Case Study of Project Design: A Small Banking System

Cheng-Chin Chiang

Steps

- 訂定功能規格
- 系統架構設計
- 類別與物件介面設計
- 函數實作

步驟一：訂定功能規格

- 找出誰是使用者（可能不只一種使用者）？
- 使用者會需要什麼功能（不同使用者需要之功能可能不同）？
- 使用者會如何使用各種功能（輸入、輸出、檢查邏輯）？

以Banking System為例

- 誰是使用者？
  - 存戶
  - 銀行行員
  - 銀行主管
- 使用者需要什麼功能？
  - 存戶：進行各種線上交易，包括申請帳號、提款、轉帳、查詢交易、取消帳號
  - 銀行行員：核計當日交易總額、列印交易清冊、查詢交易、執行存戶臨櫃交易（存戶所能執行的功能均包括在內）
  - 銀行主管：行員執行之查詢交易

以Banking System為例（續）

- 使用者會如何使用各種功能？
  - 存戶
    - 登入：
      - 輸入：帳號and密碼
      - 輸出：允許登入或拒絕登入訊息
      - 檢查邏輯：帳號格式是否正確、是否為合法客戶
    - 提款：提供資料庫中客戶提款功能。
      - 輸入：提款金額
      - 輸出：更新客戶存款餘額、新增一筆交易記錄
      - 檢查邏輯：金額是否合乎規定、存款餘額是否足夠
    - 轉帳：提供資料庫中客戶轉帳功能。
      - 輸入：轉入帳號 and 轉帳金額
      - 輸出：更新轉出客戶存款餘額、更新轉入客戶存款餘額、新增一筆交易記錄
      - 檢查邏輯：轉出/入帳號是否為合法客戶、金額是否合乎規定、存款餘額是否足夠
    - 查詢交易紀錄：提供資料庫客戶查詢資料庫中自己的交易紀錄。
      - 輸入：
        (1) 以日期查詢：輸入日期
        (2) 以交易種類查詢：輸入交易種類（存款、提款、轉帳）
      - 輸出：符合查詢標的之交易記錄列表
      - 檢查邏輯：輸入之查詢資料格式是否正確
    - …
  - 銀行行員
    - 每日經手交易總額：
      - 輸入；
        - (1) 日期
        - (2)交易類別存款
          - 提款
          - 轉帳
      - 輸出：總計金額
      - 檢查邏輯：輸入格式是否正確

      …
  - 銀行主管
    - …

步驟二：系統架構設計

- 從功能清單中有無共通性或相似性之功能？
- 考慮可否將共通性功能設計成一個模組？若規模太大可否切成數個子模組？
- 明列各模組負責什麼功能（檢視前一步驟所列之功能規格）？

以Banking System為例

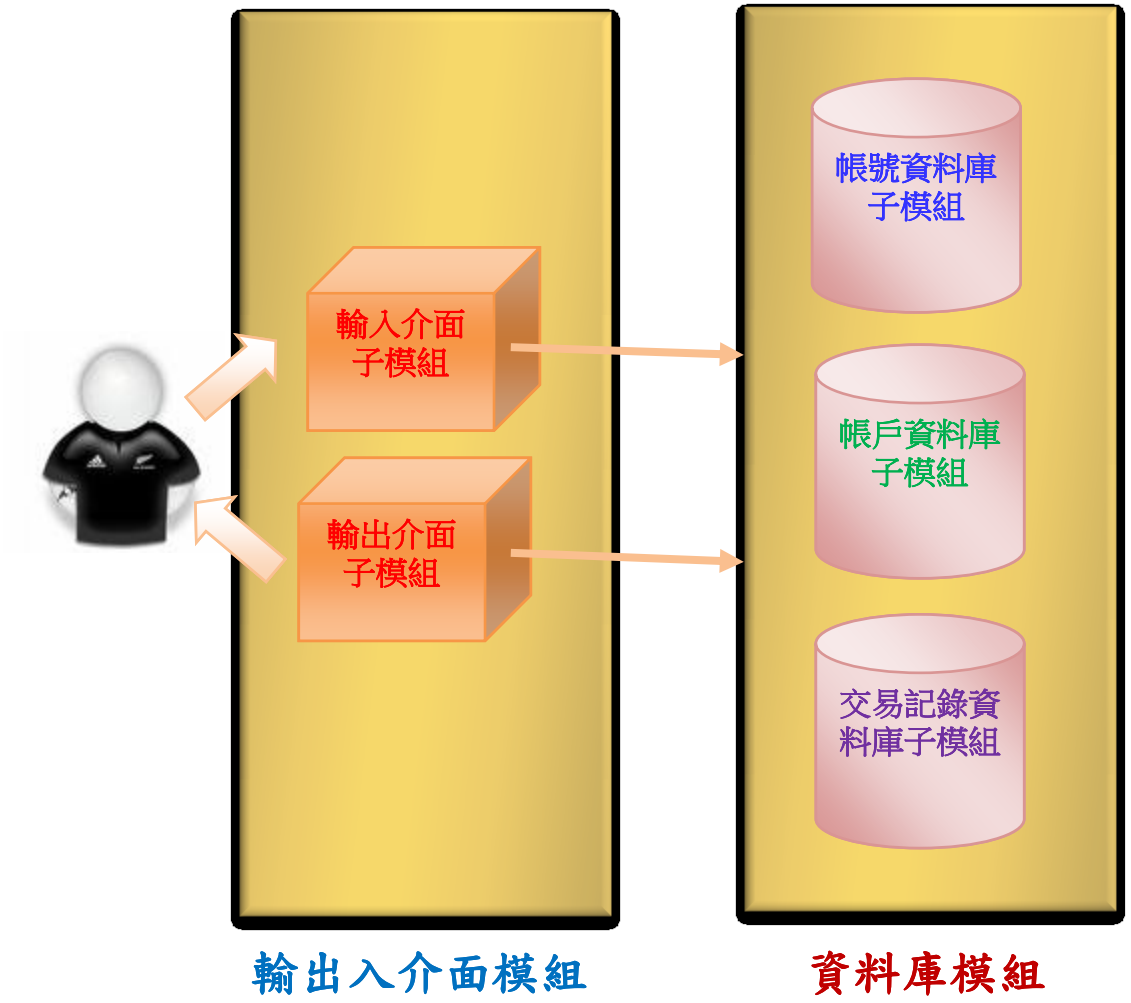
- 功能清單
  - 存戶：
    - 轉帳：登入畫面、帳號檢查、帳戶資料更新、交易資料更新、結果輸出
    - 提款：登入畫面、帳號檢查、帳戶資料更新、交易資料更新、結果輸出
    - 查詢交易：登入畫面、帳號檢查、帳戶資料搜尋、交易資料搜尋、結果輸出
    - 申請/取消帳號：帳號資料輸入畫面、重複帳號檢查、帳戶資料更新、交易資料更新、結果輸出
  - 行員：
    - 核計交易總額：資料查詢畫面、交易資料搜尋與加總、結果輸出
    - 查詢交易：存戶帳號資料輸入畫面、帳號檢查、交易資料搜尋、結果輸出
    - 執行存戶臨櫃交易：同存戶功能所列
  - 銀行主管：
    - 行員執行之交易查詢：行員帳號資料輸入畫面、帳號檢查、交易資料搜尋、結果輸出

以Banking System為例

- 共通性模組
  - ***畫面管理***➔**輸出入介面模組**
    - 輸入介面子模組
    - 輸出介面子模組
  - ***帳號管理***➔帳號資料庫子模組
  - ***帳戶資料與搜尋更新***➔帳戶資料庫子模組
  - ***交易資料與搜尋更新***➔交易資料庫子模組

資料庫模組

以Banking System為例



步驟三：類別與物件介面設計

- 各模組中牽涉到的『名詞』物件為何？
- 決定哪些適合設計為『類別』？哪些適合設計為『結構』？
- 決定類別間的關係為何（『是』（繼承）、『屬於』或是『使用』（類別遷入））？
- 各結構之資料欄位為何？
- 各類別的資料成員和函式成員為何？
- 決定類別成員之可視性（Visibility）如public, private, protected

以Banking System為例

- 各模組牽涉到的『名詞』物件
  - 輸入介面子模組：
    - 存戶用輸入資料表單（登入、提款、轉帳、查詢交易、申請帳號、取消帳號）
    - 行員用輸入資料表單（登入、存款、提款、轉帳、查詢交易、申請/取消帳號、交易總額查詢）
    - 主管用輸入資料表單（登入、行員交易查詢）
  - 輸出介面子模組：
    - 帳戶存款餘額表單
    - 交易查詢結果列表
    - 交易總額查詢結果列表
    - 登入結果訊息
    - 帳號新增與刪除結果列表

以Banking System為例

- 各模組牽涉到的『名詞』物件（續）
  - 帳號資料庫子模組：
    - 資料庫：包含所有存戶之帳號資料
    - 帳號資料：紀錄存戶之帳號（身份證）、姓名、出生年月日、住址、電話、帳號建立日期
  - 帳戶資料庫子模組：
    - 資料庫：包含所有存戶之所有存款餘額資料
    - 帳戶資料：紀錄帳號、帳戶存款餘額、最後更新日期
  - 交易紀錄資料庫子模組：
    - 資料庫：包含所有存戶、行員、主管的交易紀錄
    - 交易紀錄：紀錄存戶帳號、交易種類、交易日期、交易時間、交易操作者帳號（存戶自行操作、行員操作、主管操作）、交易結果代碼

以Banking System為例

- 決定哪些適合設計為『類別』？哪些適合設計為『結構』
  - 有操作功能性者定義為類別
  - 無功能性者（除了資料儲存功能外）定義為結構

物件	結構
輸出入介面模組	
輸入介面子模組（需要處理輸入之資料）	各種輸入表單（一表單一結構）
輸出介面子模組（需要處理格式化輸出）	各種輸出報表（一報表一結構）
資料庫模組	
帳號資料庫子模組（需搜尋/新增/刪除帳號資料）	帳號資料
帳戶資料庫子模組（需搜尋/新增/關閉帳戶資料）	帳戶資料
交易紀錄資料庫子模組（需搜尋/新增交易紀錄）	交易紀錄

以Banking System為例

- 決定類別間的關係為何（『是一種』（➔繼承）、『有』或是『使用』（➔類別遷入））？

	輸出入介面模組	輸入介面子模組	輸出介面子模組	**資料庫模組**	帳號資料庫	帳戶資料庫	交易紀錄資料庫
輸出入介面模組		有/使用	有/使用				
輸入介面子模組							
輸出介面子模組							
資料庫模組	使用				有/使用	有/使用	有/使用
帳號資料庫子模組							
帳戶資料庫子模組							
交易紀錄資料庫子模組							

以Banking System為例

- 各結構之資料欄位為何？
  - 登入表單：帳號、密碼
  - 提款表單：提款金額
  - 轉帳表單：轉出帳號、轉入帳號、轉帳金額
  - 申請帳號：身份證字號、姓名、出生年月日、住址、電話、密碼、確認密碼
  - 取消帳號：帳號、密碼
  - …（其他表單和輸出報表比照辦理）
  - 帳號資料：帳號（身份證）、姓名、出生年月日、住址、電話、帳號建立日期
  - 帳戶資料：帳號、帳戶存款餘額、最後更新日期
  - 交易紀錄資料：帳號、交易種類、交易日期、交易時間、交易操作者帳號、交易結果代碼

以Banking System為例

- 各類別的資料成員和函式成員為何？
- 決定類別成員之可視性（Visibility）如public, private, protected

Database
*pUDB:UserDatabase *pADB:AccountDatabase *pTDB:TransactionDatabase *IO: IOInterface …
+AddUser(UserRecord&):bool +RemoveUser(ID:string):bool +AddAccount(AccountRecord&):bool +CloseAccount(ID:string):bool +AddTransaction(TransactionRecord&):bool +QueryUser(ID:string):UserRecord& +QueryAccount(ID:string):AccountRecord& +QueryTransaction(ID:string):vector<TransactionRecord&> +QueryTransaction(day:Date):vector<TransactionRecord&> +GetUDB():UserDatabase* +GetADB():AccountDatabase* +GetTDB():TransactionDatabase* +OutputUser(udata:vector<UserRecord&>):void +OutputAccount(adata:vector<AccountRecord&>):void +OutputTransaction(tdata:vector<TransactionRecord&>) …

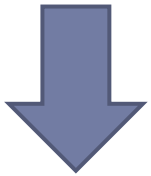
步驟四：物件實作

- 將前面的設計轉換成C++指令
  - 資料結構和類別的宣告
  - 類別中成員函式的實作
- 設計函式時適當運用以下法則
  - 從高階到低階
  - 化一繁為多簡（各個擊破法）
  - 畫出函式處理的流程圖

以Banking System為例

- 函數實作—從高階到低階
  - bool AddUser(UserRecord& urec)
    - Step 1: 檢查urec是否已在UserDatabase中
    - Step 2: 如果已經存在就傳回false
    - Step 3: 否則將urec加入UserDatabase
    - Step 4: 傳回true

高階（自然語言）



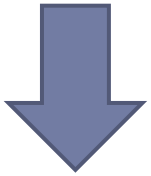
```
if Search(urec)==true
{
    return false;
}
else
    pUDB->Add(urec);
    return true;
```

低階（程式語言）

以Banking System為例

- 函數實作—化一繁為多簡
  - bool AddUser(UserRecord& urec)

一繁



```
if pUDB->Search(urec)==true
{
    return false;
}
else
    pUDB->Add(urec);
    return true;
```

多簡

以Banking System為例

- 函數實作—繪出流程圖
  - bool AddUser(UserRecord& urec)

```
if pUDB->Search(urec)==true
{
    return false;
}
else
    pUDB->Add(urec);
    return true;
```