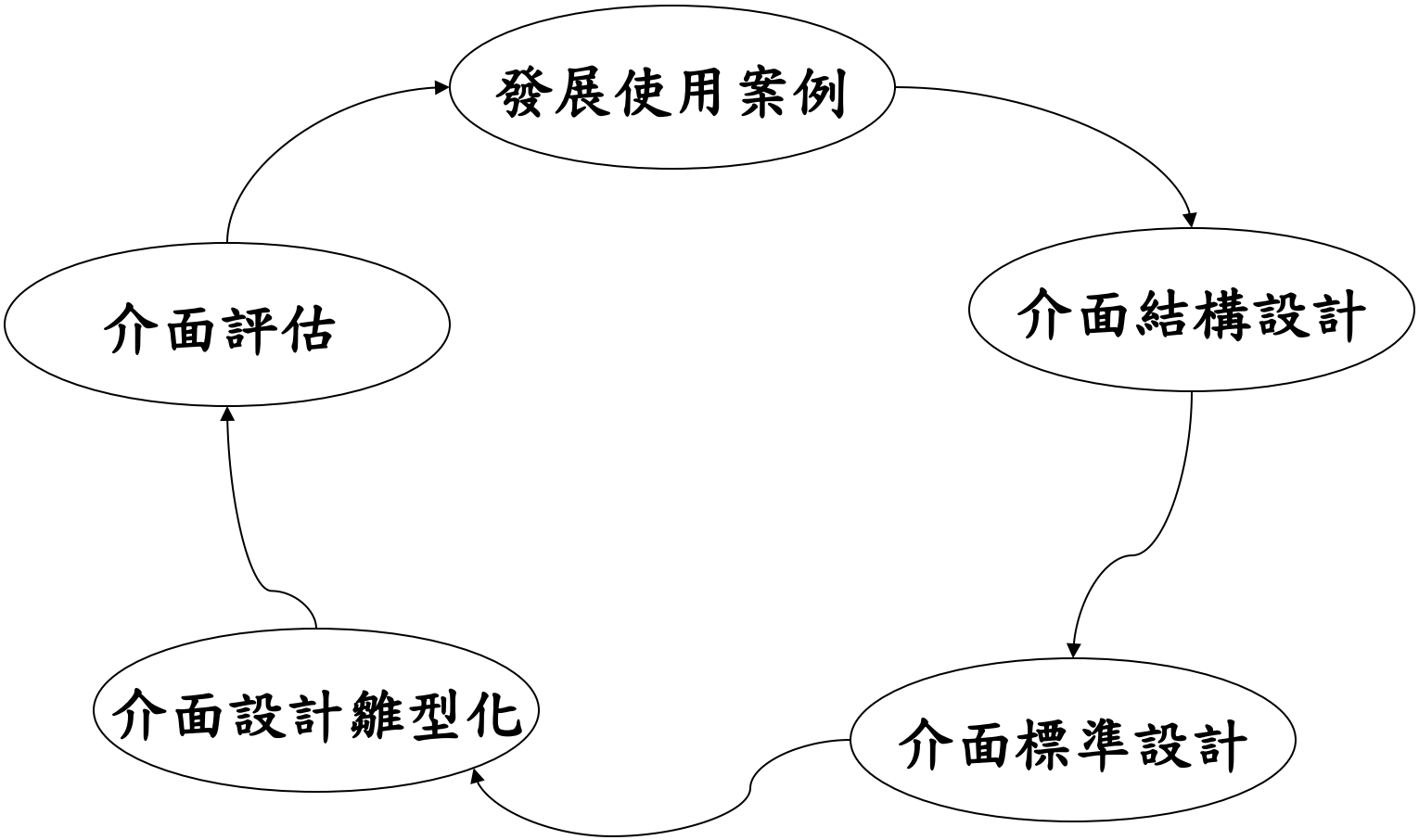
類程式碼的範例

- Count=0; //紀錄不及個人數
- Red_flag=0; //超過50人不及格則令紅旗為1
- Open student_record file; //打開學生紀錄檔案
- Read student_score;
- Repeat
- 	If student_score > 59
- 		Pass=trun //及格的情況
- 	Else { if Count>50 //不及格的情況
- 		Red_flag=1
- 		Else
- 		Red_flag=0;
- 		Count=count+1;
- 		}
- Until Eof=true;

使用者介面設計的處理與結果待1

- 使用者介面設計是以使用者為中心的設計活動。
- 介面之間的關係：介面標準的設計(interface standards design)讓介面的溝通具一致性。
- 完成設計以後可以建立雛形，做為初步評估的基礎。
- 使用者介面設計的活動

使用者介面設計的處理與結果END



物件導向軟體工程待1

- 物件導向技術。
- 物件導向模型的內涵。
- 物件導向模型的特性。
- 物件導向軟體工程。
- 物件導向分析與設計的歷史。
- 物件導向分析的方法。

物件導向軟體工程END

- 物件導向技術(Object-oriented technologies)是應用廣泛的技術。
- 物件導向技術有兩個主要的影響：應用系統的資料模型與應用系統的開發。
- 資料模型的影響來自於新興起的資料庫應用。
- 系統開發的影響來自於軟體工程及圖型化使用者介面(GUL Graphical User Interface)的進展。

基本觀念

- 物件導向技術以類別(class)與物件(object)的觀念為基礎，描述真實世界的事物。
- 類別有強大的抽象化與描述能力
- 物件有互動與動態特性。
- 系統分析與設計是物件導向技術應用的領域之一。

物件導向技術的應用

- 物件(Object)描述事物，形容實體的物質，也能說明抽象的觀念。
- 電腦作業系統的視窗(Windows)環境，是物件導向技術的一種應用。
- 視窗的視覺化使用者介面，可以完全用物件導向的觀念來詮釋。

物件導向技術應用的領域

- 物件導向技術應用最廣的領域是程式設計。
- 「物件導向程式設計」(Object-oriented programming)，運用物件導向技術來強化程式設計的效率。
- 節省系統開發的成本。
- 程式設計領域的成功經驗，使物件導向技術逐漸地被導入到各種應用上。

認識物件導向程式設計

- 「物件導向程式設計」（OOP, Object-Oriented Programming），「物件導向」與「程式設計」兩種技術的整合。
- 物件導向的觀念起源早，到近年才逐漸被理論化和系統化。
- 寫程式是很多人都有的經驗，目的是指揮電腦解決問題。

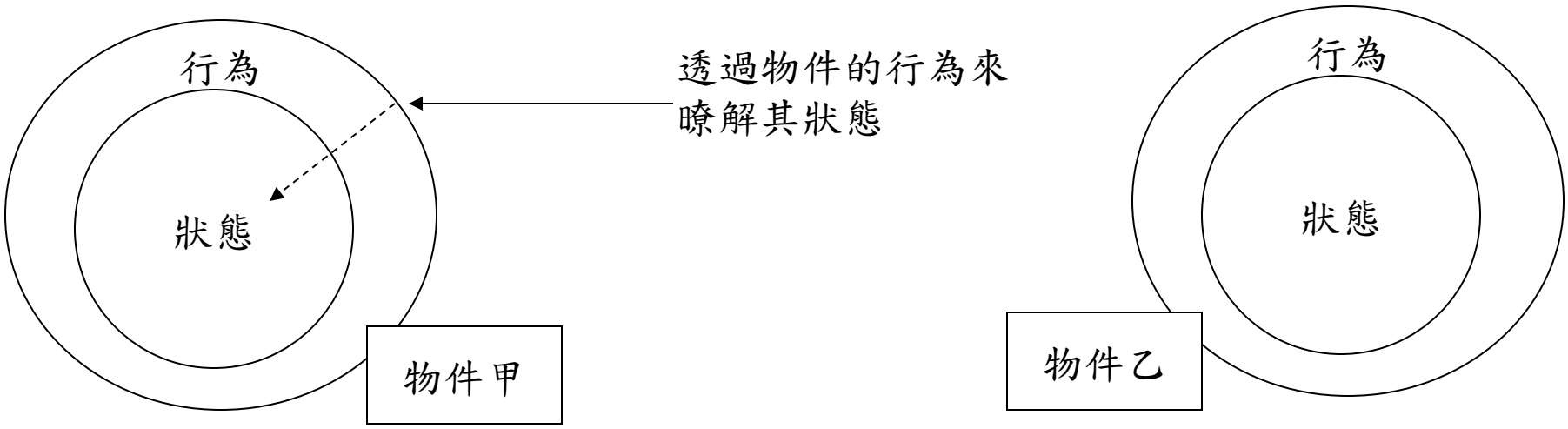
從物件與類別的故事談起

- 物件導向的程式設計是類別與物件為核心。
- 物件代表我們所要描述的世界中的事物，類別把這些物件分門別類。
- 類別之間有關聯，物件之間也會有各種關係與互動
- 分類就會產生架構。
- 問題分析後得到的類別與物件就形成了物件導向模型。

「物件」是什麼？

- 正統的說法以三個主要的特徵來描述：
  1. 物件的身份（Identity）：標示物件。
  2. 物件的狀態（State）：物件各種特性的狀況。
  3. 物件的行為（Behavior）：代表功能，或是對外來刺激的回應。
- 物件導向的封裝(Encapsulation)特性。

物件圖示



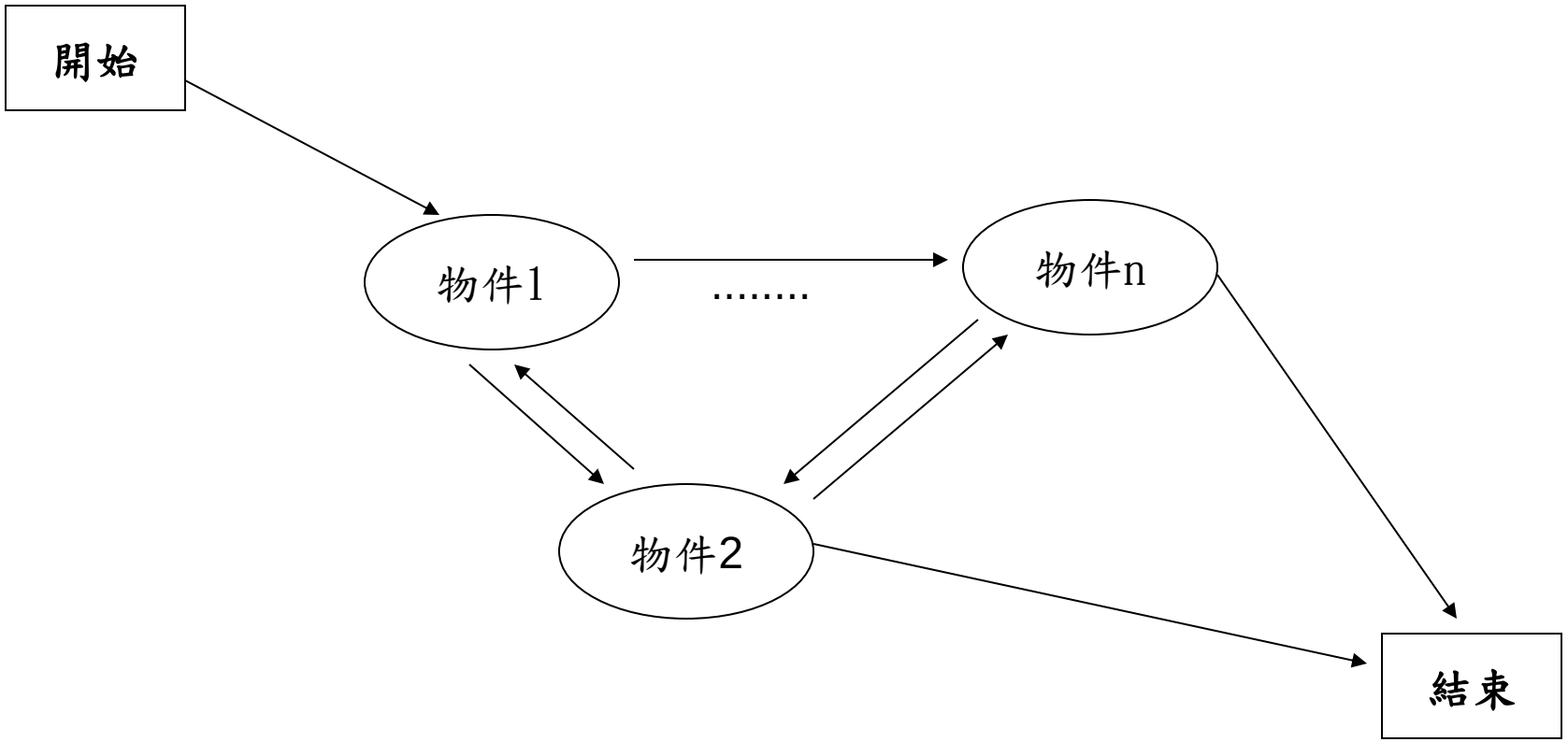
方法與屬性

- 物件的狀態用資料或資料結構來表示，資料在數值上的變化就表示物件狀態的改變，
- 物件的行為用處理(Procedure)來表示。
- 代表行為的處理常被稱為物件的方法(Method)。
- 物件的狀態則稱為屬性(Properties & Attributes) 。

「類別」是什麼？

- 「類別」用來將物件分類。
- 同一類別的物件具有很多相同的特徵。
- 類別是鑄造物件的模子，物件則是類別的實例（Instance）。
- 透過繼承(inheritance)定義新類別。新類別和舊類別之間就產生了subclass與superclass的關係。
- 繼承的好處是再利用(Reuse)，軟體再利用可提升開發生產力。

物件導向的程式設計觀念



程式語言中類別的觀念

- 物件導向程式設計的精隨在於類別(Class)與物件(objec)的運用。
- 程式語言嚴謹、精準。

程式語言的類別觀念

- 類別實際地將抽象描述給定義出來。
- 物件屬於同一類別，表現出類別的屬性與行為。
- 屬性是內涵，行為則像外在的表現。
- 在Java裡頭用程式變數(program variable)來定義屬性。
- 行為用Java的方法(method)來定義。
- 類別定義物件的對外介面，表明物件提供的服務，也定義了內部的實作(implementation)。

程式中的物件觀念

- 物件是類別的實例（instance）。
- 類別是建立物件的架構或模子。
- Java 程式中擔當大任的是物件。
- 物件的建立包括兩個主要的步驟：
  1. 參考變數(reference variable)宣告(declaration)
  2. 產生物件

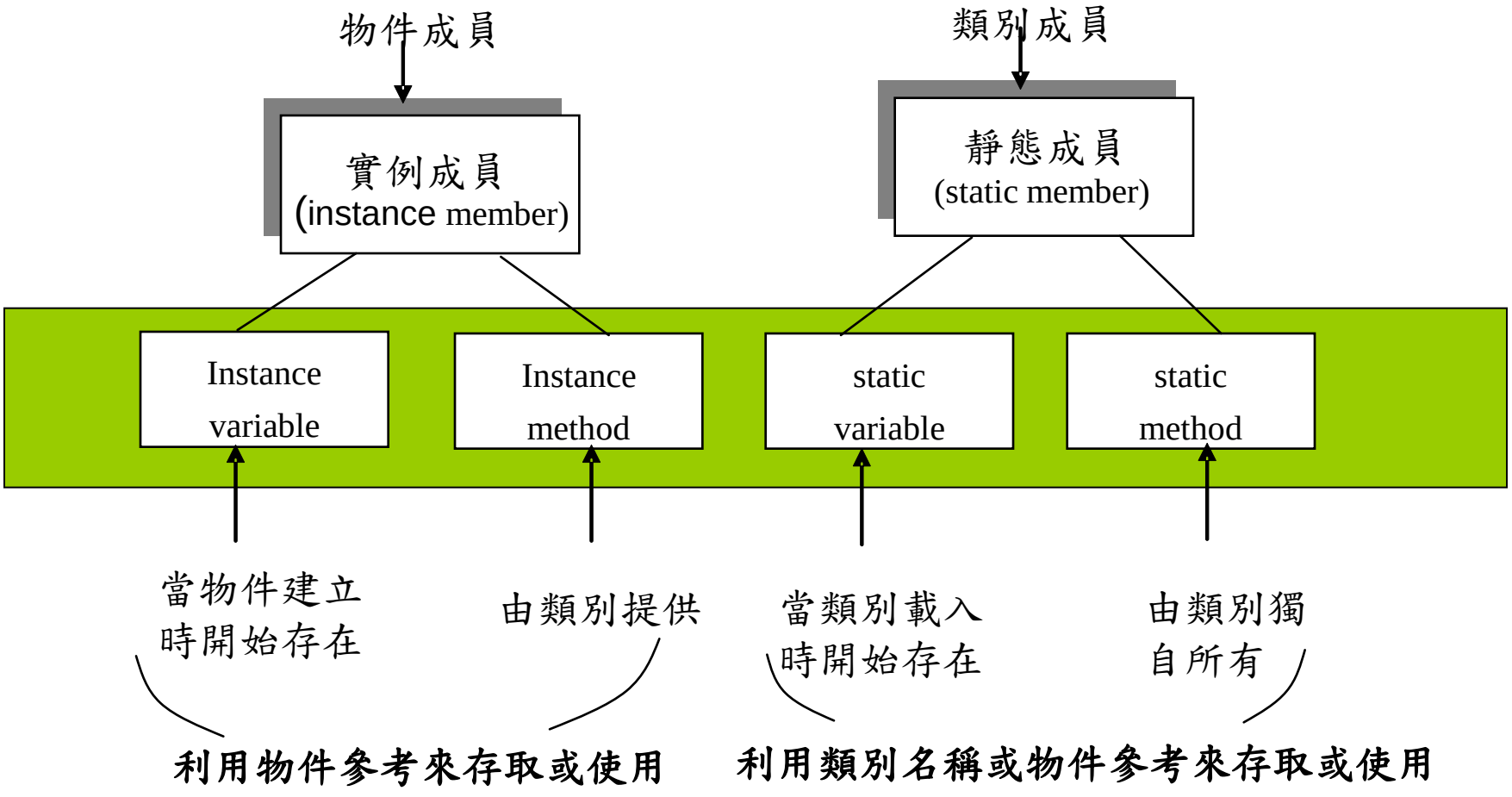
物件的建立

- 參考變數宣告：Java採用物件參照(object reference)來指名某一個物件。
- 建構物件：類別實例化(class instantiation)，利用建構子(constructor)來產生類別實例，也就是物件。
- 關鍵字new會使得系統傳回一個類別所屬物件的參照。
- 物件建構過程中，一些內部成員是因建構子的執行而建立的。
- 宣告和實例化兩個步驟在語法上可合併。
- Java，「垃圾回收」(garbage collection)，物件佔用的記憶體空間，最後能回收。

物件的成員與類別的成員

- 物件是類別的實例，擁有類別的屬性與方法，是物件成員(實例成員，instance member)。
- 物件的屬性成員的內容值，決定了物件的狀態。
- 方法成員則是同一類別的物件都相同。
- 類別可以有不屬於任何物件的屬性成員，叫做靜態成員(static member)。

物件的成員與類別的成員



物件導向模型

- 物件導向模型以物件與類別為基礎
- 運用在資料塑模上稱為物件導向資料模型(object-oriented data model)。

物件導向資料模型

- 資料模型描述應用系統的資料世界。
- 物件導向資料模型（Object-Oriented Data Model）以物件為基礎，有繼承（Inheritance）、封裝（Encapsulation）與多形（Polymorphism）等特性。
- 可用類別（Class）與型態（Type）分類物件。
- 物件由屬性（Attributes）與方法（Methods）所構成的。

物件與類別之間的關係

1. 根型態(root type)可簡化系統設計。
2. 型態(type)與類別(class)：type是抽象的定義，類別含有實作的部份。
3. 類別與物件的關係：依類別的定義產生物件。

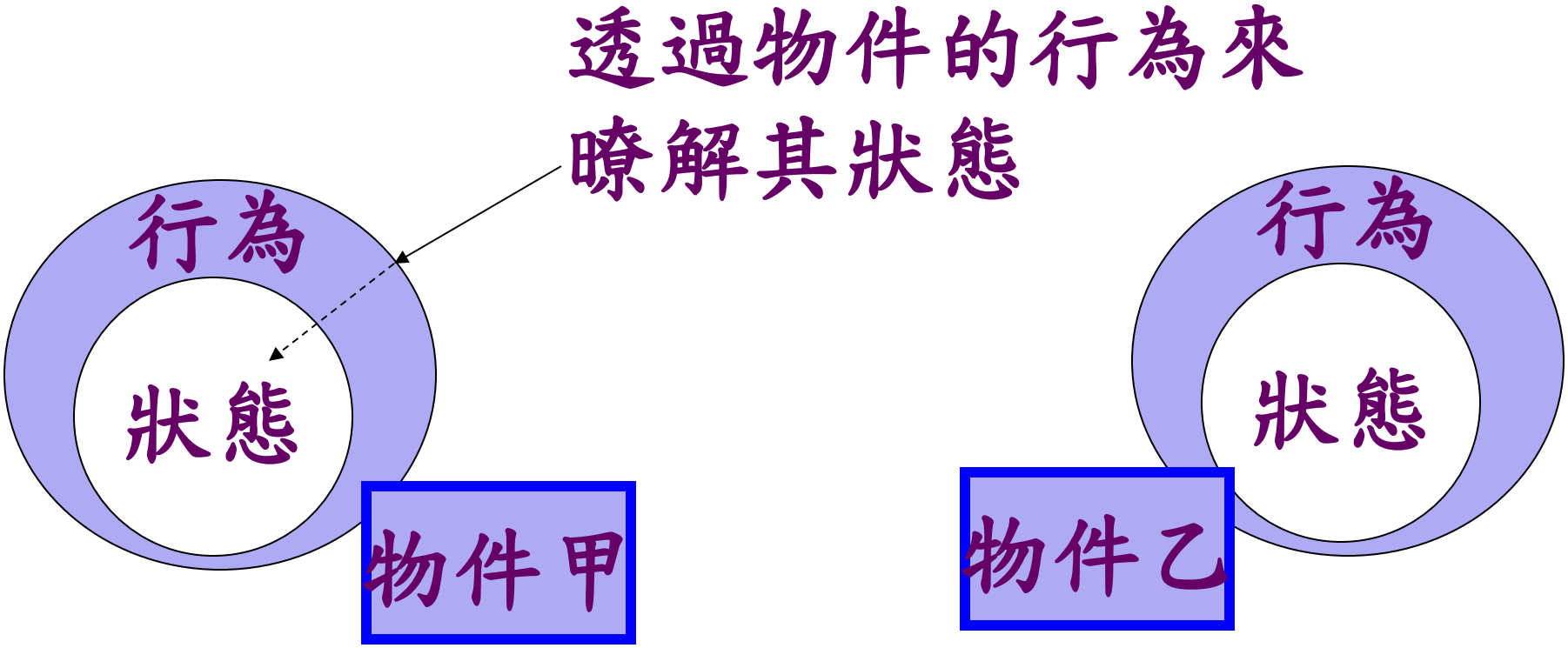
程式語言的資料模型

- 程式語言的資料模型亦可描述結構複雜的資料。
- 物件導向模型比較容易和程式語言整合。
- 關聯式資料模型則需要轉換。
- 程式語言所描述的複雜資料若無法透過關聯式資料庫管理系統處理，這現象撐為impedance mismatch。
- 物件導向資料庫系統沒有Impedance mismatch問題。

物件導向的特性

- 「物件」(Objects)描述物或觀念或事。

物件的成分與結構



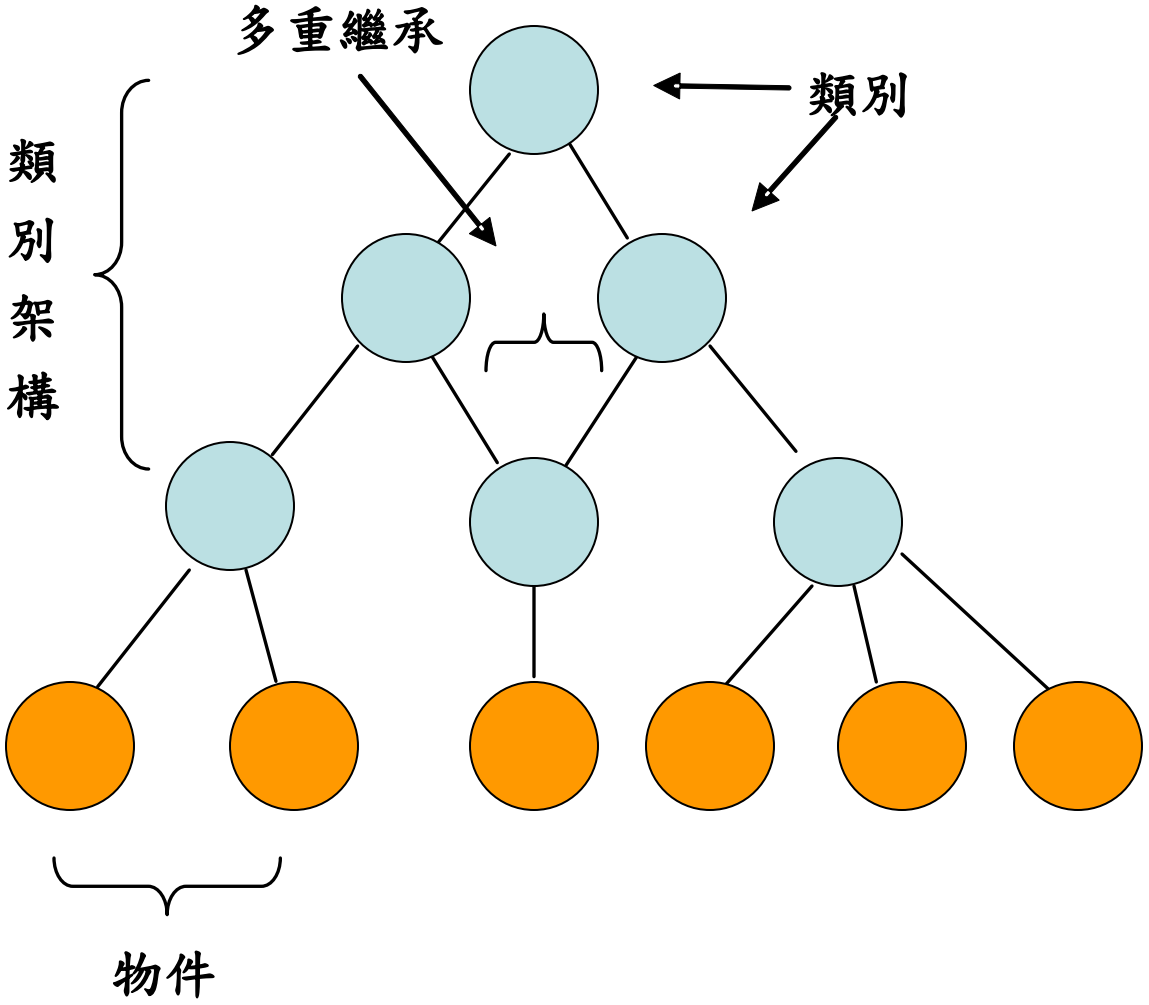
封裝(Encapsulation)

- 狀態必須經由介面與行為才能了解，狀態的改變決定於物件的行為。
- 封裝提供安全性，因為物件的狀態無法被外界直接存取，並且簡化了對物件狀態的了解。

繼承(Inheritance)

- 物件常有共通的狀態或是行為，歸納在同一個類別下，可以簡化物件的定義。
- 類別是物件的規格，物件是類別的具象化。
- 繼承是指把類別共通點抽出定義在新類別內。
- 繼承的好處:複用(Reuse)以節省重新定義類別的成本。

類別與物件



多載(override)

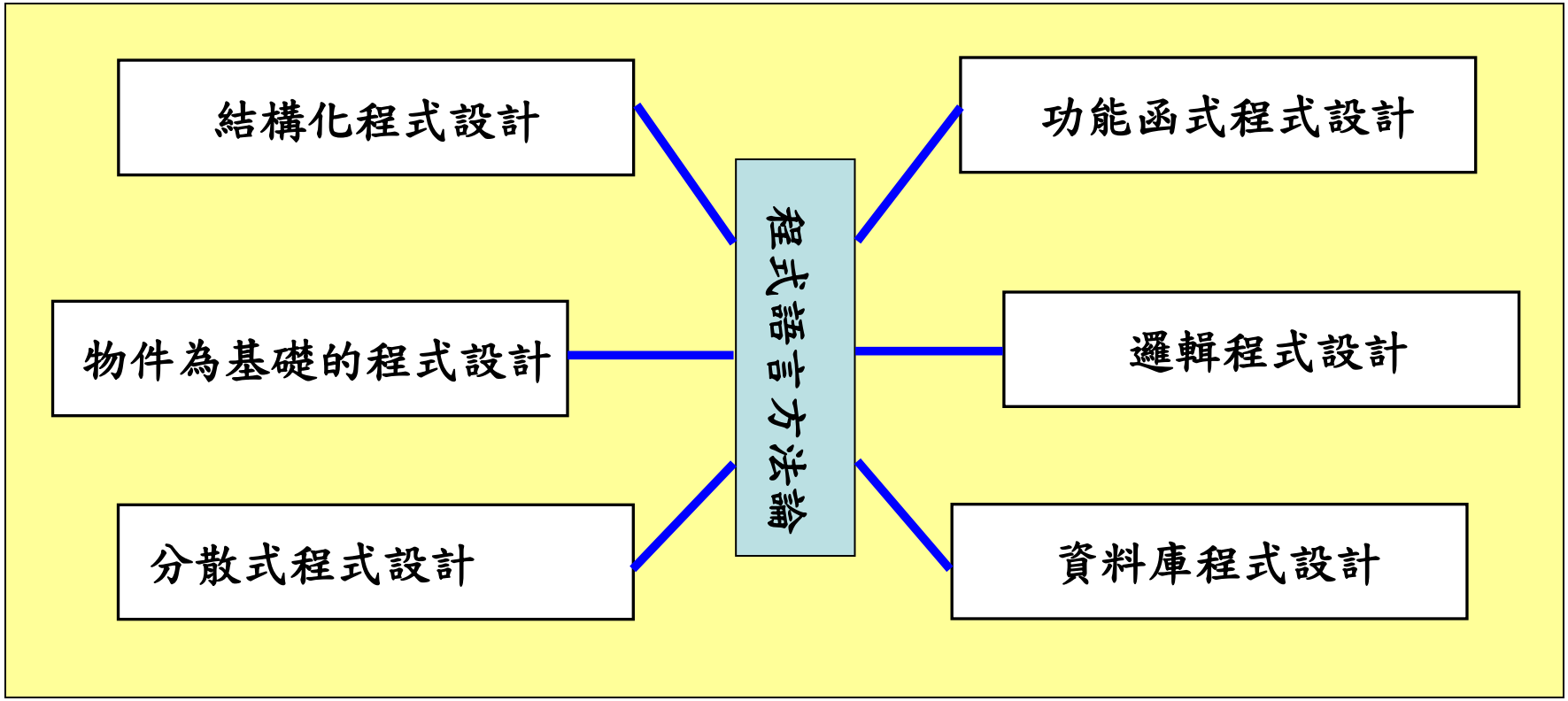
- 多載：多元性，指名稱相同的物件行為但具有不同的功能。

物件導向程式設計

- 程式語言方法論(Programming language paradigm)是程式語言所採用的基礎、原理與方法等。

- 方法論都可和物件導向技術結合，產生各種特殊的程式設計方法。

物件導向技術在程式設計上應用



物件導向技術與軟體工程

- 物件導向程式語言利用物件間的交互作用描述應用系統行為。
- 物件的定義一旦確定，就可以產生一個完整的應用系統。
- 物件代表系統裡單獨的一部分，複雜的系統以物件分成子系統，結構清楚。
- 透過繼承，類別可以重復使用其他類別裡的定義，代表程式單元的再使用(Reuse)；
- 對軟體工程而言，類別組成的類別庫(Class library)，是程式再使用的基礎

再使用的方法

- 建立新物件：新物件擁有類別定義的屬性與行為。
- 類別的繼承關係：一個類別的定義來自數個類別，稱為「多重繼承」(Multiple inheritance)。
  - 透過物件導向技術達到的軟體再使用。
  - 物件導向模型對於應用系統的描述自然有彈性。
  - 物件導向應用在軟體工程上，有時是大型軟體元件(Software Component)的再使用，。
  - 大型的系統，共通的成分找出，再利用分工加速系統開發。

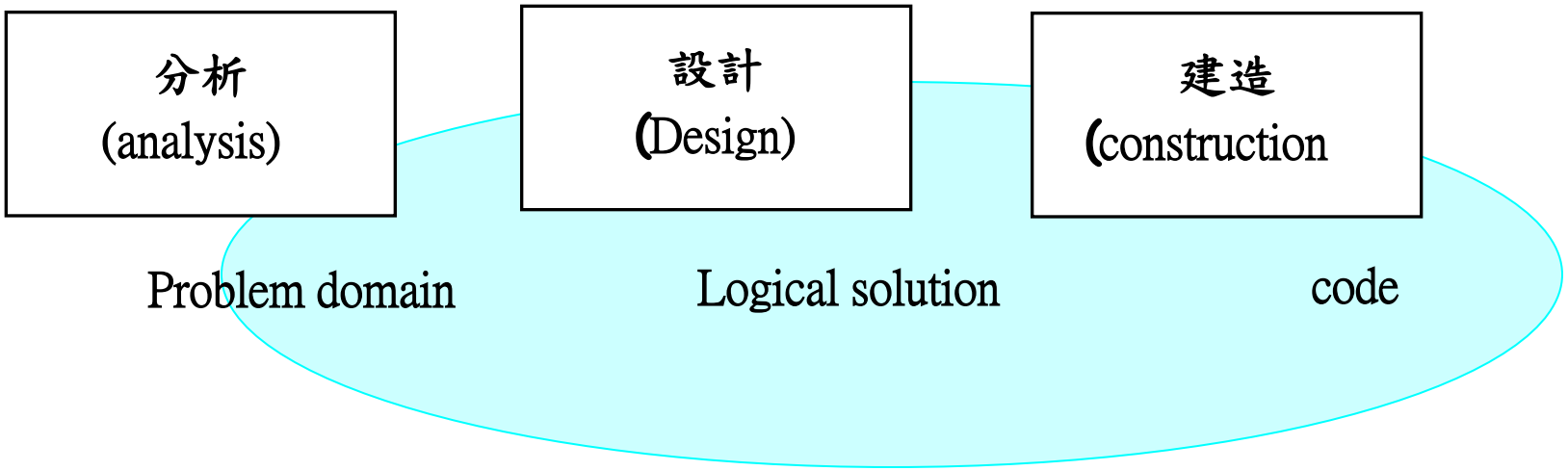
物件導向軟體工程

- 1990年代物件導向分析與設計開始發展
- 1994年IBM正要導入物件導向技術，發展SOM（system object model）
- Booch的OO大作。
- 物件導向編程觀念大約在1970年代就有了
- 物件導向的分析與設計方法， 1990年代才開始出現。
- 要認識物件導向的分析與設計方法，值得一讀的文章:
  - Fichman, R.G.er al. “Object-Oriented and Conventional Analysis and Design Methodologies,” IEEE Computer,Oct. 1992,pp.22-39.

物件導向分析

- 先描述問題與需求以及軟體系統的功能。
- 分析(analysis)的工作強調問題的探討。
- 設計(design)強調解決的方法。
- 物件導向分析與設計(object-oriented analysis and design)強調從物件(objects)觀點看問題(problem domain)與解決的方法(logical solution) 。

開發活動


分析
(analysis)
設計
(Design)
建造
(construction
Problem domain
Logical solution
code

物件導向分析

- 物件導向分析的主要目的:找出問題精確而完整的描述

OOS的7個步驟

1. 找出問題主要的entities，相當於ER model的實體，跟系統要處理的資料有密切關係。
2. 分辨active entities與passive entities：active entities 會執行一些操作；passive entities建立ERD(ER diagram)。
3. 建立active entities 之間的data flows。
4. 細分entities或functions成sub-entities與functions。
5. 檢查是否需要新的entities。
6. 分配functions到新的entities。
7. 分配entities到application domains，每個domain都建立一個ERD。

物件導向設計的4個步驟

- Booch指出物件導向設計的4個步驟：
  1. 訂出類別與物件：從問題的描述找出可能的類別與物件。
  2. 定義類別與物件的涵義；從物件的使用過程建立類別與物件的涵意。
  3. 找出類別與物件的關係：了解類別的繼承關係與物件之間的互動關係。
  4. 實作(Implement)類別與物件：建立類別與物件的細節，例如類別的行為。

RDD(RDD (responsibility-driven design)的two-phase design的六個步驟

1. 找尋類別：從需求規格中萃取名稱或名詞片語，找出可能的類別。
2. 找出responsibilities：考慮每個類別的角色與功能，從需求規格裡含有動作的部分找出class的responsibility。
3. 找出collaborations：檢視每個responsibility，看是否需要其他classes的collaborations。
4. 定義hierarchies：建立類別階層架構。
5. 定義subsystems：繪出系統的collaboration graph，將複雜的collaborations當做可能的subsystems。
6. 定義protocols：定義出class、subsystem與contracts的內涵。

OOA與OOD

- OOA與OOD各有系統化的方法。
- 分析的工作注重使用者的需求與問題。
- 設計則把需求的實作藍圖繪出，在滿足成本、效能與品質的條件下。
- OOA與OOD的工作很多交錯之處，都可對應到設計模型的某一部分。
- 設計時，對需求有了更深入的了解，再回到分析進行調整。

標準化的物件導向分析

- 物件導向分析從需求分析開始，接著是功能塑模(functional modeling)、結構塑模(structural modeling)與行為塑模(behavioral modeling)。
- UML的圖示在功能塑模時會針對商務處理(business process)找出使用案例，畫出使用案例圖(use case diagram)。甚至建立活動圖(activity diagram) 。
- 結構塑模的目的：找出基本的類別、類別的屬性，以及類別之間的關係。
- 行為塑模的目的：找出類別的行為，常會繪製合作圖(collaboration diagram)或是循序圖(sequence diagram)之類的互動圖(interaction diagram)。

需求分析

- 分析需求的文字敘述
  1. 名詞與名詞片語通常會成為物件(objects)或是屬性(attributes)。
  2. 動詞與動詞片語通常會成為方法(method)或關聯(association)。
  3. 帶所有格名詞通常是屬性而非物件。
- 分析後可以從中得到類別與物件的資訊，CRC卡可以在此時使用。
- 獲得需求的過程，使用者會提到系統的各種用途，從中可找出使用案例。

軟體系統需求敘述的範例

- The **system** shall match up **actual cashflows** with **forecasted cash flows.**
- The **system** shall automatically generate appropriate **postings** to the **General Ledger.**
- The system shall allow an **Assistant Trader** to Modify **trade data** and propagate the **results** Appropriately.

使用案例塑模

- 使用案例塑模(use case modeling)是物件導向分析階段重要的工作，一個使用案例代表一種系統的使用方式。
- 使用案例模型有多種角色(actor)當做使用者
- 找尋使用案例的方法：
  1. 藉由圖形使用者介面尋找。
  2. 從目前使用的系統尋找。

領域模型（Domain model）

- 領域模型在結構塑模的過程中建立，得到概念式的類別圖。文法分析與CRC卡可在這個階段使用。
- 概念式的類別圖描述系統使用的資料
- 分析階段主要目標:了解系統的業務處理細節。

從物件導向分析到設計

- 分析工作從需求的了解開始，在過程中建立使用案例(use case)。
- use case是對領域處理(domain process)的文字描述
- UML提供了use case的圖形表示法。
- use case描述使用者運用系統的方式
- use case多，代表使用者和系統的互動多。

use case描述的內容

Use case :
Actors :
Type :
Description :

Use case :
Actors :
Purpose :
Overview :
Type :
Cross references :

物件導向軟體系統開發待1

- 物件導向分析與設計的處理與方法。
- 視覺化塑模的用途。
- 塑模工具的功能。
- UML視覺化塑模的過程與方法。
- 運用UML輔助物件導向分析與設計工作。

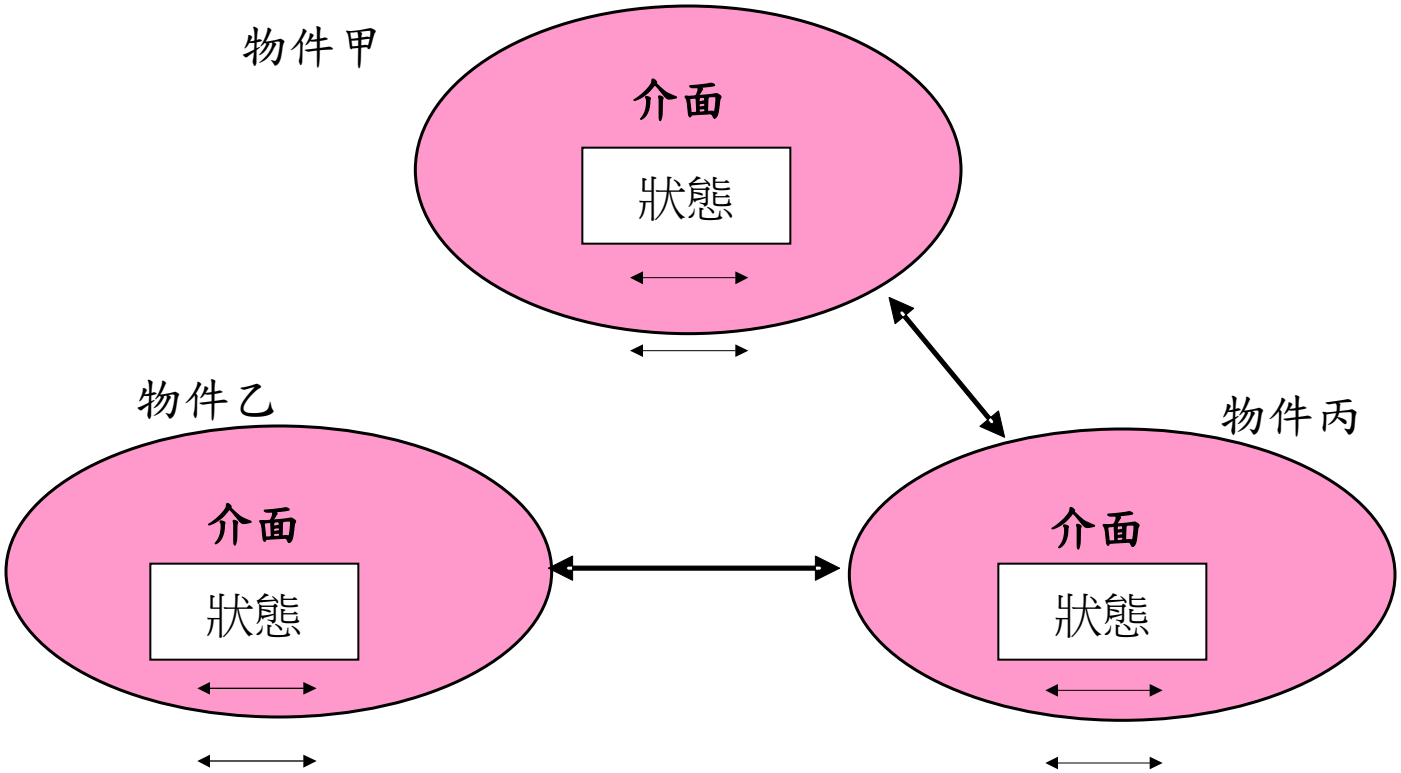
物件導向軟體系統開發的實務END

- 觀念雖然可用口語解釋清楚，但若能轉換成視覺化的表示，會更容易理解。
- 「視覺化塑模」（visual modeling ）運用圖示表達各種模型。
- 視覺化的塑模工具甚至可自動產生程式碼，大幅地簡化系統開發與維護的工作。
- 反向工程(reverse engineering)可重新建立各種描述系統的模型。

物件導向設計

- 物件導向設計（Object-oriented design）以物件為基礎
- 軟體系統是有交互作用的物件組合
- 物件具有特定的功能與內部的資訊。
- 物件具有狀態（State）與介面（Interface）
- 狀態是物件的資料屬性。
- 介面定義物件的操作。
- 透過介面可了解或改變物件的狀態。

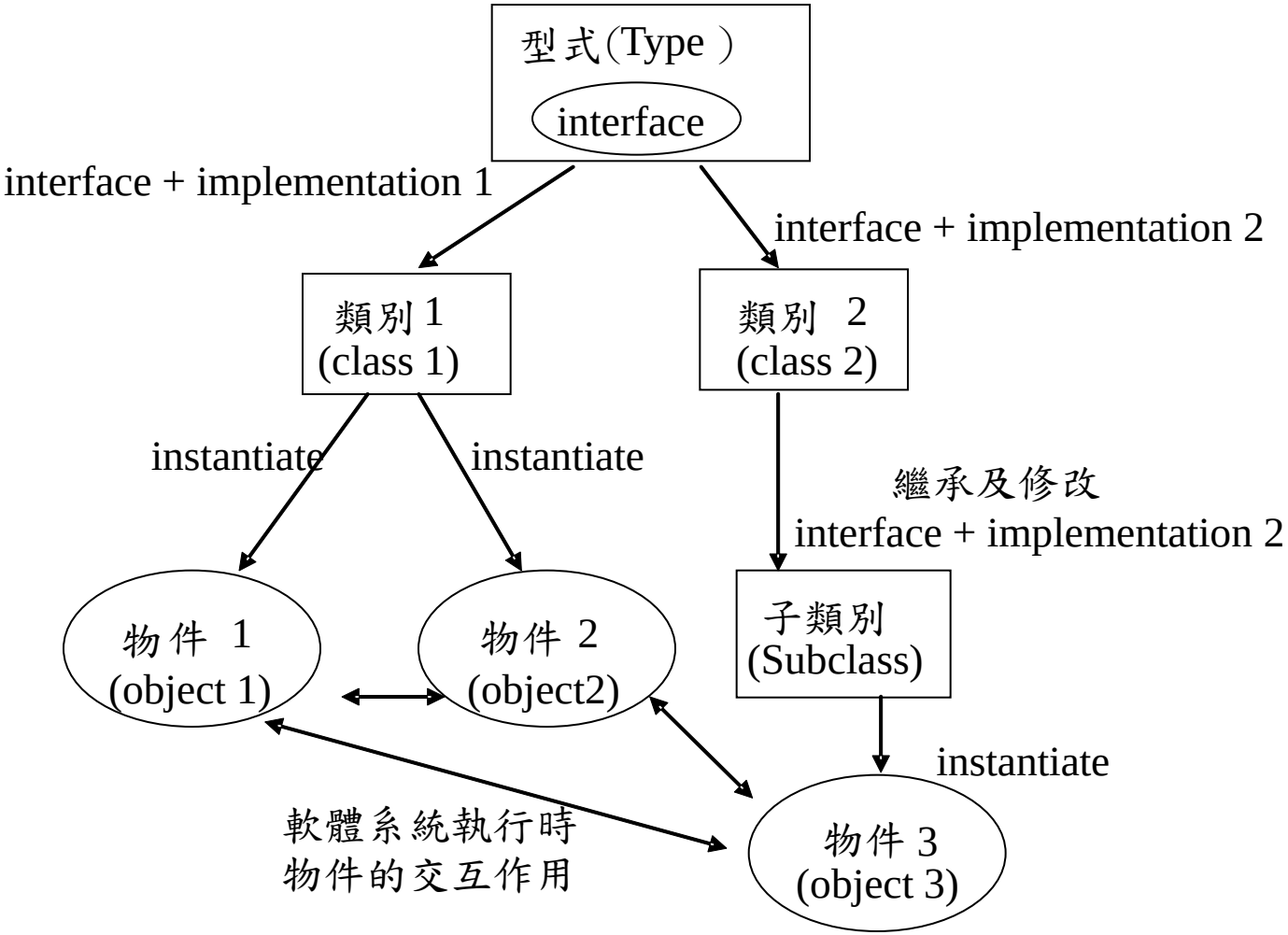
物件導向設計的結構



物件導向設計的優點

- 物件獨立自主，容易維護及修改
- 物件的變更，比較不容易造成系統的大變動
- 物件和應用系統之間有良好的對應關係
- 物件的再使用容易，可節省開發成本
- 物件導向資料模型的描述能力強
- 物件導向技術及工具的發展成熟而普遍

形式、類別與物件的觀念


型式(Type)
interface
interface + implementation 1
interface + implementation 2
類別 1
(class 1)
類別 2
(class 2)
instantiate
instantiate
繼承及修改
interface + implementation 2
物件 1
(object 1)
物件 2
(object2)
子類別
(Subclass)
instantiate
軟體系統執行時
物件的交互作用
物件 3
(object 3)

利用繼承的方式來產生次類別

- 利用繼承（Inheritance)的方式來產生次類別(subclass），再利用父類別的定義，僅需增加新功能。
- 依據類別建立物件的過程叫做實例化(Instantiate)
- 物件是類別的實例(Instance)
- 物件建立後，都有唯一的識別(Unique identification）。
- 相同的類別會有不同的實例
- 物件的組合與互動構成一個軟體系統。

抽象類別與具體類別

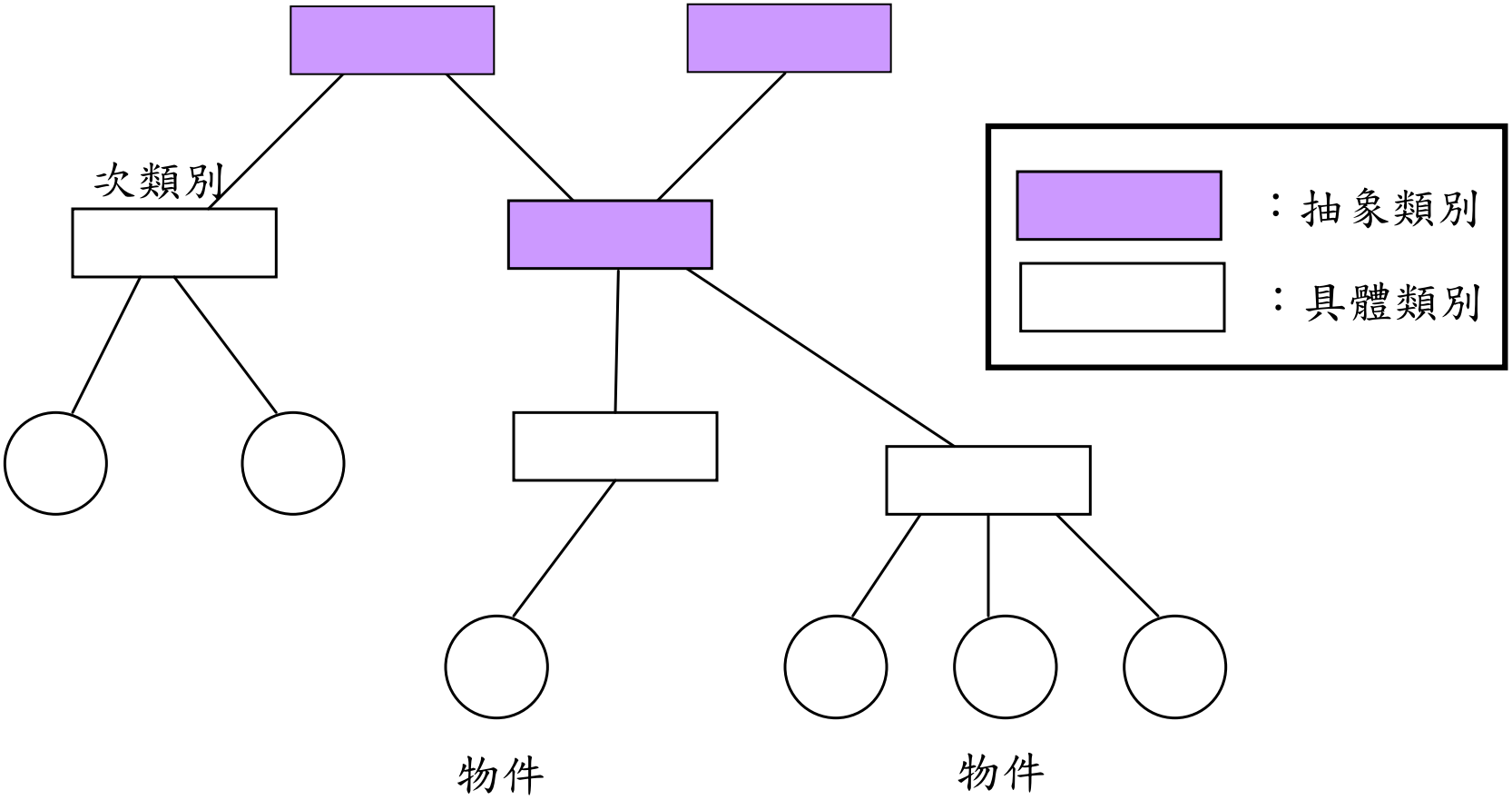
- 次類別(Subclass)由現有的類別衍生出。
- 繼承關係形成類別階層架構(class hierarchy)，可分成兩大類：

1. 抽象類別(Abstract class)：用來規範其他類別的定義。
2. 具體類別(Concrete class)：用於建立物件。

抽象設計

- 設計是由下而上還是由上而下，並不一定
- 可能是雙向同時進行，直到結果令人滿意。
- 抽象設計(Abstract design)是從具體設計(Concrete design)萃取共通點。
- 具體設計是抽象設計的再使用並加上潤飾。

類別階層



塑模(modeling)

- 「塑模」(modeling)是抽象化(abstraction)的能力:思考問題、尋求解答。
- 分析複雜的問題，讓人易於了解，是抽象化的目的。
- 從不同的角度將所要建置的系統抽象化，做為系統的藍圖。
- 塑模，利用工具，可以降低成本並提昇效率。
- 開發專案成功的主因:嚴謹的表示法、有效的工具。

視覺化塑模

- 「視覺化」(Visualization) 表示法，一圖表示千言萬語。
- UML (Unified Modeling Language)整合了視覺化塑模和物件導向分析與設計。
- 一種軟體開發的新標準。
- Rational Rose是完整支援UML的軟體工具。

UML的歷史

- 物件導向分析與設計的理論衍生出各種方法。
- Rumbaugh的OMT
- Jacobson的OOSE
- Booch的方法
- 這些方法，經過融合，在多人努力下，UML結合各家優點。
- UML V1.0版， 1997年交付OMG (Object Management Group)，標準化的基礎。
- UML的分析與設計表示法的標準化包括模型、語法和圖示
- 接受UML的越來越多，開始出現UML的軟體工具。

開發程序的反覆性

- 開發程序(Development process)，通常是反覆性的(Iterative)步驟。
- 可降低風險
- 開發初期先試著解決高風險的問題
- Rational Objectory Process ，可以從兩個角度探討

1. 時程：開發流程分成多個階段。設定目標，訂出細節，反覆地建造部分的系統功能，直到完成。
2. 實作：需求了解、分析與設計、製作和測試，每項工作都有特定的階段和時程。

系統易維護與擴充

- 軟體開發的階段，都含有各種抉擇、調適和求證的過程。
- 有關於系統的記載夠詳實，能幫助在每個過程中做到最精確的管理與聯繫。
- 使系統易維護與擴充正UML能提供的。

Rational Rose

- 描述軟體系統有多種方式
- 一種事物從不同的角度看會呈現不同的風貌。
- 軟體開發有不同的階段工作，過程中都需要系統藍圖。
- 從系統塑模的表示法介紹Rational Rose

1. 使用案例模型(Use case model)：描述系統的行為，讓使用者了解系統的運作方式，讓開發系統的人認識系統的功能。
2. 類別圖(Class diagram)：類別的內涵與類別之間的關係代表系統的組成與功能。
3. 順序圖(Sequence diagram)：使用案例模型的具體化，從使用者和開發者的角度，將系統內的行為以互動關係在循序圖表示出來。
4. 狀態遷變圖(State transition diagram)：狀態變化圖描述系統的行為。

Rational Rose

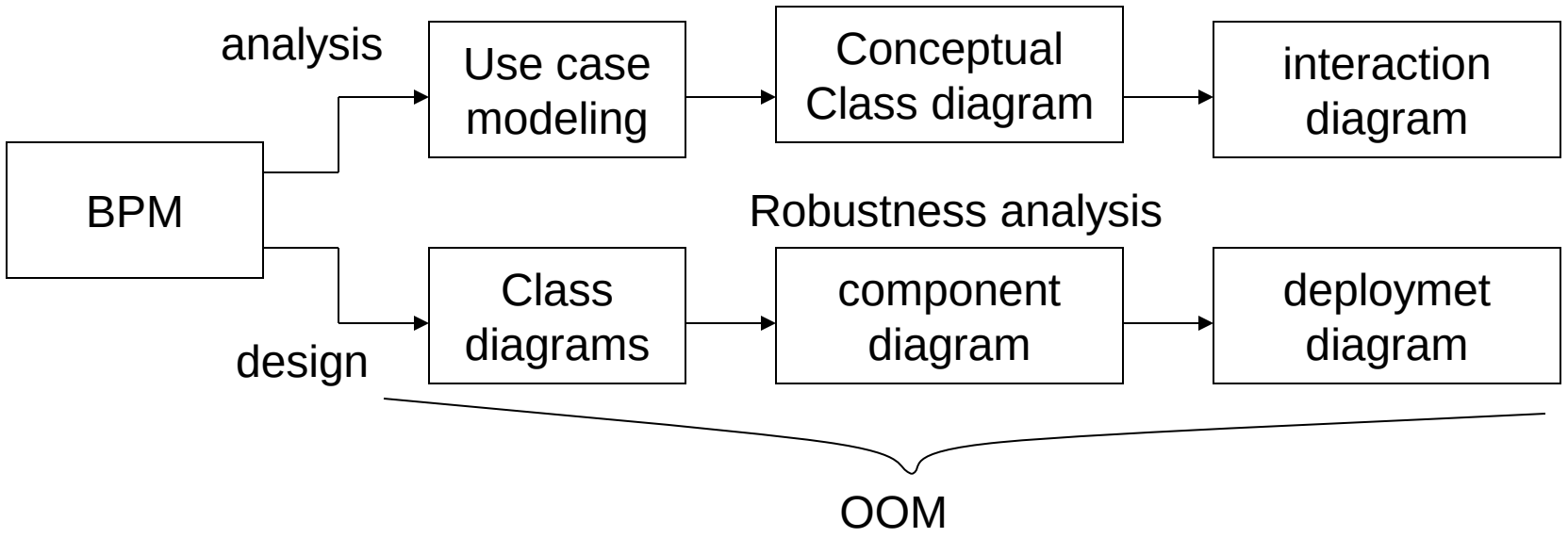
- 元件圖、部署架構圖，Rational Rose都支援
- Rational的4+1 view
- PowerDesigner是以BPM->OOM->CDM->PDM的方式，將大部分UML的支援集中在OOM，與整個開發流程結合在一起。
- 如何把UML圖形融入物件導向分析與設計的過程中
- 在那一個階段該使用那一種圖形要了解。

類別可以依照特性進一步分類

■「類別圖」(Class diagram)表示類別和類別之間的關係。

1. 實體類別(Entity class)：資訊與相關的行為，通常對應實際的事物或觀念。
2. 邊界類別(Boundary class)：負責系統內外之間的溝通，可描述系統的介面。
3. 控制類別(Control class):系統的控制邏輯

物件導向分析與設計


analysis
Use case modeling
Conceptual Class diagram
interaction diagram
BPM
Robustness analysis
Class diagrams
component diagram
deploymet diagram
design
OOM

參考資料

- 顏春煌編著，「軟體工程理論與實務應用」，碁峰資訊股份有限公司。