Chapter 7

設計與實作

本章內容

- 軟體設計與實作是軟體工程程序中，負責開發可執行的軟體系統的階段。
- 軟體設計與實作活動 (activity) 一定是交錯進行的。
- 軟體設計 (design) 是一種具創造力的活動，要根據客戶的需求找出軟體元件與它們的關係。而實作 (implementation) 則是將設計實現變成程式的過程。

- 在軟體專案早期就必須決定的最重要的實作決策之一，就是到底應該購買或自己建構應用軟體。很多領域現在已經可以購買市面上現成的應用系統來用，只要設定和稍微修改符合使用者的需求即可。例如病歷系統已經有套裝軟體，很多醫院都在使用，直接買來用會比用傳統程式語言開發一個全新系統更便宜也更快速。
- 假如是以再利用市售產品這種方式開發應用程式系統，則設計程序的重點就變成是如何使用系統的組態設定功能，來符合應用方面的需求。

7.1 使用UML進行物件導向設計

- 物件導向設計程序中包含設計物件類別 (class)，以及設計類別之間的相互關係。
- 要從概念開發系統設計成為詳細的物件導向設計，需要做下面幾件事：

  1. 瞭解與定義系統的環境 (context) 及與外部的互動情形
  2. 設計系統架構
  3. 辨識 (找出) 系統內的主要物件
  4. 開發設計模型
  5. 指定介面

- 接下來我以設計荒野測候站系統 (第1章) 的部分嵌入式軟體為例，來解釋物件導向軟體設計的程序活動。

7.1.1 系統環境與互動

- 任何軟體設計程序的第一個階段，一定是先瞭解該軟體與外部環境之間的關係。
- 第5章也提過，瞭解環境還可以讓你建立系統的邊界。
- 設定系統邊界有助於決定系統中要實作哪些功能，以及哪些功能是在相關聯的其他系統中。

7.1.1 系統環境與互動

1. **系統環境模型 (system context model)**：是一個結構化模型，用來描述環境中的其他系統。
2. **系統互動模型 (interaction model)**：是一個動態模型，用來描述系統實際上如何與它的環境互動。

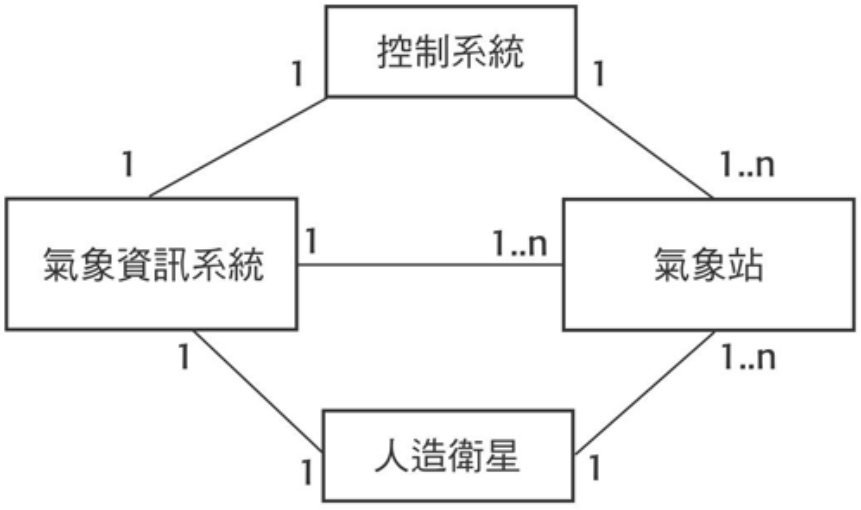

圖 7.1　氣象站的系統環境

7.1.1 系統環境與互動

氣象站的使用案例

報告氣象 (Report weather)：傳送氣象資料到氣象資訊系統
報告狀態 (Report status)：傳送狀態資訊到氣象資訊系統
重新啟動 (Restart)：假如氣象站關閉，重新啟動系統
關閉 (Shutdown)：關閉氣象站
重新設定 (Reconfigure)：重新設定氣象站軟體的組態
省電模式 (Powersave)：把氣象站設定在省電模式
遠端控制 (Remote control)：傳送控制命令給任何氣象站子系統

http://software-engineering-book.com/web/ws-use-cases/

7.1.1 系統環境與互動

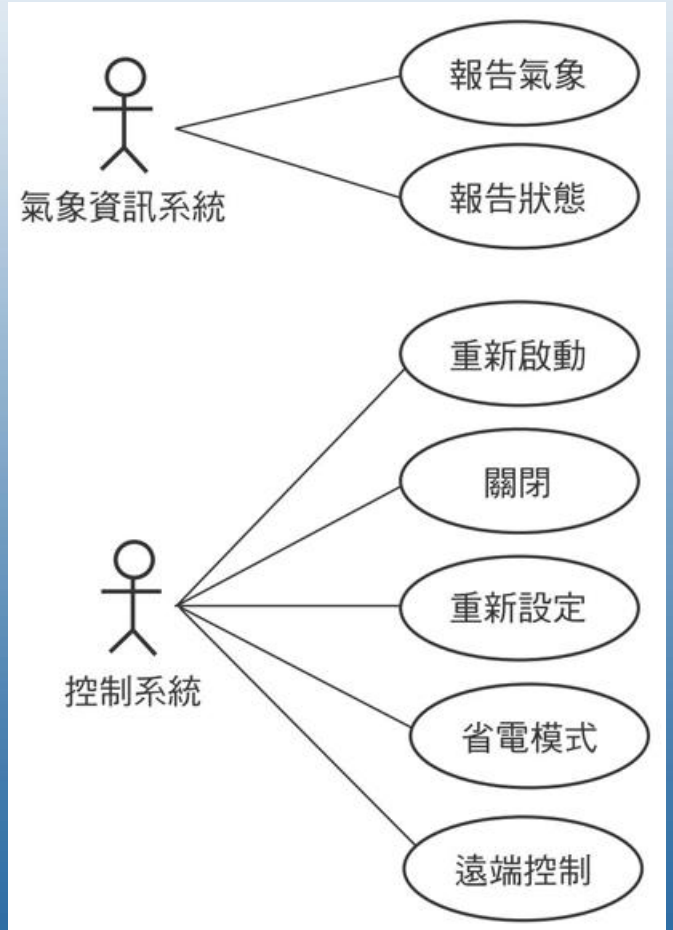

圖 7.2 氣象站的使用案例

7.1.1 系統環境與互動

系統	氣象站 (Weather station)
使用案例	「報告氣象」(Report weather)
活動者	氣象資訊系統、氣象站
資料	氣象站將各儀器在資料收集階段收集到的氣象資料，彙整之後傳送到氣象資訊系統。傳送的資料包括地面和空中的最高、最低與平均溫度；最高、最低與平均壓力：最高、最低與平均風速；總降雨量，還有每 5 分鐘收集一次風向。
刺激	氣象資訊系統與氣象站建立衛星通訊連線，並且要求傳輸資料。
回應	將彙整後的資料傳到氣象資訊系統。
備註	氣象站通常會要求每小時回報一次，不過回報的頻率每個氣象站都不同，而且有可能會修改。

圖 7.3 「報告氣象」(Report weather) 使用案例的描述

7.1.2 架構設計

- 一旦軟體系統和系統環境之間的互動定義好之後，便可以使用這些資訊當作設計系統架構的基礎。
- 首先找出組成系統的主要元件和它們的互動，之後也許可使用某種架構模式(如分層式或主從式模型)來組織元件。
- 氣象站是由獨立的子系統所組成，它們的溝通是透過在共同的基礎架構上，也就是圖7.4的通訊線路上廣播訊息。

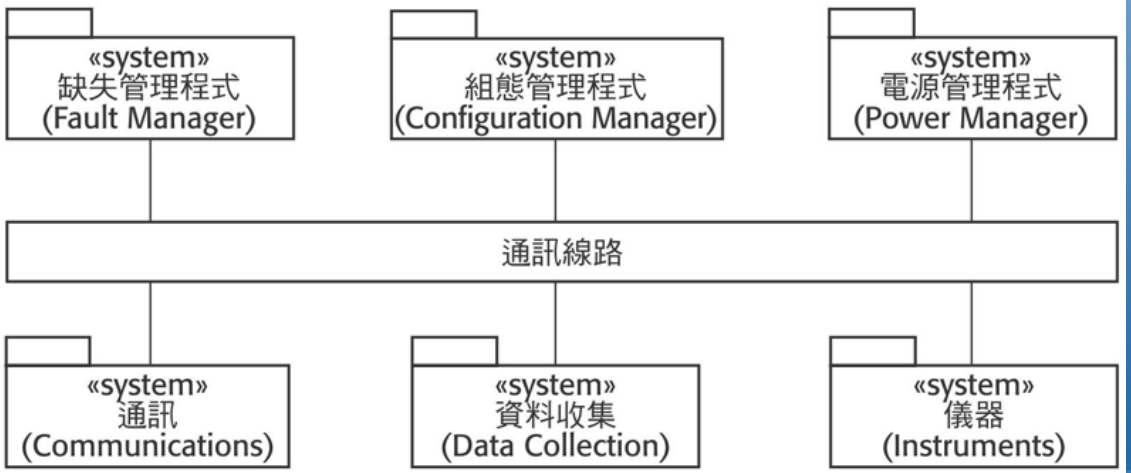

圖 7.4 氣象站的高階架構

7.1.2 架構設計

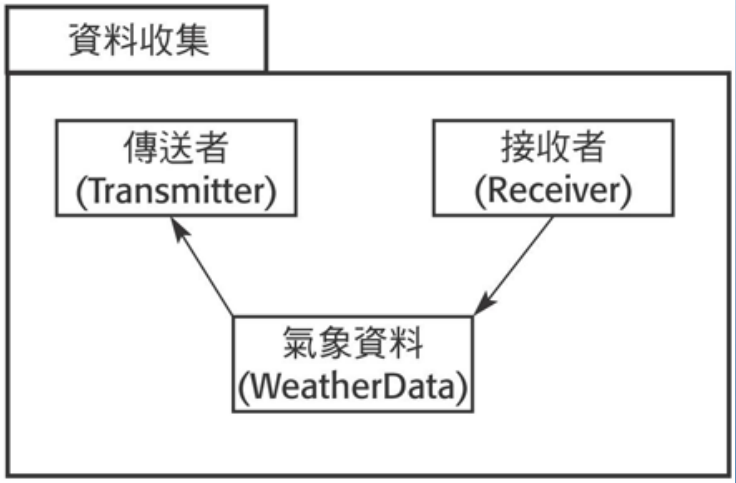

圖 7.5 資料收集系統的架構

7.1.3 辨識物件類別

- 使用案例的描述有助於找出系統中的物件和運算動作。
- 提出下列幾種找出物件導向系統中物件類別的方式：
  1. 對系統的自然語言描述進行文法分析。
  2. 找出應用領域中的有形實體(事物，如飛機)、角色(如經理人)、事件(如請求)、互動(如會議)、地點(如辦公室)和組織單位(如公司)等。
  3. 使用情境分析法，針對系統的各種使用情境 (scenario) 輪流進行辨識與分析。

7.1.3 辨識物件類別

- 氣象站的物件識別是以系統中的有形硬體為主。
- 不過在圖7.6有列出此系統的5個物件類別，其中Ground thermometer (地面溫度計)、Anemometer (風速計) 和 Barometer (氣壓計) 物件是代表應用領域的物件，而WeatherStation和WeatherData物件則是從系統描述和情境描述 (使用案例) 中找出來的：

1. WeatherStation物件類別提供氣象站和其環境的基本介面。
2. WeatherData物件類別負責處理報告氣象的命令。它會傳送來自氣象站中各個不同儀器的彙整資料到氣象資訊系統。
3. Ground Thermometer、Anemometer和Barometer物件類別則與系統中使用的各個儀器有直接的關係。它們反映系統中有形的硬體實體，以及控制這些硬體的相關運算動作。

7.1.3 辨識物件類別

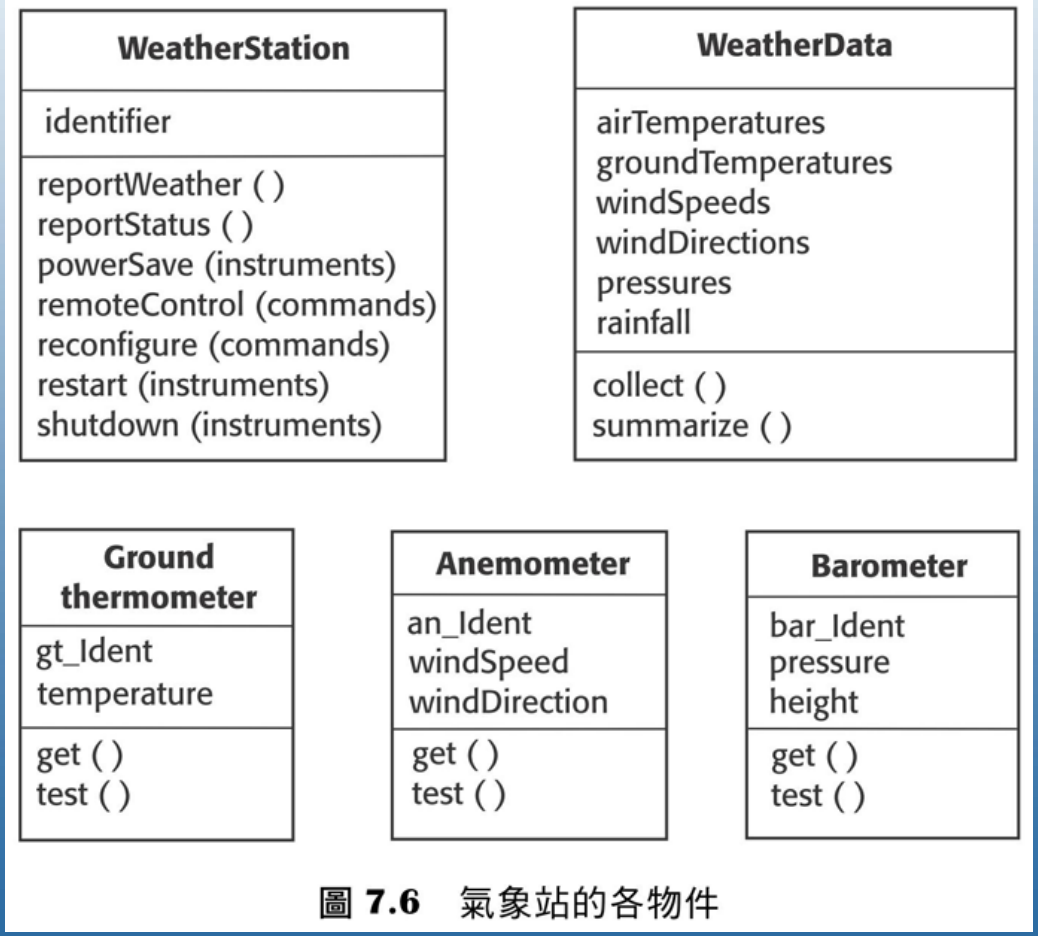

圖 7.6 氣象站的各物件

7.1.4 設計模型

- 第5章探討過的設計或系統模型可展示出系統中的物件或物件類別，以及這些實體之間的關聯性和關係。這些模型可當成系統的需求與實作之間的橋樑。
- 使用UML開發設計時，應該要產生下列2種設計模型：
  1. **結構化模型**：以系統的物件類別和它們的關係來描述系統的靜態結構。
  2. **動態模型**：是描述系統的動態結構和展示系統物件之間的執行期間互動關係。

7.1.4 設計模型

- 有3種UML模型是特別適合用來對使用案例和架構模型加上細節：
  1. **子系統模型 (subsystem model)**：將物件依照邏輯關聯分組成為子系統。
  2. **序列模型 (sequence model)**：用來表示物件互動的順序。
  3. **狀態機模型 (state machine model)**：用來表示個別物件如何回應事件而改變自己的狀態。

7.1.4 設計模型

- 子系統模型它可以表示如何將設計有系統的分成邏輯上相關的物件群組。
- 除了子系統模型外，也可以設計細部的物件模型，顯示系統中所有物件和它們的關聯性 (繼承關係、一般化關係、聚合關係等)。
- 序列模型是動態模型，它針對每個互動模式，描述將發生的一連串的物件互動。

7.1.4 設計模型

- 序列圖是由上而下讀。

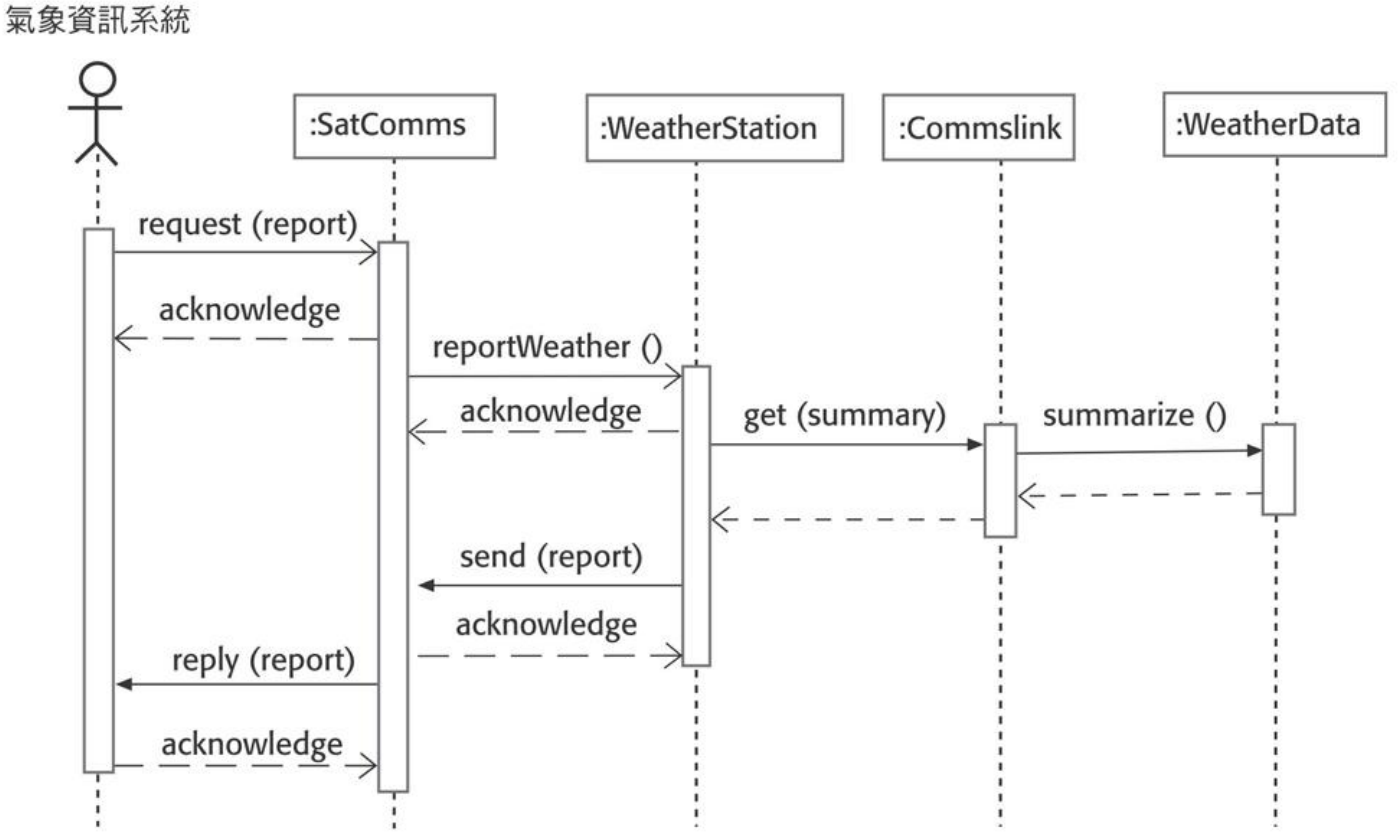

圖 7.7　描述資料收集的序列圖

7.1.4 設計模型

- 可能你也會想要摘要描述單一物件或子系統回應訊息和事件的行為，這可以使用狀態機 (state machine) 模型，來表示物件樣例如何根據接收到的訊息變更它的狀態。

7.1.4 設計模型

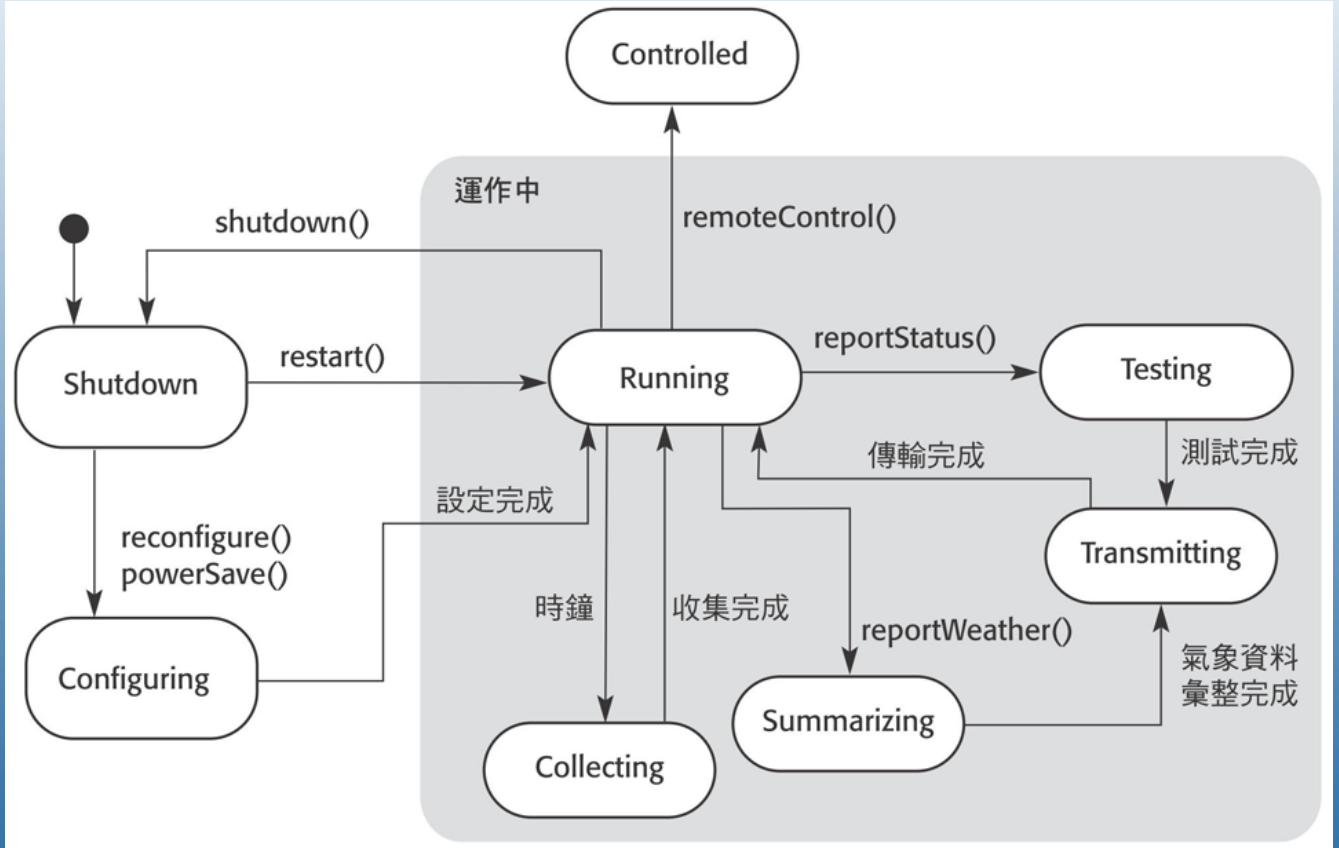

圖 7.8　氣象站的狀態圖

7.1.4 設計模型

- 狀態圖對於系統或某個物件的運作是很有用的高階模型。不過，通常不需要為系統中所有的物件都製作狀態圖。因為系統中有許多物件是非常簡單的物件，它們的運作根本不需要狀態模型也很容易想像。

7.1.5 介面規格

- 這些介面必須先明確制訂，各個不同的物件和子系統才可以分頭平行的進行設計。
- 介面可用UML來制訂，使用與類別圖同樣的符號表示法。
- 設計人員在介面設計中應該避免出現如何表達資料的細節，因此屬性不應該在介面規格中定義，但是應該要有能存取和更新資料的運算動作。
- 物件和介面之間不一定是1:1的簡單關係，同一物件也可能有好幾個介面，每一個代表它所提供方法的一個觀點。
- Java有直接支援這項功能，Java的介面宣告與物件和實作介面的物件都是分開的。

7.1.5 介面規格

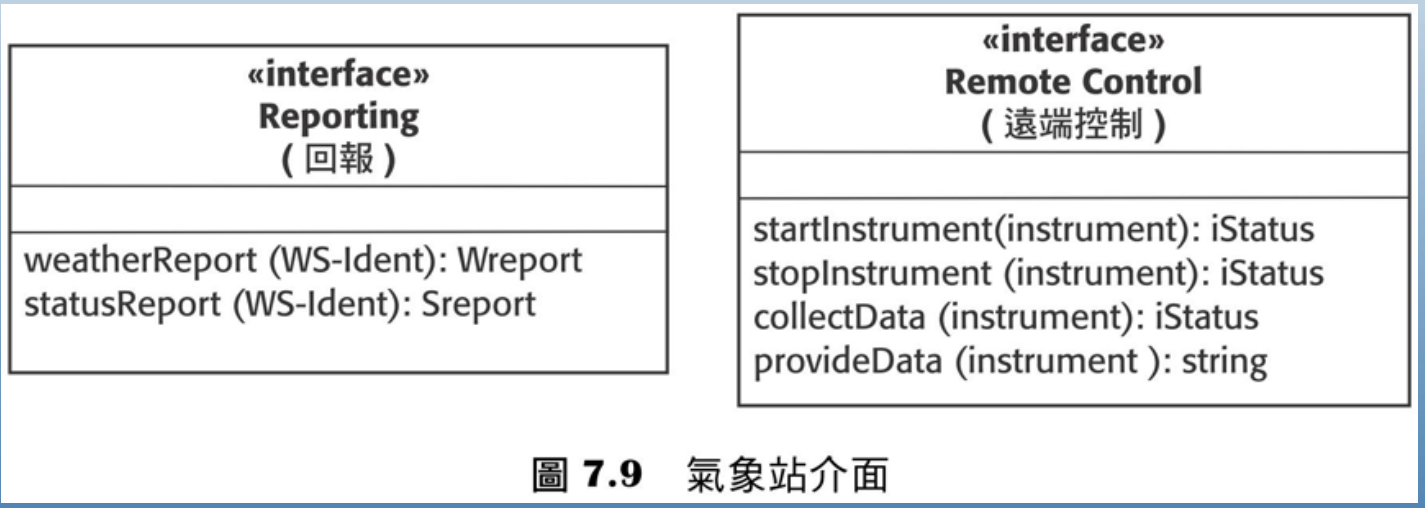

圖 7.9　氣象站介面

7.2 設計模式

- 「模式」(pattern) 是一種描述問題與其解決方案要素的方法，目的是要讓同一個解決方案可以重複應用在不同環境中。
- Hillside Group網站 (hillside.net/patterns/) 是專門維護模式的相關資訊，以下是摘錄自該網站的一段話：

> 模式 (*Pattern*) 與模式語言 (*Pattern Language*) 都是描述最佳實務經驗、良好的設計和擷取經驗的方式，這種描述方式是要儘量能讓其他人再利用 (*reuse*) 這些經驗。

- 模式是一種再利用其他設計人員的知識和經驗的方式。設計模式通常與物件導向設計有關。已發行的模式經常依靠物件的特性，例如繼承 (inheritance) 與多型 (polymorphism) 來提供一般性 (generality)。

7.2 設計模式

- 設計模式有以下4個要素：
  1. 一個代表模式的有意義名稱。
  2. 一個問題領域的描述。
  3. 一個解決方案的描述。這不是具體的設計描述，只是某個設計解決方案的樣板，能以不同方式來建立樣例(instantiate)。
  4. 一個結果的陳述，說明套用這個模式的結果和權衡得失。

7.2 設計模式

模式名稱：Observer

模式描述：將物件狀態的顯示與物件本身分開，並且提供其他不同的顯示方式。當物件狀態改變時，會自動通知並更新所有顯示，以反映新的物件狀態。

問題描述：在許多情況下，必須對某些狀態資訊提供多種顯示方式，如圖形顯式或表格顯示。在指定資訊時還不必知道所有的表示方式。所有表示方式都應該支援互動操作，而且當狀態改變時，所有顯示都必須被更新。

這個模式可以應用在需要以多種顯示格式來顯示狀態資訊的情況下，而且物件不需要知道所使用的顯示格式，也可以維護狀態資訊。

解決方案描述：它定義 2 個抽象物件：Subject 和 Observer，以及兩個具體物件：ConcreteSubject 和 ConcreteObject，這兩個物件會繼承相關抽象物件的屬性。抽象物件包含在所有情況都適用的通用運算動作。顯示的狀態是由 ConcreteSubject 負責維護，它會繼承 Subject 的運算動作，因此可以新增和移除 Observer (每種 observer 對應到一種顯示)，以及在狀態改變時發出通知。

ConcreteObserver 會複製一份 ConcreteSubject 的狀態，並且實作 Observer 的 Update() 介面，這個介面可以保持這些複製而來的狀態副本是最新的。ConcreteObserver 會自動顯示它的狀態，並且當狀態更新時反映它的改變。這個模式的 UML 模型如圖 7.12 所示。

結果：Subject 只知道抽象的 Observer，並不知道具體類別的細節。因此，在這些物件之間只有最少的耦合關係。由於缺乏這些知識，所以不容易提升顯示的執行效能。對 Subject 所做的改變可能會導致一連串相關的 Observer 被更新，其中有些可能是不需要的。

圖 7.10 Observer 模式

7.2 設計模式

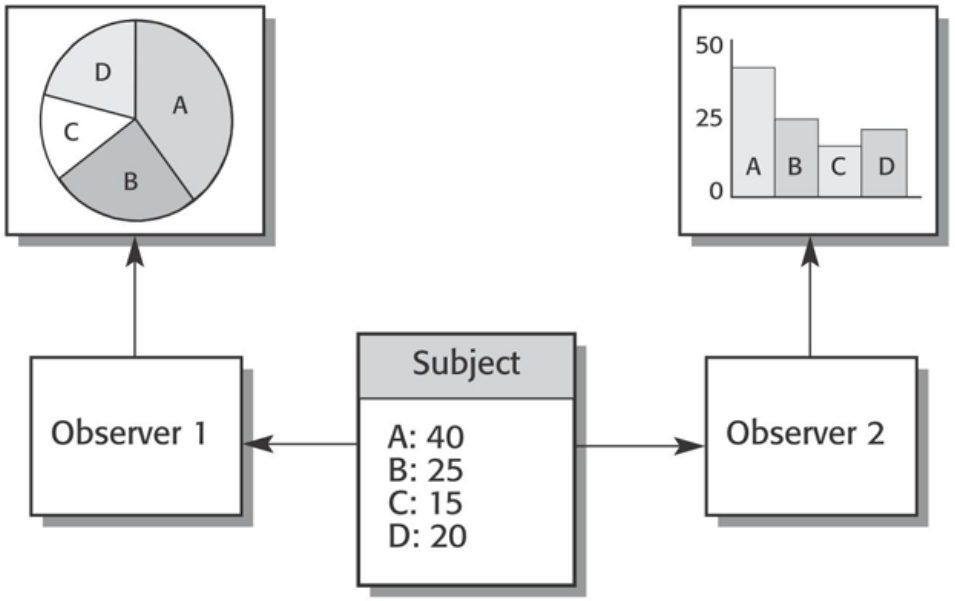

圖 7.11　多種顯示方式

7.2 設計模式

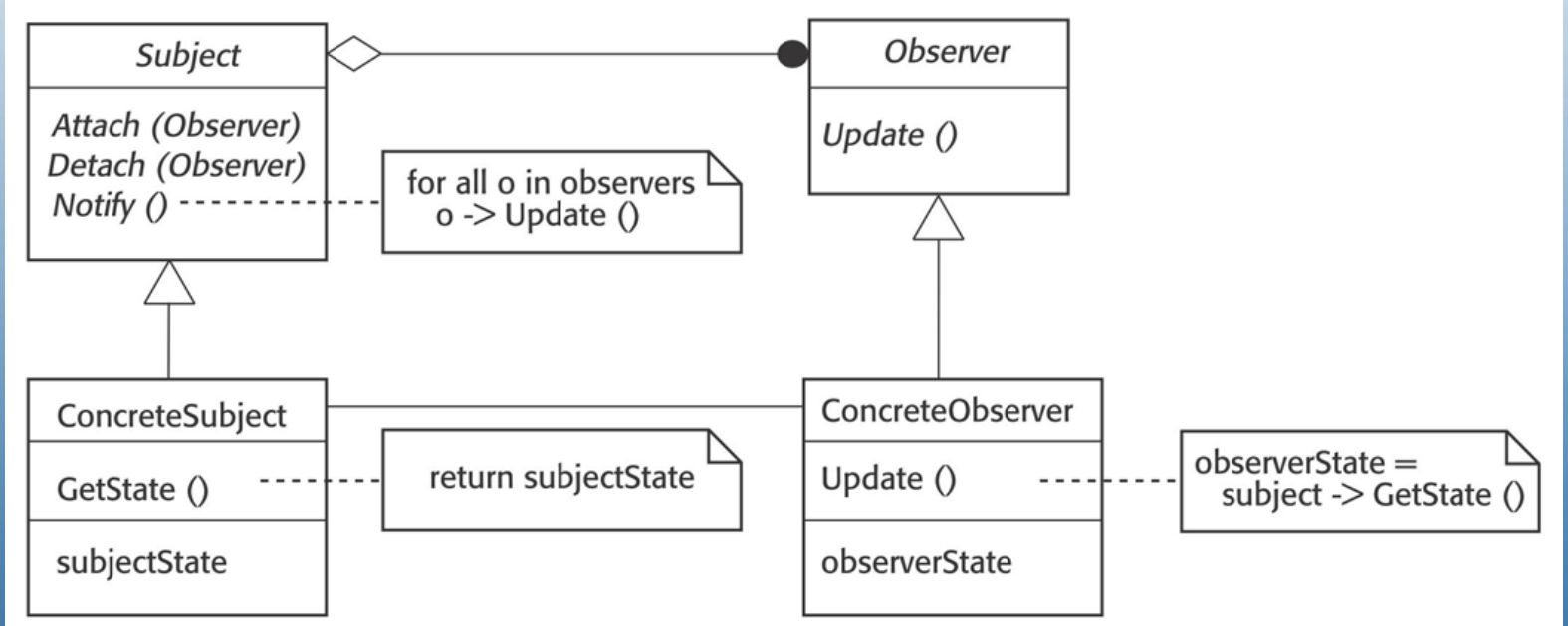

圖 7.12 Observer 模式的 UML 模型

7.2 設計模式

- 在設計中要使用模式，需要先確認你面對的設計問題有相關聯的模式可以套用。
  1. 要通知一些物件有些其他物件的狀態已經改變 (Observer模式)。
  2. 將一些逐步開發的相關物件的介面加以整理 (Façade模式)。
  3. 提供一種存取集合內元素的標準方式，無論該集合是以什麼方式實作 (Iterator模式)。
  4. 允許能夠擴充現有類別在執行期間的功能 (Decorator模式)。

7.3 實作方面的議題

- 本章的解說方式與程式語言無關，而是把重點放在一般程式設計教科書沒講，但是對軟體工程特別重要的實作方面的主題。

1. **再利用 (Reuse)**：多數現代軟體都是再利用現有的元件或系統建構而成。在開發軟體時應該儘量利用現有的程式碼。
2. **組態管理 (Configuration management)**：在開發程序期間，每個軟體元件都會產生很多個不同的版本。
3. **主機／目標系統開發 (Host-target development)**：開發完成生產出來的軟體，將來執行它的電腦不一定會和軟體開發環境一樣。反而經常是你在一種電腦上開發 (主機系統，host system)，出來的產品在另一種電腦上執行 (目標系統，target system)。

7.3.1 再利用

- 從1960年代到1990年代，大部分的新軟體都是從零開始開發，全部的程式碼都是用高階程式語言一行一行寫出來的。
- 唯一可稱得上有進行再利用的是使用程式語言內建程式庫 (library) 的函式 (function) 或物件。
- 成本和時程壓力讓這種方式變得越來越不可行，特別是商業系統或以Internet為基礎的系統。
- 結果興起這種以再利用現有軟體為主的開發方式，現在已經廣泛應用在所有以web為基礎的系統、科學軟體。

7.3.1 再利用

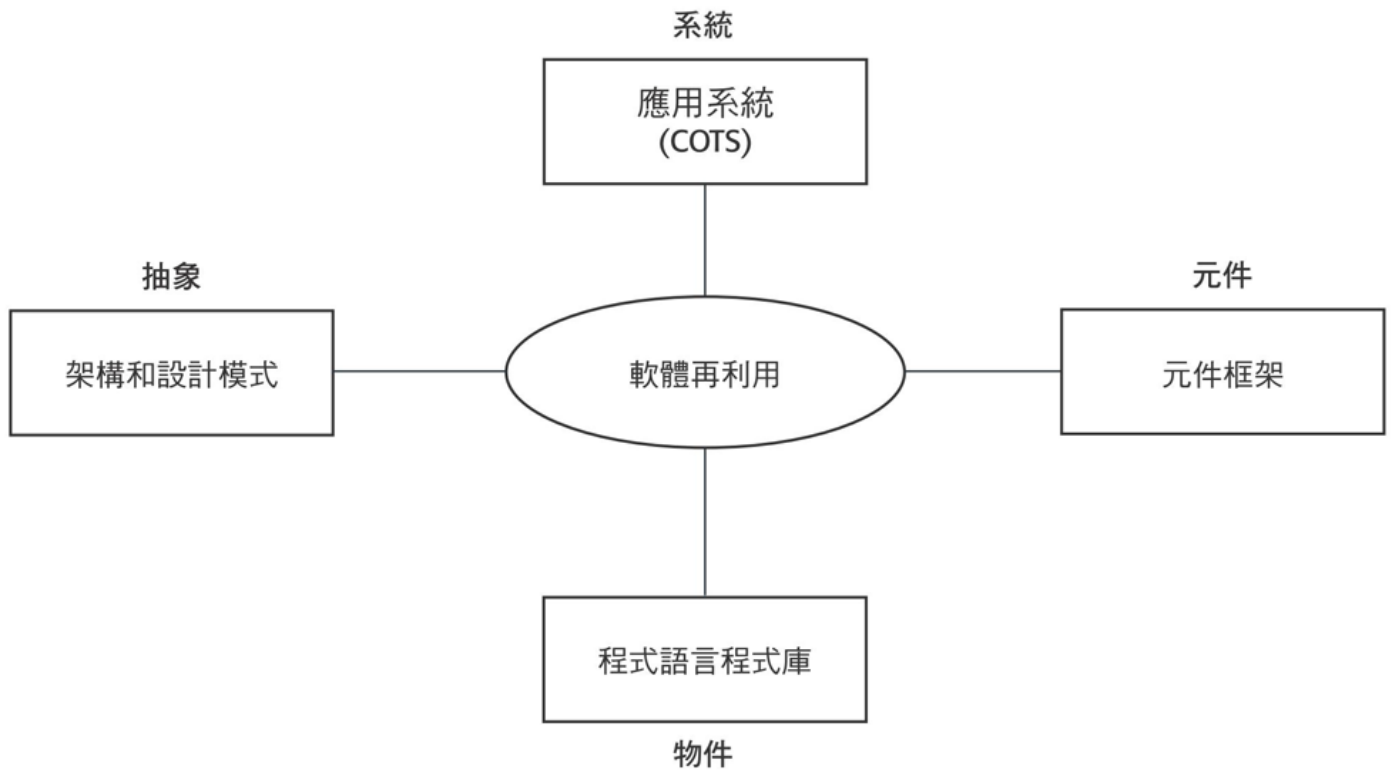

圖 7.13 軟體再利用

7.3.1 再利用

- 軟體再利用的層級可分成以下幾個不同的層級：
  1. **抽象層級**：在此層級不是直接再利用軟體，而是在設計中運用知識或成功的抽象成果。
  2. **物件層級**：在此層級是直接再利用程式庫裡的物件，而不是自己撰寫程式碼。
  3. **元件層級**：元件 (component) 是一群為了提供相關功能和服務，而一起運作的物件和物件類別的集合。
  4. **系統層級**：在此層級是再利用整個應用程式系統。

7.3.1 再利用

- 再利用也有下列相關成本：
  1. 花在尋找可再利用軟體和評估是否符合需要的時間成本。
  2. 採購可再利用軟體的成本。如果是大型的現成系統，價格可能很高。
  3. 花在調整和設定可再利用軟體的元件或系統，以符合新系統需求的成本。
  4. 每一個可再利用軟體彼此整合的成本，加上與你撰寫的新程式碼整合的成本。

7.3.2 組態管理

- 軟體開發過程一直都在發生變化，所以變更管理是絕對必要的。
- 組態管理是一種管理變動中軟體系統的程序。組態管理的目標是支援系統整合程序，讓所有開發人員都能以有管控的方式存取專案的程式碼和文件、查詢已經做了哪些變更，以及編譯和連結元件來建構出一個系統。

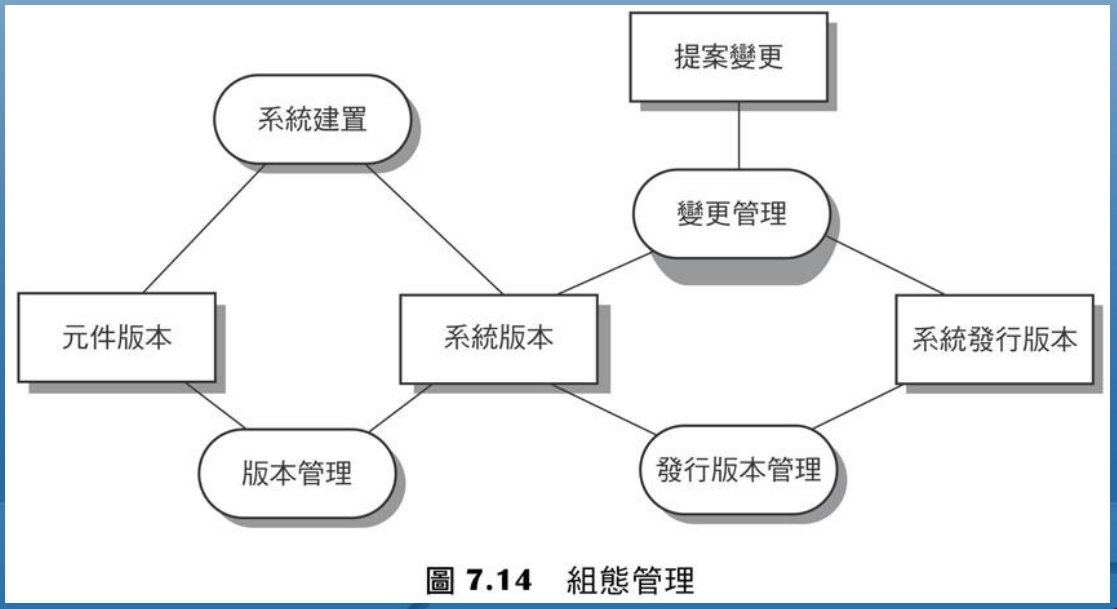

圖 7.14　組態管理

7.3.2 組態管理

- 組態管理有4個基本活動：
  1. **版本管理**：記錄軟體元件各個不同的版本。版本管理系統的功能主要是協調多位程式設計人員的開發工作。
  2. **系統整合**：協助開發人員定義要用元件的哪個版本來建立系統的每個版本。這裡的定義之後會被用來自動編譯和連結必要元件建構出一個系統。
  3. **問題追蹤**：讓使用者能夠回報錯誤和問題，並且開發人員能看到誰在負責處理這些問題以及何時被修復。
  4. **發行版本管理**：在此將軟體系統的新版本交付給客戶。

7.3.3 主機 / 目標系統開發

- 大多數專業軟體的開發都是以主機／目標 (host-target) 模型為基礎 (圖7.15)。軟體在某台電腦 (主機) 上開發，但在其他不同的機器上執行 (目標)。更廣義的講法是分別稱呼為開發平台 (主機) 和執行平台 (目標)。
- 平台 (platform) 不只是硬體而已，還包括上面安裝的作業系統加上其他的支援軟體，如資料庫管理系統；或是以開發平台而言，上面可能還會有互動式開發環境。

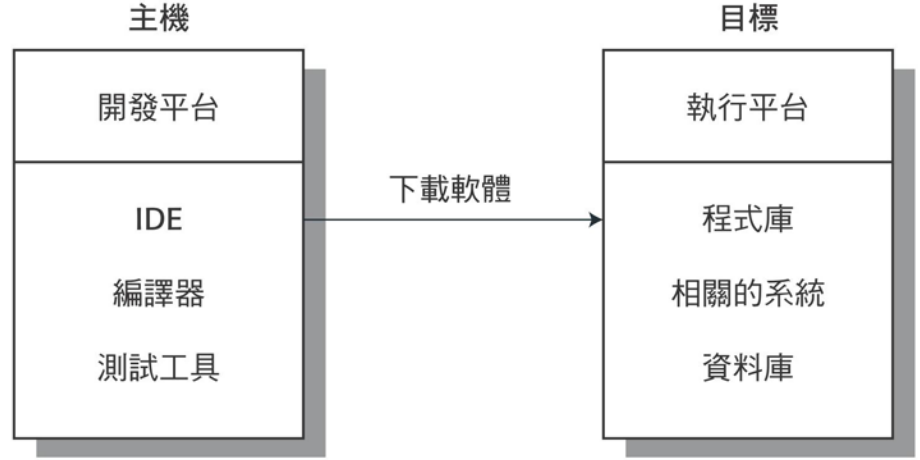

圖 7.15 主機 / 目標系統開發

7.3.3 主機 / 目標系統開發

- 軟體開發平台應該提供一些工具來支援軟體工程程序。包括可能有：
  1. 一個整合式編譯器和語法導向的編輯系統，方便輸入、編輯和編譯程式碼。
  2. 程式語言除錯系統。
  3. 圖形化編輯工具，例如編輯UML模型的工具。
  4. 測試工具如JUnit，它可以自動在程式的新版本上執行一整組測試。
  5. 支援重構和程式視覺化的工具。
  6. 用來管理不同版本的原始碼和整合與建置系統的組態管理工具。

7.3.3 主機 / 目標系統開發

- 現在的軟體開發工具通常會被組織成一個整合式開發環境(integrated development environment, IDE)。IDE是由支援軟體開發各方面的一組軟體工具所組成，它們共用同樣的框架和使用者介面。
- 通常IDE會專門支援某種程式語言的開發工作，如Java。這種綁定語言的IDE可能是專門針對它開發的，也可能是設定某種通用用途的IDE再搭配特定語言支援工具產生出來的。

7.3.3 主機 / 目標系統開發

- 不過假如是分散式系統，就需要思考哪些元件要部署在哪個平台上。需要考慮的問題包括：
  1. **元件的軟硬體需求**：如果元件是專為特定硬體架構所設計的，或是需要依賴其他軟體系統，顯然它一定要部署在有提供這些軟硬體的平台上。
  2. **系統可用性的需求**：高可用性系統可能會要求元件要部署在多個平台上。這表示萬一發生平台故障情況，元件還有另一套實作版本可用。
  3. **元件通訊**：如果元件間的通訊流量很大，通常會把它們部署在同一平台或實體位置彼此接近的不同平台上。這樣可減少通訊的延遲時間。

7.4 開放原始碼的開發

- 開放原始碼 (open source) 開發是一種軟體開發方式，完成的軟體系統它的原始程式碼會公開出來，邀請志願人士參與開發程序 (Raymond 2001)。
- 這個構想是源自免費軟體基金會 (Free Software Foundation, www.fsf.org)，它提倡程式的原始碼不應該是專屬的，而是應該可提供給使用者檢驗，並依照需要修改。

7.4 開放原始碼的開發

- 開放原始碼軟體透過Internet將這個想法發揚光大，透過Internet可以招募到人數更多的志願開發人員，其中很多人同時也是原始碼的使用者。
- 開放原始碼產品Linux作業系統是最被廣泛使用的伺服器系統，還有Apache網站伺服器也是。其他重要而且被大量採用的開放原始碼產品，包括Java、Eclipse IDE和mySQL資料庫管理系統。

7.4 開放原始碼的開發

- 軟體開發公司必須考慮下列兩個有關開放原始碼的問題：

  1. 我們的這個產品要利用開放原始碼元件來開發嗎？
  2. 我們的這個產品是否要採用開放原始碼方式來開發？

- 現在有很多軟體產品公司使用開放原始碼方式來開發，尤其是特殊用途的系統。他們的商業模式不是靠販賣軟體產品，而是販賣該產品的支援服務來營利。他們認為參與開放原始碼團體，能讓軟體以更便宜更快速的方式來開發，同時還能為這個軟體建立出一群死忠用戶。

7.4.1 開放原始碼的授權方式

- 雖然開放原始碼開發的一個基本原則是程式的原始碼是免費可用的，但這並不表示任何人可以任意處置這些原始碼。
- 在法律上原始碼的作者 (可能是公司或個人) 還是擁有 (own) 這些程式碼，透過在開放原始碼軟體授權 (license) 中附加法律條件，他們可以限制它的用途 (St. Laurent 2004)。
- 有些開放原始碼作者認為如果使用開放原始碼元件來開發新系統，那麼這個新系統也應該屬於開放原始碼。
- 另外有些作者願意無條件讓大家使用他們的程式碼，開發出來的系統可以是私有的而對外販售，也可以不公開原始碼。

7.4.1 開放原始碼的授權方式

1. **GPL (GNU General Public License)**：這是所謂的「互惠」(reciprocal) 授權。簡單的說，就是如果你使用GPL授權模式的開放原始碼軟體，那麼你完成的軟體也一定要是開放原始碼。
2. **LGPL (GNU Lesser General Public License)**：這是GPL授權的一種變化形式，你可以連結使用開放原始碼程式來撰寫元件，而不需要公開這些元件的原始碼。
3. **BSD授權 (Berkley Standard Distribution (BSD) License)**：這是非互惠的授權模式，意思就是你對開放原始碼的程式進行任何變更或修改不需要受限。

7.4.1 開放原始碼的授權方式

- Bayersdorfer (Bayersdorfer 2007) 建議想要使用開放原始碼來開發專案的公司應該做這些動作：
  1. 建立一個系統來維護已下載和使用的開放原始碼元件的相關資訊。
  2. 注意授權的不同種類，並且在使用元件前先了解它的授權方式。
  3. 注意元件的演進過程。
  4. 教育人員有關開放原始碼的觀念和授權方式。
  5. 稽核系統要到位。
  6. 參與開放原始碼團體。