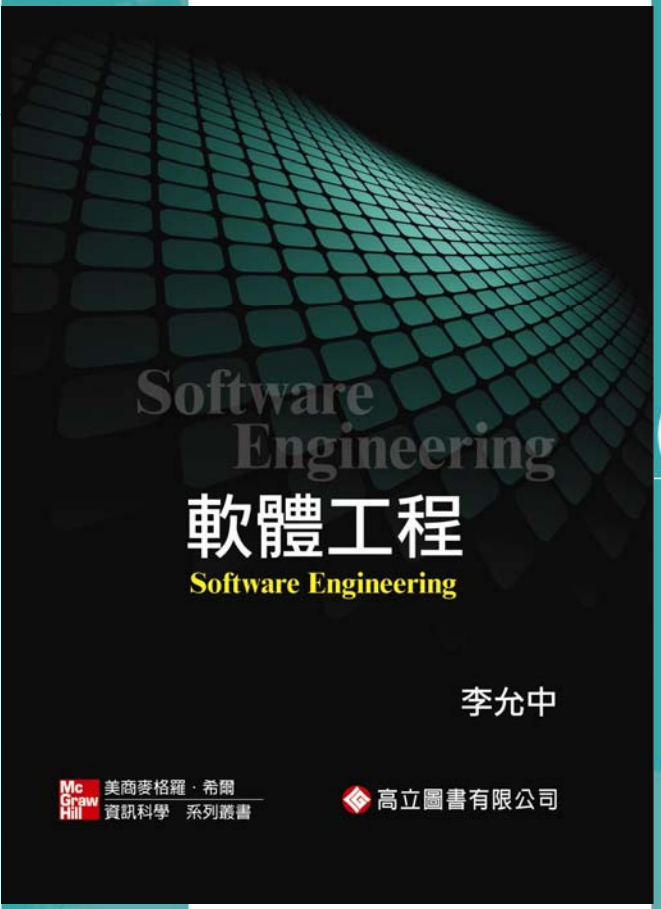


CHAPTER 1

軟體危機與流程

1

大綱

- 軟體危機
- 基本的軟體開發活動
- 軟體流程模型
- 本章總結

1 軟體危機

- 國際著名Standish Group研究機構之調查，有84%的軟體計劃無法於既定的時間及經費當中完成 。
  - 1995年，以美國境內8000個軟體專案為調查樣本。
  - 超過 30%的計畫執行到中途被取消。
  - 專案的預算平均超出189 %。
- 問題癥結
  - 軟體公司總是在不合理的期限(Unrealistic Deadline)壓力下進行開發；
  - 客戶在專案結束前要求增加新功能，或是給予不明確的需求(Vague Requirements)；
  - 軟體本身非常複雜(Complex Structure) ；
  - 專案開發過程中具有許多不確定因素(Numerous Uncertainties)。

軟體危機－真實案例

- 美國銀行信託軟體系統開發案
  - 時間：
    - 1982年進入信託商業領域，並著手規劃發展 信託軟體系統。
  - 規劃時程： 花了18個月進行該系統之規劃及分析。
    - 計畫原訂預算： 2千萬美元；
    - 原訂開發時程： 9 個月, 預計1984/12/31完成；
    - 開發期間：直至1987年三月都未完成本系統， 並已經投入6千萬美元。
  - 後果：
    - 失去了6億美元的信託生意商機，最後因為此系統並不穩定而不得不放棄此系統，並將340億美元的信託帳戶轉移出去。
- 其他案例:
  - 1996年，亞利安五號原型爆炸。
  - 1998年，波音Delta III火箭爆炸。

Fredrick Brooks所提出的軟體本質問題

- 複雜性
  - 軟體系統之複雜程度，往往隨著程式的大小及軟體元件個數以非線性的方式，甚至是等比級數的方式遞增。 例如：系統的狀態、 程式碼大小、 系統功能、 程式靜態結構、系統動態行為等各層面，會越漸複雜。
- 易變性
  - 為了滿足客戶的需求，一套成功的軟體系統，從開發到完成、從產品交付到營運維護，隨時都有可能要做變更。
- 隱藏性
  - 軟體本身是看不到、摸不著的，導致需求容易存有誤解、疏忽的地方不容易被發現，而大大的妨礙了彼此溝通的進行。
- 一致性
  - 在大型的協作環境下發展軟體系統，介面跟介面間、模組跟模組間、系統跟系統間的介接，便都存有一致性的問題需要解決，因此需要透過各種方式來轉換或是介接不一致的地方。

基本的軟體開發活動

- 軟體開發主要活動有—
  - 需求分析
    - 了解客戶的需求、分析系統的可行性、分析需求的一致性，以及正確性等。
  - 設計
    - 將需求轉換為系統的重要過程。
    - 包含架構設計、模組間的介面設計、資料庫設計、演算法設計與資料結構設計等。
  - 實作
    - 透過程式語言將所設計的內容轉化為可以執行的軟體系統。
    - 「除錯」是實作活動中無可避免的工作，主要是移除程式撰寫過程中所產生的錯誤。
  - 測試與維護
    - 測試是對實作活動階段所產出的程式碼模組進行檢測，檢驗其功能是否正確、效能是否符合要求。
    - 「軟體維護」的目的是要確保已經發行的軟體系統可以持續滿足客戶需要。

軟體測試

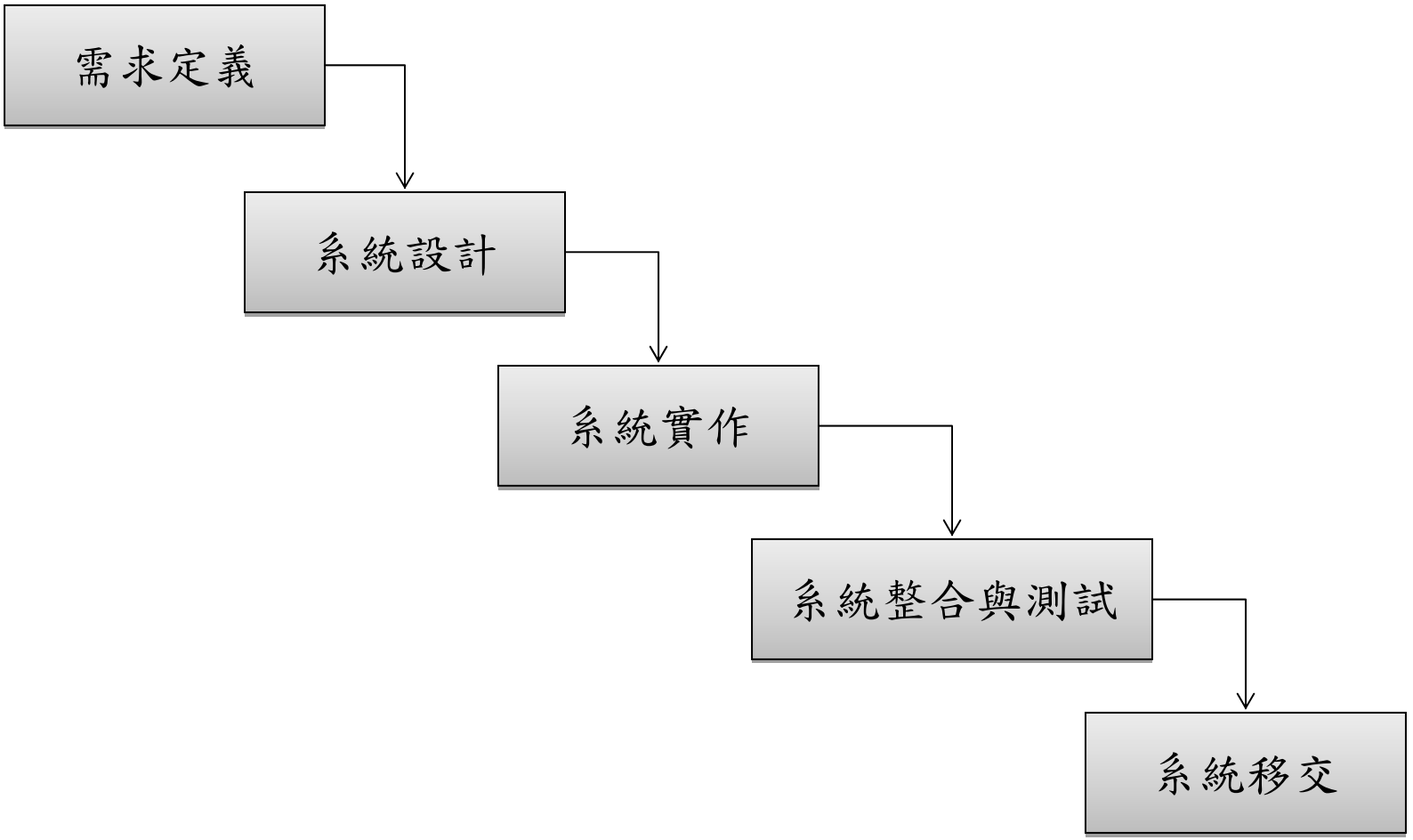
- 軟體測試包含有—
  - 單元測試：
    - 測試單元模組功能是否正常運作。
  - 整合測試：
    - 測試模組或子系統的介面整合是否正常運作。
  - 系統測試：
    - 測試整體系統的效能、安全性、穩定度等非功能性需求是否如預期。
  - 接受測試：
    - 測試系統的整體性運作是否符合使用者的需求。

軟體流程模型

- 軟體流程模型是
  - 抽象的軟體流程樣板，提供組織定義軟體開發的流程指引。

- 常見的軟體流程模型有：
  - 瀑布式開發流程
  - 統合流程
  - 極限製程

瀑布式開發流程



瀑布式開發流程

- 瀑布式開發流程的概念是Winston W. Royce 於1970 年提出，因為其描述各開發階段的順序性相當明確，所以稱為瀑布式開發方式。
- 此模型分成五個階段：
  - 需求定義
    - 主要目的是在了解顧客的需求或建立產品的功能需求，許多的活動，例如，需求擷取、需求訪談、需求分析等都會在這個階段進行。這個階段典型的產出是軟體需求規格書。
  - 系統設計
    - 依照上一階段所產出的軟體需求規格書進行設計。包含架構設計與細部設計（例如，介面設計、資料庫設計等）。此階段主要產出為系統設計規格書。
  - 系統實作
    - 將系統設計規格書落實為可以執行的軟體程式碼，並進行單元測試。這個階段的產出主要為程式碼與單元測試的結果。

瀑布式開發流程

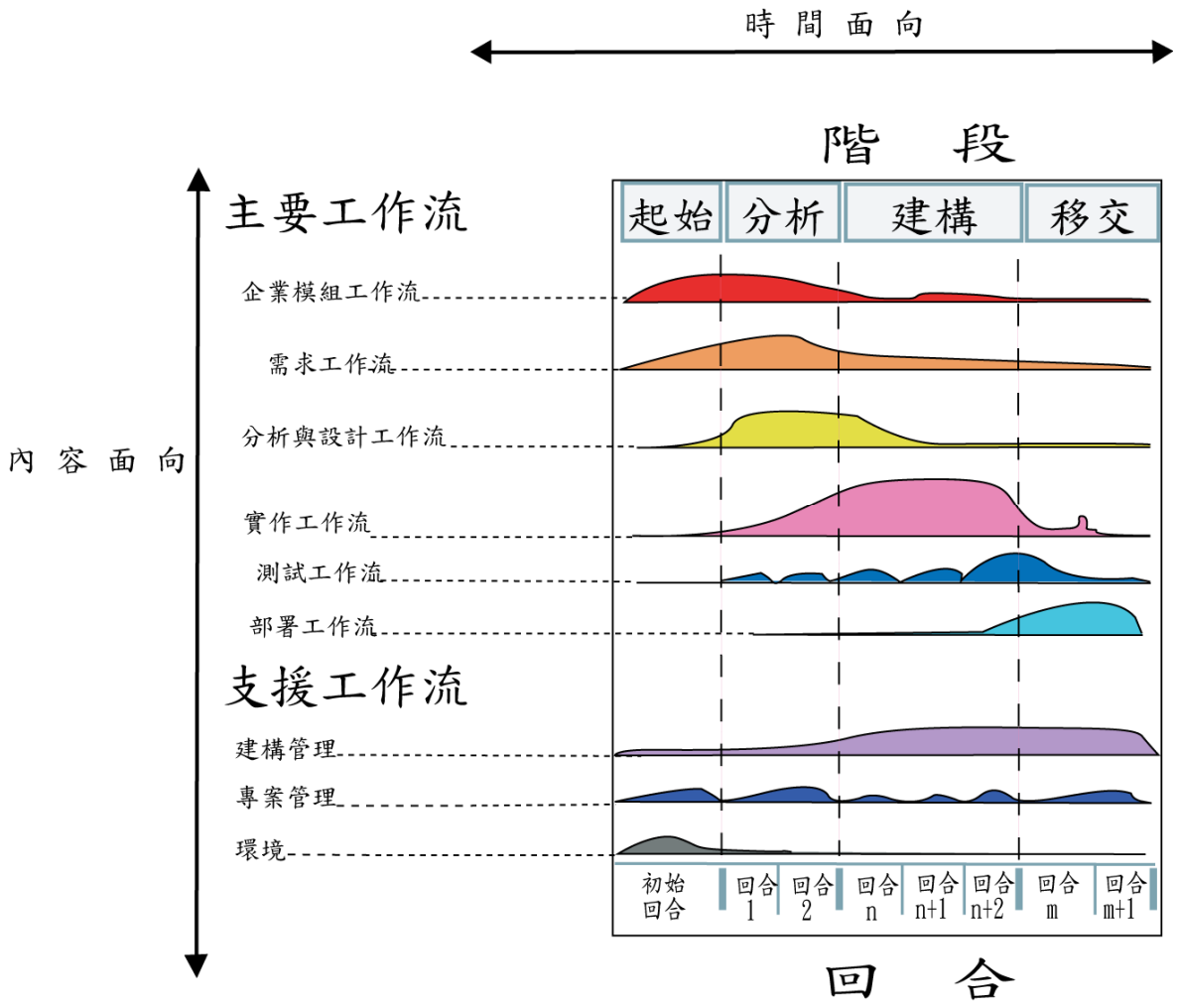
- 系統整合與測試
  - 系統整合係指依據系統設計規格書的架構逐步整合各子系統或模組，並進行整合測試，以確定各子系統可以正確無誤地整合。系統測試係針對整個系統進行整體性的測試，以確保其功能性、效能性都可以符合需求規格書的描述。此階段的主要產出為系統測試報告。
- 系統移交
  - 當系統測試無誤並進行移交後，此軟體系統也進入維護階段。維護階段通常很長，主要在處理錯誤的修復以及功能的增強。
- 瀑布式開發時程規劃範例

	1	2	3	4	5	6	7	8	9	10	11	12
需求定義	●	●										
系統設計			●	●	●							
系統實作						●	●	●				
系統整合與測試									●	●	●	
系統移交												●

瀑布式開發流程

- 瀑布式開發流程方式的優點有：
  - 清楚的階段分割；與一般的工程方法（例如，建築、土木、化學工程等）一樣有明確的流程階段，使專案的控管更為容易。
  - 明確的文件產出；使得合約的簽訂、技術或管理的審查更為容易，也有利於後續專案的維護。
- 限制瀑布式開發流程方式的因素有：
  - 需求提供者沒有辦法正確及完整地表達需求。
  - 使用者、系統分析師、系統設計師、程式設計師之間傳遞訊息上的誤解。
  - 使用者的需求經常改變。
- 該流程模型之適用範圍
  - 瀑布式開發流程方式較適用於大型專案，且需求變更幅度不大的系統。

統合流程


時間面向
階段
起始
分析
建構
移交
內容面向
主要工作流
企業模組工作流
需求工作流
分析與設計工作流
實作工作流
測試工作流
部署工作流
支援工作流
建構管理
專案管理
環境
初始回合
回合1
回合2
回合n
回合n+1
回合n+2
回合m
回合m+1
回合

統合流程

- 統合流程強調反覆(Iterative)、遞增(Incremental)與演進(Evolutionary)。
  - 「反覆」表示系統分析、設計、實作、測試與整合是反覆不斷進行的；
  - 「遞增」表示系統的需求是逐步漸增，並非一開始就必須全部收集完整；
  - 「演進」表示系統在開發過程中是不斷的演進，而非僅在後期建置。
- 統合流程之特色
  - 使用者案例導向(Use Case Driven)：強調使用者的價值，以使用者的觀點思考使用者所想要的是什麼、所需要的又是什麼，開發者在專案開發過程中將會不斷地檢視其設計是否符合使用者案例模型。
  - 以架構為中心(Architecture-Centric)：強調儘早建立一個以元件為基礎的架構(Component-Based Architecture)。統合流程強調如何使用目前現有元件建立系統的架構。

統合流程

- 統合流程以兩個面向來描述
  - 在橫軸方面，代表時間面向，分為四個階段。
  - 在縱向方面，代表流程工作內容面向，於核心流程工作可以分為六個工作流。
- 此模型在時間面向分四個階段
  - 起始階段
    - 計畫申請、風險評估、可行性分析、初步計畫執行時間、資源概估及專案規劃。
  - 分析階段
    - 分析需求、了解問題領域、建立系統架構。
  - 建構階段
    - 系統設計、系統實作、單位測試。
  - 移交階段
    - 移機測試、移機安裝、文件製作。

統合流程

- 此模型在內容面向之核心流程工作分六個工作流
  - 企業模組工作流
    - 企業模組工作流的主要目的是透過此流程活動的進行，讓系統開發端能深入了解企業內部相關的專業領域知識及相關專業用語，並為開發端、需求端與使用端建立溝通管道。開發端會嘗試著了解組織的願景、企業電腦化的動機，並分析可能遇到的困難。
  - 需求工作流
    - 需求工作流的主要目的是讓系統開發端透過此流程活動的進行，以便於了解及確認該系統所需提供的功能。為獲得明確且切實的需求，分析師必須訪談客戶與未來該系統建置完成後的相關使用人員，然後整理並記錄於組織所收集到的系統需求文件，並進一步與客戶確認。

統合流程

- 分析與設計工作流
  - 分析與設計工作流的目的是系統開發人員將客戶端所認可的需求內容，轉換成真正可以部署執行之完整系統前的準備工作，以系統開發者的觀點，對所要開發的系統加以分析及設計，產生各個設計模組以實踐需求。所產生的設計模組將做為下階段實作的依據。
- 實作工作流
  - 實作工作流的目的是將系統設計模組轉換為可以執行的程式碼。通常包含對單元模組（例如，某個函數或類別的單元）進行測試。由於統合流程強調「反覆性」，系統實作活動在先前的回合就會進行，如此可及早發現可能的設計錯誤，某些不可行的技術方案也可以及早發現並適時調整。

統合流程

- 測試工作流
  - 測試工作流的主要目的在確認每個單元可以正確地整合，特別是要確認單元之間的介面是相容的，可以相互銜接溝通。測試也包含需求測試，檢驗實作出的產品是否符合需求規格所訂之內容。由於統合流程是採用反覆的方式，測試在先前的回合就會進行，亦具有可以及早發現缺失(Defects) 的優勢。
- 部署工作流
  - 此活動的目的是釋出(Release) 可以提供使用者安裝部署並且執行的版本。大部分的部署活動都會在移交階段進行，但先前回合會進行部分的部署準備，例如，準備軟硬體設備、系統的資源配置等。

統合流程

- 統合流程開發時程規劃範例

	1	2	3	4	5	6	7	8	9	10	11	12
回合一／企業模組	●											
回合一／需求擷取		●										
回合一／分析設計			●									
回合一／實作				2w								
回合一／測試				2w								
回合一／部署				2d								
回合二／企業模組					2w							
回合二／需求擷取					2w							
回合二／分析設計						●						
回合二／實作							●					
回合二／測試								3w				
回合二／部署								1w				
回合三／企業模組									2d			
回合三／需求擷取									.5w			
回合三／分析設計									.5w			
回合三／實作									3w	●		
回合三／測試											●	
回合三／部署											2w	2w

統合流程

- 統合流程開發方式的優點有：
  - 具瀑布式開發方式之優點，進而因強調反覆、遞增與演進的開發方式，能儘早建立一個以元件為基礎的架構。
  - 以使用者的觀點思考使用者所想要的是什麼、所需要的又是什麼，藉由反覆的系統展示與架構檢視，確定設計者與客戶對系統的共識，便能減少客戶與所開發系統之間的落差。

極限製程

- 極限製程(XP) 是Kent Beck 於1999 年提出， 目的是提倡更能「擁抱改變」(Embrace Changes)的敏捷開發方式(Agile Method)。
  - 極限製程的「極限」有著極端的含意，因為它提出許多極端的做法，例如，極端地倡導多回合開發方式、極端地要求顧客參與、極端地強調測試的重要性等。
- 極限製程之特色
  - 客戶駐點
    - 顧客代表也是開發團隊成員之一，在開發過程中必須全職參與開發團隊的討論。如此可以省去需求文件化的時間與閱讀需求文件可能產生的錯誤。隨時溝通、快速回饋是XP 的特性。
  - 漸進式的規劃
    - 顧客代表參與開發團隊一起訂定需求。需求不是條列式的功能列表，而是一個個像故事般的故事卡(Story Card)。按照故事卡的輕重緩急和風險，快速訂出專案的範圍。

極限製程

- 頻繁改版
  - 快速將簡單的系統上線，並在極短時間內更換新版本。
- 簡單設計
  - 任何時候，系統都應該盡可能地設計簡單。XP 強調設計並不是一次就可以到達完美，透過簡單的設計、測試與設計的改善，逐步修正設計，使系統可以切合使用者需求與系統品質。一開始過於複雜的設計會使設計所花費的時間過長，使用者及架構師無法立即對系統產生回饋。
- 測試先行
  - 先撰寫單元測試程式，確保每單元程式皆為正確。
    - XP 十分強調回饋，而良好、正確的回饋需要好的測試。為了達到有效的測試，XP 建議在撰寫程式以前先設計測試案例，並在程式撰寫結束後立即進行測試。當測試符合所定義的測試策略，例如，敘述包含度到達95% 時，才可以將程式碼check in 到程式庫中進行整合。
    - XP 強烈建議使用自動化的測試，例如，使用Unit Testing 的工具撰寫測試案例，如此即可在程式完成後，或每次修改後進行測試，以確定所修改的內容不會產生新的錯誤。

極限製程

- 極限製程開發時程規劃範例

	1	2	3	4	5	6	7	8	9	10	11	12
回合一／規劃	2w											
回合一／設計測試案例	2w											
回合一／設計與實作		2w										
回合一／執行測試與整合		2w										
回合二／規劃			2w									
回合二／設計測試案例			2w									
回合二／設計與實作				2w								
回合二／執行測試與整合				2w								
…												
回合六／規劃											2w	
回合六／設計測試案例											2w	
回合六／設計與實作												2w
回合六／執行測試與整合												2w

極限製程

- 極限製程開發方式的優點：
  - 較敏捷的開發方式，有快速的回饋機制來修正過程中產生的各項衝突或錯誤。
- 極限製程開發方式的限制：
  - XP 的各項原則都有互補作用，一旦某項原則沒有確實做好，就有可能帶來危機。
    - 例如，若測試程式撰寫不完整，系統便無法靠著通過測試來達到頻繁改版的穩定性。且XP所強調的是程式，而非文件，若程式重整得不夠，使得設計不夠簡單，便有可能會造成日後維護的困擾。
  - 實施上若沒有確實很容易流為「沒有流程」，造成流程的失效。

本章總結

- 流程看似簡單，卻是無數專案的結晶，如何定義合適的軟體流程來協助團隊有效率地開發系統，是軟體工程最重要的議題之一。
  - 軟體流程就像演算法一樣，合適的流程可以有效率地產生系統，不合適的流程則會延宕專案的進行。
- 在這章中，首先介紹軟體流程幾個基本的活動，多數的流程都是這些活動的組合。
  - 這些組合形成了幾種流程的通用模式，例如，瀑布式開發流程與統合流程等。
- 組織或開發團隊必須依照自己的特性去設計自己的流程，本章所提到的僅是大架構上的流程模式。
  - 目前台灣與全世界所極力推行的CMMI (Capability Maturity Model Integration)流程改善機制，即是透過定義合適的流程來改善軟體開發的效率與產品的品質。

作業練習 (Exercises)

1. 比較瀑布式開發流程、統合流程與XP三種開發方式的優缺點。
2. 哪種流程模式比較適合大型專案，為什麼？
3. 你的顧客想開發一個前所未有的軟體系統，他只知道大約的功能，你會採用哪種流程模式來進行這個專案？為什麼？
4. XP的原則中有哪些是你認為較難實施的？為什麼？