Software Engineering

U-02 軟體開發程序

2024.04_V1.4

©Templateswise.com

Agenda

- 瞭解軟體程序與軟體程序模型的概念
- 瞭解3種通用軟體程序模型與適用時機
- 概略瞭解軟體需求工程、軟體開發、測試及演進所涵蓋的活動
- 明瞭Rational Unified Process方法如何整合好的軟體程序經驗，建立出現代而通用的程序模型
- 瞭解支援軟體程序活動的CASE技術

2.1 軟體程序

- 軟體程序是生產軟體產品的一組活動。包含這些活動的軟體開發程序，可能是使用Java或C這類標準程式語言從頭開始，也可能是將現有的系統擴充或修改，或者是買現成的軟體或系統元件整合而成，不過現在後者會越來越多。

軟體程序具有幾項基本活動

- 軟體規格制定（software specification）：定義軟體的功能以及運作上的限制。
- 軟體設計與實作（software design and implementation）：產生符合規格的軟體。
- 軟體確認（software validation）：軟體必須經過確認是否符合客戶的需求。
- 軟體演進（software evolution）：軟體必須持續維護及改進，以符合客戶一直在變動的需求。

資訊系統開發模式

- 常用之資訊系統開發模式有八種：

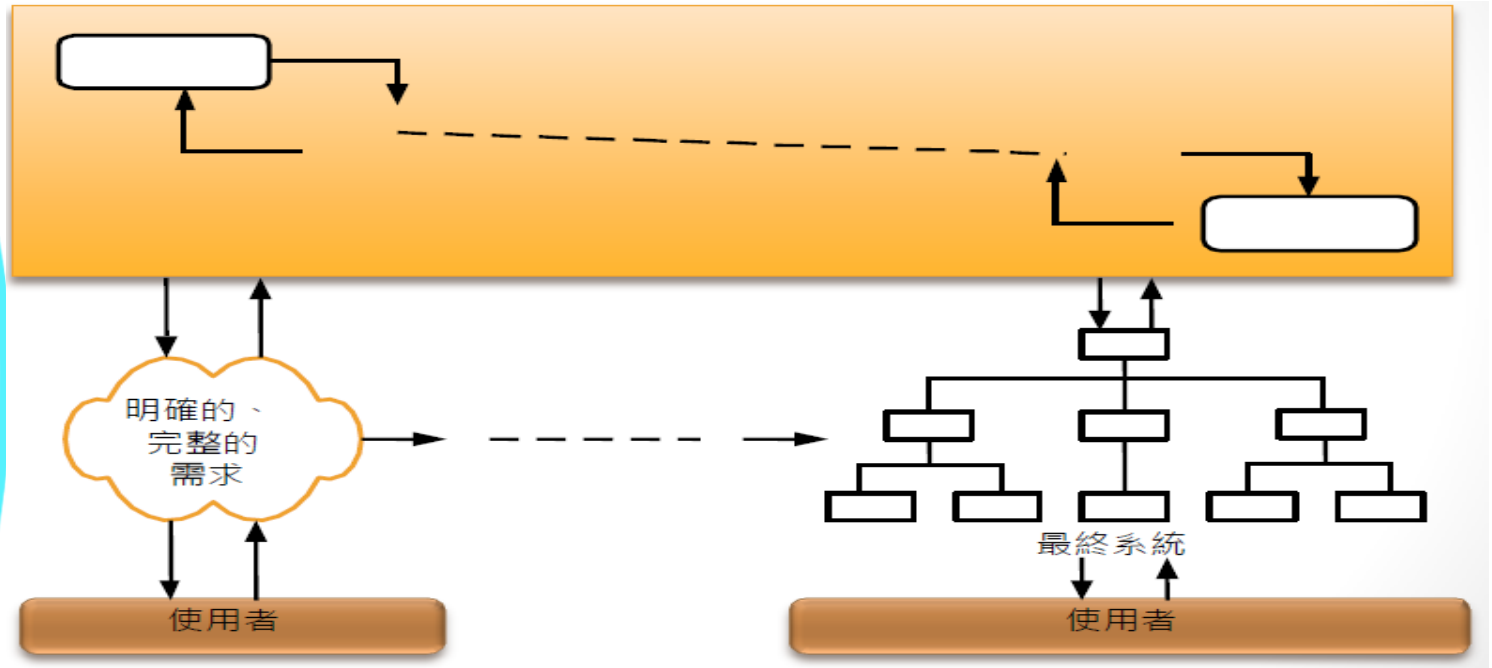
1. 瀑布模式 (Waterfall Model)
2. 雛型模式 (Prototyping Model)
3. 漸增模式 (Incremental Model)
4. 螺旋模式 (Spiral Model)
5. 同步模式 (Concurrent Model)
6. RUP模式 (Rational Unified Process Model)
7. 敏捷軟體開發 (Agile Software Development)
8. MDA (Model Driven Architecture) 軟體發展生命週期

2.2 瀑布式模型

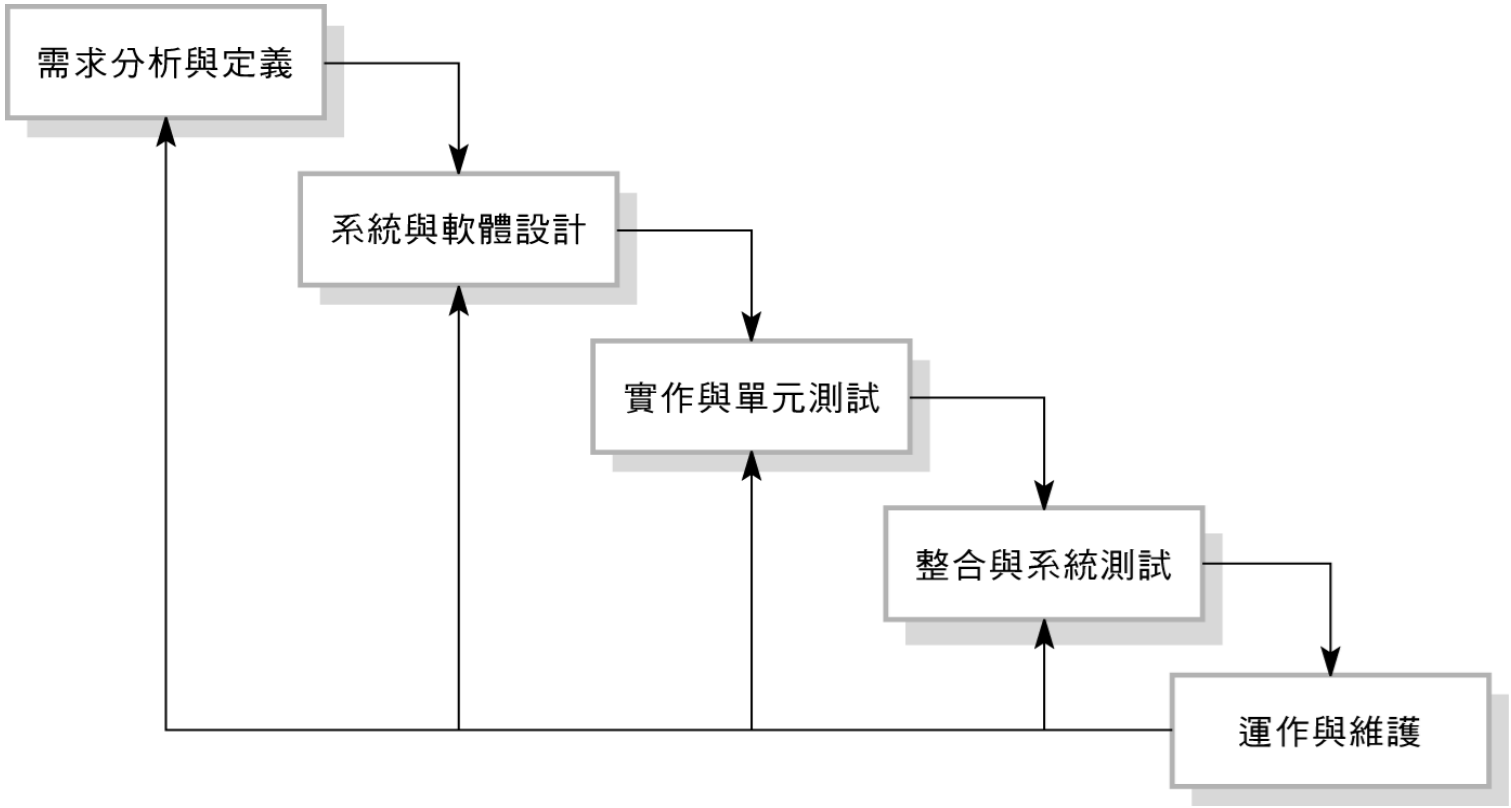
- 因為從某個階段進入下一個階段是以類似瀑布的方式進行，所以這個模型被稱為「瀑布式模型」（waterfall model）或軟體生命週期（software life cycle）。這個模型中的主要階段剛好對應到軟體開發的基本活動：
  - 需求分析與定義：與系統使用者進行諮商後，定義出系統的服務、限制和目標，之後再詳細定義成系統的規格。
  - 系統與軟體設計：系統的設計程序會將需求分割成硬體和軟體系統，以建立整體的系統架構。
  - 實作與單元測試：在此階段將軟體設計實作成一組程式或程式單元，單元測試則是驗證每個單元是否符合制定的規格。
  - 整合與系統測試：將獨立的程式單元或程式整合，並且視為完整系統進行測試，以確保系統符合軟體的需求。
  - 運作與維護：這個階段通常是最久的，安裝好該系統並開始實際運作。維護則包括修正之前沒有發現的錯誤、改善系統單元的實作，以及針對新需求改進系統的服務。

瀑布式模型

分　析	設　計	實　施
1. 可行性分析 2. 需求分析 3. 系統分析	4. 概念性設計 5. 細部設計	6. 程式編輯與單元測試 7. 整合測試 8. 安裝與系統測試 9. 教育訓練 10. 操作與維護

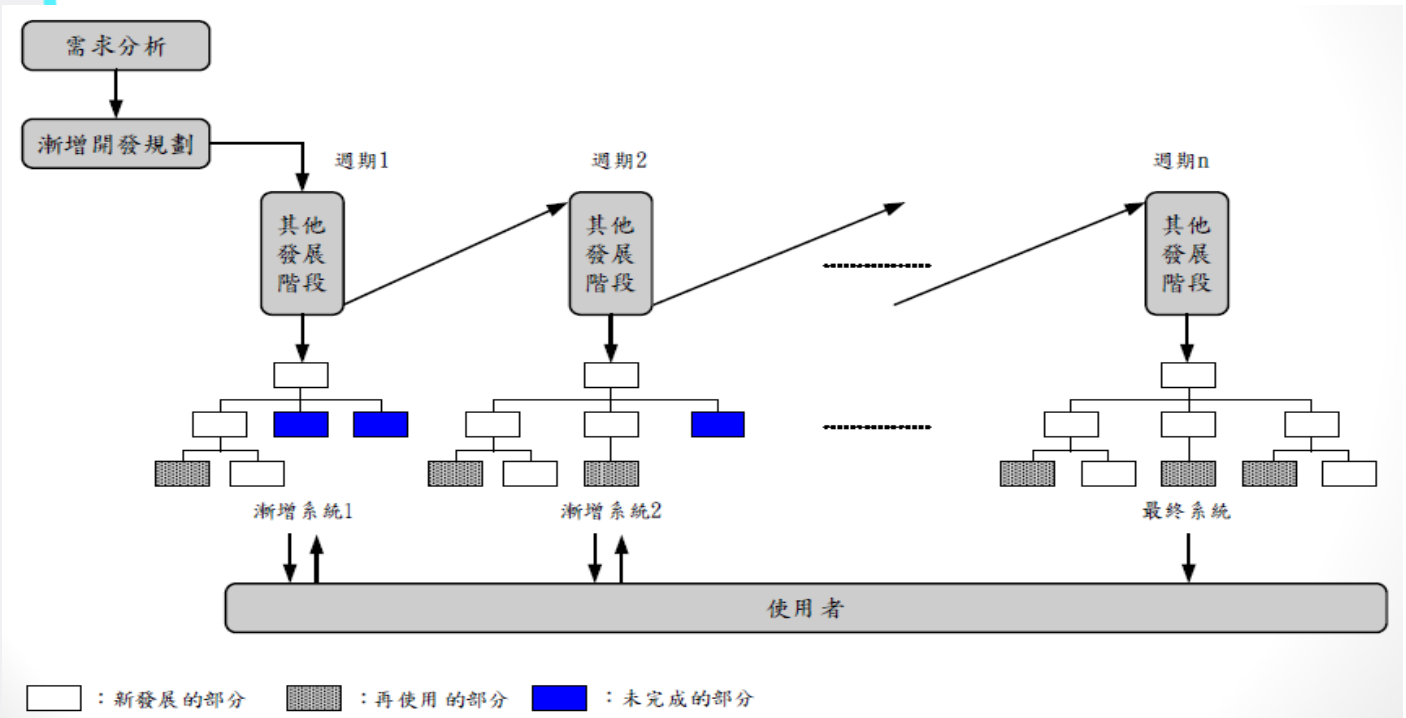


軟體生命週期



2.3 漸增模式 (2/4)

圖2-5 漸增模式之系統開發程序



2.3 漸增模式 (3/4)

- 漸增模式與瀑布模式大部分相同，但是仍有一些地方是不同的，例如：
  - 系統被分成幾個子系統或功能，各子系統可獨立依序開發；而瀑布模式則是各個子系統需同時開發。
  - 系統開發可由多個週期完成，每個週期表示不同版本之系統，因為每個週期均有程式編輯及上線實施，使用者均有參與，故漸增模式之風險較低。

2.3 漸增模式 (4/4)

- 漸增模式適用於下列情況：

  1.組織的目標與需求可完全且清楚地描述。

  2.預算須分期編列。

  3.組織需要時間來熟悉與接受新科技。

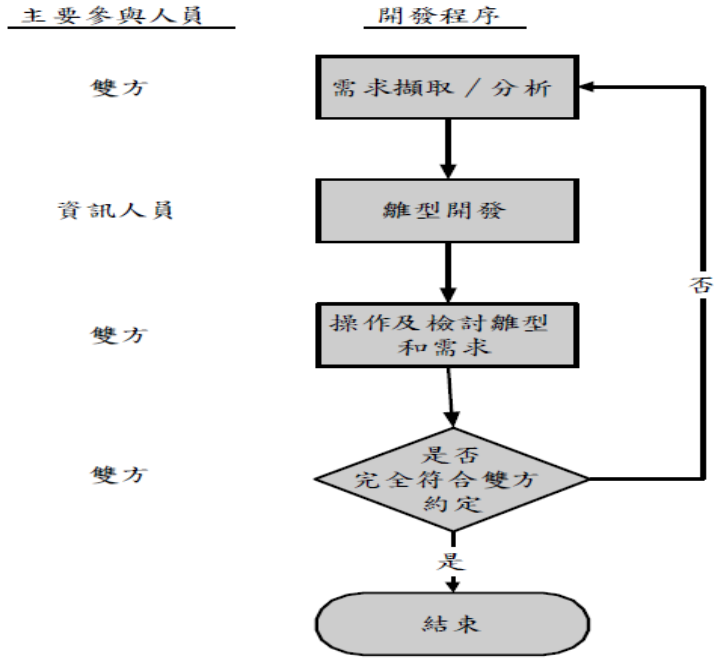
2.4 雛型模式 (1/4)

- **雛型模式**是一種系統開發方法，該方法先針對使用者需求較清楚的部分或資訊人員較能掌握之部分，依分析、設計與實施等步驟快速開發雛型。

- 開發過程中，強調盡早以雛型作為使用者與資訊人員需求溝通與學習之工具，雙方透過雛型之操作與回饋，釐清、修改及擴充需求，並藉以修改與擴充雛型。上述步驟反覆進行，直到系統符合雙方約定為止。

2.4 雛型模式 (2/4)

圖2-6 雛型模式之系統開發程序及參與人員



2.4 雛型模式 (3/4)

- 雛型模式之主要特性與原則如下：
  - 強調雛型之快速開發及使用者高度參與。
  - 強調以雛型作為使用者及系統開發者之需求溝通與學習機制。
  - 從需求最清楚的部分著手開發雛型，並透過使用者對雛型之操作與回饋，反覆修改與擴充，每次反覆時間間隔（週期）要盡可能縮短。

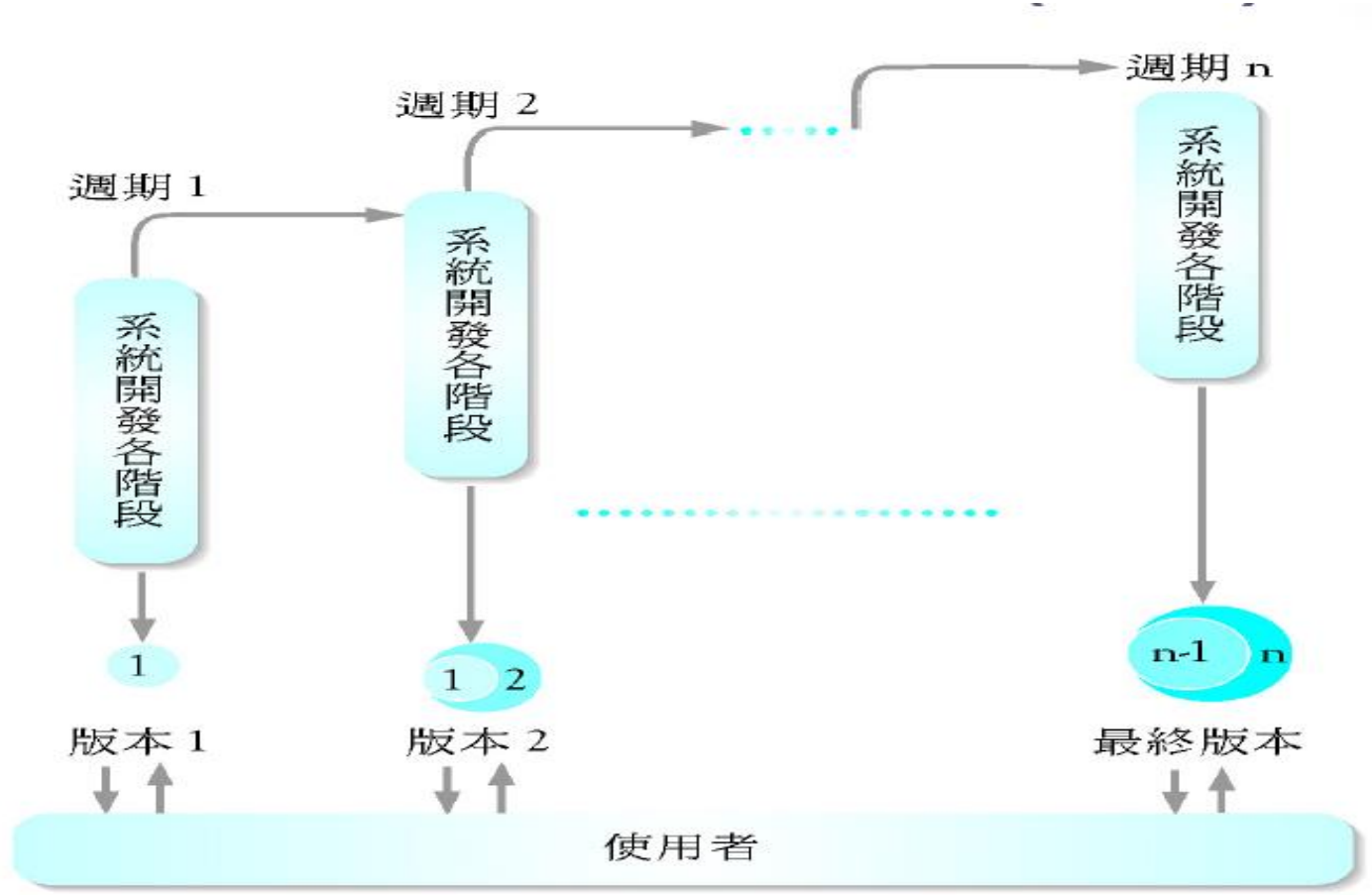
2.4 雛型模式 (4/4)

- 雛型模式的潛在問題：
  - 因強調以「雛型演進」代替完整之分析與設計，故系統文件較不完備，程式亦可能較難維護。短期而言，可能較能滿足使用者需求；但對長期而言，系統較易失敗。
  - 因缺乏整體之規劃、分析與設計，故較不適用於大 型及多人參與之系統開發專案。
- 雛型模式有兩種常見之應用策略：
  - 演進式雛型策略
  - 用後丟棄式雛型策略

2.4 1 演進式雛型策略 (1/2)

- 演進式雛型策略主要係將所有需求看成一個整體，從需求最清楚的部分先快速經歷一系統開發週期，以完成初版雛型系統之開發。
- 再利用該雛型與使用者溝通，以確定、修改和擴充需求，並藉以作為下一週期雛型演進之依據。
- 該週期不斷地反覆進行，直到雛型系統符合雙方約定為止。

2.4 1 演進式雛型策略 (2/2)



2.4 2 用後丟棄式雛型策略 (1/2)

- **用後丟棄式雛型策略**一般是以一種快而粗糙的方式建立雛型，以促使使用者能夠盡快藉由與雛型之互動來決定需求項目，或允許資訊人員藉以研發問題之解決方法與資訊科技之應用等。

- 這種雛型因為用後即丟，所以不需要考慮雛型系統之運用效率與可維護性，也不需要考慮容錯的能力。

2.4 2 用後丟棄式雛型策略 (2/2)

- 雛型需丟棄之原因很多，例如可能因所用之開發工具非最終所決定之工具，也可能因經由R&D 研發出新方法，但新研發出之方法與原來之方法可能完全不同或不相容，或發現修改原雛型所需之成本比重新開發來得高。

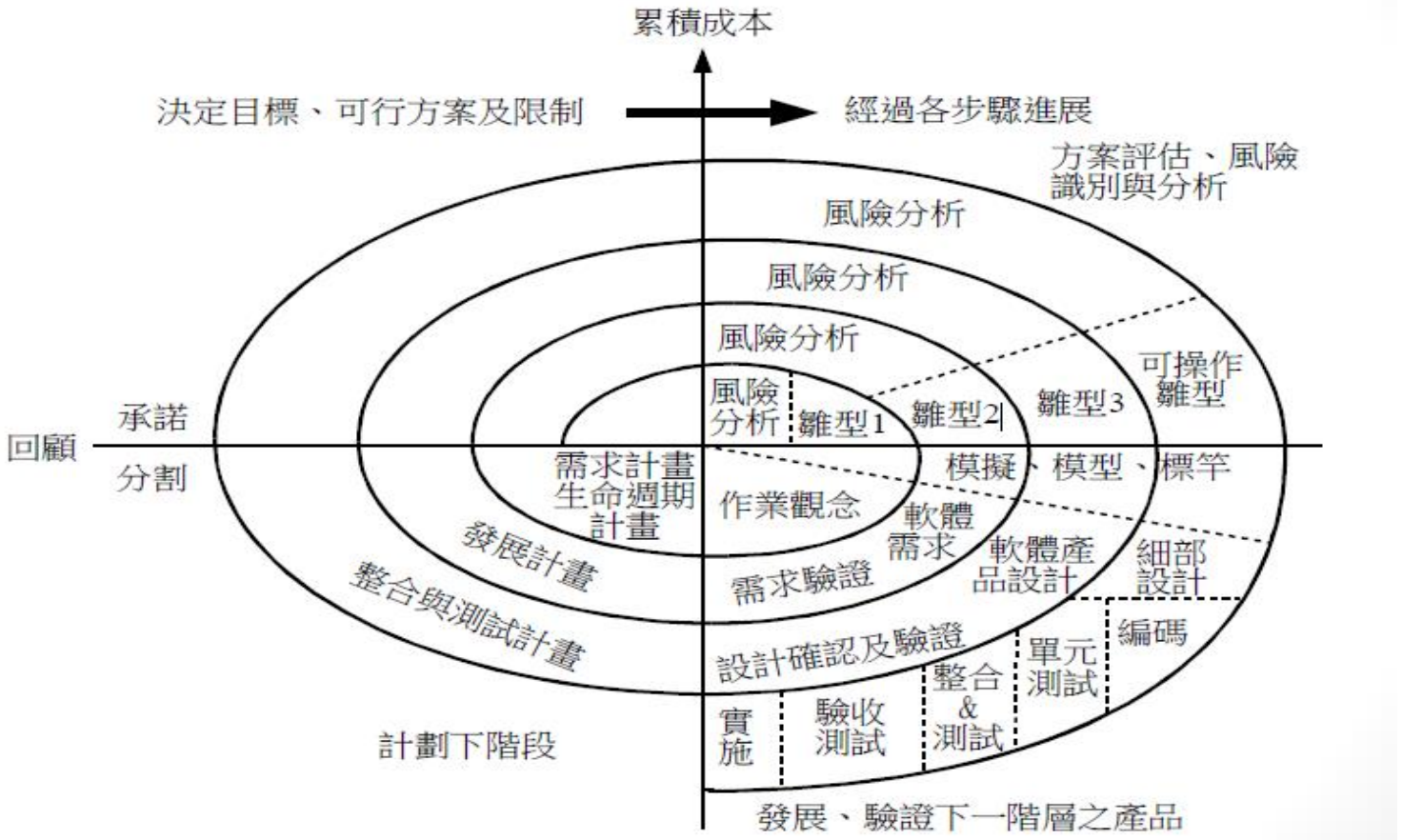
- 用後丟棄雛型策略僅實施在風險程度最高的地方，例如在使用者需求或解決問題之知識、概念與資訊科技整合最不清楚的情況，而其他情況則盡可能地採用演進式雛型策略，因為雛型之丟棄也意味著成本的浪費。

2.5 螺旋模式 (1/7)

- 螺旋模式之執行由三個步驟形成一週期：
  - 找出系統的目標、可行之實施方案與限制。
  - 依目標與限制評估方案。
  - 由剩下之相關風險決定下一步驟該如何進行。
- 此週期反覆進行，直到系統開發完成為止。

2.5 螺旋模式 (2/7)

螺旋模式之開發程序



2.5 螺旋模式 (3/7)

- **步驟一：找出系統的目標、可行之實施方案與限制**

  1.找出系統的目標

  • 系統目標之評核因素很多，例如系統的績效、功能與容忍改變之能力等。

  2.找出系統之實施方案

  • 系統實施方案會因問題而異，例如找出之實施方案有設計方案A、設計方案B、重用、購買等。

  3.實施方案之限制

  • 實施方案之限制可能為專案之成本、時程、系統介面等。

2.5 螺旋模式 (4/7)

- **步驟二：依目標與限制評估方案**

  主要是找出各方案之不確定處，並設法解決其步驟如下：

  1.找出不確定的部分，也就是專案風險之重要來源。

  2.解決風險來源：

  - 可用雛型、模擬、**標竿(Benchmarking)、參考點檢查(Reference Checking)**、問卷、分析模式、上述方式之綜合或其他技術以解決風險。

2.5 螺旋模式 (5/7)

- **步驟三：由剩下之相關風險決定下一步驟**
  - 若績效或使用者介面風險將強力影響程式開發或內部介面控制，則下一步驟可能是採取演進式雛型策略。
  - 若該雛型使用性佳且夠強韌 (Robust)，足以當作未來系統發展之基礎，則往後的步驟將是一系列的雛型演進。
  - 假如先前之雛型已解決所有的績效或使用者介面之風險，且程式開發及介面控制之風險獲得掌控，則下一步將遵循基本的瀑布模式，亦可適當的修飾以整合漸增模式。

2.5 螺旋模式 (6/7)

- 螺旋模式之特色與應用原則：
  - 在高風險部分之設計尚未穩定前，規格之發展不需要一致、詳盡或正式，以避免不必要之設計修改。
  - 在開發之任一階段，螺旋模式可選擇整合雛型模式以降低風險。
  - 當找到更吸引人之方案或需解決新風險時，螺旋模式整合重做或回到前面之階段。

2.5 螺旋模式 (7/7)

- 螺旋模式包容了現有軟體流程模式之大部分優點，且其風險導向之方法解決了許多系統開發模式所存在之問題。
- 在某些條件下，螺旋模式相當於某一現有之流程模式。例如：

若專案在使用者介面或綜合績效需求方面屬於低風險，且在預算及時程控制方面屬於高風險，則這些風險之考量會使螺旋模式之執行相當於瀑布模式或漸增模式。

- 若專案在預算及時程控制、大型系統之整合或需求變動方面之風險較低，且在使用者介面或使用者決策支援需求方面之風險較高，則這些風險之考量會使螺旋模式之執行類似於雛型模式。

2.6 同步模式 (1/6)

- **同步模式**源自於製造業的同步工程，其目的在於縮短開發時間、加速版本之更新。

- 同步模式是基於三個主要的構想來達到縮短時程的目標：

  1. 多個團隊同時開發。這種多組人同時工作的方式稱為**活動同步 (Activity Concurrency)**。

2.6 同步模式 (2/6)

2. 資訊同步。不同團隊的資訊互相交流與共享，稱為**資訊同步 (Information Concurrency)**。

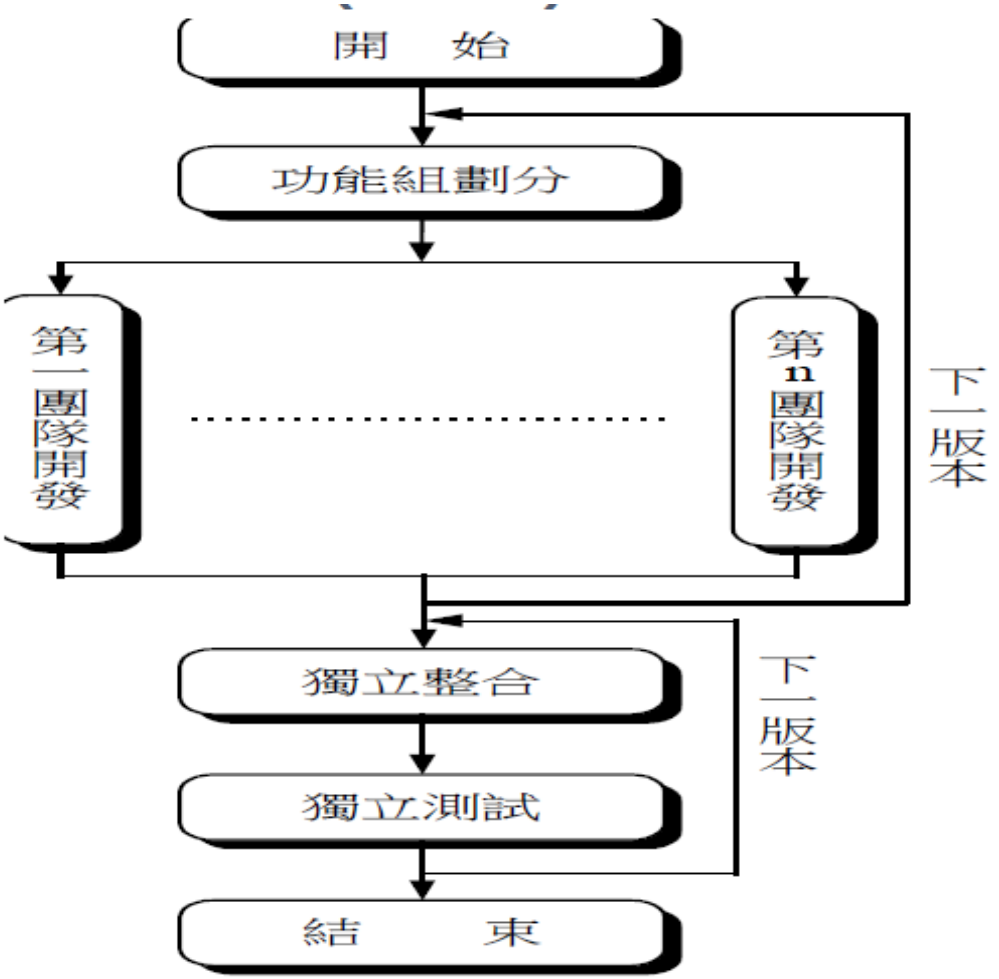
   資訊同步有三個技巧：

- 向前傳遞 (Front Loading)
- 向後傳遞 (Flying)
- 建立一個有效的資訊交換網路及支援群體工作的環境

3. 整合性的管理系統。同步模式的管理比一般的開 發模式複雜，必須開發一個管理系統以協調人員、資源、過程及產品間複雜的互動關係。

2.6 同步模式 (3/6)

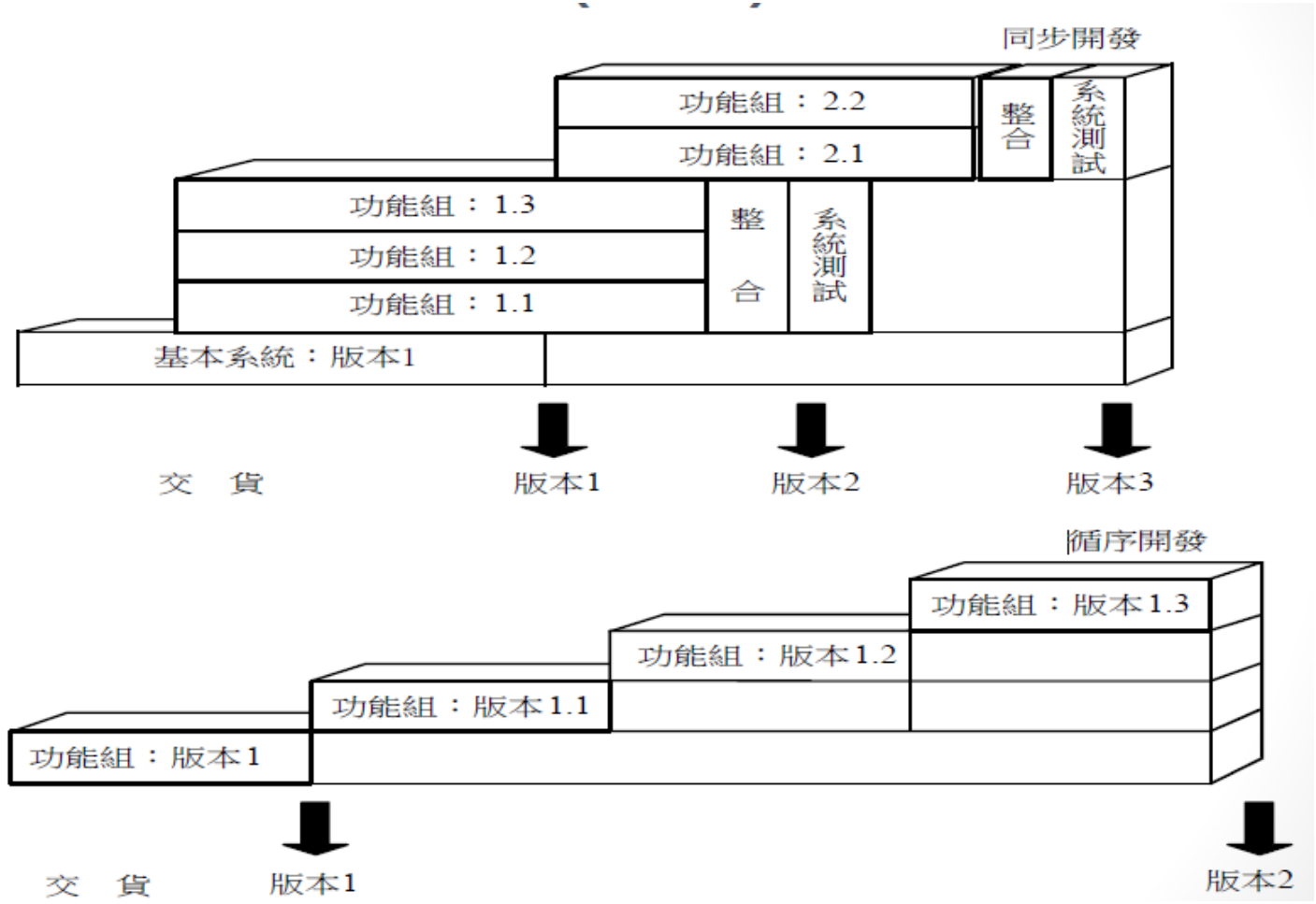
同步模式之開發程序



2.6 同步模式 (4/6)

- 同步模式的發展主要是為了因應商業套裝軟體的市場競爭。
- 其優點是開發時間的縮短可提高產品的競爭力。
- 其缺點則是緊湊的步驟及資訊溝通的頻繁，使得專案管理的複雜度大幅提高，人力成本也相對提高，若沒有輔以良好的工具及管理方法，則不易達成目標。

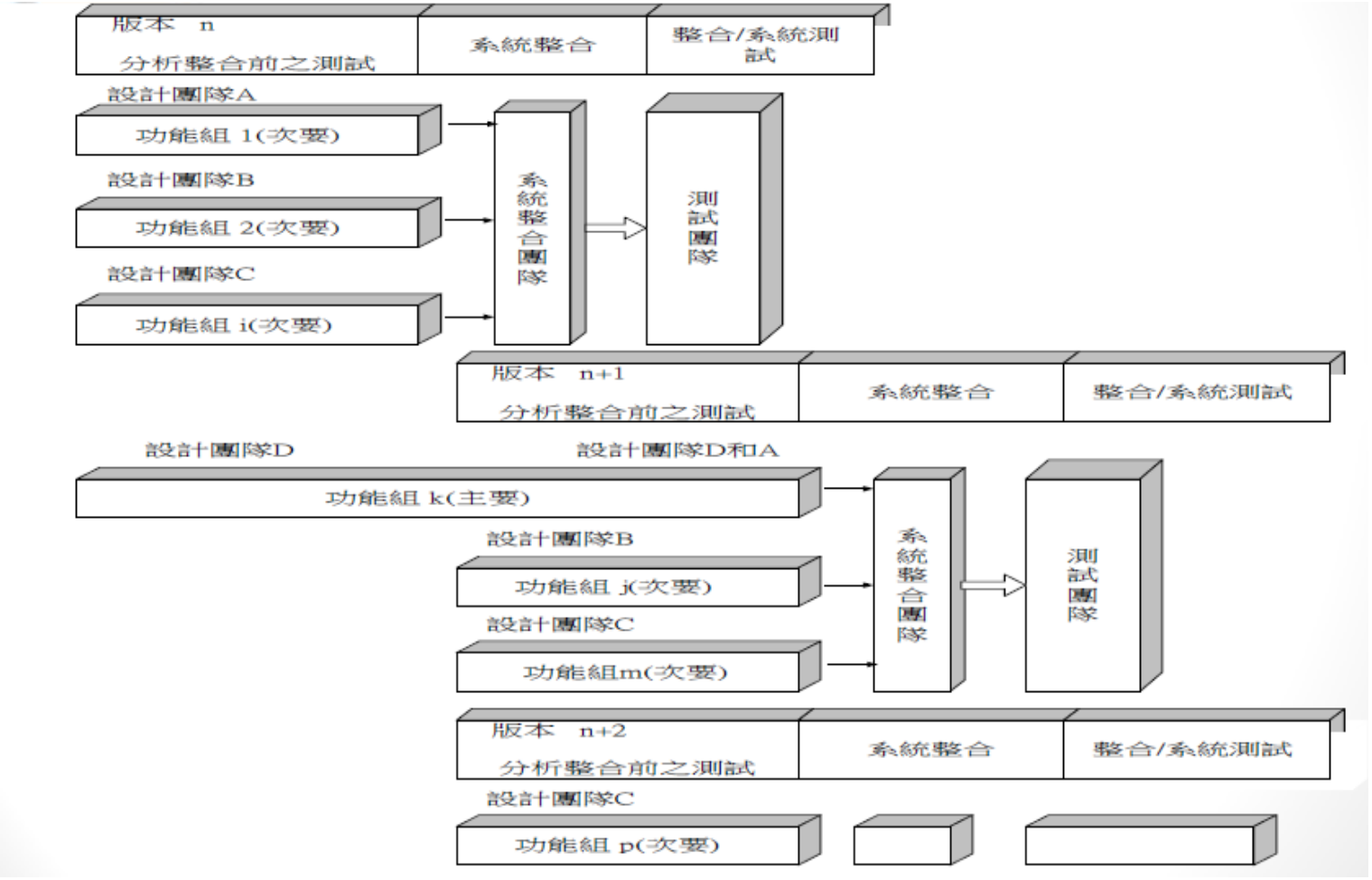
2.6 同步模式 (5/6)



同步開發與循序開發方法比較

2.6 同步模式 (6/6)

同步開發模式



2.7 Rational統一流程模式

- **RUP 模式**於1998年由Jacobson 等人提出。該模式結合螺旋模式的概念，以反覆與漸增的軟體開發原理進行軟體發展，且每一次的反覆後需產出一個可運作的系統版本，並在每一個反覆週期中評估風險，以盡早發現問題。

- RUP 模式可由動態與靜態兩個構面來說明系統開發專案之實施階段與核心工作。

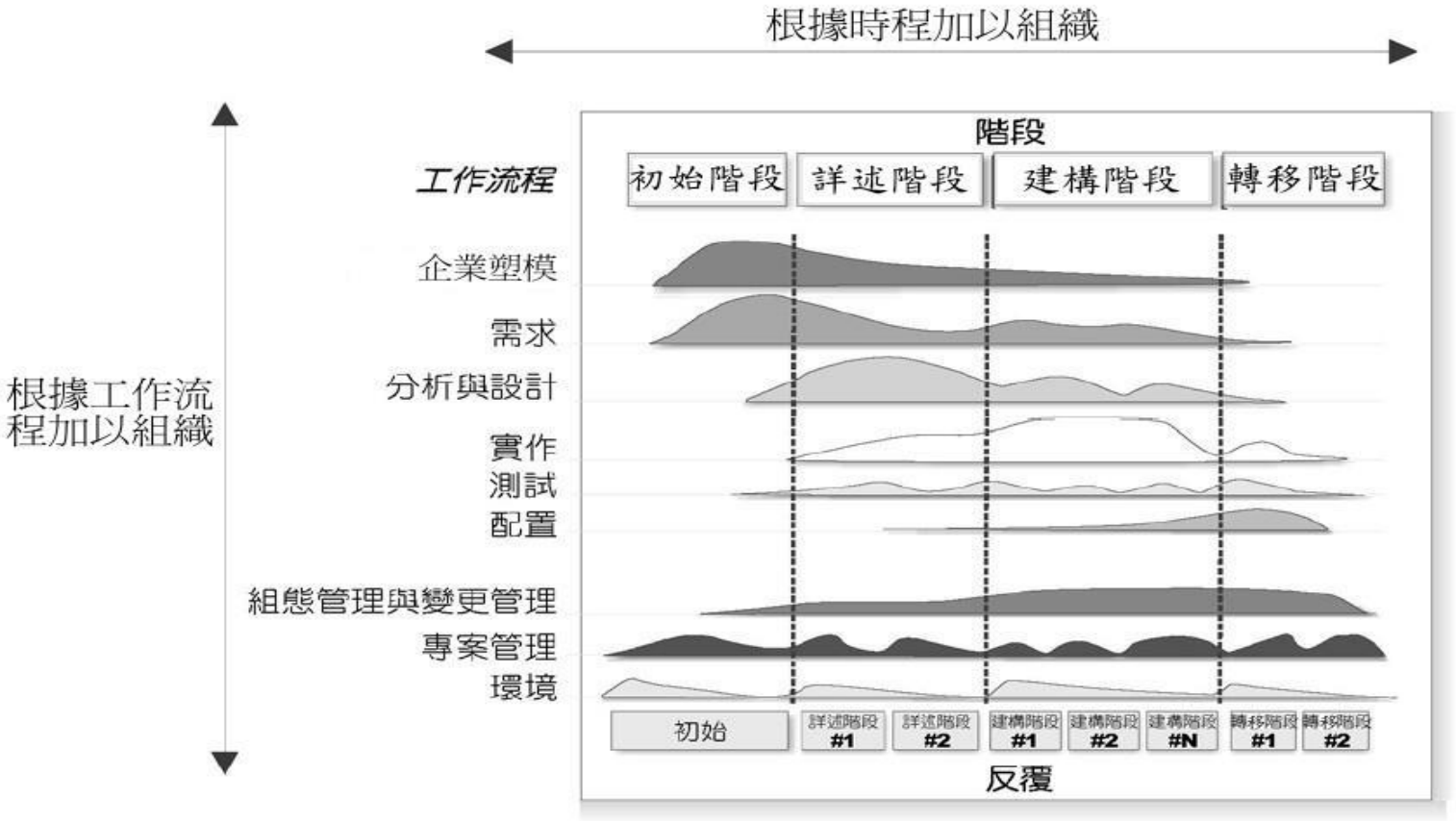
2.7 RUP模式 (1/2)

- RUP 模式的動態面把軟體開發依序分成四個主要階段：初始、詳述、建構與轉移。這四個階段構成一個週期，週期可反覆進行，每個週期內之各階段也可視情況反覆進行。

- RUP 模式的靜態面結構主要處理依邏輯順序將軟體開發與管理支援工作表達成九個核心工作流程：企業塑模、需求、分析與設計、實作、測試、配置、組態管理與變更管理、專案管理、 環境等，其中前六項是軟體工程工作，而後三項是管理支援工作。

2.7 RUP模式 (2/2)

RUP 模式之二維構面

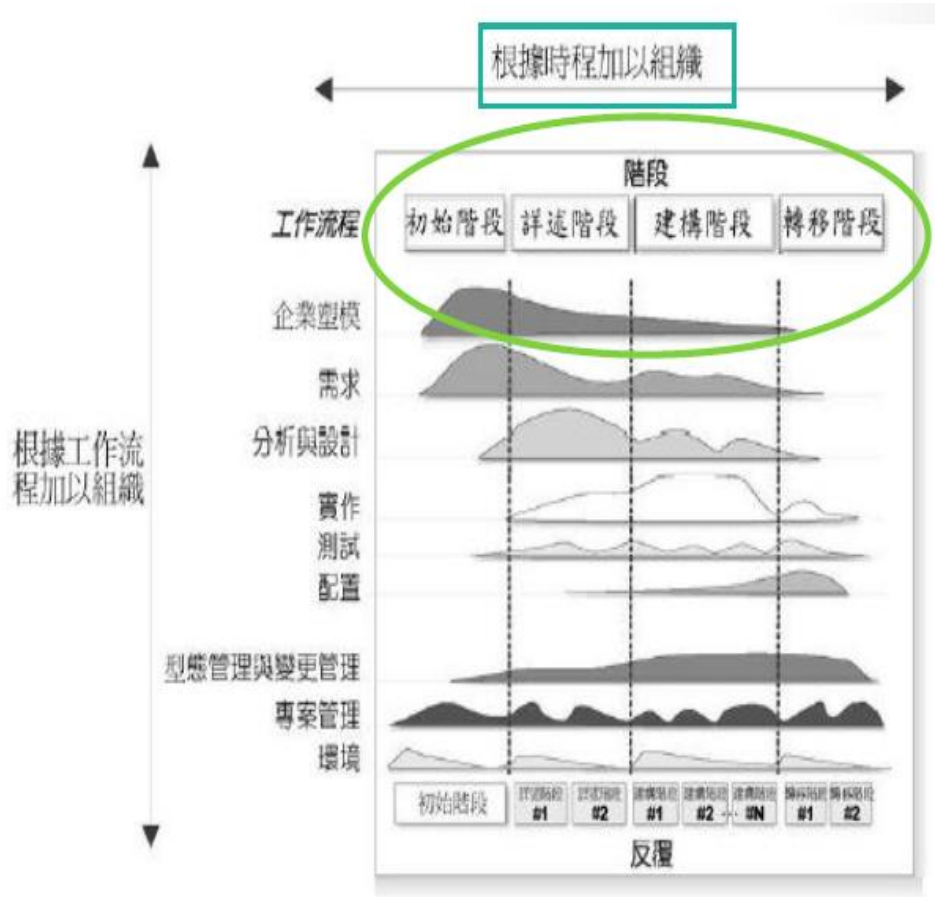


2.7 RUP模式的構面

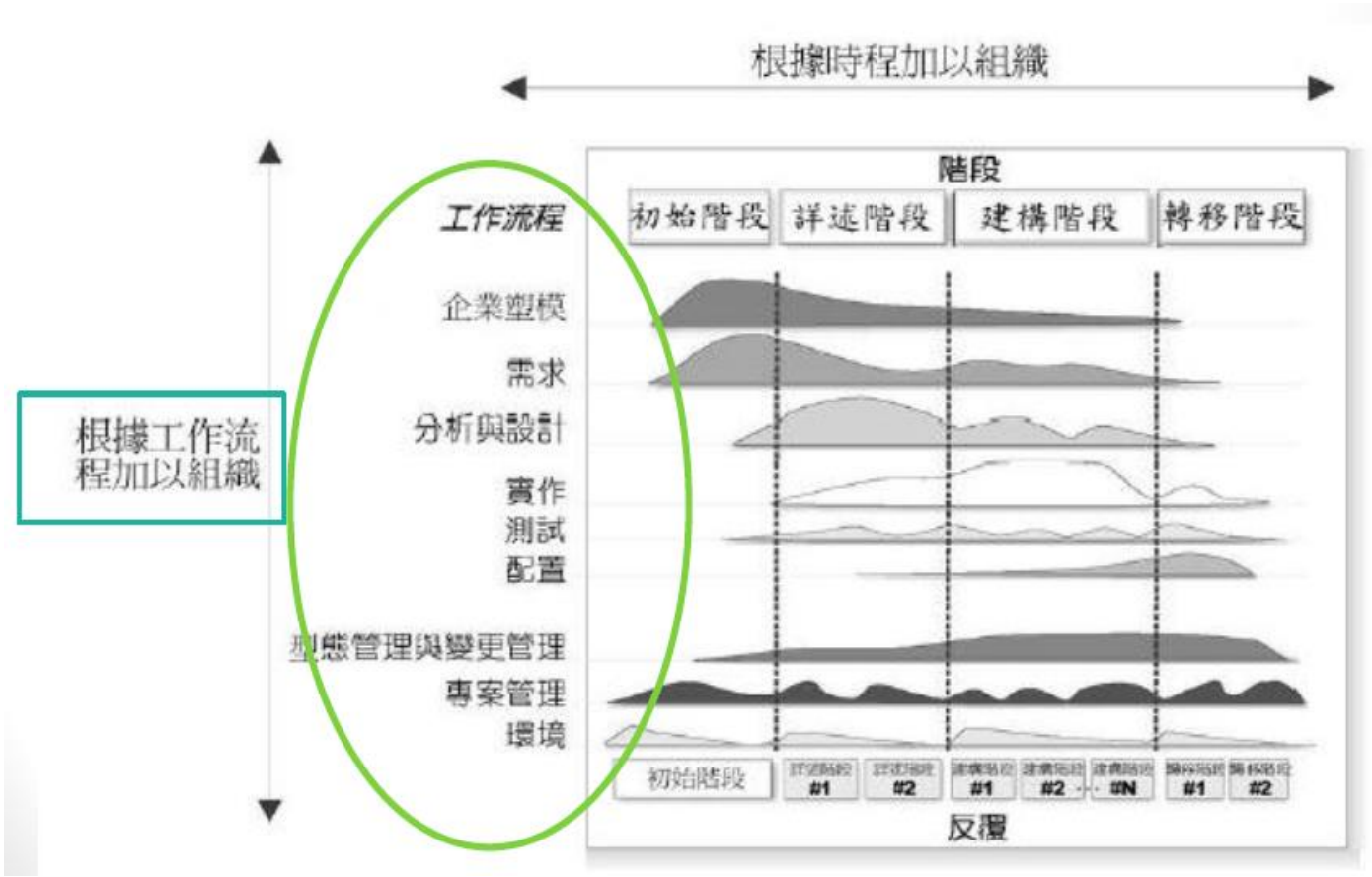
- 動態面（水平軸）主要依專案週期之時程表達系統開發之週期、階段、反覆與里程碑。
- 靜態面（垂直軸）主要依核心工作流程之邏輯順序表達系統開發之活動、**領域範圍 (Discipline)**、工作產出與角色。
- 水平軸與垂直軸交叉格上的圖形面積代表其所對應工作之估計工作量或比率。

2.7 RUP模式--動態面

- 初始階段：建立企業個案 (business case)。
- 詳述階段：處理主要的技術工作 。
- 建構階段：建構一個初步可運作的系統版本。
- 轉移階段：將系統產品移交客戶使用。



2.7 RUP模式-靜態面


根據時程加以組織
階段
初始階段
詳述階段
建構階段
轉移階段
工作流程
企業塑模
需求
分析與設計
實作
測試
配置
型態管理與變更管理
專案管理
環境
根據工作流程加以組織
初始階段
#1
#2
#1
#2
#N
#1
#2
反覆

2.8 敏捷軟體開發 (1/2)

- 一群不同軟體開發方法的領域代表於2001年共同推出**敏捷宣言 (Agile Manifesto)**。
- 其主要目的為提出一套較傳統軟體開發方式更為簡捷且快速的軟體開發概念，此即**敏捷軟體開發 (Agile Software Development)**。

2.8 敏捷軟體開發 (2/2)

- 敏捷軟體開發的主要開發理念和價值觀如下：
  1. 個體與互動勝於流程與工具。
  2. 可運作的軟體勝於全面性的文件。
  3. 與客戶的協同合作勝於契約談判。
  4. 因應變化勝於遵循計畫。

- 目前有多種軟體開發方法，包括動態系統開發方法、Scrum 、精實軟體開發和極限編程等，皆以上述敏捷軟體開發概念為基礎。

動態系統開發方法 (1/2)

- 開發過程主要以反覆與漸增的開發方式進行，並強調使用者在開發過程中的參與。

- 在開發的過程中會隨著需求改變而反覆調整，其目的在於準時且於預算內將軟體開發完成，主要應用於時程緊湊且預算有限之專案。

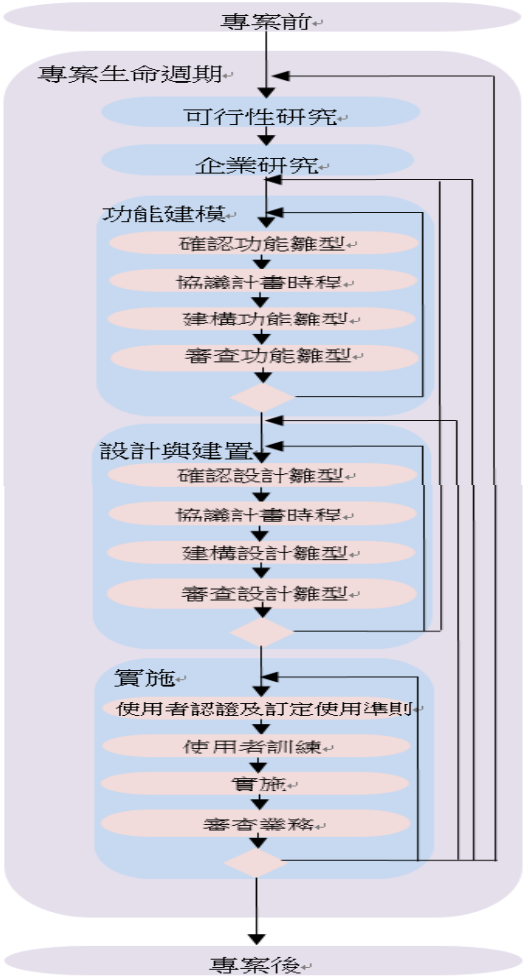
動態系統開發方法 (2/2)

- 實施的過程分為**專案前 (Pre-Project)**、**專案生命週期 (Project Life-Cycle)** 和**專案後 (Post-Project)** 三個階段。

- 專案生命週期之主要工作可分為五個階段，包括：

- **可行性研究 (Feasibility Study)**
- **企業研究 (Business Study)**
- **反覆功能建模 (Functional Model Iteration, FMI)**
- **反覆設計與建置 (Design and Build Iteration, DBI)**
- **實施 (Implementation)**

動態系統開發方法


專案前
專案生命週期
可行性研究
企業研究
功能建模
確認功能雛型
協議計畫時程
建構功能雛型
審查功能雛型
設計與建置
確認設計雛型
協議計畫時程
建構設計雛型
審查設計雛型
實施
使用者認證及訂定使用準則
使用者訓練
實施
審查業務
專案後

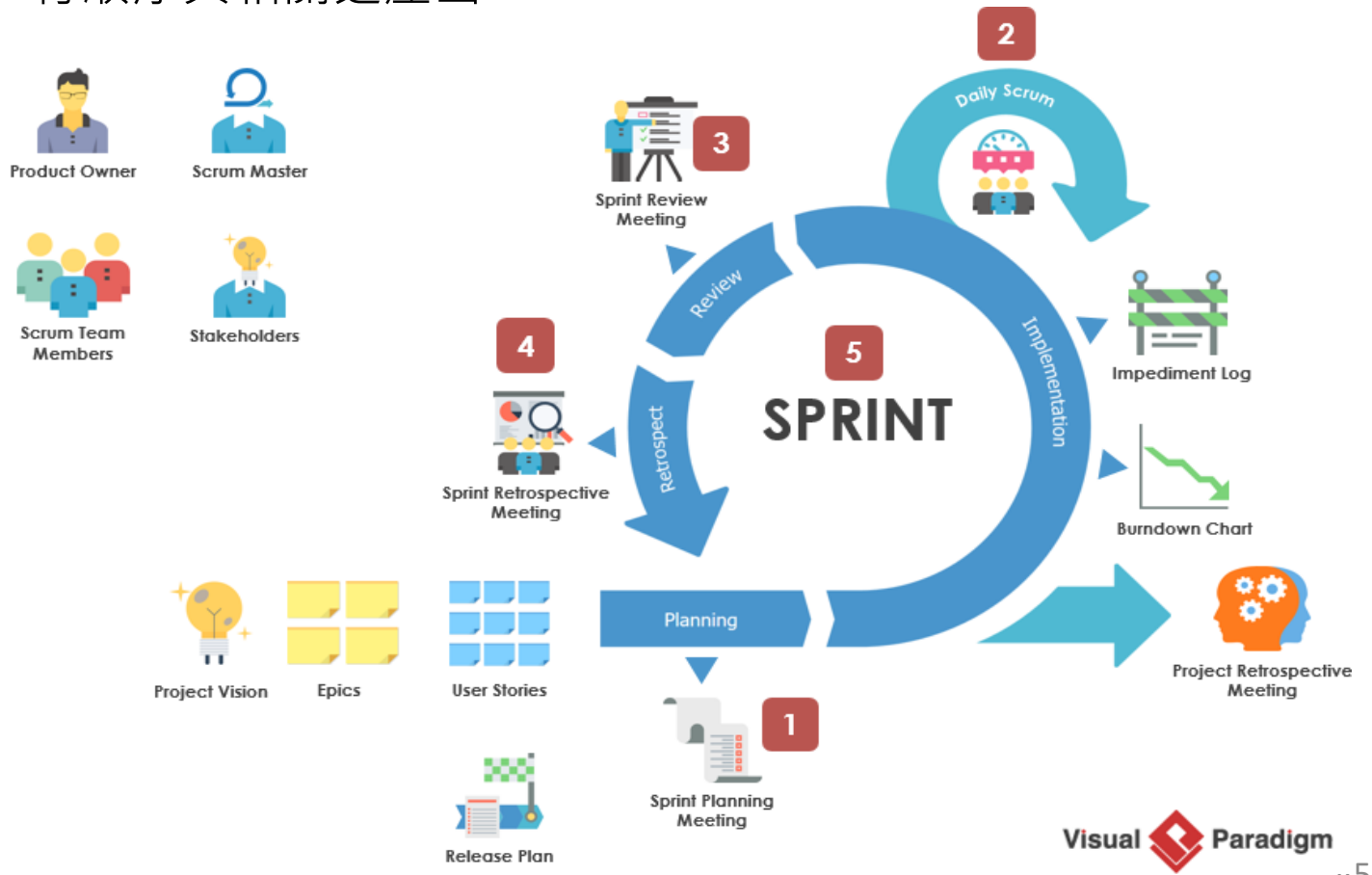
Scrum軟體開發(1/3)

- Scrum軟體開發是敏捷 (Agile)開發方法論的實施方式之一，Scrum強調反覆地短期發展、檢討與調整、以自我組織與管理的目標來建立開發團隊，主張以若干個固定時間長度的衝刺 (Sprint)進行開發工作，並在每一個衝刺結束時需展示個別完成的功能。

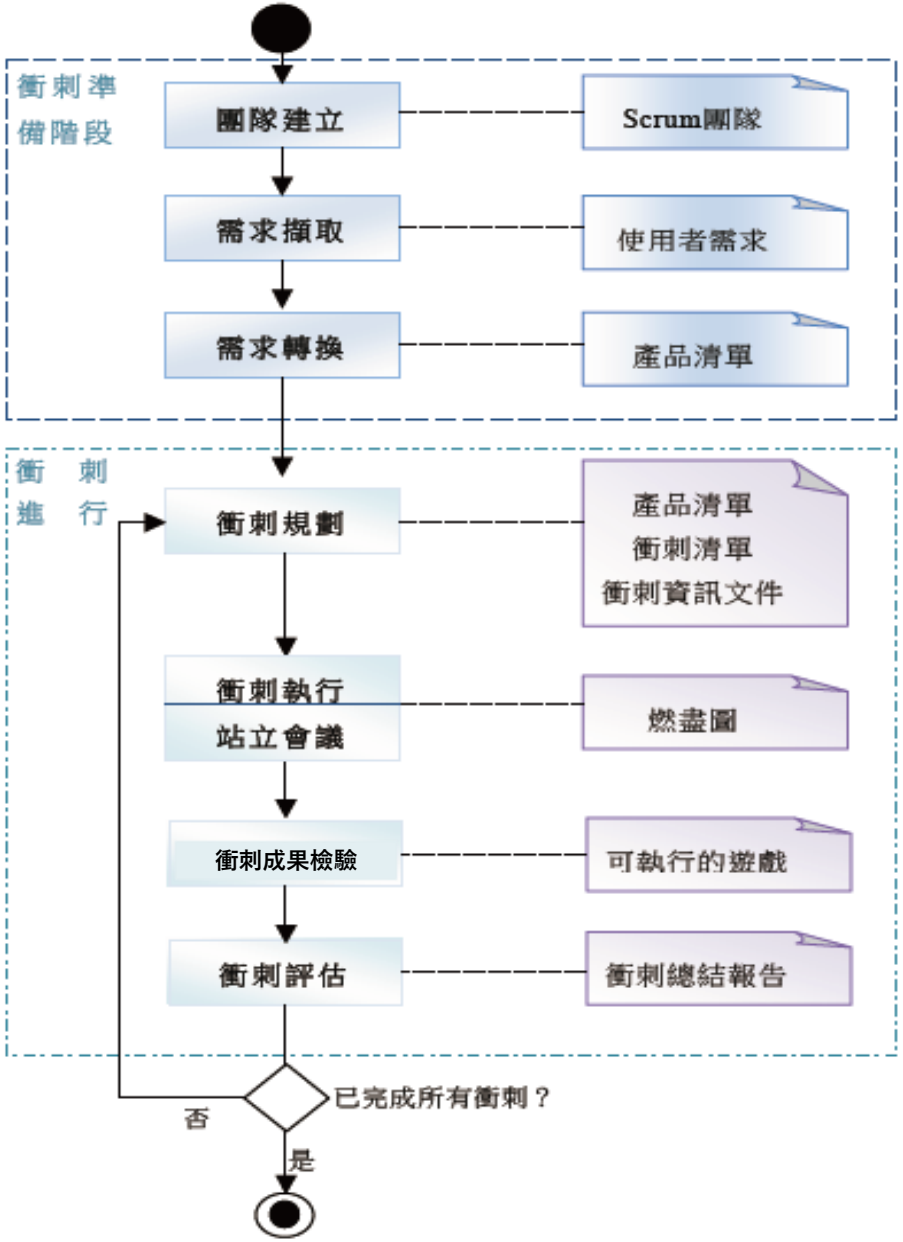
- 應用Scrum進行系統開發需將一項任務（例如一個故事）分成很多個子工作，且每一個子工作須能在一個衝刺時間（例如1~4週）內被完成，並透過反覆進行衝刺與檢討的方式，至該任務完成

Scrum軟體開發(2/3)

- Scrum的實施流程可分為兩階段：
  1. 衝刺準備階段包含Scrum團隊建立、需求擷取與需求轉換。
  2. 衝刺進行階段包含衝刺規劃、執行、成果檢視與評估等，其執行順序與相關之產出



Scrum軟體開發 (3/3)



精實軟體開發

- 為將豐田生產系統 (Toyota Productive System, TPS) 提出之**精實生產 (Lean Manufacturing)** 原則與方法，應用於軟體開發領域。
- 包括七項原則概念：
  1. 消除浪費 (Eliminate Waste)
  2. 增進學習 (Amplify Learning)
  3. 延遲決定 (Delay Commitment)
  4. 快速遞送 (Deliver Fast)
  5. 團隊授權 (Empower the Team)
  6. 建置完整 (Build Integrity In)
  7. 全盤檢視 (See the Whole)

極限編程

- 為Kent Beck 於1996年提出之軟體開發方法。與其他敏捷軟體開發方法相似，極限編程亦著重於開發流程是否對使用者或企業產生價值，強調以有效率且富彈性（反覆與漸增）之方式，開發高品質之軟體系統。

- 提出四項軟體開發基本行為：
  1. **編碼 (Coding)**
  2. **測試 (Testing)**
  3. **傾聽 (Listening)**
  4. **設計 (Design)**

2.9 MDA發展生命週期(1/6)

- **模式驅動結構 (Model Driven Architecture, MDA) 是由OMG (Object Management Group)** 定義的一種軟體開發架構，其關鍵是軟體開發過程中每個階段（或步驟）的產出均須建構出模式，且該模式之產出為下一個階段的輸入。

- MDA 的發展生命週期其實與其他系統開發模式（例如瀑布模式或RUP 模式）的系統發展生命週期並沒有差別，主要的差別是在發展過程中步驟之產出，強調該產出是由電腦可理解的正規模式 (Formal Model) 表達。

2.9 MDA發展生命週期(2/6)

- MDA 有三個核心模式：PIM、PSM 與Code。
- **平台獨立模式 (PIM)**

- PIM 是一種高階抽象的模式，該模式與開發技術獨立。PIM 是分析與設計結果的重要產出，主要根據需求塑模的結果，從如何支援企業運作的觀點描述一個軟體系統，並不涉及描述系統開發與運作之平台。

- PIM 必須以有完整定義 (Well-Defined) 的語言來描述，一個具有完整定義的語言具有完整定義的**語法 (Syntax)** 與**語意 (Semantics)**，且適合用電腦來自動解譯 (Automated Interpretation)。因此，以UML 來描述PIM 應是目前最好的選擇。

2.9 MDA發展生命週期(3/6)

- 特定平台模式 (PSM)

- PSM 是一種特定平台的模式，也就是該模式相依於軟體開發技術。對某一種PSM 而言，可能僅具有該特定平台知識的開發者才能理解。

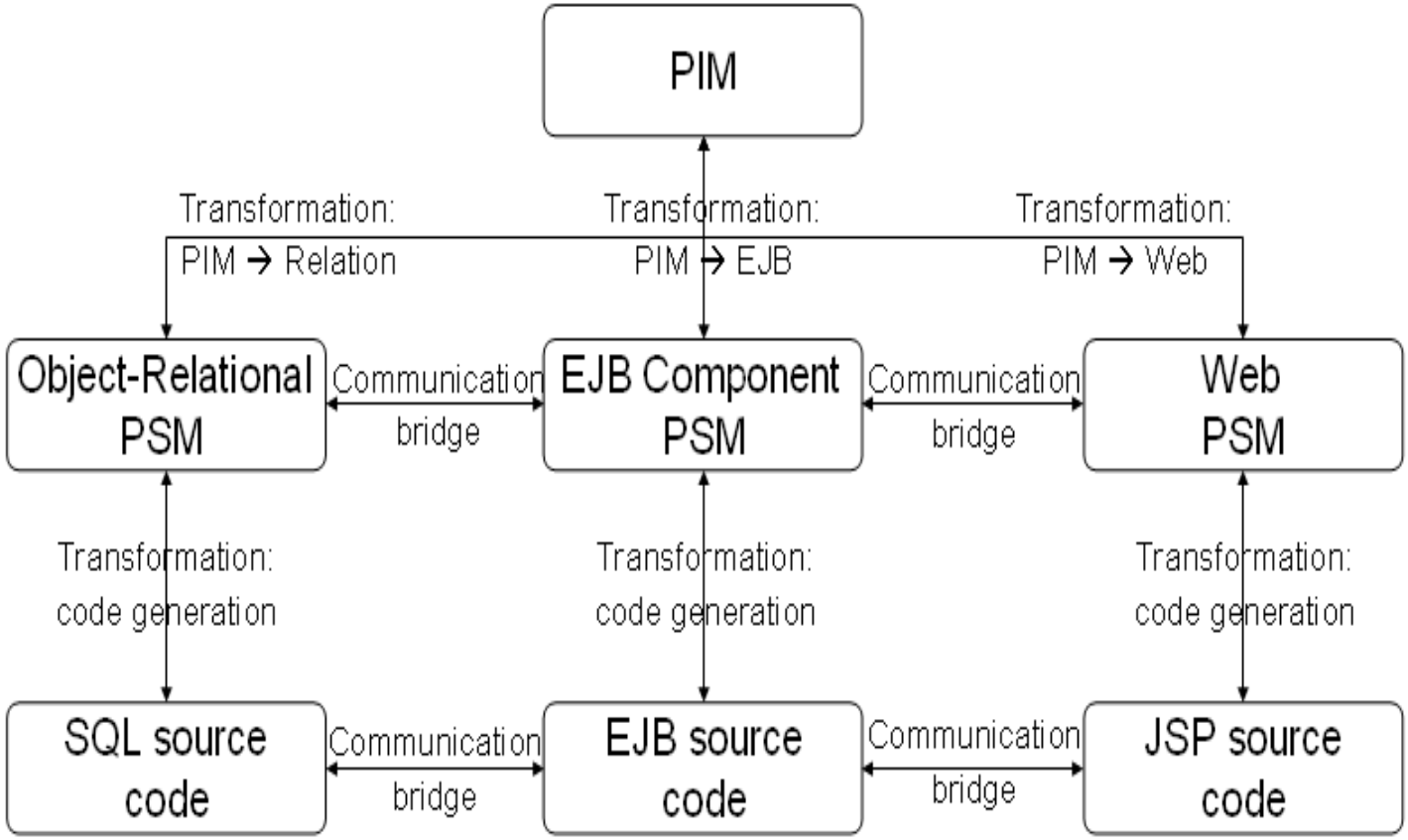
- 一個PIM 可被轉成一個或多個PSM，因為一個系統可能由數種技術（例如Relational PSM、EJB PSM、Web PSM）開發而成，對每一個特定的技術平台需產生一個與其他技術分開的PSM，PSM 間可藉由溝通橋樑 (Communication Bridge) 的機制來互動。

2.9 MDA發展生命週期(4/6)

- **程式模式 (Code)**
  - 每一個PSM 都需被轉成程式模式（或簡稱程式碼），因為一個PSM 相依於其開發技術，因此PSM 轉成程式碼之步驟非常直接。
  - 若有多個PSM 則會轉出多種的程式碼，不同的程式碼間也須藉由溝通橋樑的機制來互動。

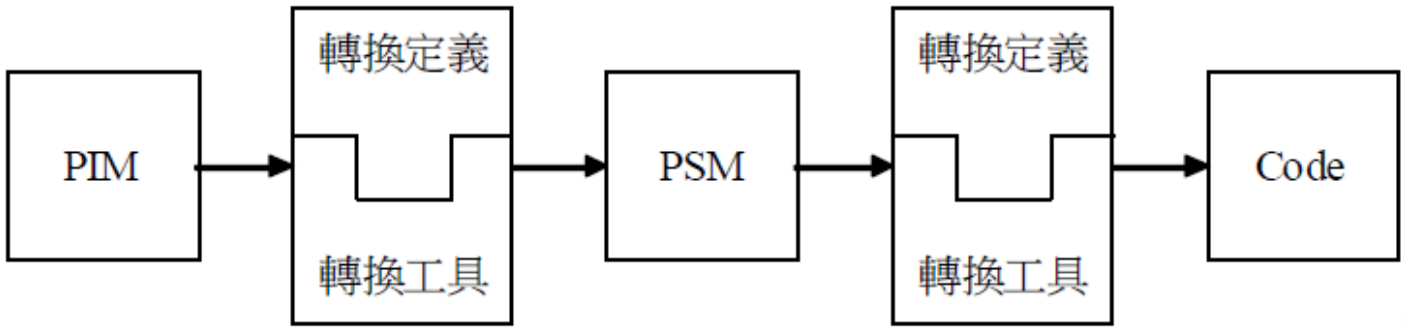
2.9 MDA發展生命週期(5/6)

PIM 轉PSM 轉code



2.9 MDA發展生命週期(6/6)

- MDA 的每一個轉換（例如PIM→PSM，PSM→Code）須有清楚的轉換定義，且該轉換的工作主要是藉由CASE Tool 來執行，也就是PIM 可藉由CASE Tool 轉換成PSM，再轉換成Code。



MDA 之三個主要模式與轉換步驟

2.10 八種系統開發模式之比較

模式	年代	基本假設 / 適用情況	主要特徵
瀑布模式	1970	1. 使用者需求可完整且清楚地描述。 2. 解決問題之知識(例如模式或方法)可以得到。 3. 軟硬體之技術與支援沒問題。	1. 開發階段有清楚的定義，每階段均需考量完整的系統範圍，且各階段僅循環一次。 2. 強調先有完整的設計與規劃，再進行編碼。 3. 重視設計與規劃之文件。 4. 一階段的完成需經驗證通過，才能進入下一階段。
漸增模式	1971	同上。	1. 開發階段有清楚的定義，把整個系統範圍分解成若干個子系統，各子系統之開發可依序以瀑布模式進行，亦可平行進行再整合。 2. 強調先有完整的設計與規劃，再進行編碼。 3. 開發週期反覆的進行。

2.10 八種系統開發模式之比較

雛型模式	1977	1.使用者需求無法完整且清楚地描述。 2.解決問題之模式或方法無法立即得到。 3.軟硬體之技術與支援不確定。	1.系統開發階段無清楚之分野，且開發週期反覆的進行。 2.不強調先有完整的設計與規劃再進行編碼。 3.強調快速地完成雛型且盡早使用，以作為雙方需求溝通與學習的工具。
螺旋模式	1986	適用於上述各情況。	1.綜合上述各情況。 2.強調各開發週期之規劃與風險評估。

2.10 八種系統開發模式之比較

同步模式	1993	*1.*需求可明確與完整的描述。 *2.*有足夠的人力參與。 *3.*團隊間有良好的溝通、資訊交換與專案管理。	*1.*將開發工作分割並同時進行。 *2.*系統測試不可分割，且各功能組都要執行。
RUP模式	1998	適用於上述各情況。	*1.*綜合上述各情況。 *2.*強調反覆與漸增的開發，及各開發週期之規劃與風險評估。 *3.*強調流程、工作產出與專案管理。

2.10 八種系統開發模式之比較

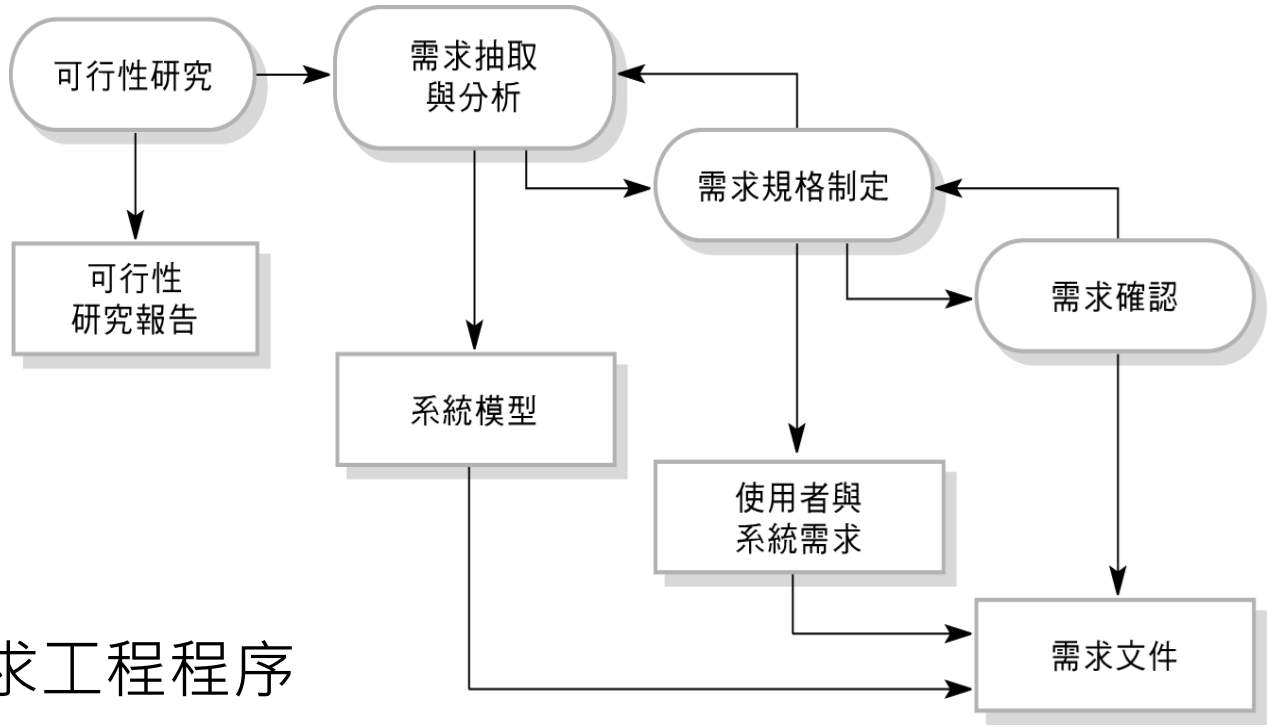
敏捷軟體開發	2001	*1.*使用者需求於開發過程中不斷變化。 *2.*開發團隊與使用者需有良好溝通和互動的機制。	*1.*強調開發團隊與使用者間協同合作。 *2.*強調反覆與漸增的開發方式。 *3.*強調隨時因應變化。
MDA模式	2001	適用於上述各情況。	*1.*綜合上述各情況。 *2.*每個階段的產出均須建構模式，且該模式是下一個階段的輸入。 *3.*各階段之產出是由電腦可理解的正規模式表達。

程序活動

- 軟體程序的4個基本活動分別是規格制定、開發、驗證與演進，在不同的軟體程序中有不同的組織方式。

軟體規格制定

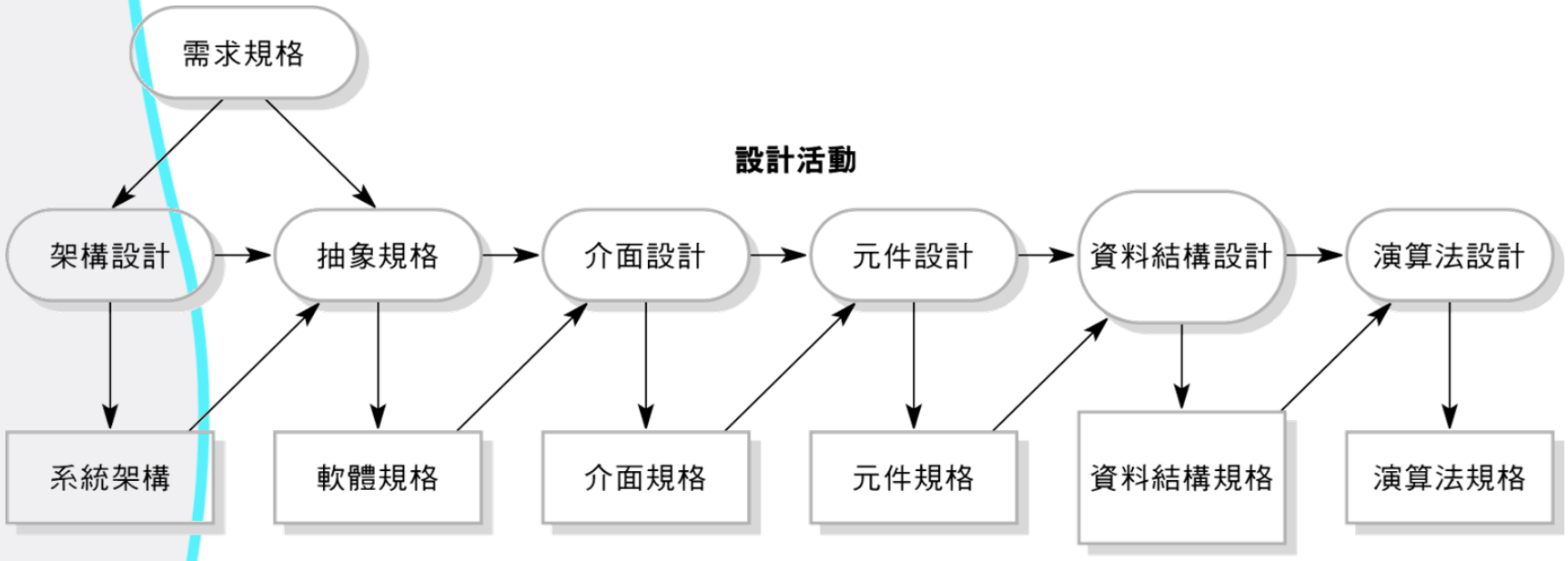
- 軟體規格制定（software specification）或稱需求工程（requirements engineering）這個程序，目的是瞭解與定義系統所需的服務，以及系統運作與開發的一些限制條件。



需求工程程序

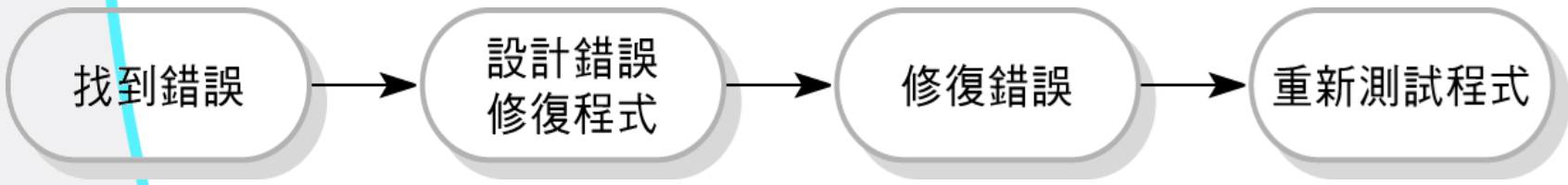
軟體設計與實作

- 軟體開發程序中的實作階段，是將系統規格轉換成可執行系統的過程，它通常包含軟體設計與程式撰寫的程序。



設計程序的一般模型

偵錯程序



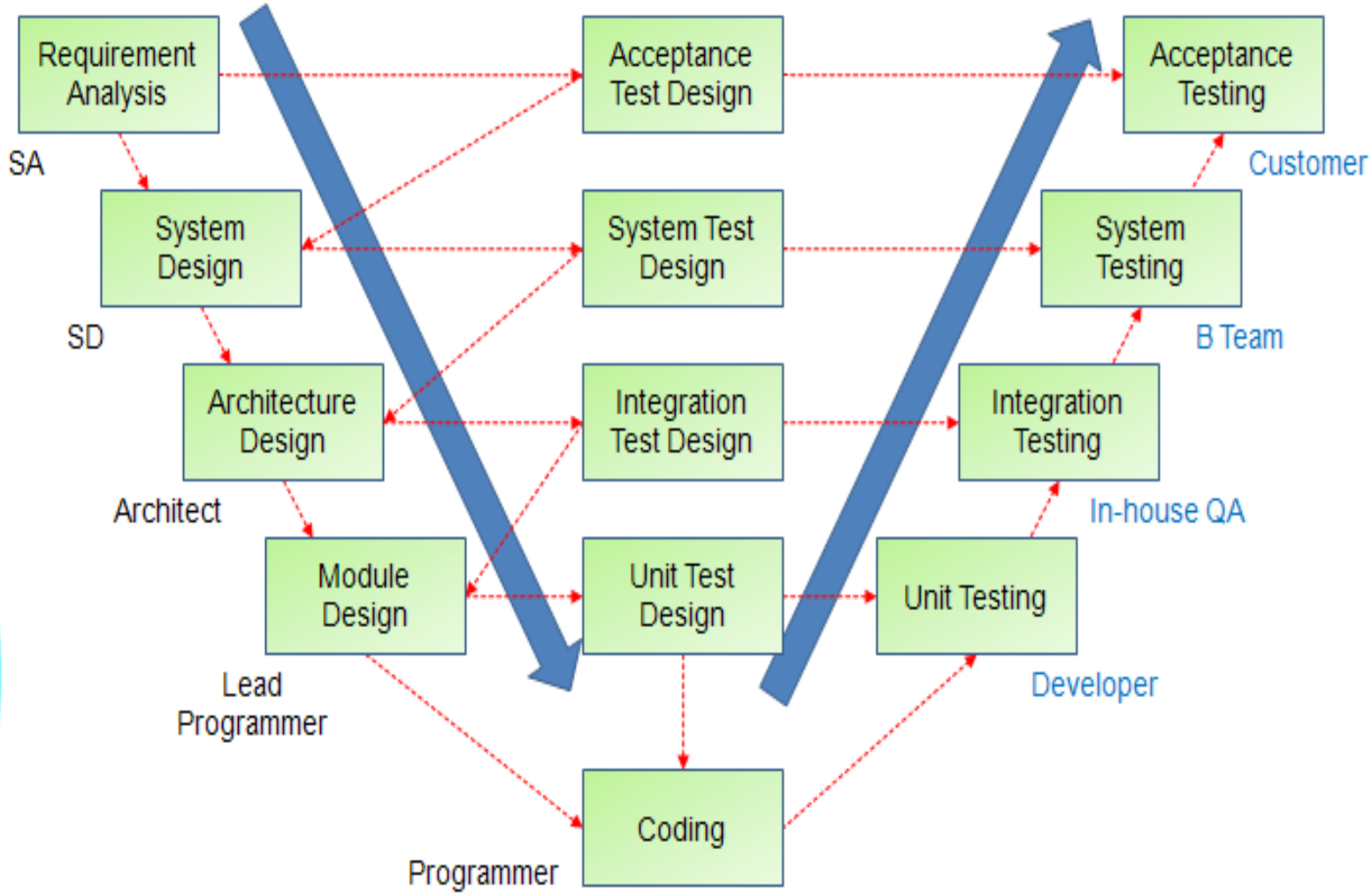
軟體確認

- 軟體確認（software validation）一般都稱為「驗證與確認」（Verification and Validation, V & V），目的是確保系統符合規格與客戶的期望。

驗證與確認的比較

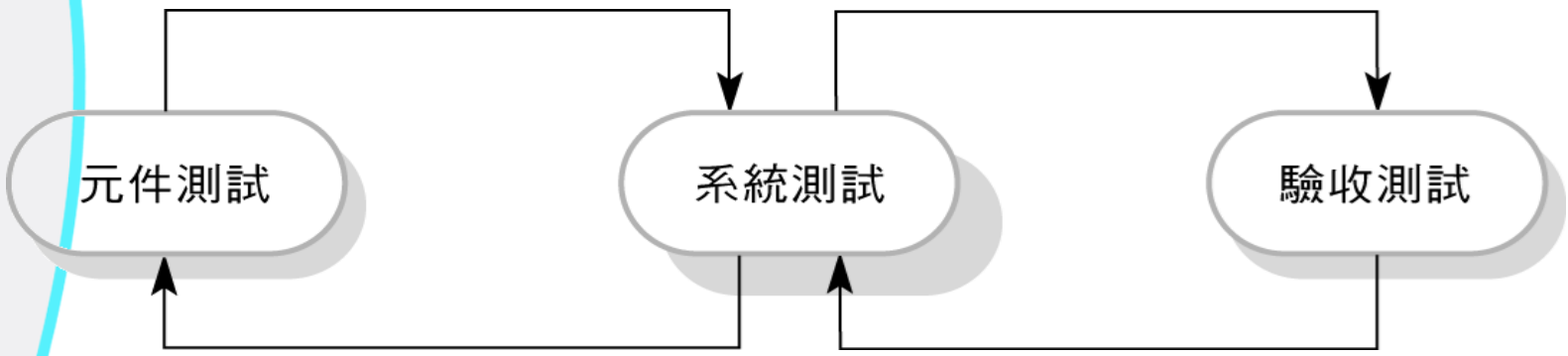
- **驗證(Verification)**：
  "我們是否正確的開發了產品？"
- 軟體應該與規格相符
- **確認(Validation)**：
  "我們是否開發了對的產品？"
- 軟體應該執行使用者真實的需求

V Model

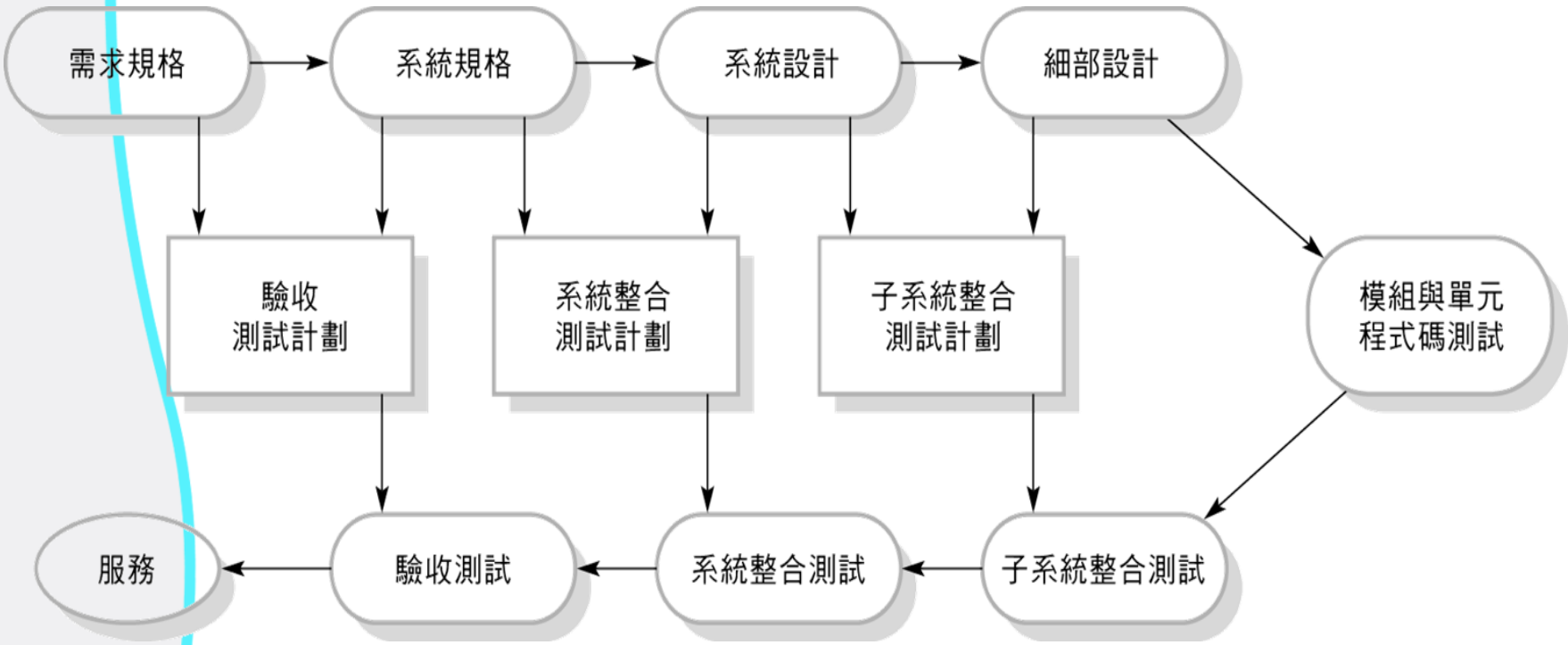

Requirement Analysis
SA
System Design
SD
Architecture Design
Architect
Module Design
Lead Programmer
Coding
Programmer
Acceptance Test Design
System Test Design
Integration Test Design
Unit Test Design
Unit Testing
Developer
Integration Testing
In-house QA
System Testing
B Team
Acceptance Testing
Customer

測試程序階段

- 元件測試（component testing）或單元測試（unit testing）：測試個別元件，確保它們能夠正確的運作。每一個元件是獨立測試的，與其他系統元件無關。
- 系統測試（system testing）：將元件組合成完整的系統。這個程序的重點在於找出子系統之間因為非預期的互動所引發的問題，還有元件介面問題。
- 驗收測試（acceptance testing）：這是測試程序的最後一個階段，系統被接受之後就可以開始上線運作。

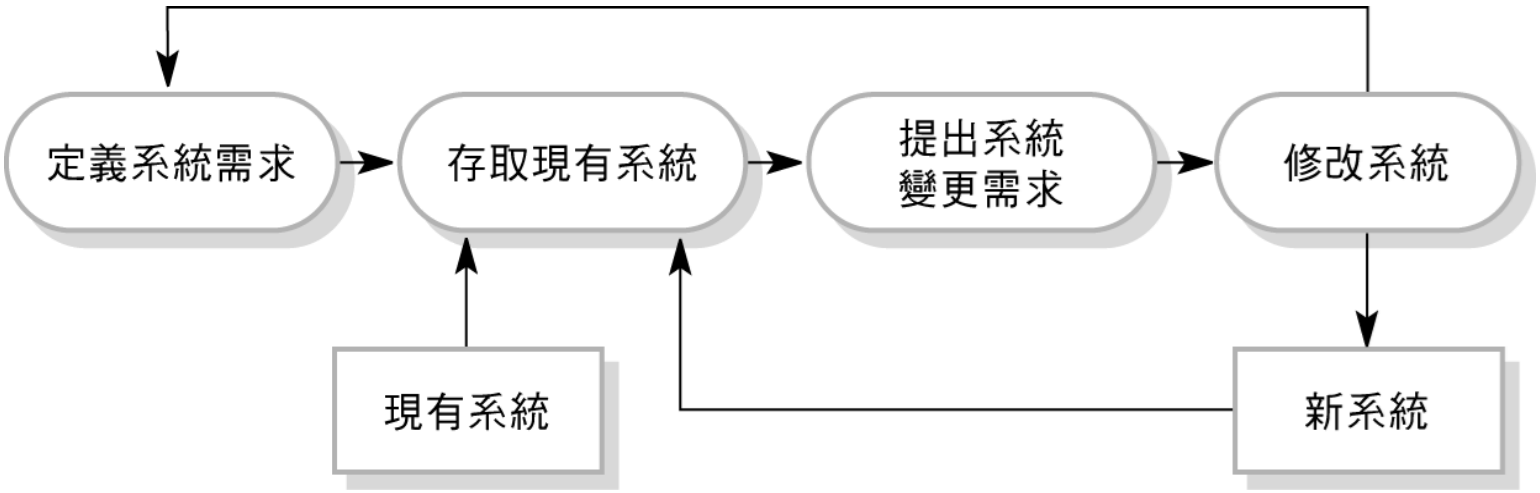


軟體程序中的測試階段


需求規格
系統規格
系統設計
細部設計
驗收
測試計劃
系統整合
測試計劃
子系統整合
測試計劃
模組與單元
程式碼測試
服務
驗收測試
系統整合測試
子系統整合測試

軟體演進

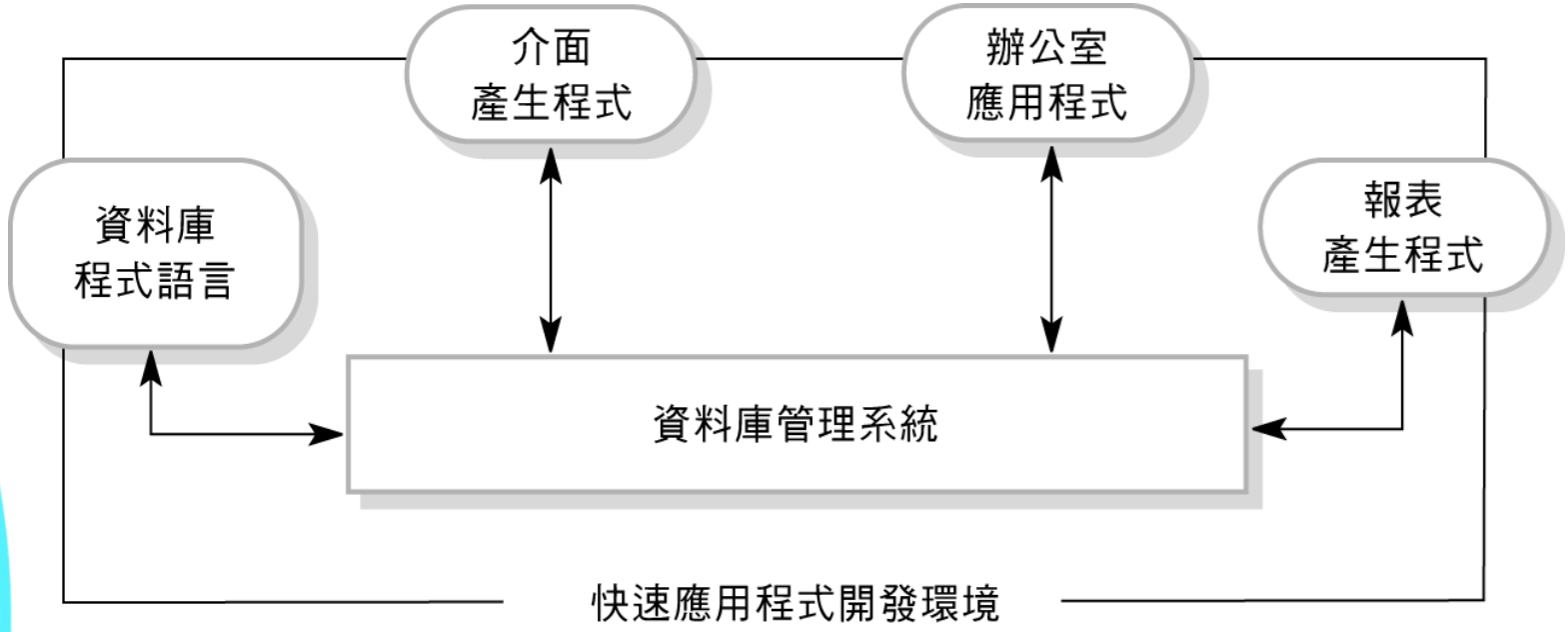
- 傳統上通常會將程序劃分為軟體開發程序與軟體演進（software evolution）程序，後者或稱軟體維護（software maintenance）程序。
- 不過開發與維護之間的區別已經漸漸不明顯了。因為現在很少有軟體系統是完全新的系統，而且將開發與維護視為連續性的動作似乎比較合理。



快速應用程式開發

- 快速應用程式開發（rapid application development, RAD）是從1980年代所謂的4GL（fourth-generation language, 第4代語言）發展出來的。
- 4GL是用來開發資料量龐大（data-intensive）的應用程式。
- 它們通常是組織成一組工具，它們能建立、搜尋、顯示資料和製作報表。

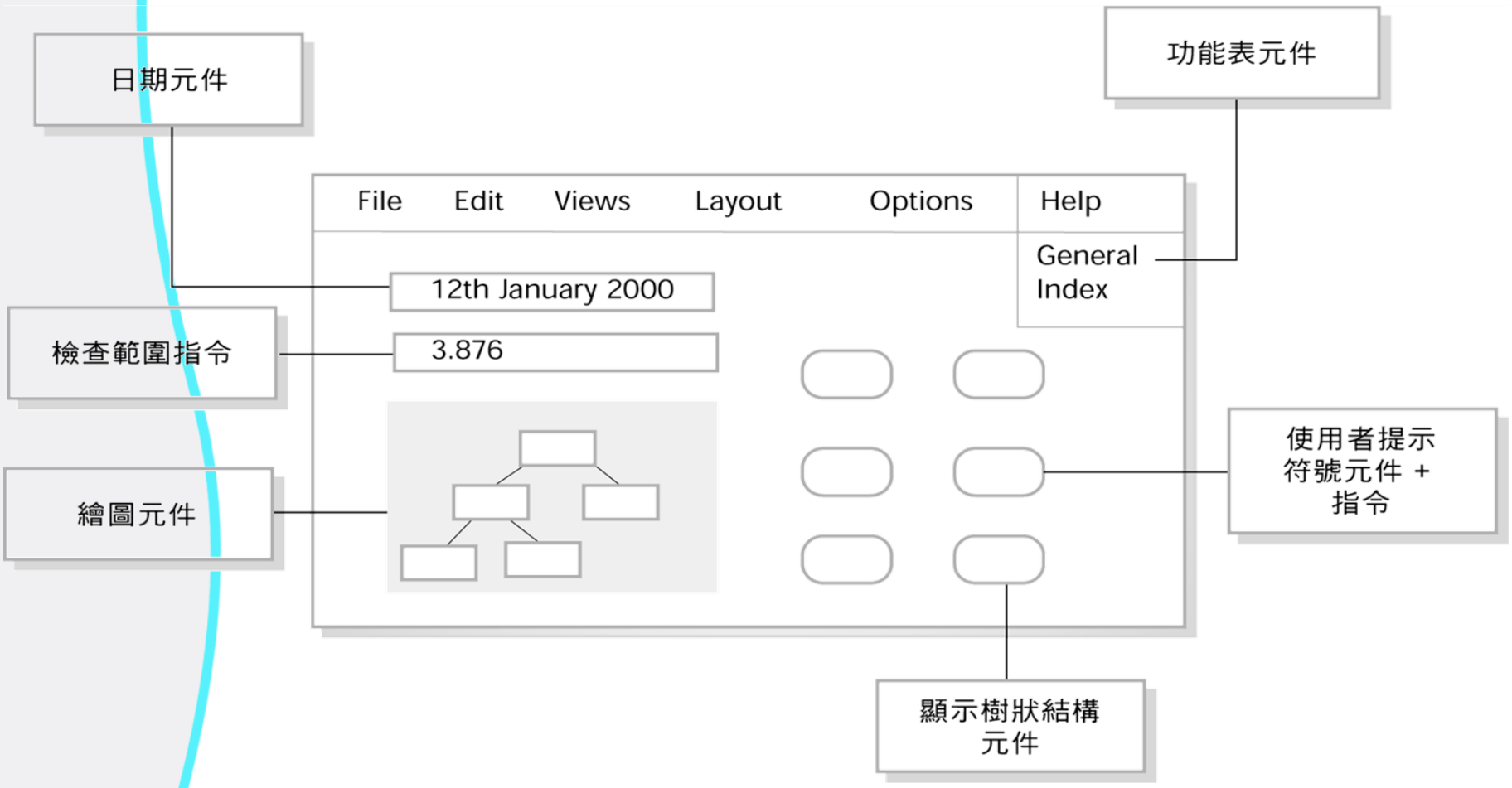
快速應用程式開發環境



快速應用程式開發

- 在RAD環境中包含的工具如下：
  - 資料庫程式語言：它內建資料庫結構的知識，並包含基本的資料庫處理運算。SQL（Groff et al., 2002）是標準的資料庫程式語言。SQL命令可能是直接輸入到系統中，或是由使用者所填寫的表單自動產生。
  - 介面產生程式：用來建立資料輸入和顯示的表單。
  - 連結到辦公室應用程式：例如試算表或文書處理程式。
  - 報表產生程式：使用資料庫的資訊來定義和建立報表。

視覺化程式設計搭配再利用技術



視覺化開發方法

- 視覺化開發方式（visual development）是一種建構RAD系統的方式，藉由整合較小的再利用（reusable）軟體元件來達成。
- 另一種以再利用技術為主的方式，它所再利用的「元件」是整個的應用程式系統，這有時被稱作以COTS為基礎的開發方式，其中的「COTS」代表「現成的商業軟體」（Commercial Off-the-Shelf），也就是已經可用的應用程式。
- 以COTS為基礎的開發方式，讓開發人員可以使用應用程式的所有功能。

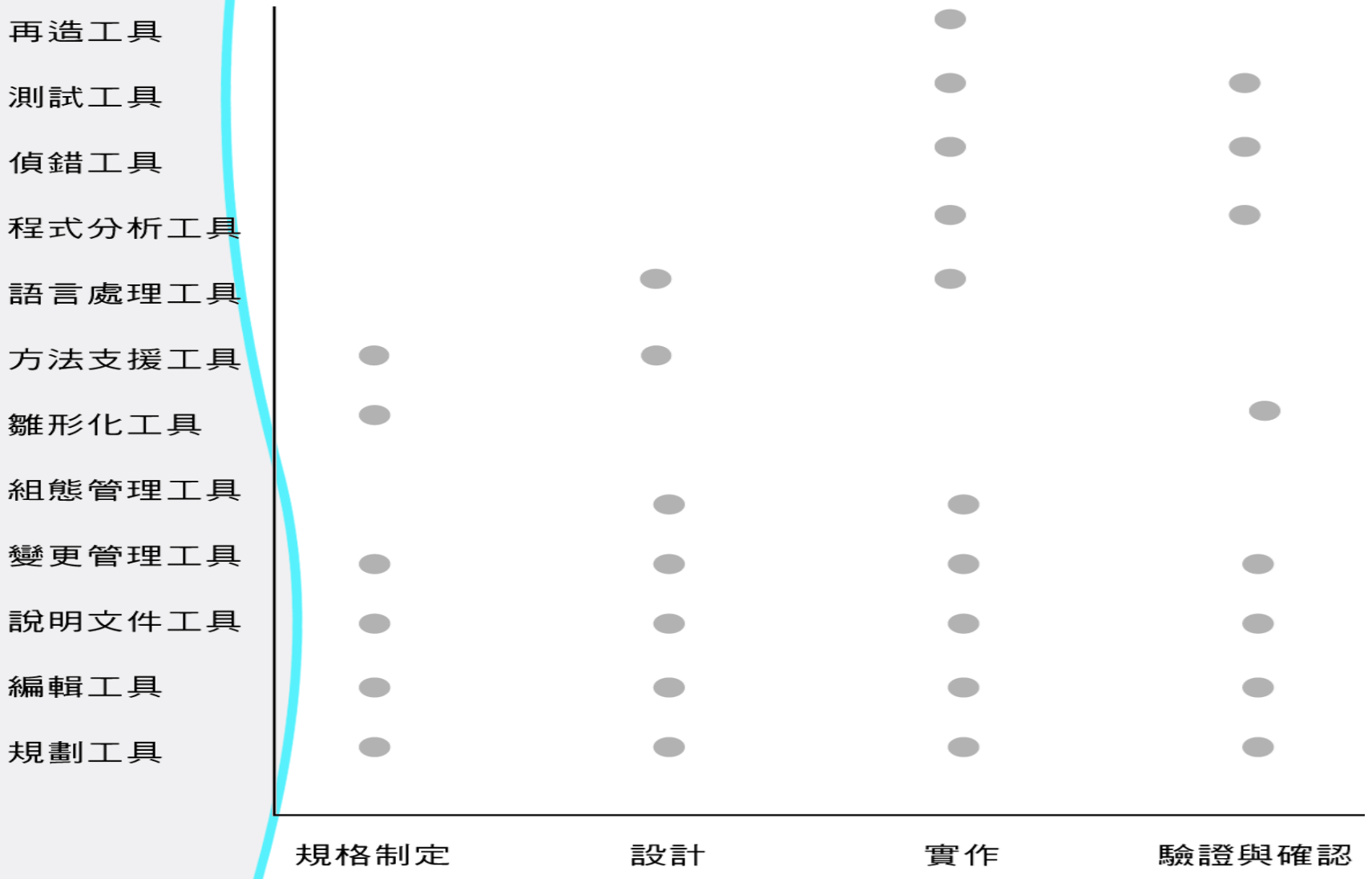
電腦輔助軟體工程(CASE)

- 電腦輔助軟體工程（Computer-aided Software Engineering, CASE）是一種軟體，用來支援軟體程序的各項活動，如需求工程、設計、程式開發和測試等。因此CASE工具包括有設計編輯程式、資料字典、編譯器、偵錯程式以及系統建置工具等。

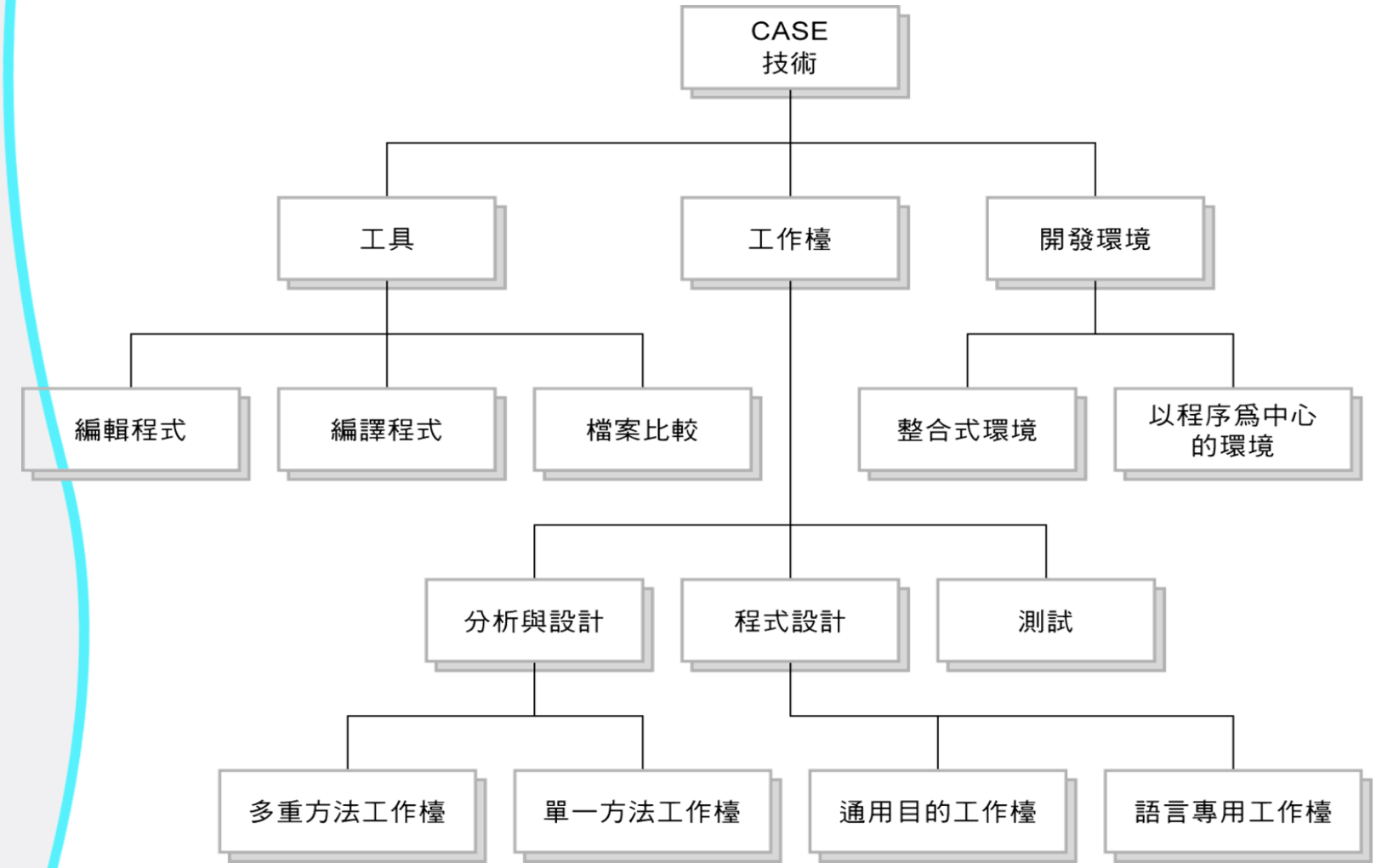
CASE分類

- 分類CASE工具的方法有很多種，每種分類法都是從不同的觀點出發。本節將從3種不同觀點來討論CASE工具：
  - 功能觀點：依據CASE工具的功能來分類。
  - 程序觀點：依據工具所支援的程序活動來分類。
  - 整合觀點：根據這些工具如何組織成不同的整合單元，針對一或多個程序活動提供支援。

以活動來分類CASE工具



工具、工作檯與開發環境


CASE
技術
工具
工作檯
開發環境
編輯程式
編譯程式
檔案比較
整合式環境
以程序爲中心
的環境
分析與設計
程式設計
測試
多重方法工作檯
單一方法工作檯
通用目的工作檯
語言專用工作檯