Chapter 15

軟體再利用

本章內容

15.1 再利用技術綜覽
15.2 應用程式框架
15.3 軟體產品線
15.4 應用系統再利用

- 以再利用為基礎的 (reuse-based) 軟體工程，是一種讓開發程序儘量再利用現有軟體的軟體工程策略。
- 如今在開發新系統時，都已經是大量使用這種開發方式。大家為何逐漸轉移到以再利用為基礎的開發方式，其動機包括希望降低軟體的生產和維護成本、能夠更快交付系統，以及提高軟體品質。

1. **系統再利用**：由幾個應用程式所組成的完整系統也可被再利用。
2. **應用軟體再利用**：整個應用程式系統可能都可再利用，方式包括不做修改就將它納入到其他系統。
3. **元件再利用**：應用程式的元件有各種不同的大小，從子系統到單一物件都可以拿來再利用。
4. **物件與函式再利用**：專門實作單一功能的軟體元件，例如數學函式 (function) 或物件類別，也可以被再利用。

好處	說明
加速開發	讓產品儘早問市通常比降低整體開發成本更重要。再利用現有軟體將可加快系統開發的速度，因為開發與確認的時間都可以縮短
有效的利用專家	與其讓某些應用領域的專家，在不同專案做相同的工作，不如讓這些專家專注開發一些可再利用的軟體元件，將他們的知識全部投注在這些軟體上
增加可信賴度	再利用那些在已上線系統中使用過的軟體，應該比使用新軟體更可靠，因為這些軟體在設計和實作上的缺失都已經被找出來且加以修正
減少開發成本	開發成本會隨著被開發軟體的規模大小成正比增加。能再利用一些軟體意味著需要編寫的程式碼行數會變少
降低程序風險	現有軟體的成本是已知的，而開發成本卻是未知。這對專案管理而言是重要的考慮因素，因為它可以降低專案成本預估時的誤差。假如能重複使用較大型的軟體元件，例如子系統，這項優點會更加明顯
符合標準	有些標準可實作成一組可再利用元件，如使用者介面標準。舉例來說，使用者介面中的功能表，就可以製作成可再利用元件，讓所有應用程式都呈現相同的功能表格式。使用標準的使用者介面可以提升可信賴度，因為使用者對熟悉的介面比較不會出錯

圖 15.1　軟體再利用的好處

問題	說明
建立、維護與使用元件程式庫	要引進一個再利用元件的程式庫，並且確保軟體開發人員能使用這些程式庫，可能成本會很高。開發程序必須調整以確保程式庫有被使用
尋找、理解並調整可再利用元件	軟體元件必須從程式庫中尋找，而且必須花時間瞭解這些元件，有時候還必須針對新環境修改這些元件。工程師們必須將在程式庫中尋找元件這個動作，當成正常開發過程的一部分
增加維護成本	如果沒有元件的原始程式碼，則維護的成本就會增加，因為系統中的再利用元件可能會逐漸變得與系統的變更無法相容
缺乏支援工具	有些軟體工具集並不支援再利用的開發方式。這些工具可能很難或甚至不可能和元件的程式庫系統整合，因此這些工具所支援的軟體程序，可能沒有將軟體再利用考慮進去。特別是支援嵌入式系統工程的工具，比物件導向開發工具更容易有這種現象
非自創的心態	有些軟體工程師比較喜歡重寫元件，因為他們認為自己可以寫的比這些再利用元件好。部分原因是由於自信，部分原因則是撰寫軟體比使用別人寫的軟體感覺起來更具挑戰性

圖 15.2　軟體再利用的問題

15.1 再利用技術綜覽

- 因為在相同應用領域的系統常會很類似，值得重複使用，因此可以在不同層次上應用再利用技術(從簡單的函式到完整的應用程式都有可能)，以及制訂促進再利用的可再利用元件的標準。

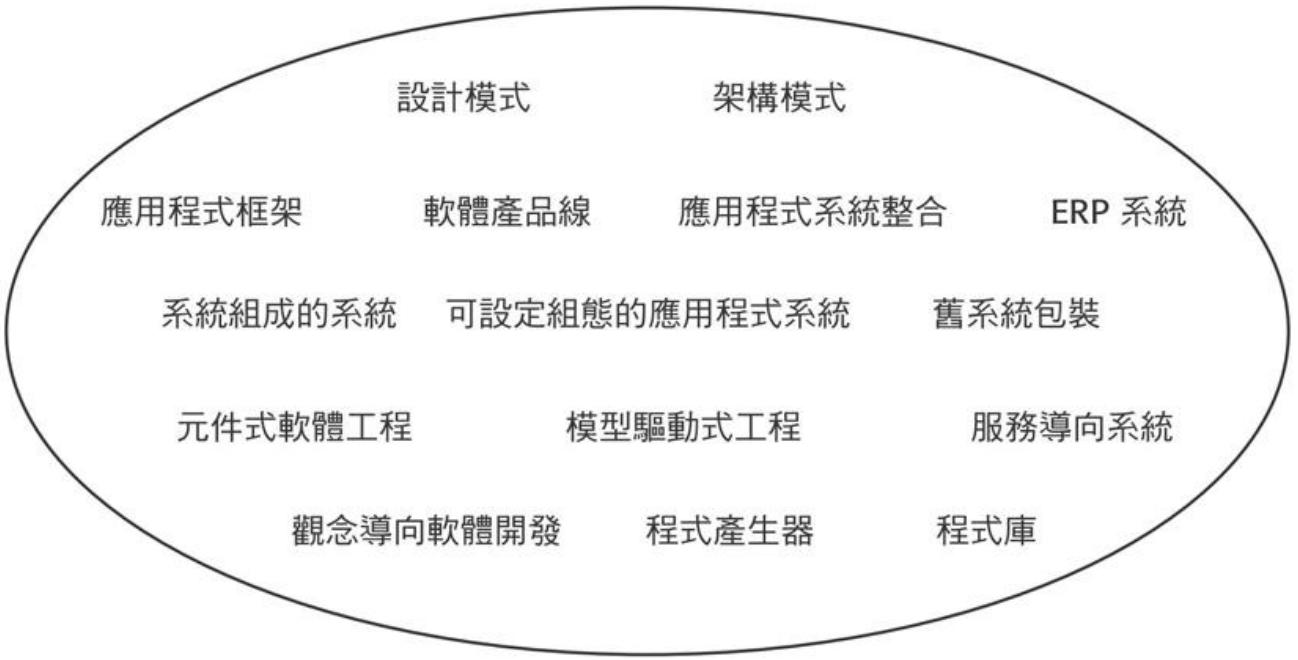

圖 15.3 再利用技術綜覽

15.1 再利用技術綜覽

方式	說明
應用程式框架 (application framework)	一組抽象和具體類別的集合，可經由調整和擴充而建構出應用系統
應用系統整合 (application system integration)	將二或多個應用系統加以整合，來提供延伸的功能
架構模式 (architectural pattern)	將支援共同類型應用系統的標準軟體架構，當作應用程式的基礎。參見第 6、11 和 17 章
觀念導向軟體開發 (aspect-oriented software development)	在程式編譯時將共用元件編織到應用程式的不同位置
元件式軟體工程 (component-based software engineering)	系統是以整合元件（物件的集合）的方式來開發，這些元件必須符合元件模型標準。這部分參見第 16 章
可設定組態的應用系統 (configurable application system)	特定領域的系統，它的設計是可經由組態設定來符合特定客戶的需求

圖 15.4 支援軟體再利用的各種方式

15.1 再利用技術綜覽

方式	說明
設計模式 (design pattern)	將橫跨應用程式的通用抽象觀念，以設計模式來表達，顯示出抽象和具體物件以及互動方式。參見第 7 章
ERP 系統 (ERP system)	是一種大規模的系統，用來封裝為機構而設定的通用企業功能和規則
舊系統包裝 (legacy system wrapping)	舊系統（參見第 9 章）可經由定義一組介面加以「包裝」(wrapped)，透過這些介面可存取舊系統
模型驅動式工程 (model-driven engineering)	將軟體表達成領域模型和與實作無關的模型，而從這些模型產生出程式碼。參見第 5 章
程式產生器 (program generator)	將某種應用程式的知識嵌入產生器系統，就可以從使用者提供的系統模型產生出那個領域的系統
程式庫 (program library)	實作讓大家共同使用的抽象觀念的類別庫和函式庫，透過再利用方式可以讓大家共用
服務導向系統 (service-oriented system)	系統是以連結共用服務 (shared service) 的方式來開發，這些服務可由外界提供。參見第 18 章
軟體產品線 (software product line)	針對某個共同架構產生出某種應用程式類型，可根據不同客戶的需求來調整
系統組成的系統 (systems of systems)	將二或多個分散式系統加以整合來建構出新系統。參見第 20 章

圖 15.4（續）

15.1 再利用技術綜覽

- 在規劃再利用方式時，應該考慮的關鍵因素如下：
  1. 軟體的開發時程
  2. 軟體的預期壽命
  3. 開發團隊的背景、技能和經驗
  4. 軟體的關鍵性與它的非功能需求
  5. 應用領域
  6. 系統將在哪個平台上執行

15.2 應用程式框架

- 現在顯然物件導向再利用技術在物件導向開發程序中，搭配大小較大的抽象物件效果最佳，這種抽象物件稱為框架(framework)。
- 所謂的框架是一種通用的結構，以它為基礎可延伸擴充而建立出某個特定的子系統或應用程式。

> ……是一種將各軟體作品(如類別、物件和元件)整合後的成果，它們共同合作為一系列相關應用程式提供一個可再利用架構。

15.2 應用程式框架

- 框架可支援設計再利用，它提供某個骨架給應用程式套用，還有可以再利用系統中某些特定類別。
- 目前常用的物件導向程式語言都有框架可用，包括Java、C#、C++ 還有如Ruby和Python這類動態語言。
- 現在最被廣泛使用的應用程式框架是web應用程式框架 (web application framework, WAF)，它支援動態網站的建構。WAF的架構通常是以Model-View-Controller (模型-檢視-控制物件，MVC) 複合模式為基礎，參見圖15.5。
- MVC模式最早是在1980年代所提出的一種GUI設計，它可以讓同一物件有多種呈現方式，而且對每一種呈現方式都有不同的互動方式。

15.2 應用程式框架

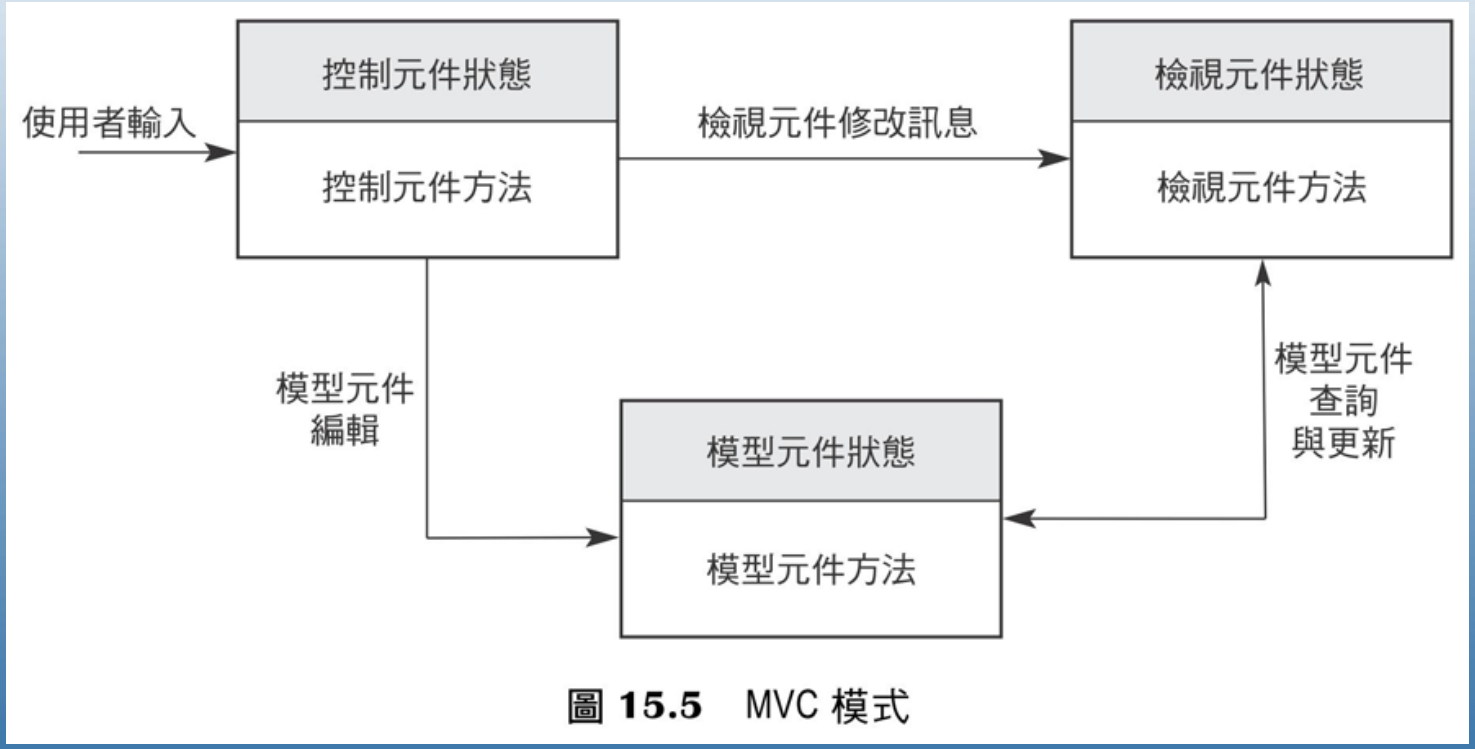

圖 15.5 MVC 模式

15.2 應用程式框架

- 框架經常是設計模式 (design pattern) 的實作成果，參見第7章。
- 不過大多數WAF都提供能支援下列功能的元件和類別：

1. **保全性 (Security)**：WAF可能含有能協助實作使用者驗證 (登入) 和存取控制方面功能的類別。
2. **動態網頁 (Dynamic web page)**：提供協助定義網頁範本，並能動態的從系統資料庫存取資料的類別。
3. **資料庫整合 (Database integration)**：框架通常不包括資料庫，不過會假定有個另外的資料庫可用，如MySQL。此框架可能針對各種不同資料庫提供抽象介面的類別。

15.2 應用程式框架

4. **工作階段管理 (Session management)**：負責建立和管理工作階段 (使用者與系統之間的許多互動) 的類別，通常也是WAF的一部分。
5. **使用者互動 (User interaction)**：現在多數的web框架有支援AJAX (Holdener, 2008) 和HTML5 (Sarris 2013)，使用它們能建立互動性更佳的網頁。

15.2 應用程式框架

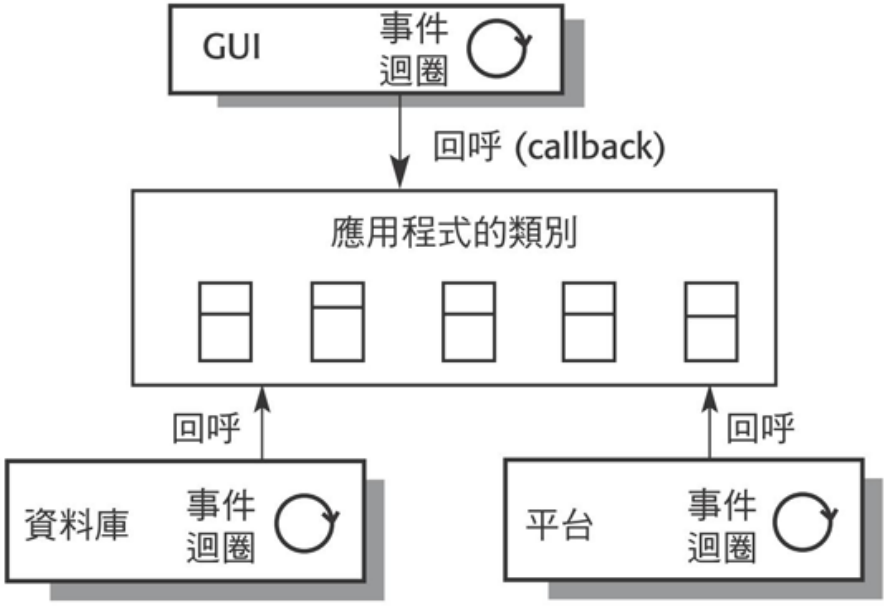

圖 15.6　框架中的反向控制

15.2 應用程式框架

1. **系統基礎架構框架**：這些框架支援系統基礎架構 (infrastructure) 的開發，如通訊、使用者介面和編譯器等。
2. **中介軟體整合框架**：這些框架是由一組支援元件通訊與資訊交換的標準和相關物件類別所組成。
3. **企業應用程式框架**：這些框架專門針對某些特定的應用領域，例如電信或財務系統 (Baumer et al. 1997)。

15.2 應用程式框架

- 以框架建構而成的應用程式，也可以透過軟體產品線或應用程式系列產品概念，成為進一步再利用的基礎。
- 做法是覆寫之前新增到框架裡的具體類別和方法。
- 框架是一種很有效率的再利用方式，但是要將它引進軟體開發程序中需要很高的成本，原因是它們天生就很複雜，而且學習使用它們可能要花好幾個月。

15.3 軟體產品線

- 所謂軟體產品線 (software product line) 是指一組具有共同架構和共用元件的應用程式，其中每個應用程式專門針對不同的需求。
- 其核心系統 (core system) 的設計可讓我們設定組態和調整，以符合不同客戶和設備的需要。
- 這包括可能需要設定某些元件的組態、撰寫一些額外元件，以及調整某些元件讓它符合新的需求。

15.3 軟體產品線

- 一般來說，基底應用程式會包含：
  1. 提供基礎架構支援的核心元件。在開發產品線的新樣例時，通常不會動到它們。
  2. 可組態設定的元件，經由修改和設定它們而建構出新應用程式。有時候藉由使用內建的元件組態設定語言，不必修改程式碼，就能重新設定這些元件的組態。
  3. 領域相關的元件，建立產品線的新樣例時可能會全部或部分被替換。

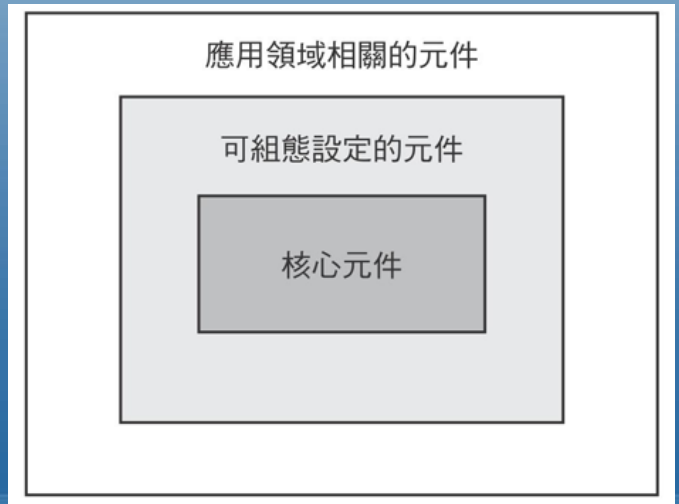

圖 15.7　產品線的基底應用程式的架構

15.3 軟體產品線

- 應用程式框架與軟體產品線有很多共同的特點。這兩種方式的主要差異如下：
  1. 應用程式框架是依靠物件導向功能如繼承和多型 (polymorphism) 來實作框架的擴充功能。軟體產品線不一定要使用物件導向方法來建立。
  2. 大多數應用程式框架是提供一般的支援，而非特定領域方面的支援。例如有專門用來建立web應用程式的應用程式框架。例如可能有軟體產品線是針對病歷資料管理的web應用程式。
  3. 軟體產品線常用在設備的控制程式上。
  4. 軟體產品線是由同一系列的相關應用程式所組成，由同一家企業或機構所擁有。

15.3 軟體產品線

- 交通工具派遣系統是一種資源分配與管理系統(圖15.8)。

圖 15.8 資源管理系統的架構

15.3 軟體產品線

- 而將這個4層式結構實例化之後的成果如圖15.9，圖中顯示在交通工具派遣系統的產品線中可能會包含的模組。在此產品線系統中每一層所包含的元件如下：

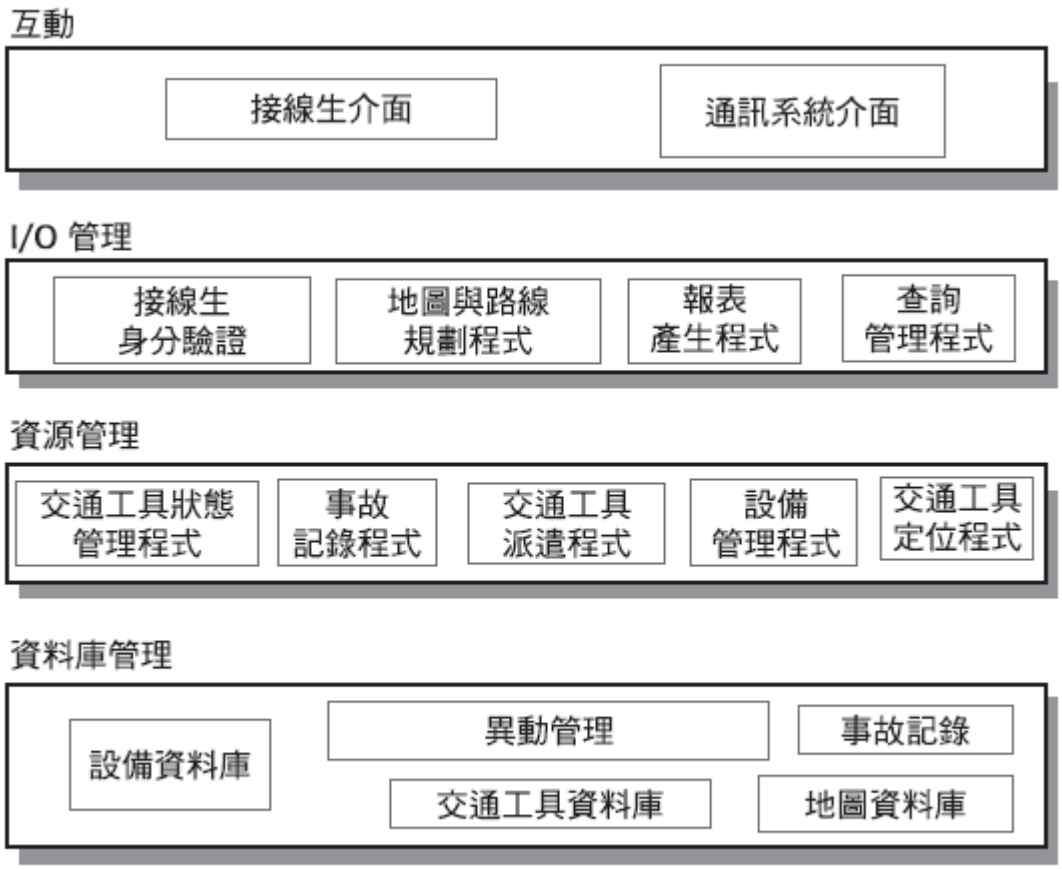

圖 15.9　交通工具派遣系統的產品線架構

15.3 軟體產品線

1. 在互動層級中，包括接線生所使用的顯示畫面元件，還有與通訊系統之間的介面。
2. 在I/O管理層級中(第2層)，包含處理接線生身分驗證、產生事故與派遣交通工具的報表、地圖輸出與路線規劃，以及提供接線生查詢系統資料庫等功能的元件。
3. 在資源管理層級中(第3層)，包含的是定位和派遣交通工具、更新交通工具與設備狀態，以及記錄事故詳細資料的元件。
4. 在資料庫層級中，除了一般的異動管理(transaction management)支援外，還分別有交通工具、設備和地圖的資料庫。

15.3 軟體產品線

- 軟體產品線可能會針對以下類型開發出不同的產品：
  1. **不同的執行平台**：針對不同執行平台開發不同版本的應用程式。
  2. **不同的環境**：針對不同的作業環境和週邊裝置，可製作不同版本的應用程式。例如，提供緊急服務的系統可根據不同服務單位使用的通訊硬體，建立不同版本的應用程式。
  3. **不同的功能**：針對客戶的不同需求，建立不同版本的應用程式。
  4. **不同的程序**：系統必須適應不同企業的流程。

15.3 軟體產品線

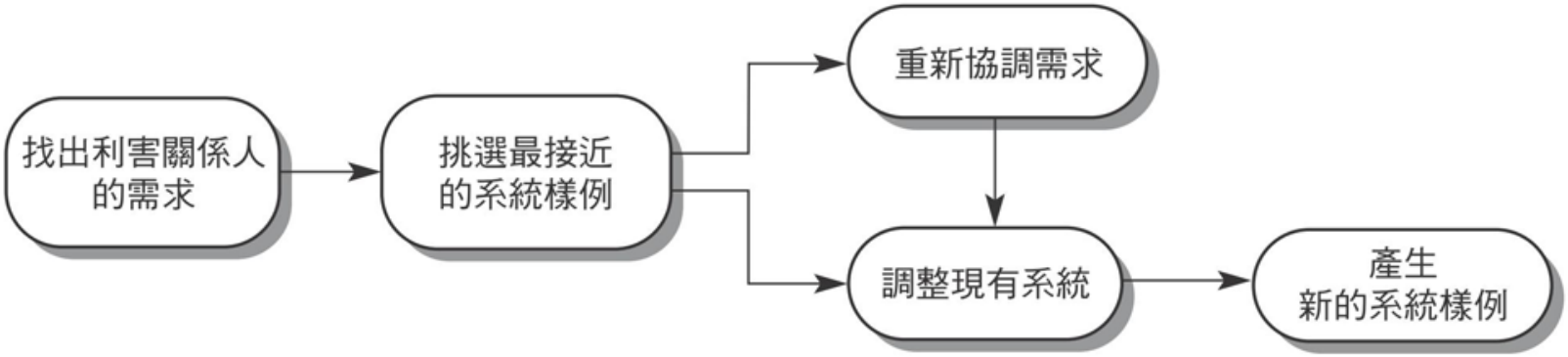

圖 15.10　產品樣例的開發

15.3 軟體產品線

1. **找出關係人的需求**：一開始可能使用一般正常的需求工程程序。
2. **挑選最接近需求的現有系統**：分析需求並挑選出最適合的系列成員進行修改。
3. **重新協調需求**：隨著更多變更的細節出現，而且專案已經規劃好時，其中一些需求可能必須與客戶重新協調，以便將需要改變的程度降到最低。
4. **調整現有系統**：對現有系統開發新的模組，並且修改現有系統的模組，以符合新的需求。
5. **產生新的系列產品成員**：將產品線新成員交付給客戶。在這個階段應當把重要的功能記錄成文件，以便未來作為其他系統開發的基礎。

15.3 軟體產品線

- 軟體產品線可以在開發程序的不同階段進行組態設定：
  1. **在設計期間進行組態設定**：此時是由開發該軟體的機構來修改共同的產品線核心，包括開發、挑選或調整元件，來為客戶建構出一個新系統。
  2. **在部署期間進行組態設定**：此時的通用 (generic) 系統是設計成由客戶或顧問來設定組態。客戶的特定需求和系統的運作環境這方面的知識，都是內嵌在由通用系統所使用的組態資料中。

15.3 軟體產品線

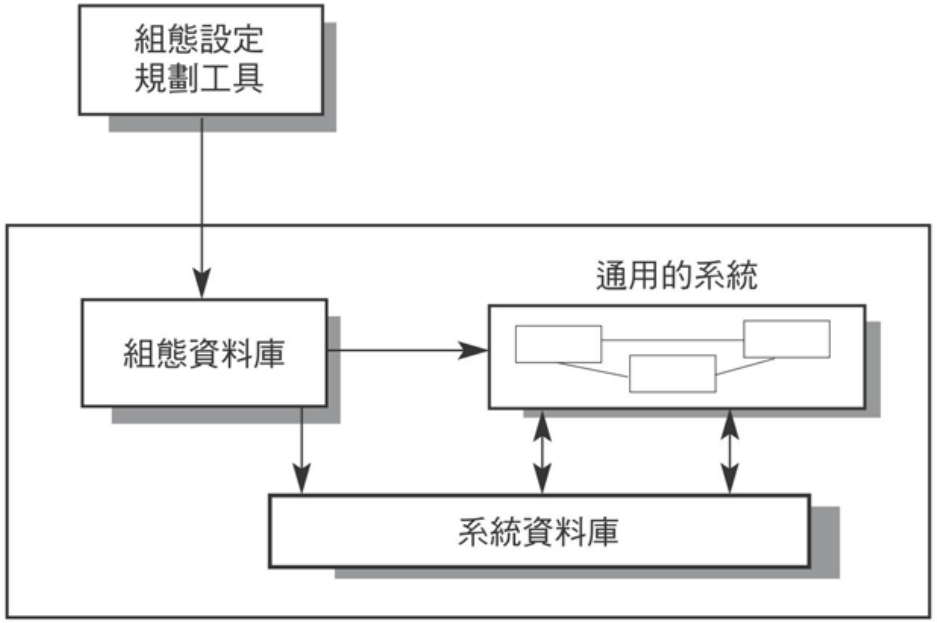

圖 15.11　在部署期間進行組態設定

15.3 軟體產品線

- 系統在部署期間進行組態設定還可以分成幾個層次：
  1. **挑選元件**：此時挑選系統中哪些元件能提供需要的功能。
  2. **定義工作流程和規則**：在此定義工作流程（資訊如何經過處理的步驟），以及針對使用者輸入或系統產生的資料應該確認符合的規則。
  3. **定義參數**：在此指定新建立的應用程式樣例的參數值。

15.4 應用系統再利用

- 應用系統 (application system) 產品是一種不需要修改系統原始碼，只要調整設定就可以符合不同客戶需求的軟體系統。
- 因為這種軟體是為了讓大眾使用而設計的，因此包含許多功能，而這些功能也許可以在不同的環境下再利用，而成為不同應用程式的一部分。
- 應用系統產品的調整是透過內建的組態設定機制，它能把系統的功能依照客戶的需要來修改。例如醫院的病歷系統，需要針對不同類型的病患，分別定義不同的輸入表單和輸出報表。

15.4 應用系統再利用

- 它有些優點遠勝過開發訂做軟體：
  1. 與其他類型的再利用方式一樣，它能更快開發出更可靠的系統。
  2. 因為能看得到應用程式提供什麼功能，所以比較容易判斷它是否合適。
  3. 使用現有軟體可避免某些開發風險，不過這種方式本身也有風險存在，稍後會討論。
  4. 企業可以專注在他們的核心活動，而不必投注一大堆資源在IT系統開發上。
  5. 隨著作業環境演進，技術的更新也可能會變簡單，因為跟上技術是應用系統廠商的責任，而不是客戶的責任。

15.4 應用系統再利用

- 也有它自己的問題：
  1. 為了因應現成應用系統的功能和運作模式，需求通常不得不進行調整。
  2. 應用系統所根據的假設可能無法變更。
  3. 要為企業選出最合適的應用系統可能不容易，特別是很多這類系統並沒有完善的說明文件。
  4. 企業自己可能缺乏經驗，因此必須仰賴廠商和外面的顧問，但是他們給的建議可能會為了賣產品或服務而不夠公正客觀。
  5. 系統的支援和演進是控制在廠商手中，但廠商可能倒閉、被合併或出問題。

15.4 應用系統再利用

組態可設定的應用系統	應用系統整合
單一產品提供客戶所需的功能	整合數個不同的應用系統來提供客戶所需的功能
以某個通用的解決方案和標準化程序為基礎	可因應客戶的程序開發出有彈性的解決方案
開發重點在於系統的組態設定	開發重點在於系統整合
由系統廠商負責維護	由系統擁有者負責維護
由系統廠商為此系統提供平台	由系統擁有者為此系統提供平台

圖 15.12　個別與整合式應用系統

15.4.1 組態可設定的應用系統

- 組態可設定的應用系統 (configurable application system) 是通用的應用系統，設計成可支援指定的企業類型、企業活動或有時是整個企業。例如有個系統是專為牙醫行業設計的，負責處理預約、病歷、回診、帳務等。
- 企業資源規劃 (Enterprise Resource Planning, ERP) 系統就是用來支援大公司內所有關於製造、訂單處理和客服方面的活動。
- 專門領域的應用系統，例如支援企業某方面工作的系統 (如文件管理)，它們提供的功能對某些使用者可能是必要的。
- ERP系統，這類系統是一種大型的整合式系統，專門設計來支援企業程序，如採購和開發貨單、庫存管理和製造流程安排等。

15.4.1 組態可設定的應用系統

- 它的組態設定程序包括挑選模組、設定個別模組、定義企業程序和企業規則 (business rule)，以及定義系統資料庫的結構和架構。

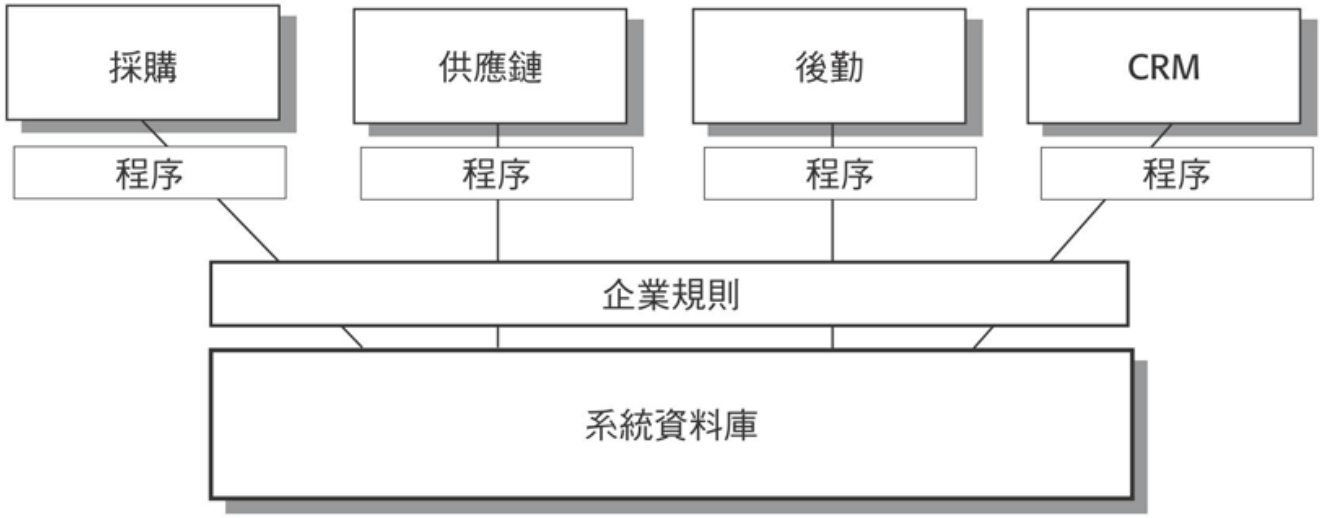

圖 15.13　ERP 系統的架構

15.4.1 組態可設定的應用系統

- 這種架構的主要功能如下：
  1. 有很多模組分別支援不同的企業功能。
  2. 每個模組都有一組相關聯的企業既定程序模型，這與模組中的活動有關。
  3. 有個負責維護企業所有相關功能的資料庫。
  4. 資料庫中的所有資料都必須遵循的一組企業規則。

15.4.1 組態可設定的應用系統

- ERP系統通常都需要進行詳盡的組態設定，來調整系統符合個別客戶的需求。這些組態設定動作可能包括：
  1. 從系統中挑選需要的功能
  2. 建立一個定義企業的資料該如何組織成系統資料庫的資料模型。
  3. 定義資料要遵循的企業規則。
  4. 定義與外界系統預期的互動。
  5. 設計由系統產生的輸入表單和輸出報表。
  6. 設計符合系統所支援的程序模型的新企業程序。
  7. 設定那些用來定義系統如何部署在平台上的參數。

15.4.2 整合式應用系統

- 整合式應用系統 (integrated application system) 內含兩個或更多個應用系統，有時還包括舊系統。
- 採用這種方式的時機，可能是因為沒有任何單一應用系統能符合全部的需要，或者是想要將某個新的應用系統與已經在使用的系統整合。

15.4.2 整合式應用系統

- 假如想要開發整合式應用系統，必須先決定一些設計上的決策：
  1. **哪個個別應用系統所提供的功能最適合？**一般而言，應該會有好幾個系統產品可選。
  2. **資料要如何交換？**一般而言，不同的系統所使用的資料結構和格式都是獨一無二的，而你必須撰寫轉換程式 (adaptor) 來進行轉換。
  3. **產品到底有哪些功能會真正被用到？**個別的應用系統所包含的功能也許會比你所需要的更多，而且在不同產品間的功能經常是重複的。

15.4.2 整合式應用系統

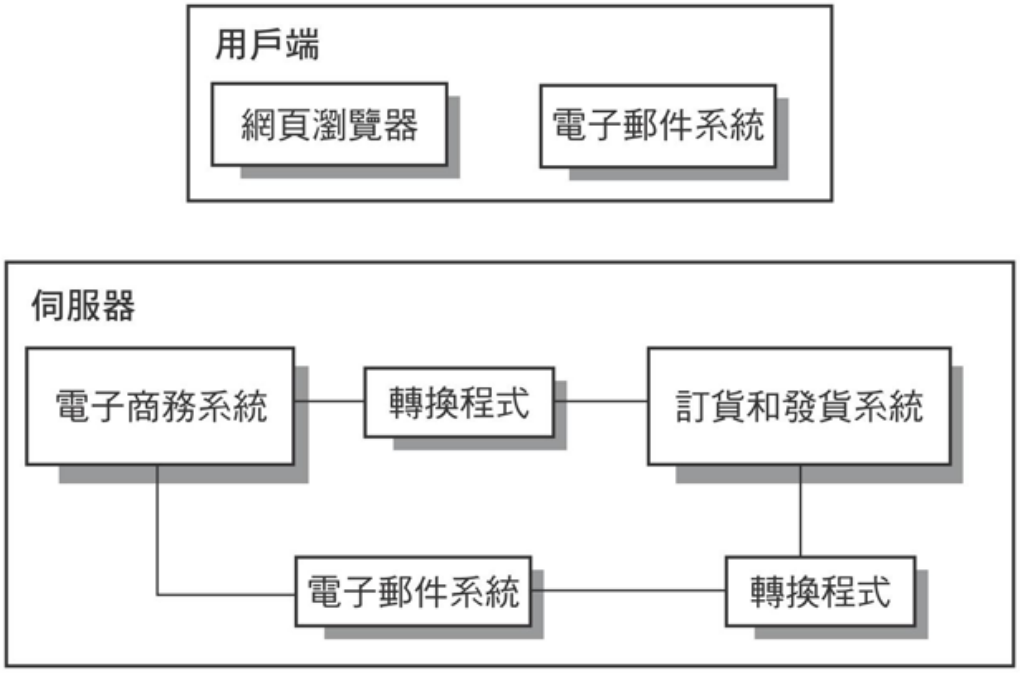

圖 15.14　整合式的採購系統

15.4.2 整合式應用系統

- 假如有使用服務導向 (service-oriented) 方法，就能簡化應用系統的整合。
- 服務導向方法的意思是能夠透過標準的服務介面，每個服務針對一個功能，來存取應用系統的功能。
- 有些應用程式可能會提供服務介面，不過有時服務介面必須由系統整合人員來實作。
- 所以必須撰寫一個包裝程式 (wrapper) 來隱藏應用程式，並提供外界可見到的服務 (圖15.15)。

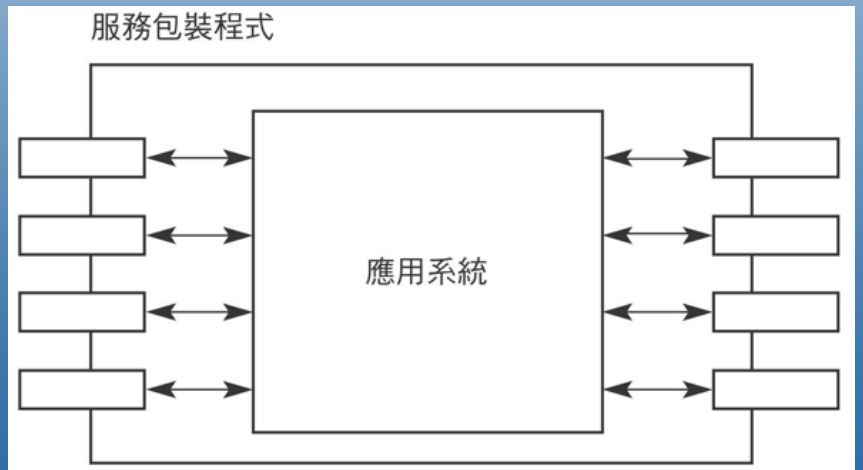

圖 15.15　應用程式的包裝

15.4.2 整合式應用系統

- 系統的整合有以下4個重要問題：
  1. **缺乏對功能與效能的控制**：雖然產品對外發佈的介面有提供一些必要的功能，不過可能設計不當或是執行效能不佳。
  2. **系統交互運作性的問題**：有時候很難讓多個不同的應用系統在一起運作，因為每個系統對自己的使用方式都有不同的假設。結果整合工作變得非常困難。
  3. **無法控制系統的演進**：應用系統的開發廠商會根據市場的反應自行決定系統的變更。
  4. **需要系統廠商的支援**：各個系統廠商對其產品的支援程度差異非常大。