Software Engineering

U-03 軟體開發程序

Neter Chao Ph.D.

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

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。

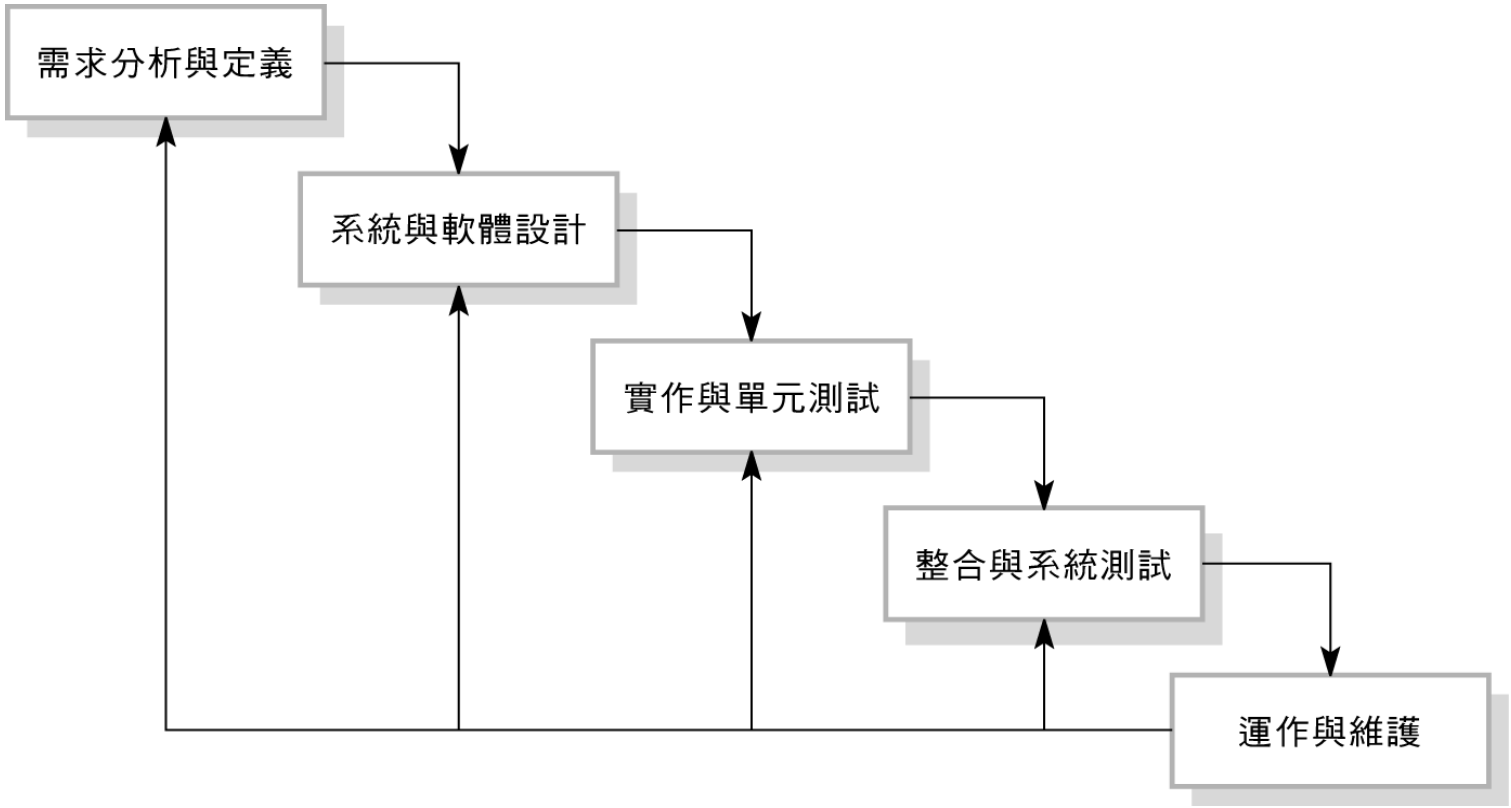
軟體程序模型

- 軟體程序模型（software process model）是軟體程序的抽象表示方式。
- 在此將討論下列幾種程序模型：
  - 瀑布式模型（waterfall）：這種方式將規格制定、開發、驗證和演進等幾個基本的程序活動，表示成各個分開獨立的程序階段。
  - 演進式開發（evolutionary development）：這種方式是讓規格制定、開發和確認等活動交互進行。
  - 元件式軟體工程（component-based software engineering）：這種開發方式是以現有可重複使用的元件為基礎。系統開發的過程主要是整合這些元件，而非從零開始開發。

瀑布式模型

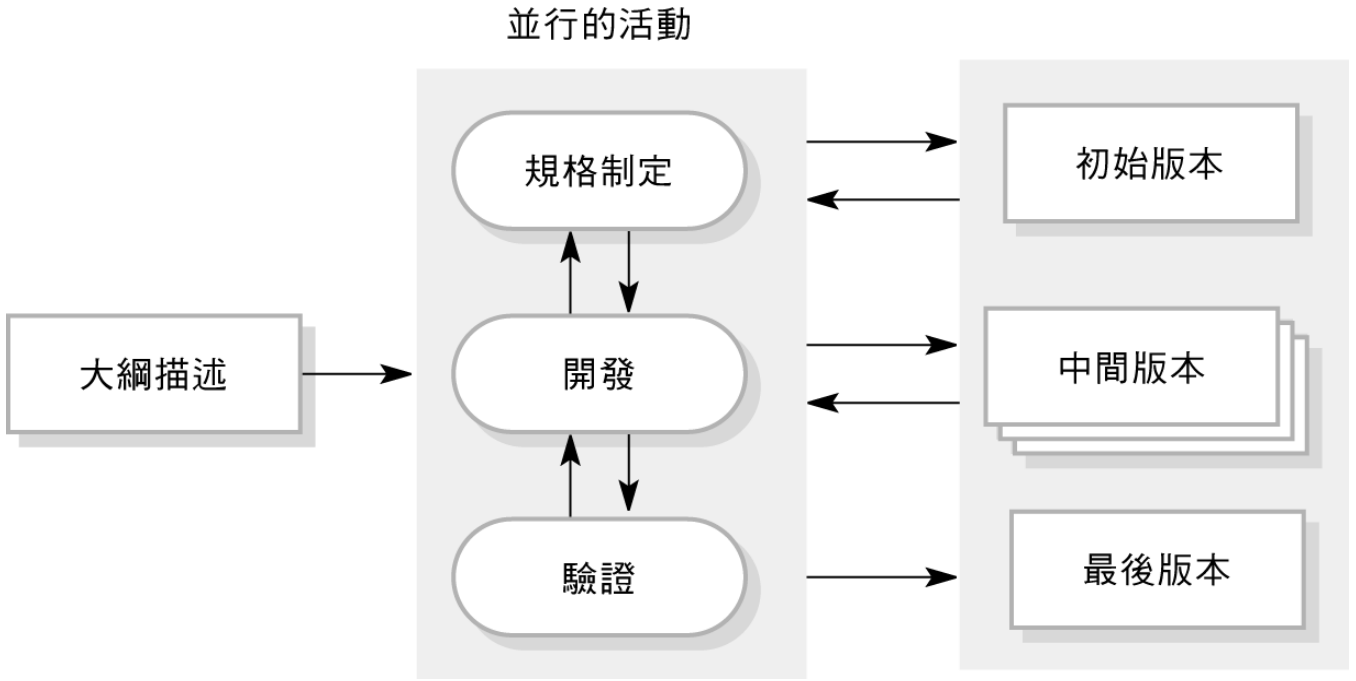
- 因為從某個階段進入下一個階段是以類似瀑布的方式進行，所以這個模型被稱為「瀑布式模型」（waterfall model）或軟體生命週期（software life cycle）。這個模型中的主要階段剛好對應到軟體開發的基本活動：
  - 需求分析與定義：與系統使用者進行諮商後，定義出系統的服務、限制和目標，之後再詳細定義成系統的規格。
  - 系統與軟體設計：系統的設計程序會將需求分割成硬體和軟體系統，以建立整體的系統架構。
  - 實作與單元測試：在此階段將軟體設計實作成一組程式或程式單元，單元測試則是驗證每個單元是否符合制定的規格。
  - 整合與系統測試：將獨立的程式單元或程式整合，並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。

軟體生命週期



演進式開發

- 演進式開發（evolutionary development）的基本概念是，先開發出一個初始的版本給使用者看，然後根據使用者的意見再做修改，接著不斷重複這些步驟，直到開發出最適當的系統為止。



演進式開發兩種基本類型

- 探索式開發(exploratory development):

  目標是與客戶合作，找出客戶的需求，然後開發出最後的系統。開發過程會先從系統需求最清楚的部分開始，然後以系統演進的方式，根據客戶提出的需求新增功能。

- 建立拋棄式雛形(throw-away prototyping):

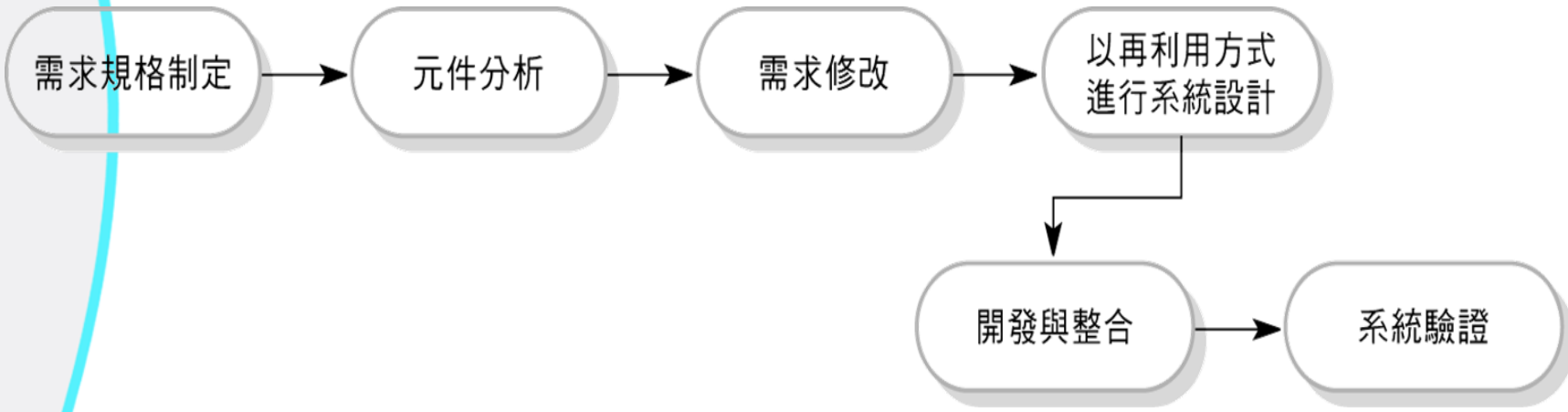
  目的是瞭解客戶需求，制定出更好的需求定義。雛形會專注在實驗那些還不是很清楚的客戶需求。

演進式開發有兩個問題

- 程序不是很透明化：經理人需要定期看到成果才能評估專案進度。如果系統開發太快，對每個版本都要製作說明文件就不是很合乎成本效益。
- 系統的結構通常很差：因為系統不斷的修改，可能會損害到系統的結構。因此之後對軟體進行變更也會愈來愈困難，成本也會愈來愈高。

元件式軟體工程

- 元件式軟體工程（component-based software engineering, CBSE）的軟體開發方法，它是以再利用技術為基礎，現在已經逐漸被廣泛使用。
- 這種再利用的方法必須依賴許多可再利用的軟體元件，還有一些能整合這些元件的架構。

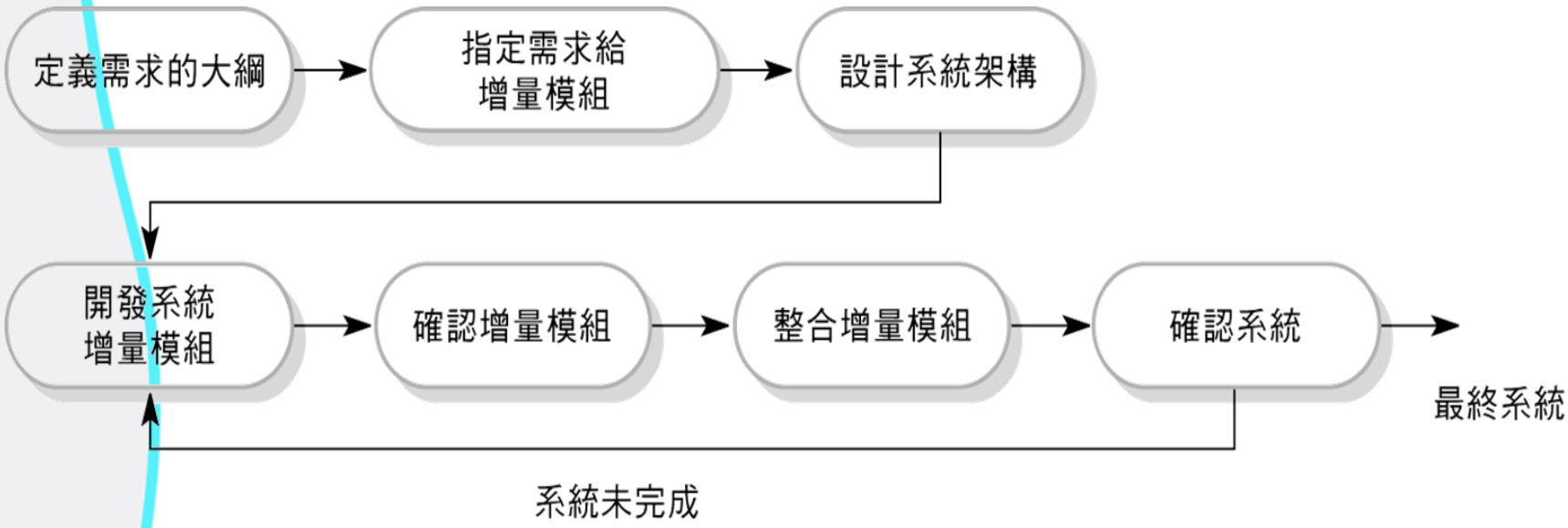


反覆式程序

- 軟體開發程序不是「一次定終生」的程序，因為系統要應付變更需求，所以會一直反覆執行開發程序的活動。
- 特別設計用來支援反覆式程序的兩種程序模型：
  - 增量式交付（incremental delivery）：將軟體的規格制定、設計與實作過程分解成一系列的增量模組（increment），然後輪流開發這些增量模組。
  - 螺旋式開發（spiral development）：系統的開發是以螺旋狀的方式，從最初始的輪廓向外逐漸發展成最後的系統。

增量式交付

- 增量式交付方法是介於瀑布式模型與演進式開發兩個模型之間的開發方法，它結合了這兩個模型的優點。



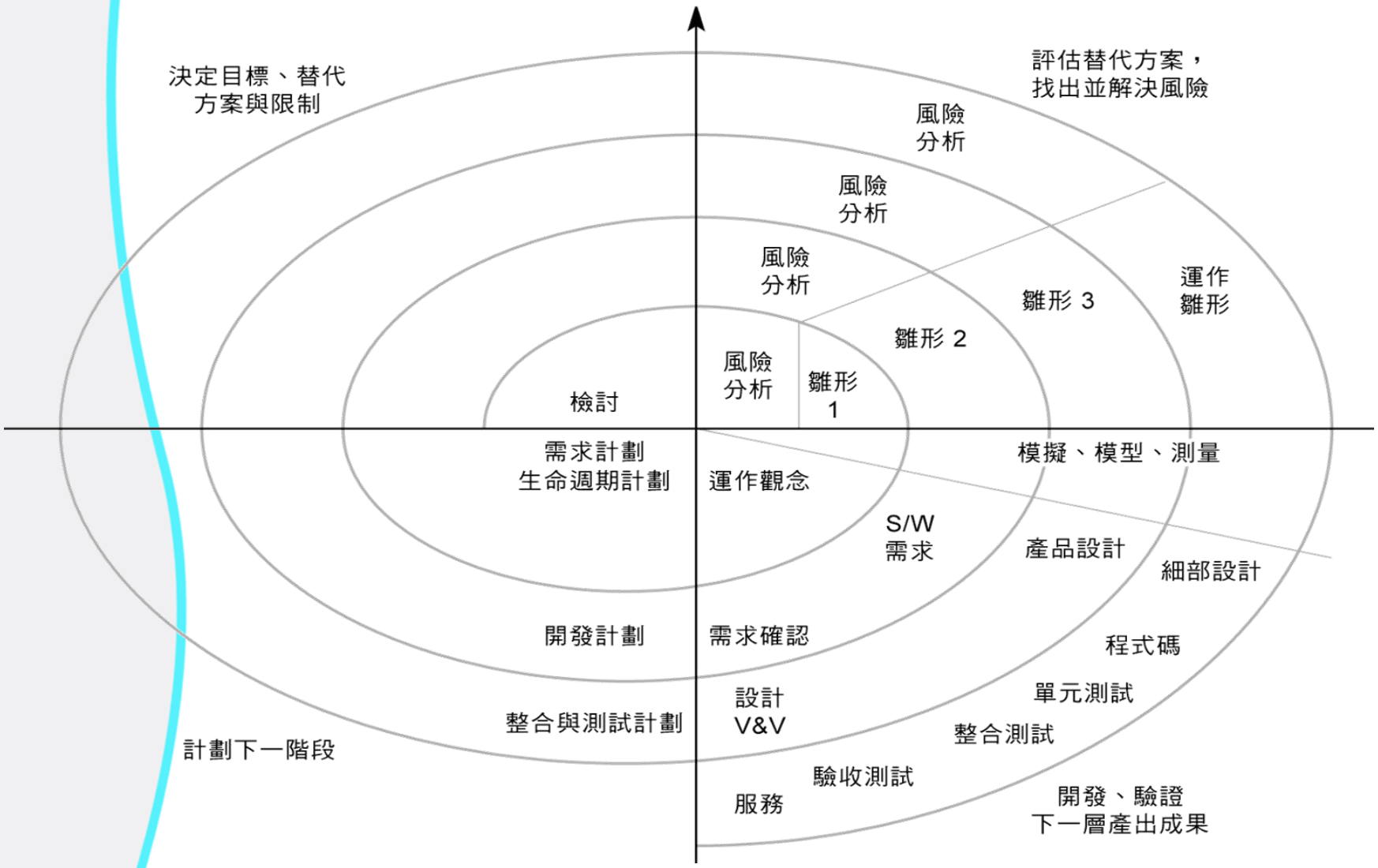
增量式交付程序優點

- 客戶不需要等到整個系統完成之後才開始使用。因為第一個增量模組是針對客戶最重要的需求，所以當第一個增量模組完成後，客戶就能馬上使用該項功能。
- 客戶可以把最早的增量模組當成雛形，從這個雛形中獲得一些開發後續增量模組的經驗。
- 整體專案的失敗率非常低。雖然在某些增量模組中可能會遇到問題，但是大部分的增量模組還是可以成功的交付給客戶。
- 因為具有最高優先順序的服務會先交付給客戶使用，後續的增量模組再與它整合，所以最重要的系統服務會被測試最多次。這表示系統最重要的部分比較不會出問題。

螺旋式開發

- 這個程序是以螺旋的方式來表示，螺旋中的每一個迴圈代表軟體開發程序中的一個階段。因此，最內圈可能是針對系統的可行性研究階段，而下一圈則與需求定義有關，再外一圈則與系統的設計有關，以此類推。
- 螺旋中的每一個迴圈可以分成四個部分：
  - 目標設定：定義專案目前階段的特定目標、找出程序和產品的限制、規劃詳細的管理計劃、確認專案的風險，以及根據風險可能需要規劃出替代策略。
  - 風險的評估與降低：對專案的每個風險進行詳細分析，並且採取降低風險的步驟。
  - 開發與確認：風險評估後，就可以為此系統選擇適當的開發模型。
  - 計劃：審查此專案，並且決定是否繼續進行下一個螺旋迴圈。若決定繼續進行，必須擬出此專案下一個階段的計劃。

Boehm軟體程序的螺旋式模型

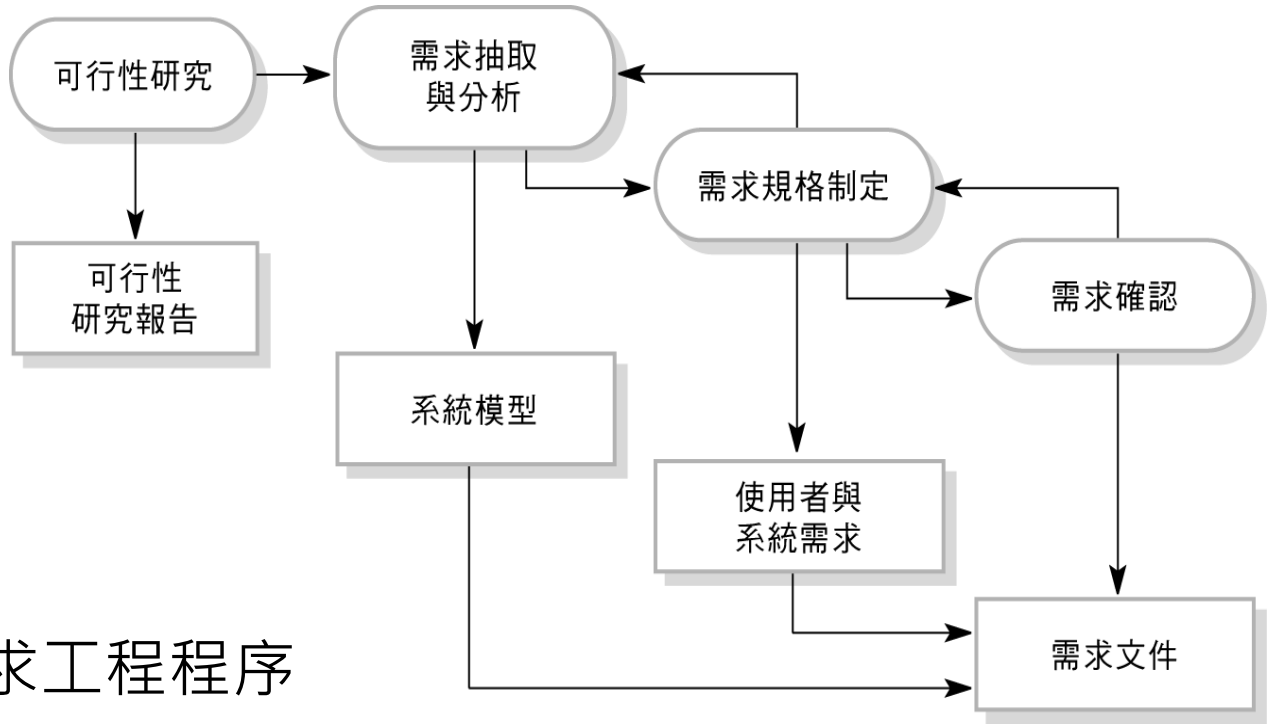


程序活動

- 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

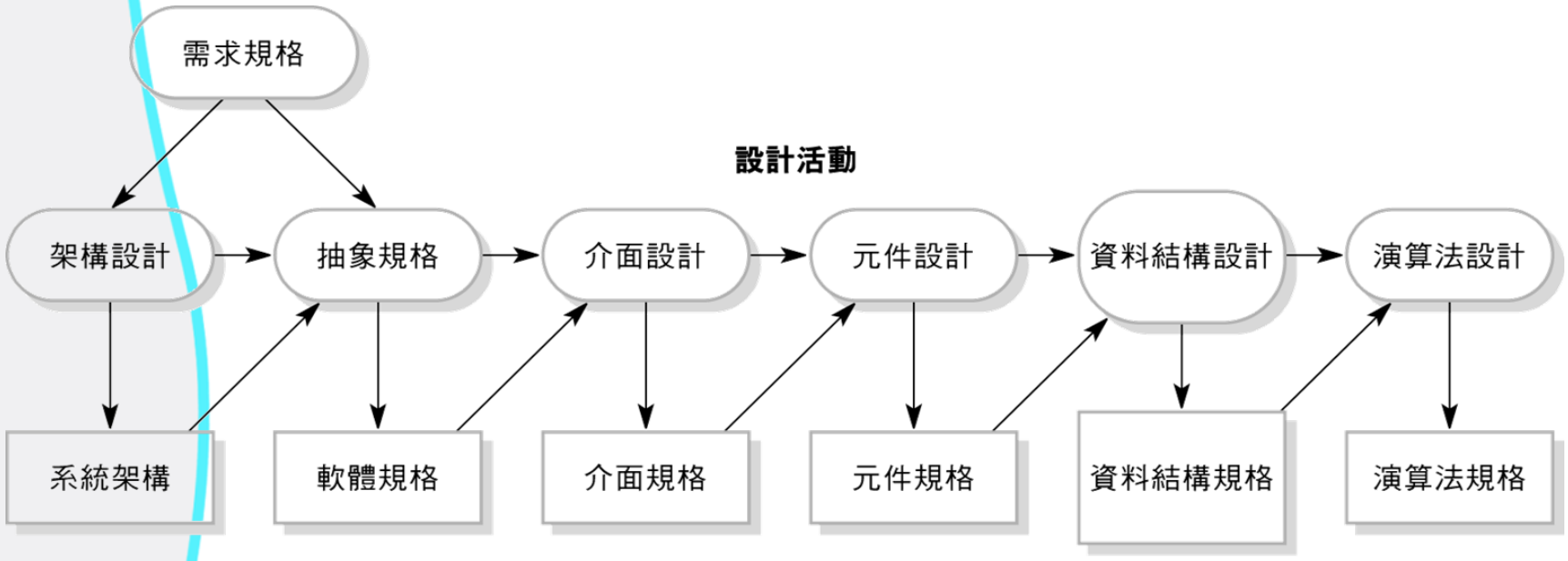
- 軟體規格制定（software specification）或稱需求工程（requirements engineering）這個程序，目的是瞭解與定義系統所需的服務，以及系統運作與開發的一些限制條件。



需求工程程序

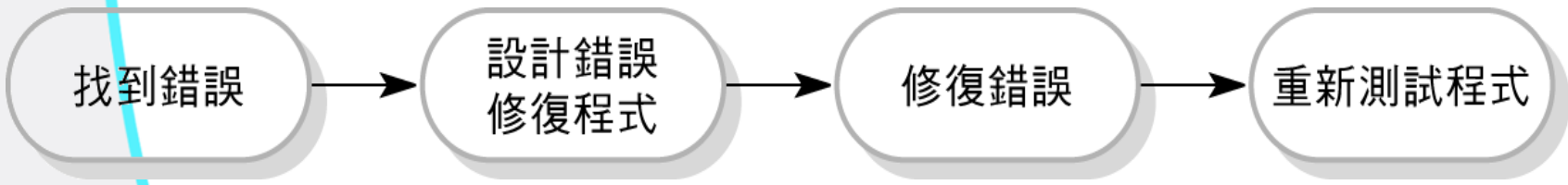
軟體設計與實作

- 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



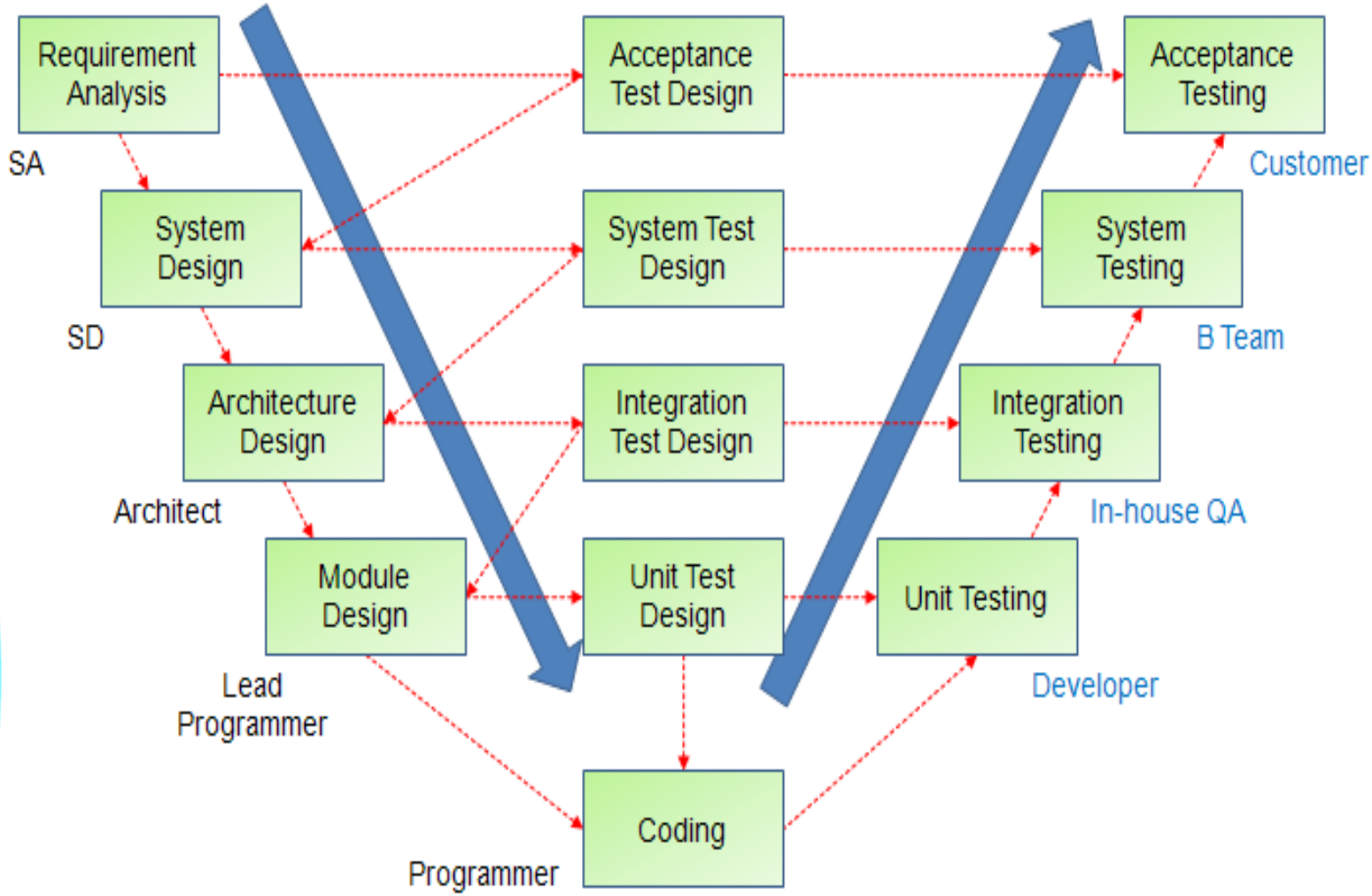
軟體確認

- 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

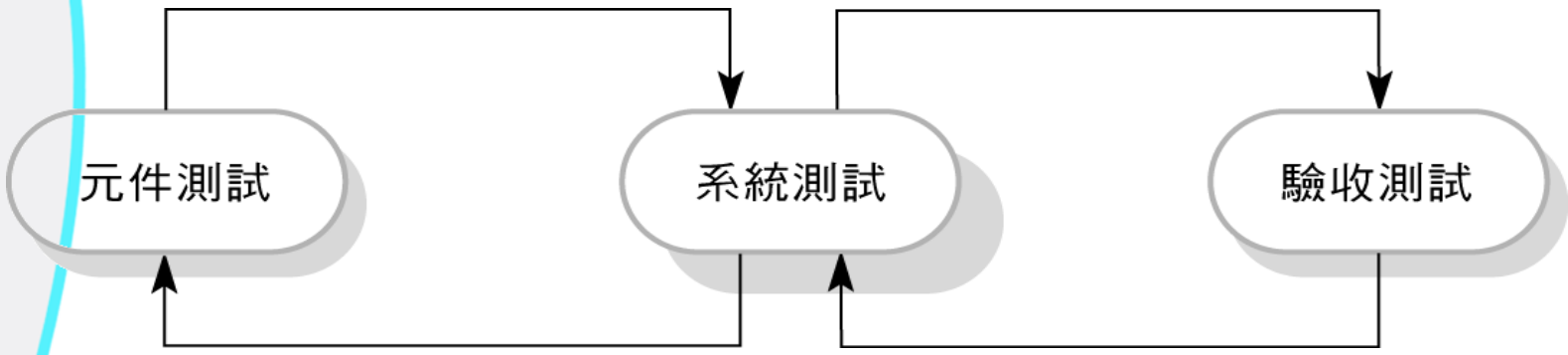
- **驗證(Verification)**：
  "我們是否正確的開發了產品？"
- 軟體應該與規格相符
- **確認(Validation)**：
  "我們是否開發了對的產品？"
- 軟體應該執行使用者真實的需求

V Model

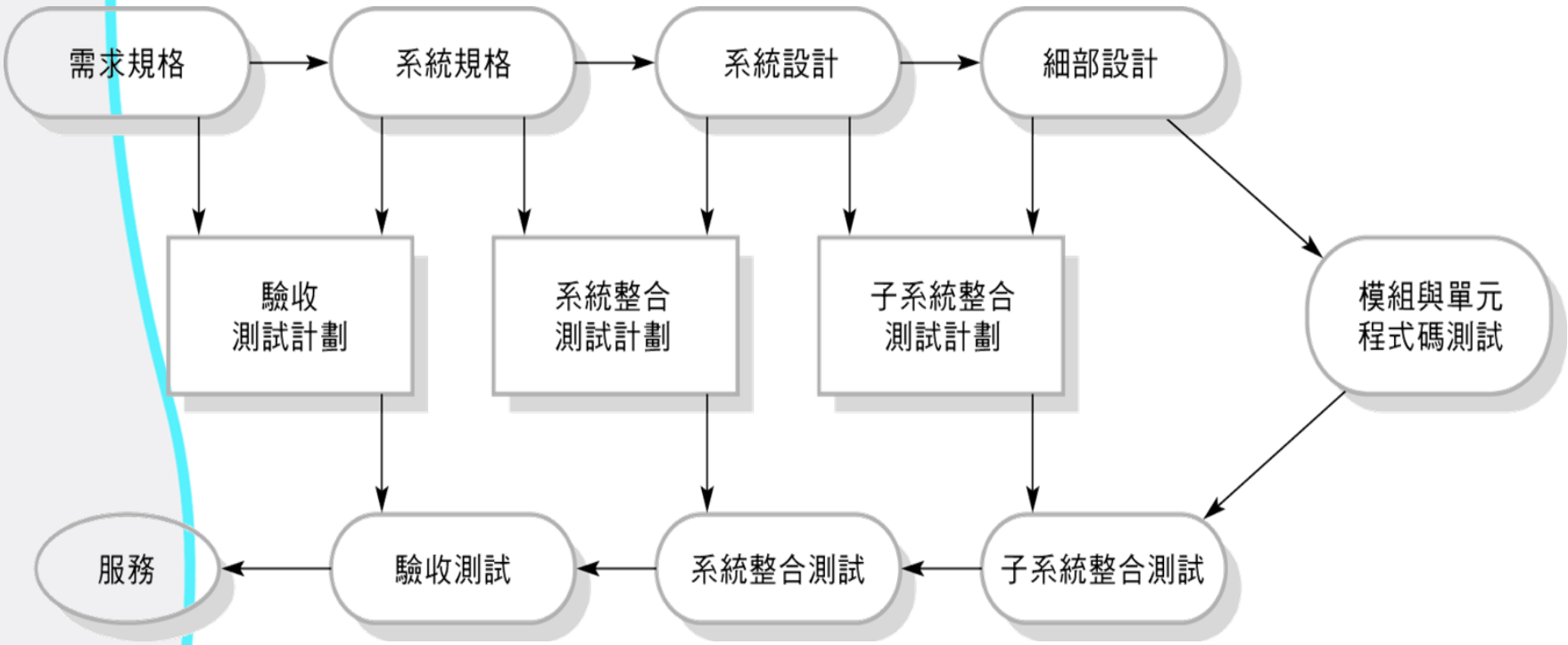

Requirement Analysis
SA
System Design
SD
Architecture Design
Architect
Module Design
Lead Programmer
Coding
Programmer
Acceptance Test Design
System Test Design
Integration Test Design
Unit Test Design
Unit Testing
Developer
Integration Testing
In-house QA
System Testing
B Team
Acceptance Testing
Customer

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。

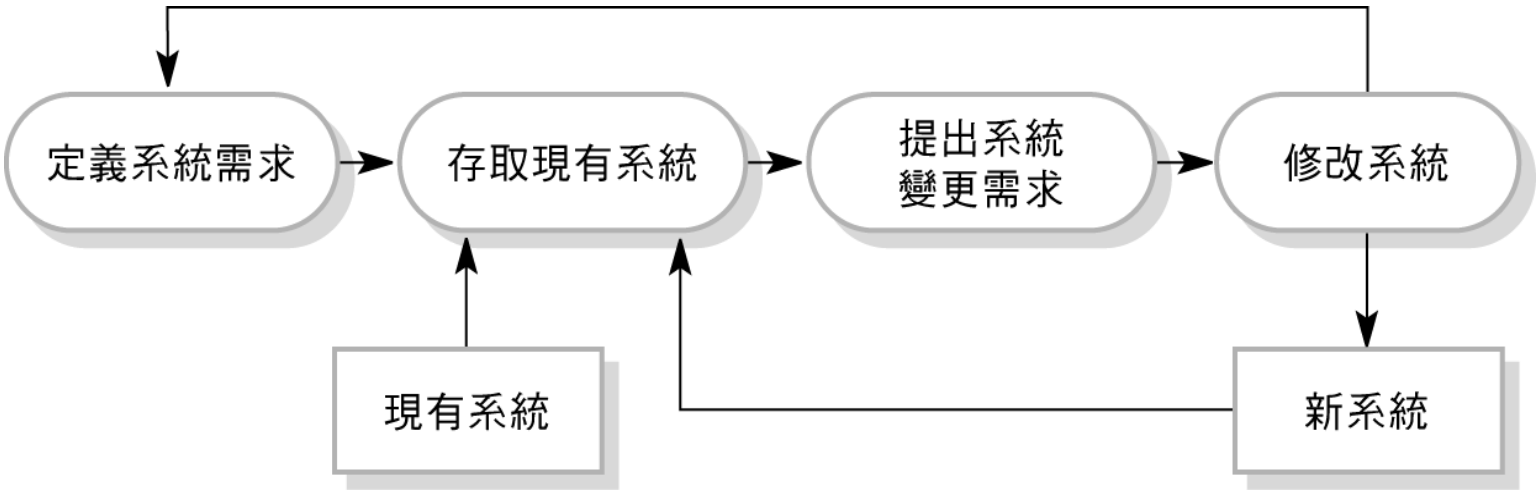


軟體程序中的測試階段


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。

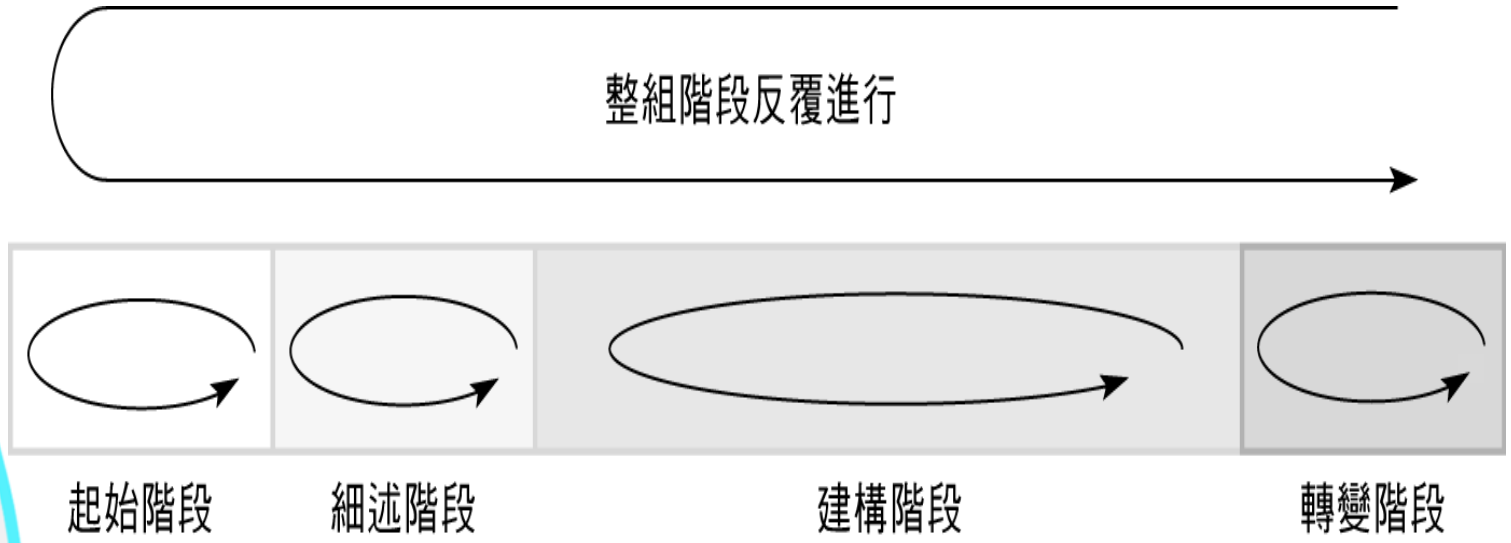


Rational Unified Process(RUP)

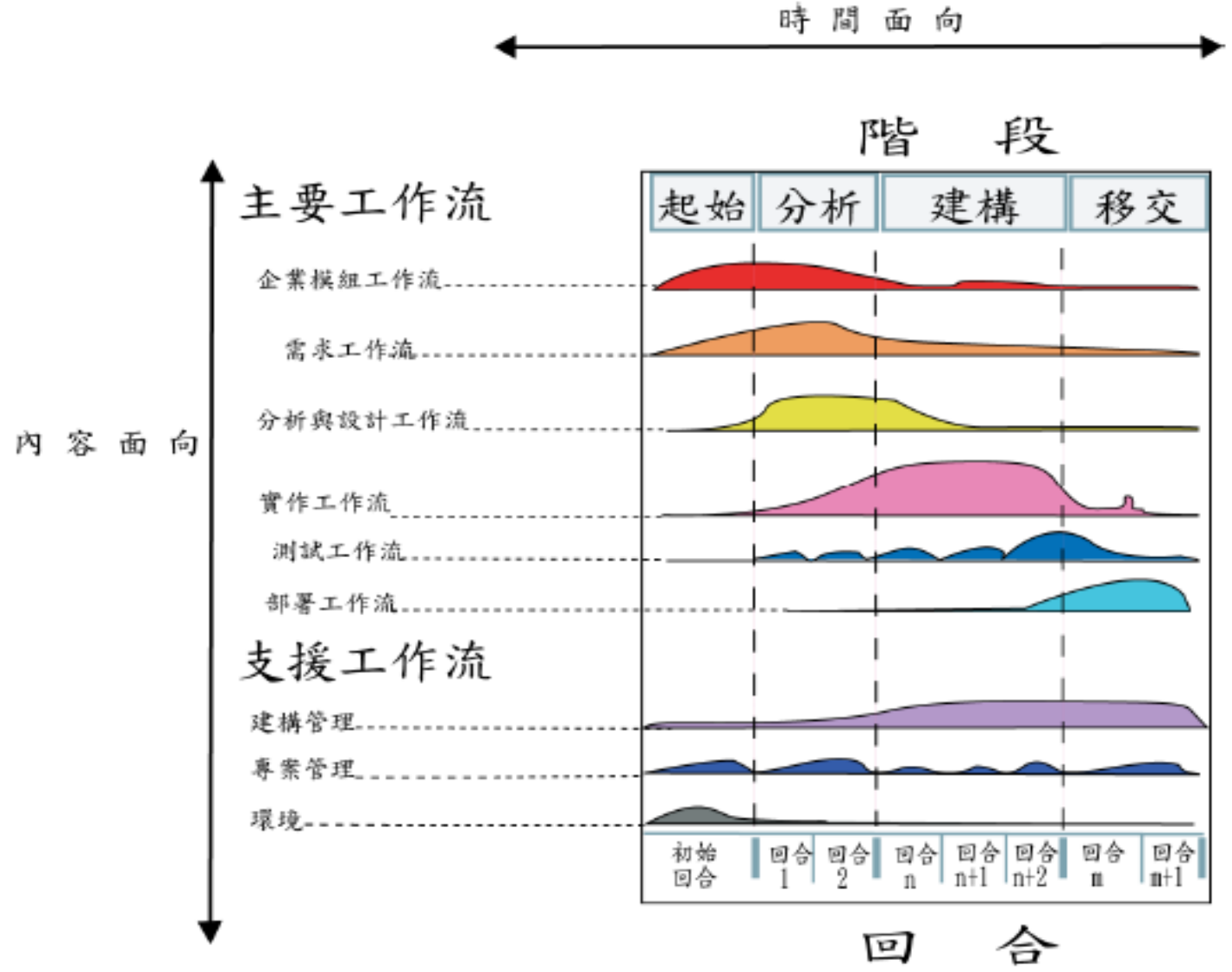
- Rational Unified Process（RUP）是現代程序模型的範例之一，它是由UML和相關的Unified Software Development Process（Rumbaugh, et al., 1999b）衍生而來。

- 而RUP一般都是從3個觀點來描述：
  - 動態觀點：顯示模型隨著時間經過的各階段。
  - 靜態觀點：顯示進行的程序活動。
  - 經驗觀點：建議在程序期間可採用的良好經驗。

RUP的各階段



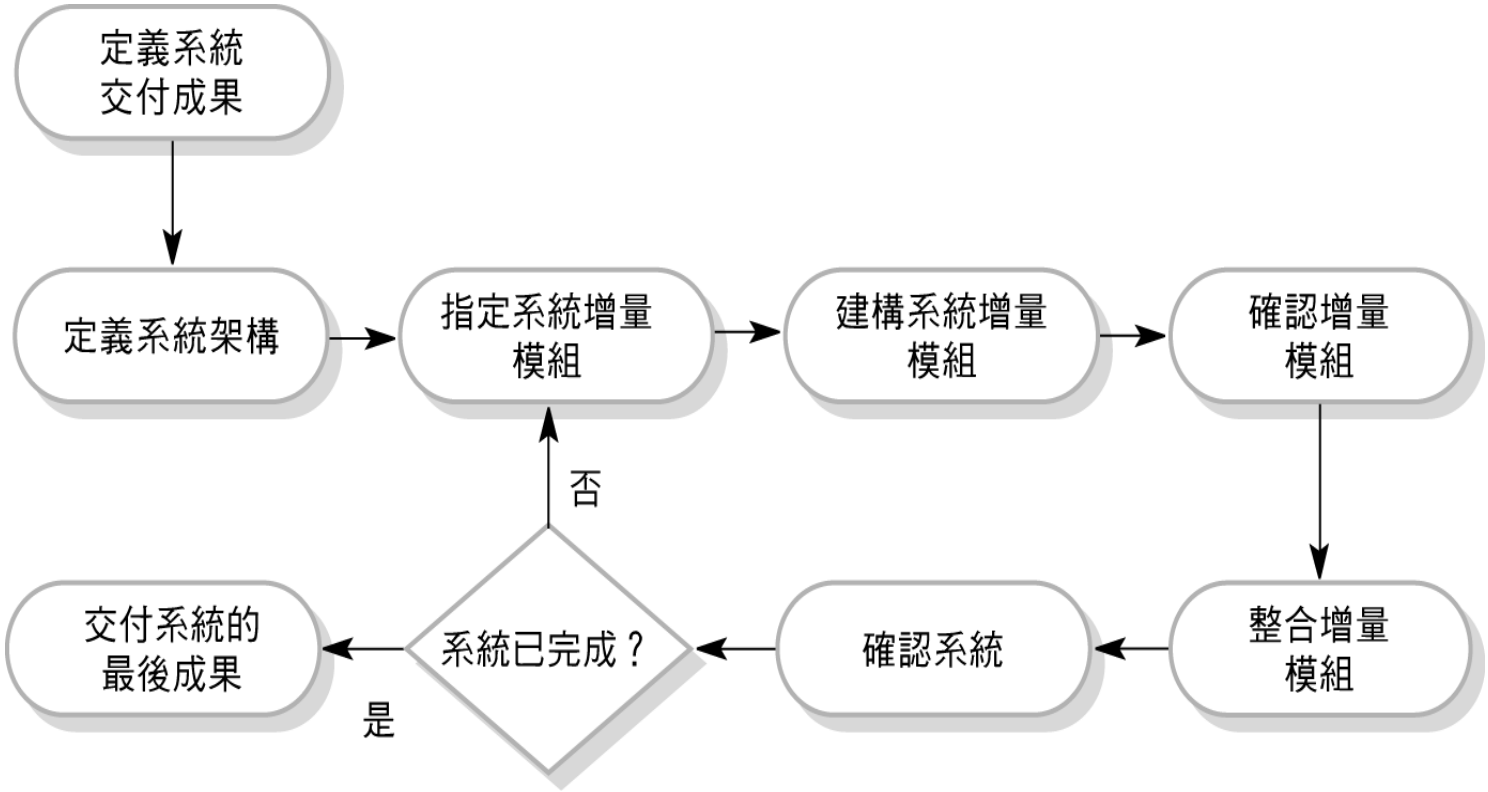
RUP的各階段


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

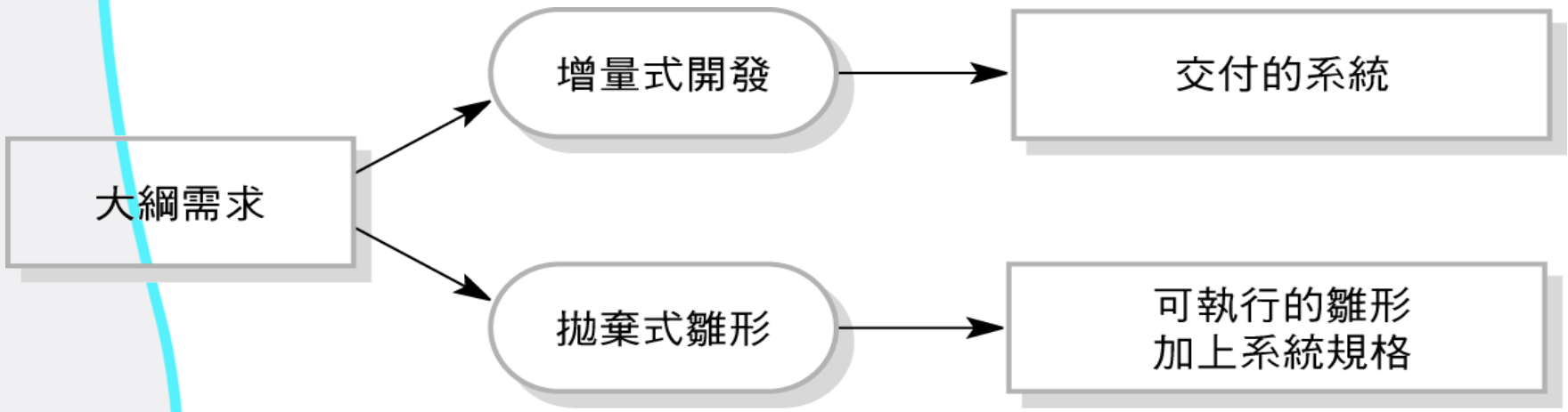
敏捷式開發方法

- 雖然快速軟體開發的方式有很多種，不過都有以下幾個共同的特色：
  - 規格制定、設計和實作程序是並行的（concurrent）。它沒有詳細的系統規格，而且設計文件也是盡量減少，或者是由程式設計環境來自動產生。使用者需求文件也只定義系統最重要的特性。
  - 系統是以一連串的增量模組來開發。終端使用者和其他的系統關係人都會參與指定和評估每個增量模組。
  - 系統使用者介面經常都是使用互動式系統來開發，此時的介面設計只要在介面上畫出和放置圖示即可。

反覆式的開發程序


定義系統
交付成果
定義系統架構
指定系統增量
模組
建構系統增量
模組
確認增量
模組
整合增量
模組
確認系統
系統已完成？
否
是
交付系統的
最後成果

增量式開發與拋棄式雛形



敏捷式方法

- 在1980年代和1990年代早期，當時覺得要開發出品質好的軟體，最好的做法是經由細心的專案規劃、正規化的品質保證、使用CASE工具所支援的分析和設計方法，並嚴加控管軟體開發程序。其實這個觀點主要是針對要開發大型而生命週期很長的軟體系統，但是對於**中小型系統**而言，這種開發程序的額外負擔（overhead）太大了。

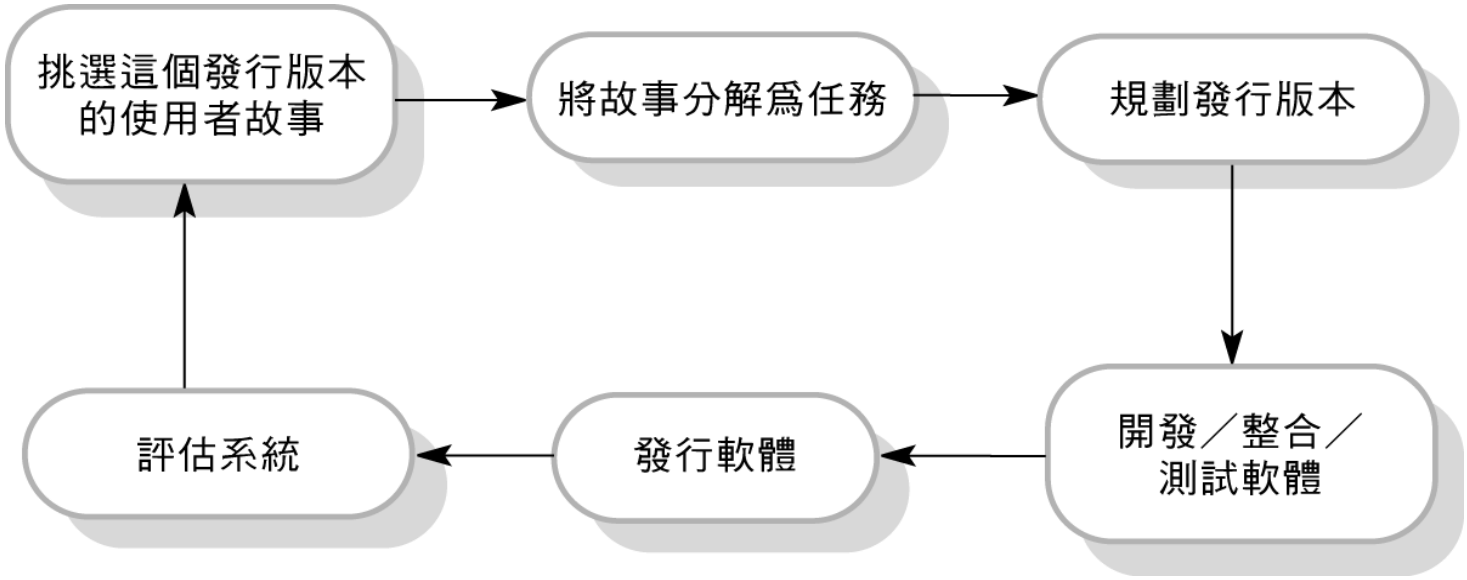
- 因為對這些繁重的方式感到不滿意，因此在1990年代有許多軟體開發人員提出各種新的敏捷式方法（agile method）。這類方法能讓開發團隊把重點放在軟體本身，而不是它的設計和說明文件。

敏捷式方法的原則

原則	說明
客戶參與	在開發程序期間會讓客戶密切參與，由他們負責提供新的系統需求和安排優先順序，並評估系統的成果
增量式交付	軟體是以一個個增量模組來開發，客戶所指定的需求會被包含在每個增量模組中
是人員而不是程序	開發團隊的技能應該被肯定與善用。團隊成員應該能依照他們的工作方式來開發，而不必拘泥於程序
擁抱改變	預期系統需求一定會變更，因此系統要設計成可以適應這些變更
維護簡單化	無論是被開發的軟體或開發程序都應該盡量簡單。可能的話，盡量採取行動降低系統的複雜度

極致程式設計

- 極致程式設計（Extreme programming, XP）可能是最有名也最受歡迎的敏捷式方法。
- 在極致程式設計中，所有的需求都是表達為情境（scenario），稱為使用者故事（user story），這些故事再直接實作成一系列的任務（task）。



極致程式設計的原則

原則	說明
增量式規劃	記錄在故事卡上的需求，是根據何時可用和相對的優先順序來安排在每一次的發行中。
小型的發行	每次發行的功能集合要儘可能的少，對於公司最具價值的優先開發。系統的發行頻繁，而且會逐漸增加功能
簡單的設計	實行的設計只要能符合目前的需求即可，不要再多
測試優先	一種自動化的測試架構，在新功能尚未實作前，就先撰寫其測試
重構（refactoring）	所有的開發人員都希望只要有找到可以改善程式碼的地方，就能持續不斷的重構程式碼。這會讓程式碼簡單而且好維護
雙人組程式撰寫	開發人員都是兩兩一組工作，互相檢查對方的工作並提供支援
集體擁有權	所有開發人員共同擁有程式碼，每個人都能修改
持續性的整合	每完成一項任務，就將它的程式碼整合到系統中，整合後所有的單元測試都要通過才行
維持正常的步調	大量加班是不被贊同的，因為通常這會降低程式碼的品質，影響中長期的生產力
在開發現場的客戶	客戶必須有個使用者代表全天候與XP團隊一起工作。

XP中的測試

- 在XP中的測試的重要功能如下：
  - 測試優先（test-first）的開發方式
  - 以增量方式從情境來開發測試案例
  - 使用者參與測試的開發與確認
  - 使用自動化的測試控制程式（test harness）

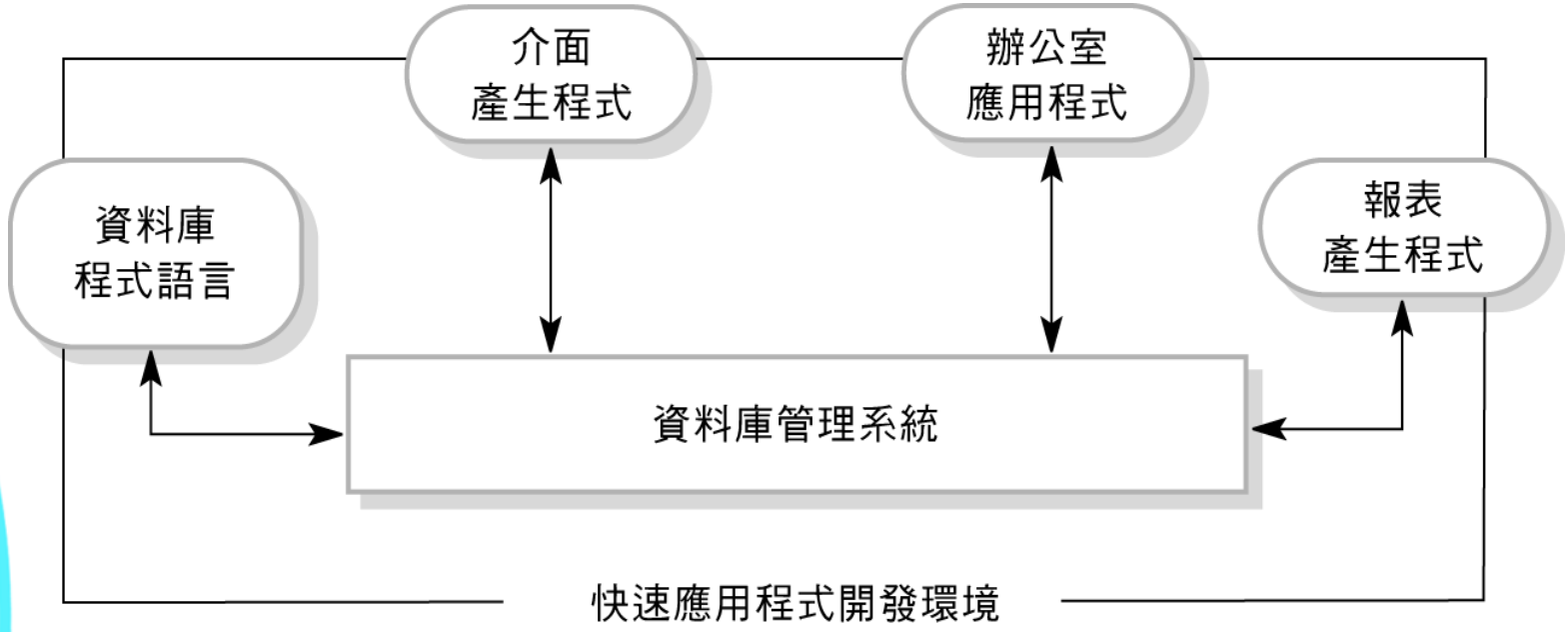
雙人組程式撰寫

- 使用雙人組程式撰寫技術有以下幾個優點：
  - 它支援共同擁有程式碼責任的想法。這反映出Weinberg的無私（egoless）程式設計（Weinberg, 1971）理念，也就是軟體是由整個團隊共同擁有的，因此程式碼的問題不是由某個個人來負責，而是整個團隊有共同責任要來解決。
  - 它其實就等於是一次非正式的審查動作，因為每行程式碼至少都有兩個人看過。而雙人組程式撰寫雖然比較不正式，找出來的錯誤比例沒那麼高，但是成本要比正式的程式檢查便宜多了。
  - 它有助於重構，這是軟體改善程序的一部分。由於雙人組程式撰寫是共同擁有，因此其他人可以馬上感受重構的好處。

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。

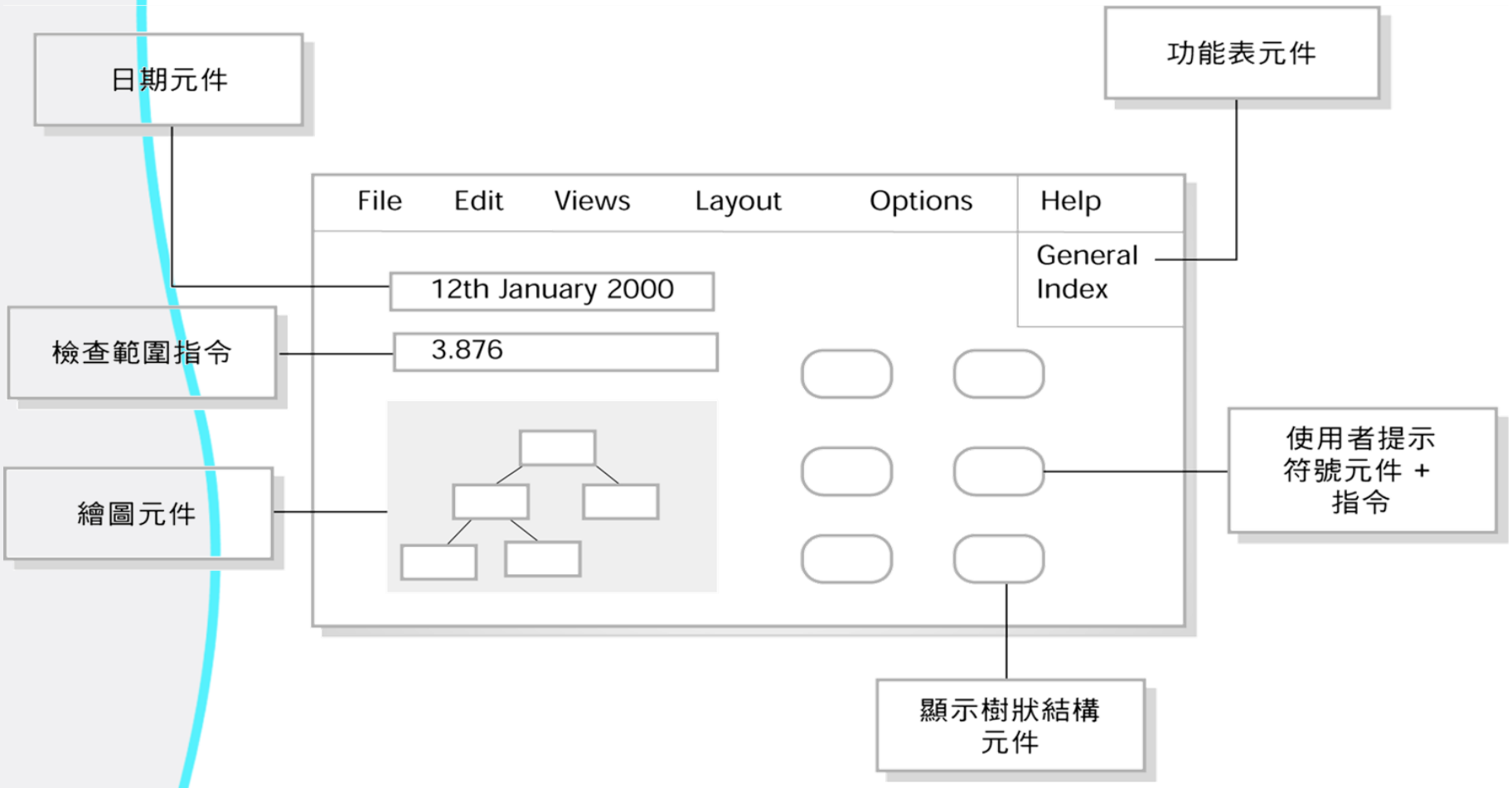
快速應用程式開發環境



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。

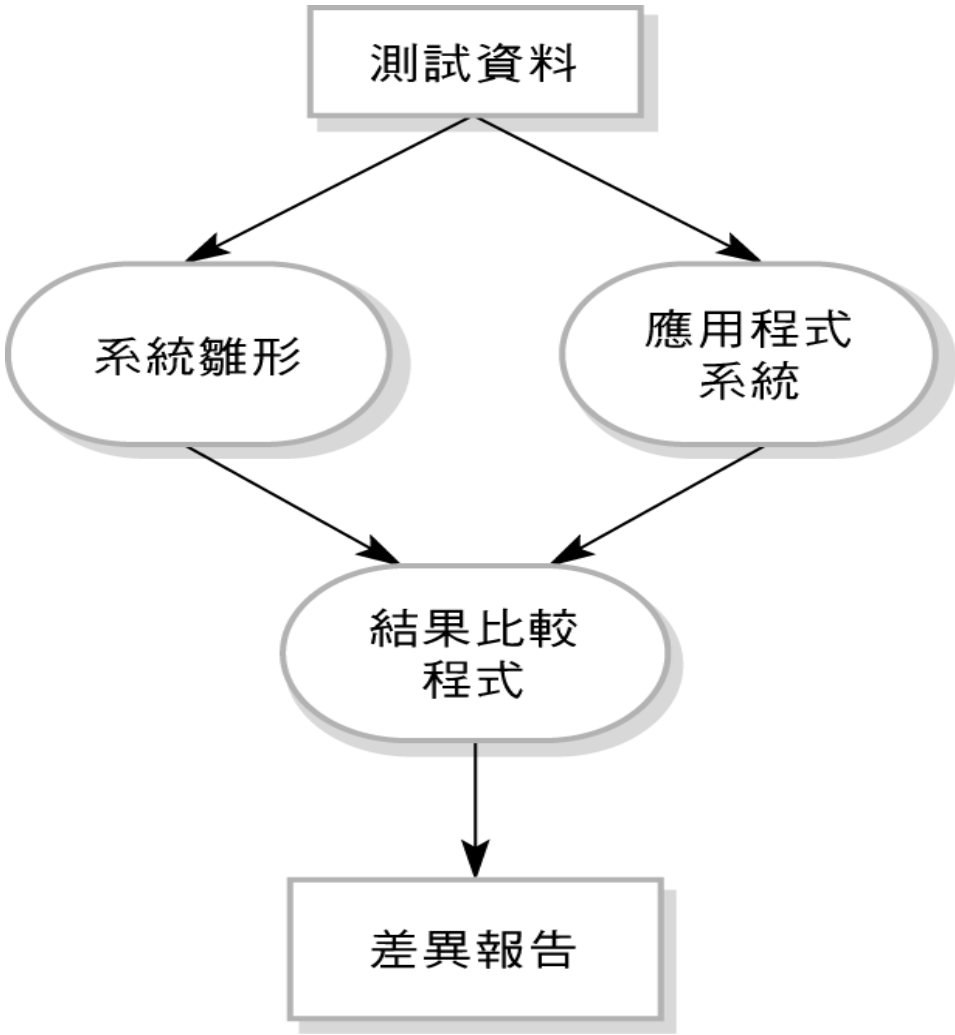
建立軟體雛形

- 有些情況雖然不適合使用增量式開發程序，但是建立軟體的雛形（prototype）也可以得到一些好處。
- 這種方式有時稱作建立拋棄式雛形（throwaway prototyping），因為這個雛形不會交付給客戶，開發人員也不需要維護它。
- 雛形（prototype）是指軟體系統某個初始版本，通常只是用來展示軟體系統的概念、嘗試設計的選項，以及用來瞭解問題和找出解決方案。

建立軟體雛形

- 在軟體開發程序中使用軟體雛形有以下幾種方式：
  - 在需求工程程序中，使用雛形可以幫助擷取和確認系統需求。
  - 在系統設計程序中，使用雛形有助於尋找軟體的解決方案和支援使用者介面設計。
  - 在測試程序中，雛形可以用來為即將交付給客戶的系統進行「背對背測試」（back-to-back test）。

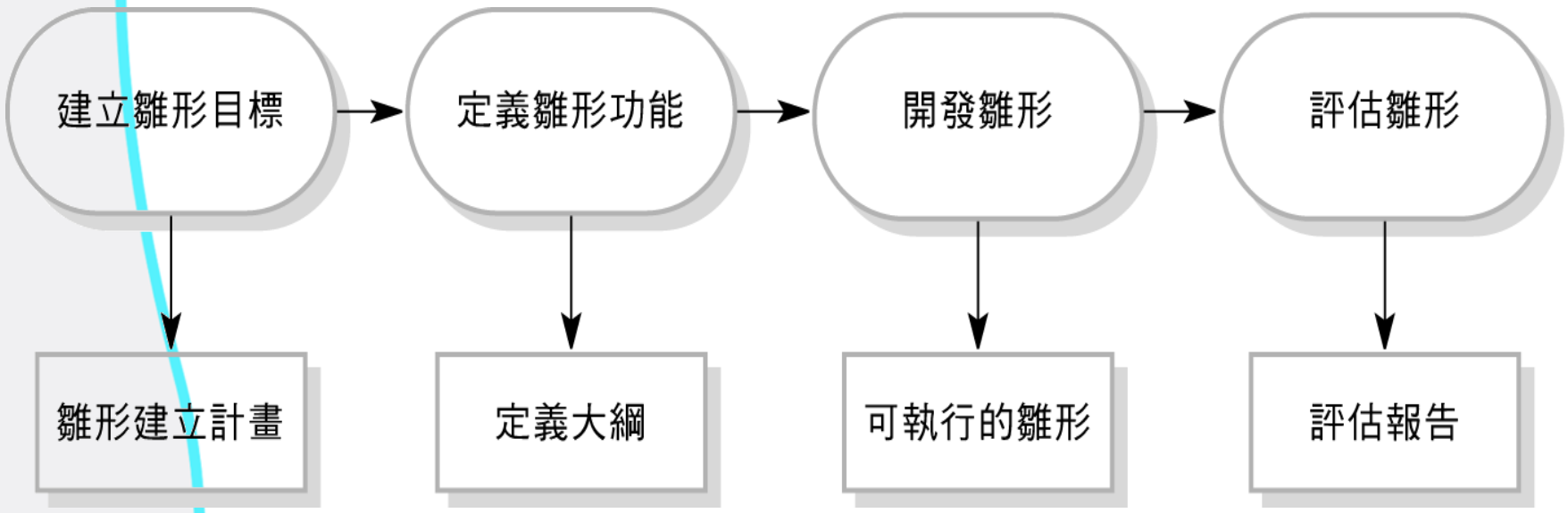
背對背測試



使用雛形法好處

- 改善系統的易用性
- 系統可以更接近使用者的需求
- 改善設計的品質
- 改善易維護性
- 降低開發工作量

雛形開發的程序



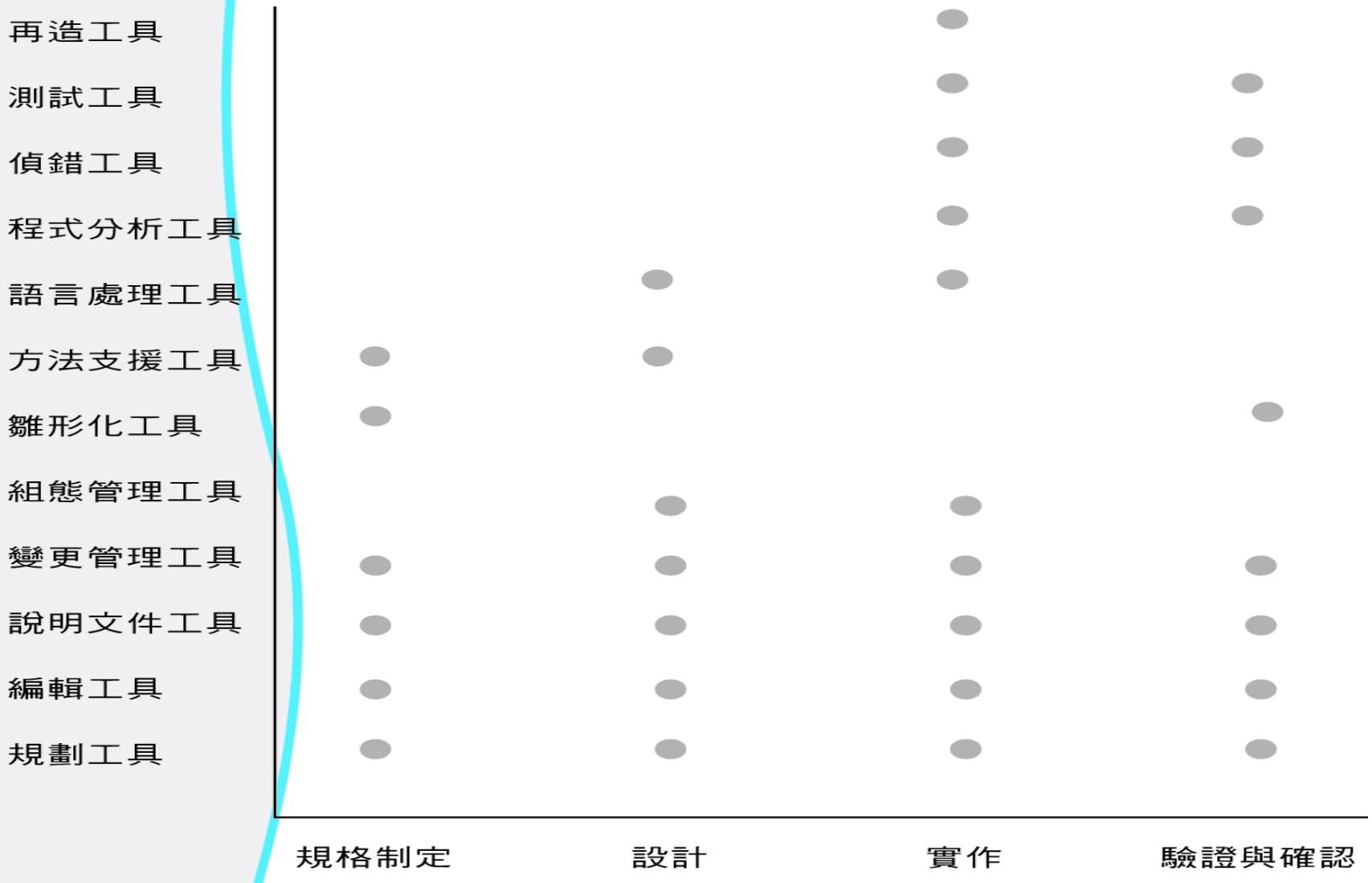
電腦輔助軟體工程(CASE)

- 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。

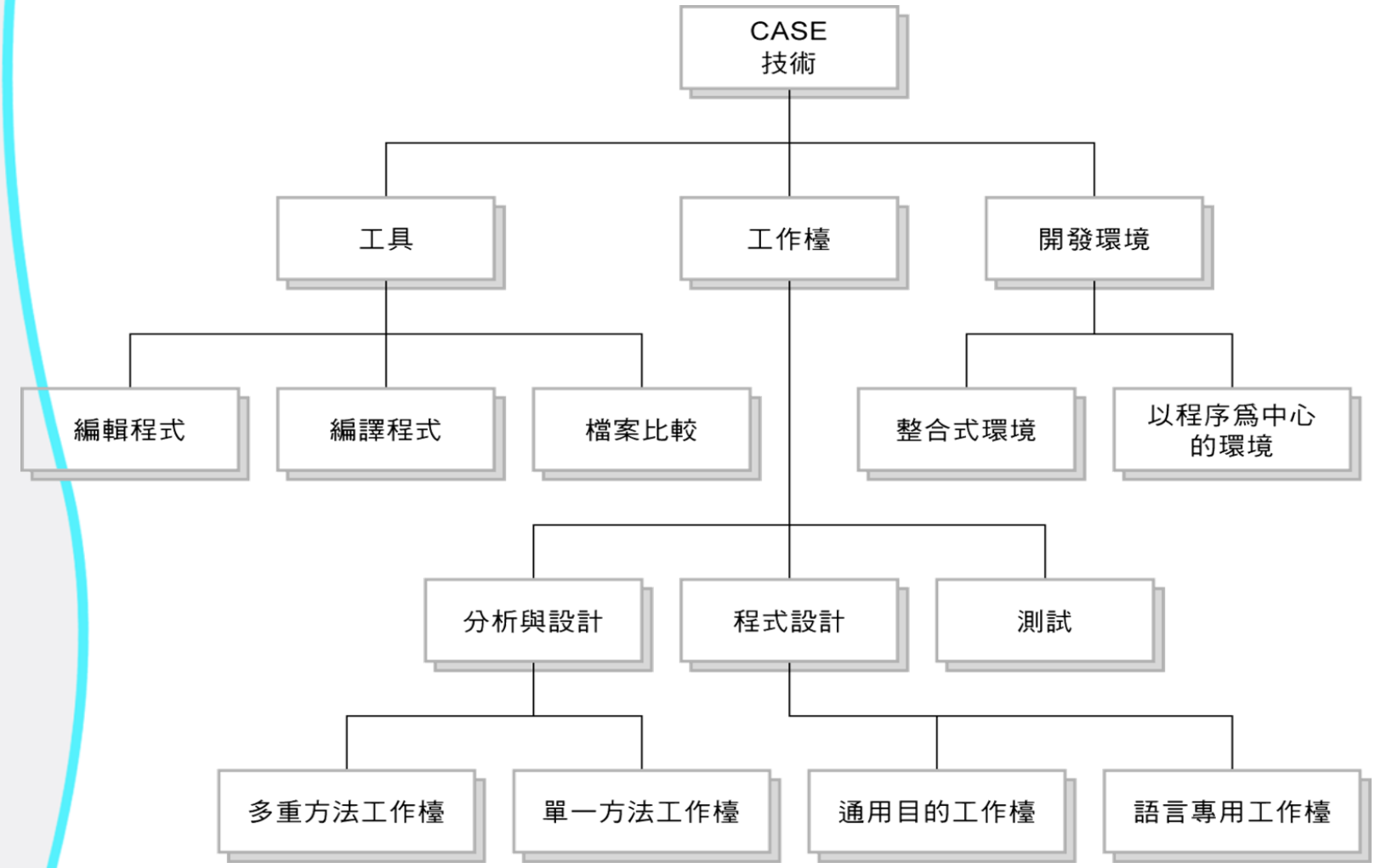
CASE分類

- 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