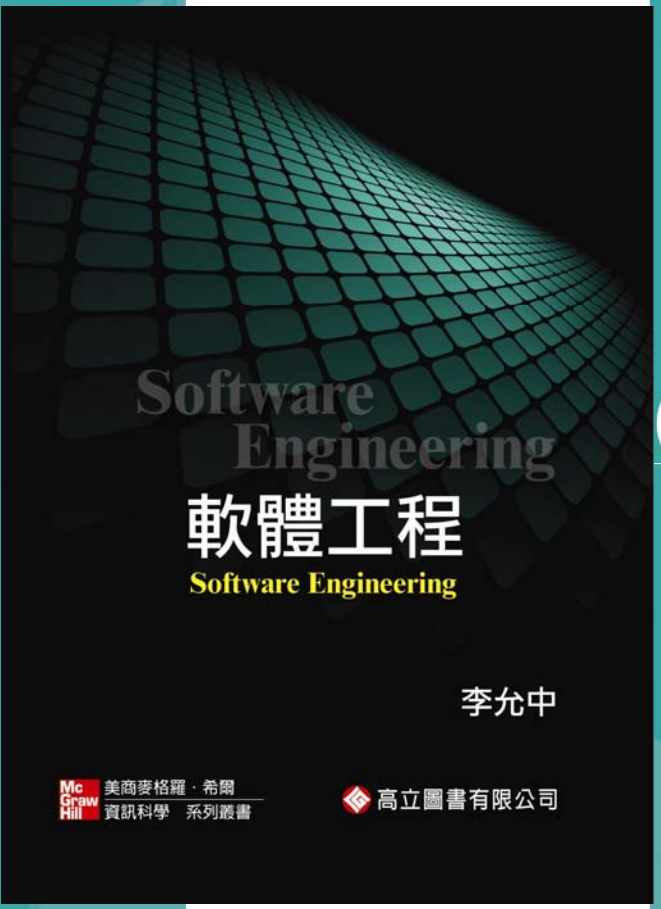


CHAPTER 6
軟體測試

6

大綱

- 軟體測試的基本觀念
- 軟體測試的實務作業程序
  - 測試規劃、準備、執行與監控
- 軟體測試計畫書、測試案例、測試記錄與報告的撰寫
- 軟體測試靜態分析方法的應用
- 軟體動態測試的測試案例設計之技術
  - 黑箱與白箱測試
- 軟體動態測試的類別
  - 單元、整合、系統與驗收測試

軟體測試的基本觀念

- 軟體測試是項複雜且龐大的工程，更是軟體產品執行品質控管的重要關鍵。
- 正確地規劃軟體測試計畫，可以協助軟體專案在開發與維護過程中，有效地執行軟體測試工作，提高軟體測試效益。
- 廣義的軟體測試，包括軟體品質的分析與檢驗，即所謂的**驗證與確認(Verification and Validation, V&V)**。

驗證與確認 (1)

- Barry Boehm 對「驗證」與「確認」的描述。
- 驗證：「我們是否正確地建構產品？」(Are we building the product right？)
  - 用正確的方法與步驟建構產品，例如需求分析規格是否正確與嚴謹的描述、規格設計是否遵循需求分析規格，以及是否採取適當的產品發展技術完成規格設計的開發等。
- 確認：「我們是否建構正確的產品？」(Are we building the right product？)
  - 建構的軟體系統與產品是否滿足使用者真正的需求，例如軟體系統所表現出的功能是否符合於使用者的期望與運作環境的限制，主要包含功能性需求測試與非功能性需求測試等。

驗證與確認 (2)

- ISO/IEC 12207:1995標準對「驗證」與「確認」的描述。

- 驗證：檢查軟體產品並提出客觀證據，以證實達成訂定的規格要求。
  - 判斷某發展階段的軟體產品，達成前項發展階段訂定的需求或限制。

- 確認：檢查軟體產品並提出客觀證據，以證實達成某一特定預期功用的需求
  - 確定依據需求規格製作的最終軟體產品，是否滿足特定使用目的。

驗證與確認 (3)

- CMMI(Capability Maturity Model Integration)的第3等級成熟度對「驗證」與「確認」的描述。
- 驗證的目的，在確保工作產品(Work Product)符合其指定的規格。
- 確認的目的，在展示最終軟體產品或產品元件在需求環境中，實現客戶使用需要。
- 每一個發展階段結束時，實施驗證以確保此階段的產品與其規格一致，而在發展過程結束，實施確認以確保每一個工作產品都和使用者的使用需求一致。
- CMMI的工作產品:
  - 專案計畫書、軟體需求文件、設計文件、測試文件、程式碼等等。

驗證與確認範例說明

- 需求規格書
  - 需求分析階段的工作產品，也可能是交付給客戶的最終產品之一 。
  - 輸入規格：合約工作條款、RFP(Request for Proposal)或客戶需求訪談紀錄。
  - 成為以下工作產品的輸入規格：軟體初步設計階段、軟體測試計畫書、建置管理(Configuration Management)計畫書。
  - 驗證工作：由利害關係者(Stakeholders)檢視合約書、RFP是否都與需求規格書一致，各項目是否都正確。
  - 確認的工作：需求規格書架構是否合乎標準、慣例、文字是否可讀、敘述是否明確而不會被誤解、是否可以測試以符合使用者使用要求等。

驗證與確認之策略原則

- 需執行**靜態分析**與**動態測試**。
- 動態測試計畫需包括測試案例設計、測試執行以及結果資料的收集與評估。
- 各種測試方式的測試數據必須可量度，以便能分析與改善測試方法，並及早發現潛在的測試作業程序問題。
- 測試技術應隨不同的軟體與測試階段而調整。
- 為避免可能產生的盲點，各階段可由不同人員執行及採用不同測試方法。

靜態分析

- 不直接執行軟體系統，而是以人工或自動化方法評估軟體開發各階段的工作產品，檢查是否滿足所制訂的需求規格，並嘗試找出錯誤與及早修正。

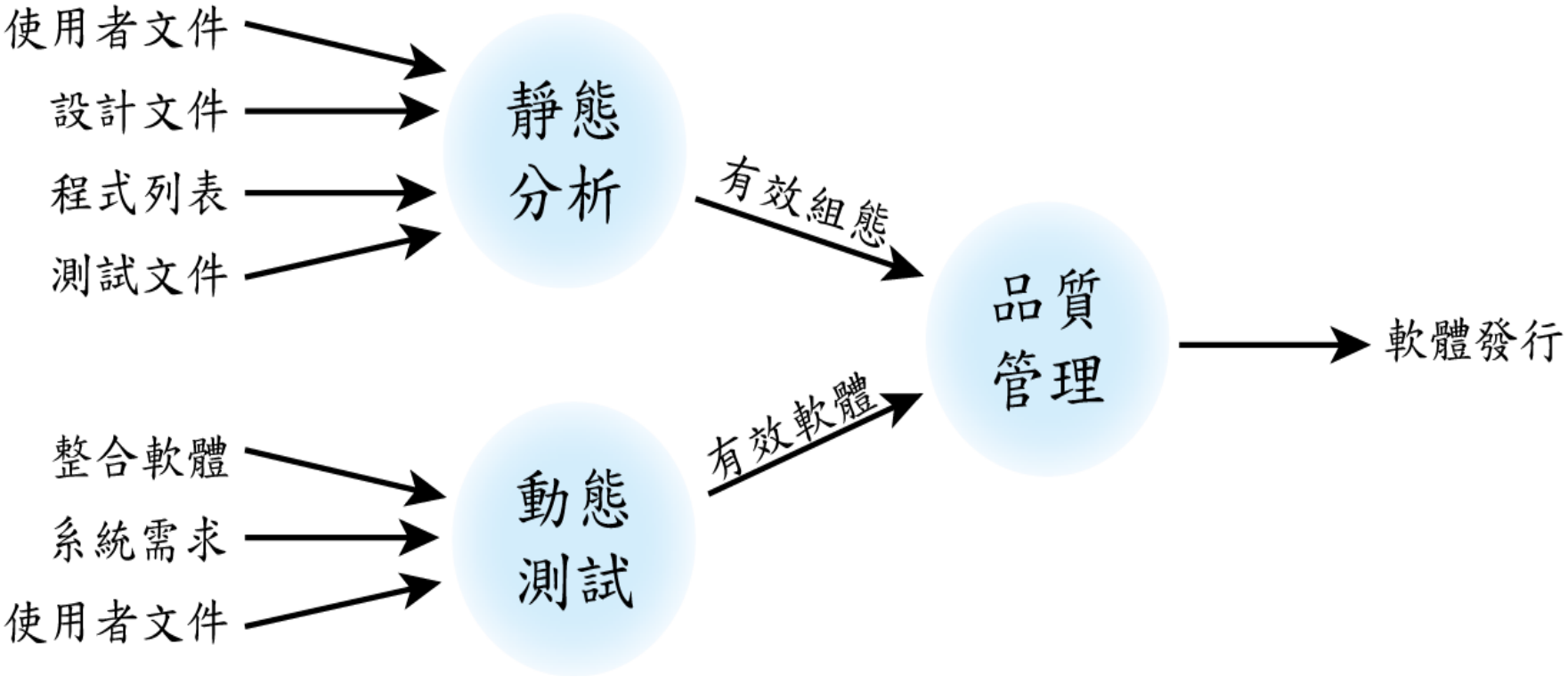
- 執行分析的方法包含檢視(Inspection)、結構化逐步審查(Structured Walkthrough)和主動審查(Active Review)等。

- 分析的工作產品可以包含建構(Configuration)計畫、需求規格、設計規格、程式碼及測試計畫書等軟體工作產品建構(Configuration)。

動態測試

- 程式撰寫完成後才能進行動態測試，執行軟體程式碼。
- 動態測試計畫需規劃測試案例(Test Case)的設計，測試的執行以及結果資料的收集和評估。
- 一般使用黑箱(Black Box)測試與白箱(White Box)測試這兩種技術來設計測試案例。
- 軟體開發過程中，透過單元測試(Unit Testing)、整合測試(Integration Testing)、系統測試(System Testing)與驗收測試(Acceptance Testing)等各種軟體測試階段，來檢查軟體介面、功能、與非功能需求，以及選擇性地確保軟體內部程式敘述執行路徑與條件判斷的正確性。

驗證與確認程序


使用者文件
設計文件
程式列表
測試文件
靜態分析
整合軟體
系統需求
使用者文件
動態測試
有效組態
有效軟體
品質管理
軟體發行

軟體測試的基礎

- 軟體測試是種用來檢驗軟體品質優劣的程序與方法，而軟體品質的衡量則包括正確性、完整性與安全性等面向。

- ISO 9126標準所制訂的軟體品質模型，共定義六個軟體品質特性，包括了功能性(Functionality)、可靠性(Reliability)、可使用性(Usability)、效率(Efficiency)、可維護性(Maintainability)和可移植性(Portability)。

- 軟體測試是以系統化的方法檢查軟體系統，發現潛藏的錯誤，以期提升軟體的品質。

軟體動態測試的問題1

```
int scale(int m, int n, int p) {
  double j=0.0 ;
  for (i=0 ; i<n ; i++) {
    if (m>n) {
      j=j-1;   // 正確應為j=j+1;
      j=j/p;
    }
  }
  return j;
}
```

如何針對上述程式碼片段進行完整的軟體動態測試?

軟體動態測試的問題2

- 在電腦中使用32位元，若要完全測試所有m, n, p的輸入組合，則必須測試$2^{32} \times 2^{32} \times 2^{32}$次（約$10^{29}$次）。
  - 假如執行1組測試數據需耗費10^{-5}秒，1年共有3.1536×10^{7}秒，則測試所有輸入組合需花費10^{16}年的時間

- 每次迴圈都有2個執行路徑（if敘述），若要完全測試程式中所有執行的路徑，需要測試$2^{n} + 1$次（變數i由0～n）。
  - 同上測試效能，假設$n = 50$，則總共需測試$2^{50} + 1$次（約10^{15}次），約需花費10^{2}年的時間完成。

軟體動態測試的主要目的

- 為了能夠**在有限的時間內，以最有效率的方式發現最多的軟體錯誤**，而發現1個或多個尚未發現的錯誤，都可算是一次成功的測試結果。

- 測試案例設計的好壞決定整個動態測試程序的成效。
  - 越早發現錯誤(缺失)，修正的成本越低。

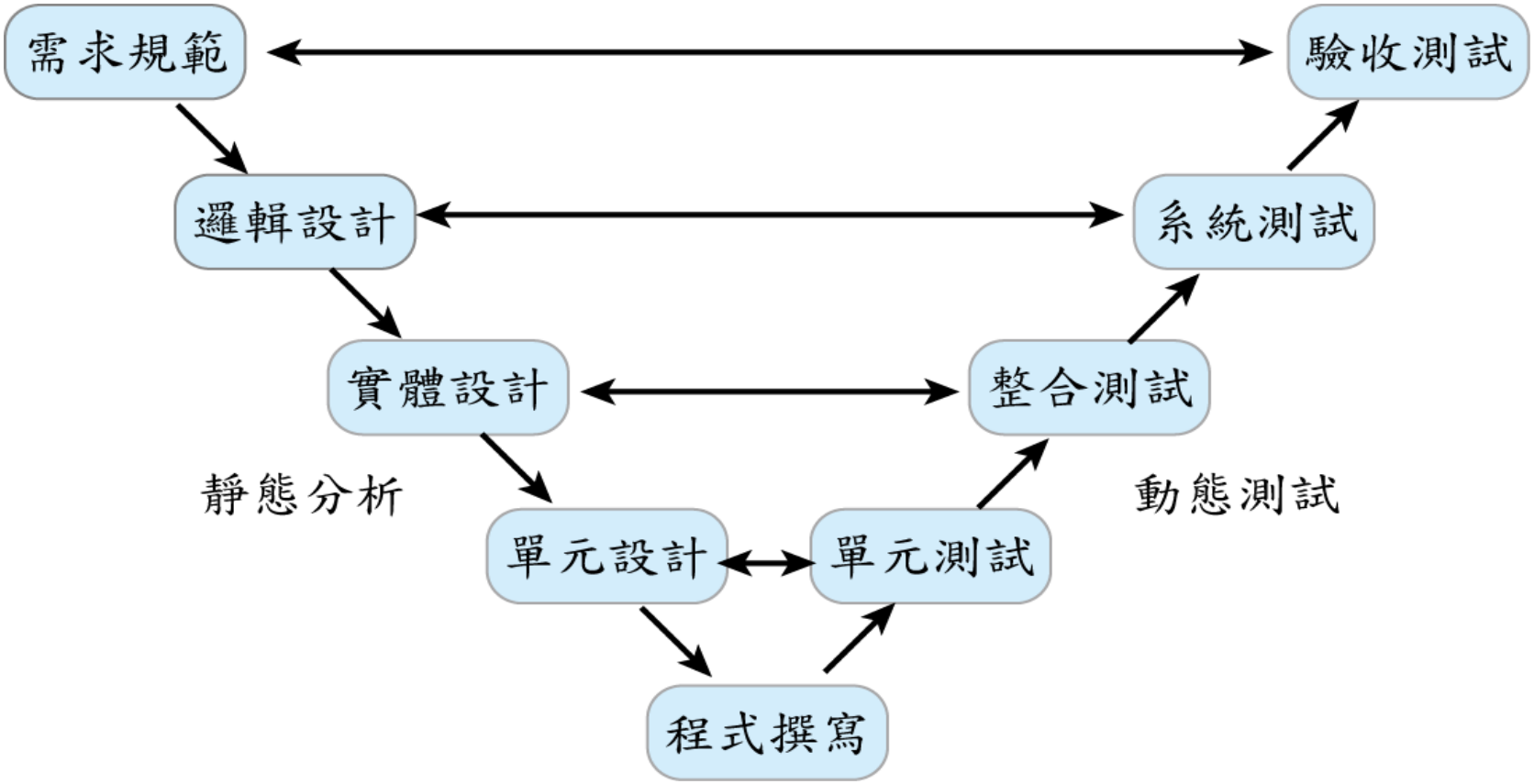
- 測試中只發現少量的錯誤，其可能表示的意義有以下兩種可能：
  - 1. 表示此軟體系統的品質已相當可靠及穩定。
  - 2. 測試工作準備不夠充分或執行不夠確實，這是產品將來可能產生危機的警訊。

缺失的產生比例與排除的相對代價

軟體發展階段	平均缺失的產生比例%	平均缺失排除的相對代價
需求規格	**15%**	**1**
設計	**35%**	**2.5**
單元編碼	30%	6.5
整合編碼	10%	16
文件撰寫	10%	40
系統測試	__	40
系統運作	__	110

註：有超過一半在動態測試時所發現的軟體缺失（錯誤），其來源在於需求規格或設計錯誤。

軟體測試V模型



註：左半部表示軟體開發生命週期的各階段，右半部表示動態測試程序的各階段。

軟體動態測試程序

測試方向與工作產品

軟體動態測試的準則

- 實際進行的測試次數將依照個別軟體系統的**特性與需要**進行規劃。
- 各**測試階段**可由不同人員執行測試及採用不同測試方法，以避免可能產生的盲點。
- 每個階段的測試時機各不相同，應視時程安排及人力考量做適度的調整。一般而言，軟體測試所花費的人力，占整個軟體發展工作至少30%以上的時間以及成本。
- 除了單元測試的工作外，最好整個測試工作能大部分交由**專門且獨立的測試人員**主導進行，尤其以具有程式設計經驗的人員為佳。
- 測試工程師能根據測試案例的執行結果，提供**完整的軟體問題報告**，軟體開發人員才可能更快的修復軟體錯誤。

軟體測試規劃

- 軟體測試工作也需要**完整詳細的測試規劃**，而且要愈早進行規劃愈好，最好在著手軟體專案規劃時就開始進行。

- 測試規劃需要考慮配合整個軟體專案的開發時程，以及移交產品給客戶的時限，而規劃的產出為**測試計畫書**。

- 測試計畫書是整個測試工作的基礎，如果在測試進行中發現任何問題，必須隨時檢視測試計畫書，必要時予以修正，避免測試計畫書已無法滿足軟體系統的開發時程需要，或是無法針對軟體系統的需求或設計變更進行測試案例的修正。

軟體測試步驟

- 軟體測試程序中的任一軟體測試階段，一般都可以分成以下四個主要的執行步驟：
  - (1)**測試規劃**：測試前應擬定測試計畫及測試目標；
  - (2)**測試準備**：準備所有參考資料及測試工具與環境；
  - (3)**執行軟體測試**：執行實際測試與記錄錯誤與缺失；
  - (4)**評估與控制**：監控與檢討測試成效。

「測試規劃」步驟

- **對受測軟體進行研究**：包括企劃書、計畫執行書、軟體需求規格書，甚至軟體原始程式碼等資料，以了解測試目標。
- **確定測試工作目標**：測試目標應視軟體重要性而有所不同。例如，證券公司的客戶股票下單系統，其測試目標可能為「即時顯示目前股價」與「安全而正確執行股票下單」。
- **設定停止測試條件**：可依據計畫的時程、測試案例完成率、使用的測試方法、發現錯誤的數量，或以是否開始出現測試生產力大量降低的情況等，做為停止測試條件。
- **規劃工作項目**：包括配置測試環境的資源、規劃與安排測試時程、制訂測試人力的資格、訓練測試人員、準備使用的測試輔助工具、確認建構管理工作與開發人員在開發及除錯時所可能延遲的時間等。
- **完成測試計畫規格書**：計畫書內容應包括整個測試活動的相關資訊，例如，測試目的、測試方法、測試時程、人力與工作分配、測試案例與測試項目。

「測試準備」步驟

- **依測試計畫書進行準備**：包括測試人員的相關訓練、測試需要的軟硬體設備、文件資料的準備、測試輔助工具的準備以及教育訓練工作。

- **完成測試案例設計**：測試設計文件中應詳細說明測試步驟與測試資料，並有預期結果與實際測試結果之比較。

- **撰寫測試程式或採購測試工具**：依據測試的要求，開發額外的測試程式以便完成測試步驟；或是採購現有的測試輔助工具以進行測試。

6

「執行軟體測試」步驟-1

- **測試輸入**：包括兩種組態，一種是軟體組態，例如，軟體需求規格、設計規格和原始碼等；另一種為測試組態，例如，測試計畫和過程、工具、測試案例、測試資料和預期結果等。

- **進行測試取得測試結果，並比較測試結果與預期結果**。預期結果與實際結果若存在萬分之一的差異，都可能需花費相當長時間進行診斷和修正，不可以輕易的忽略。

- **找到錯誤或缺失，並輸入追蹤系統，交由開發人員除錯**。若已於下一版本修正問題，則測試工程師需再次使用同一測試案例測試此問題，直到確認問題已修正完畢為止。

- **收集測試結果與進行評估，以獲得軟體品質的定性表示**。必要時，可以利用歷史資料(Historical Data)，建構軟體可靠性模型，並使用模型中錯誤率資料來預測將來可能發生的錯誤數量。

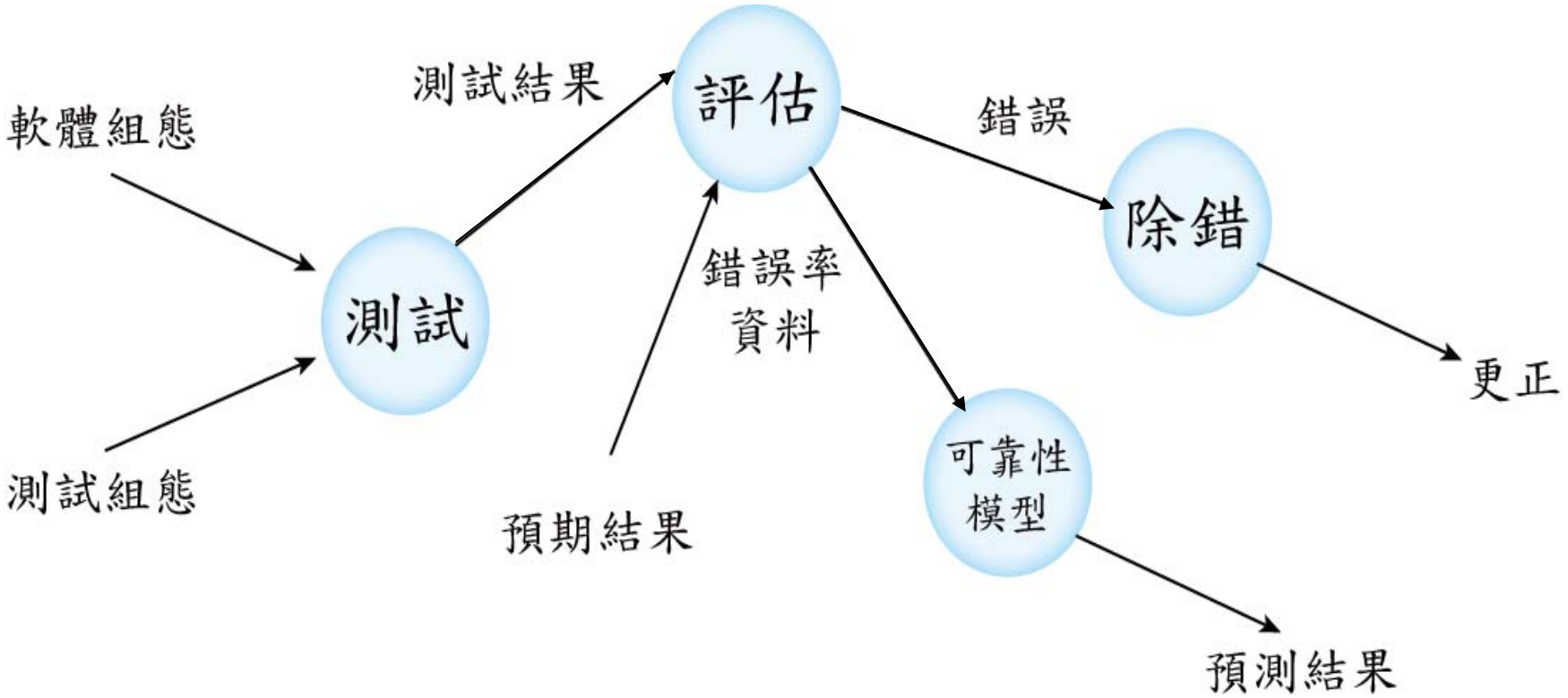
「執行軟體測試」步驟-2

- 每階段進行完畢後，應檢視執行結果與測試計畫或測試設計文件的**差異**，若存在著顯著的差異，即應針對差異部分做適度調整，例如，修改測試設計文件內容，包括測試目標與停止測試條件等，或調整測試計畫書的時程安排。

- **規劃軟體測試的標準原則**：先測核心功能，再測輔助功能；先測功能，再測性能；先測常見情況，再測異常情況；先測影響層面大的問題，再測影響層面小的問題；先測必須測試的部分，再測可選擇或未要求測試的部分；先測變更過的部分，再測沒有變更的部分。

- **執行軟體測試的指導原則**：測試環境應獨立於發展環境，以免團隊間的互相干擾，並可完整地確認與模擬軟體執行環境；隨時彙總測試結果、統計測試進度及產出測試資料；定期召開測試工作會議，進行測試經驗的記錄與交流，並提出可能的改善建議；若有程式模組修正缺失，應執行迴歸測試(Regression Testing)的工作，用以確認修正舊問題後不會再次引發新問題。

「執行軟體測試」的主要程序



「評估與控制」步驟

- **工作控制**：包括工作追蹤(Tracking)、監督(Monitoring)、工作分配列表、及工作里程碑的設定與狀況報告。

- **測試完成後評估工作的檢討**：包括評估測試工作的涵蓋度、評估測試方法的有效性以及評估測試人員的工作情況。

軟體測試計畫書

- 測試工作必須配合計畫的開發時程，因此透過撰寫測試計畫書，才能以有系統化的方法準備與進行測試工作。

- 撰寫測試計畫書的主要目的為：
  - 訂定要採用的測試策略，與預估測試工作的大小，讓軟體測試進行更加順利。
  - 讓參與的測試人員彼此間的溝通與分工更加順利，以達成測試目標。
  - 以系統化的方式進行軟體測試，便於管理。精確產生測試回饋(Feedback)、覆蓋率(Coverage Rate)、測試的深度等測試數據，並遵守測試時程。

測試計畫書的種類

- 根據軟體產品的複雜度與大小，可以撰寫較簡單扼要的測試計畫書，或是內容複雜的測試計畫書。

- **單一文件測試計畫書(Single Test Plan, STP)**：將測試工作中所有的議題，都撰寫在同一份文件。集中所有軟體測試的時程、變數資料、測試案例，以方便管理掌控。因此，此種測試計畫書內容較複雜，包含各種類的測試工作，牽涉的工作產品也較廣泛，比較適合較小型的軟體系統，或是人數較少的開發團隊使用。
- **詳細運作測試計畫書(Detail Test Plan, DTP)**：依軟體開發的進行階段，針對各階段的工作產品特色與重要工作進度，分別撰寫詳細的測試工作內容。由於將測試階段分開處理，因此適合大型或子系統模組多且複雜的軟體系統，或是對系統功能的測試要求較嚴謹的專案使用。
- **主要方針測試計畫書(Master Test Plan, MTP)**：將測試分為不同進行階段，規劃每個階段概略的測試方針，並經由不同的測試方法達成測試目標。因此，一份MTP文件常伴隨（參考）好幾個DTP文件使用，適合大型軟體開發。

IEEE 829測試計畫書的撰寫規範

- 測試計畫書的大綱中應包含以下項目：
  - 測試計畫書的名稱代碼(Test Plan Identifier)。
  - 基本介紹(Introduction)。
  - 測試項目(Test Items)。
  - 需要測試的產品功能(Features to Be Tested)。
  - 不需測試的產品功能(Features Not to Be Tested)。
  - 採用的測試方法(Approach)。
  - 通過／未通過測試的準則(Item Pass/Fail Criteria)。
  - 停止測試條件與重新測試的規範(Suspension Criteria and Resumption Requirements)。
  - 測試交付項目(Test Deliverables)。
  - 測試工作項目(Testing Tasks)。
  - 測試環境的設定(Environmental Needs)。
  - 人員工作分配(Responsibilities)。
  - 人員能力與所需訓練(Staffing and Training Needs)。
  - 測試時程(Schedule)。
  - 潛在風險與防治措施(Risks and Contingencies)。
  - 文件核可(Approvals)。

軟體的靜態分析

- 軟體的工作產品，除了一般人所熟悉的軟體系統外，還包含各式各樣不在軟體交付清單中的內部工作產品(Work Product)，例如，專案計畫書、系統規格設計書、細部設計文件等，可視為**軟體的靜態資訊**。

- 不同軟體開發階段所產生的工作產品，包含專案企劃書、專案計畫書、建構管理計畫書、契約書、需求規格書、設計規格書、軟體雛形、程式原始碼列表、測試計畫書（驗收測試計畫書、系統測試計畫書、整合測試計畫書、單元測試計畫書、測試紀錄、錯誤報告）、建構紀錄以及使用者文件等。

軟體的靜態分析的好處

- 軟體的靜態分析對於提升軟體的品質有相當大的助益，其優點如下：
  - **降低專案整體成本**：有超過一半的軟體測試錯誤造因於需求規格或設計錯誤，透過審查與檢視等過程，可以在程式撰寫前及早發現錯誤。
  - **協助經驗傳承與訓練溝通**：在審查與檢視過程中，透過小組討論，可以有效分享成員觀念與想法。
  - **及早驗證軟體品質**：藉由審查與檢視，可清楚發現錯誤所在及原因，不需經過繁複鑑定錯誤、分析原因、修改與重測，就可迅速修正錯誤。

- 各軟體開發階段對於產出的工作產品與其應注意的審查重點也各有不同 。

軟體開發階段的審查重點 -1

- **需求規格階段**：著重於系統功能與效能描述的審查，目標是在初步設計階段之前找出需求描述有問題的部分，包括模糊不清、不適當、無法測試、模稜兩可以及不正確的需求等，而審查的標的可以是全部或部分關鍵的需求規格敘述。

- **設計階段**：著重的部分可視設計的層次有所不同，典型的軟體設計層次有架構設計、細部設計、介面設計、資料庫設計或演算法設計等。此階段的審查目的在於發現設計描述上較為不足的部分，包括架構概念錯誤、模組切割錯誤或是與需求規格不符之處、甚或過度設計以至影響系統效能，或檢查產品之設計規格是否確實考量相關領域等重要問題。另外，關鍵性設計審查則著重於細部設計的正式查核。

軟體開發階段的審查重點 -2

- **程式撰寫階段**：著重於細部邏輯設計或程式實作，目的是確認細部設計符合需求規格或程式碼符合設計規格，審查的項目可以是虛擬碼、細部設計圖或原始碼列表。

- **測試階段**：不同測試階段，包括模組單元測試、整合測試、系統測試及驗收測試等，皆有不同的審查清單，其目標為確認每階段的測試可以覆蓋使用者需求與設計項目，以找出缺失、錯誤或其他忽略的測試案例，或與客戶達成驗收條件等。審查標的則可以為測試計畫書、V&V計畫、測試案例、測試資料、測試程式原始碼等。

靜態分析的方法 -1

- 靜態分析方法有許多種類，且各個文獻所使用的術語也略有差異，主要分為**團隊開會審查**以及**個人單獨審查**兩種形式。

- CMMI的驗證流程領域中提到的**同儕審查(Peer Review)**，是由產品作者的同儕，透過審查的方式，找出產品缺失並建議改善的方法，主要包括檢視(Inspection)、結構化逐步審查(Structured Walkthrough)和主動審查(Active Review)等常用的審查方式。

- PSP (Personal Software Process) 對於個人軟體程序則提出檢視、逐步審查(Walk-Through)和個人審查(Personal Review)三種審查方法。

靜態分析的方法 -2

- Karl Wiegers在軟體的同儕審查一書中，依據審查嚴謹度，由高到低整理出各種方法，依序為：
  - (1)檢視；
  - (2)團隊審查(Team Review)；
  - (3)逐步審查；
  - (4)配對程式開發(Pair Programming)；
  - (5)同儕桌上檢查(Peer Desk Check)和傳遞循環檢查(Pass Around)；
  - (6)特定問題審查(Ad hoc Review)。

審查的方法說明 -1

- **檢視**：是種最正規結構化(Formal Structure)的團隊審查程序，包括計畫、準備、會議、更正、追蹤驗證等五個階段。每位參與者都有固定的角色，並依據清楚的規格與檢查清單(Check List)事前單獨審查，之後再以會議方式共同討論找出問題，最後指定修正解決人員，並產生報告。

- **團隊審查**：即結構化逐步審查，是種輕量化檢視(Inspection-Lite)方法，要求必須有計畫、有結構的進行審查，但此檢視方法不正式與不嚴謹。此外，部分的檢視程序容易遭簡化或省略，檢視方法定義的角色也可能被合併。

- **逐步審查**：是種非正式(Informal)的審查方法，不需要遵循既定的程序，一般由產品開發者主導進行，開發者逐步說明軟體產品如何執行典型的應用案例，審查者適時的發問以追蹤產品設計，找出並記錄可能錯誤，並指定錯誤處理負責人員，不需要產生管理報告和度量(Metrics)數據。

審查的方法說明 -2

- **配對程式(Pair Programming)開發**：此即敏捷開發方法(Agile Methodology)的極限程式開發(Extreme Programming)，每個開發工作都由兩位開發者一起進行。這種方法實質上比較像是軟體發展策略，而非審查方法。

- **同儕桌上檢查**：由開發人員的同儕以自己的方法，或根據缺失檢查清單，檢查軟體產品以找出其缺失，產生缺失清單給開發者，或直接在工作產品上註記錯誤所在。

- **傳遞循環檢查**：即循環檢查，也就是多人依照順序執行的同儕桌上檢查。首位審查者找出缺失並加以記錄，接著整個資料夾傳給下一位審查者，並加入其個人所找出的缺失，如此一直循環，直到所有審查者執行完畢。最後，再將資料夾回傳到原始作者。

審查的方法說明 -3

- **特定問題審查**：開發人員要求同儕協助追蹤某個難以理解的問題，透過另一個觀點的思考，常常可以很快的解決開發者思考很久的問題。

- **個人審查**：產品開發者依照檢查清單審查自己的產品，目標是在程式第一次編譯和測試之前找出錯誤並完成修復。

不同審查方式與對應的活動

審查方式／活動	計畫	準備	會議	更正	追蹤驗證
檢視	有	有	有	有	有
團隊審查	有	有	有	有	沒有
逐步審查	有	沒有	有	有	沒有
配對程式開發	有	沒有	不斷進行	有	有
同儕桌上檢查	沒有	有	可能有	有	沒有
傳遞循環檢查	沒有	有	可能有	有	沒有
特定問題審查	沒有	沒有	有	有	沒有

檢視、逐步審查、傳遞循環檢查特性的比較

特性	檢視	逐步審查	傳遞循環檢查
審查者充分準備	有	有	有
開發者報告產品內容	沒有	有	沒有
讀者報告產品內容	有	沒有	沒有
使用檢查清單	有	沒有	可能有
歸類所發現的缺失	有	沒有	沒有
開立問題清單	有	有	有

6

檢視、團隊審查、逐步審查的會議內容特性的比較

特性	檢視	團隊審查	逐步審查
主導者	協調者	協調者或作者	作者
材料報告	報告者	協調者	作者
材料報告的大小	問題	每一頁或每一章節	
紀錄	有	有	可能有
遵循制訂的程序	有	可能有	可能有
規範參與者角色	有	有	沒有
缺失清單的使用	有	有	沒有
資料收集與分析	有	可能有	沒有
產品認證決策	有	有	沒有

6

審查會議 -1

- 對於重要的工作產品，除了個人的審查外，都會進行**正式的審查會議**，以確定每位審查人員的審查紀錄是否有誤，若有，可由作者或是與會的成員共同討論，以確認每一個找出的問題，並指派相關人員進行修正。

- 在進行審查之前，必須注意以下幾項原則：
  - **審查標的是軟體開發程序或工作產品**，而非審查開發者個人。
  - 此為**錯誤偵測而非問題解決**會議。
  - 給批評或給建議很容易，但是真正執行修正是困難的。
  - 不要陷入辯論與反駁的口頭之爭。
  - 慎選審查人員，並施以**審查訓練**，內容包含資訊技術與評論技巧。

審查會議 -2

- 在進行審查之前，必須注意以下幾項原則：(接續)
  - 尊重每位參與者發表意見的權利與內涵智慧。
  - 信賴每位成員的責任感。
  - 針對每項工作產品製作相對的**檢查清單**。
  - 給審查者應有的審查工作時間與資源。
  - 限制參與人員數目或資格。
  - 設定議程時間，並在預定議程時間內完成會議。
  - **事前做充分準備**。
  - **記錄會議中所有決議**。

審查會議角色與工作事項 -1

- **報告者(Reader)或作者(Author)**
  - 與專案經理或專案領導者組織審查會議。
  - 與專案經理或專案領導者、審查者決定審查時程。
  - 設定審查目標。
  - 準備審查資料。
  - 與審查者確認審查結果以及後續處理動作。
  - 執行後續處理動作。

- **協調者(Moderator)**
  - 指定審查者、審查主委。
  - 協調會議，讓會議依照議程順利進行。

審查會議角色與工作事項 -2

- **記錄者(Recorder)**：記錄會議中所有的建議與結論、後續處理動作與負責人員，並在會後將整理完畢的紀錄分送給與會者。
- **審查者(Reviewer)**：貢獻必要的時間與精力，並根據檢查清單，審查工作產品，並試著找出問題。
- **軟體品質保證人員（若有必要時）**：檢驗流程是否滿足品質管理標準程序。
- **使用者或客戶代表（若有必要時）**：檢驗審查目標是否滿足使用者需求。

審查會議各角色工作指導規則

- 盡可能扮演好所指定的角色。
- **會議之前**準備好所有的相關材料。
- 保持正確的態度，與他人合作且圓融。
- 準時參與會議，直到會議結束才離開。
- **處理後續的待處理項目(Action Item)並參與後續會議。**
- **服從審查協調者與團隊的意見。**

審查程序

- 一個完整的審查程序可以包括以下的階段：
  - (1)計畫階段(Planning Stage)；
  - (2)準備階段(Preparation Stage)；
  - (3)執行審查(Conducting Review)；
  - (4)處理與追蹤(Rework & Follow up)。

「計畫階段」工作事項

- 選擇**審查團隊**，平均3～4人，以不超過8人為原則，並分派角色。
- **訓練**審查團隊所需要的技巧。
- 排定並**公告審查時間**。
- **準備審查素材**，包括規格書、工作產品、審查清單。審查項目及資料不要太多也不要太少。
- **分發**審查資料給相關人員。

「準備階段」工作事項

- 審查者利用時間審查資料，記錄問題或錯誤。
  - 記錄於初版的**審查報告**中。
- 準備會議場所與器材。
- 報告者（審查資料的作者）準備好報告資料。應注意所附參考資料是否正確？規格文件輸入、輸出是否一致？品質是否遵循相關標準？

審查報告內容

- 報告撰寫日期、審查會議日期。
- 審查團隊姓名(Team Name)。
- 審查產品名稱(Product Name)。
- 審查項目(Review Items)。
- 審查目標(Objectives)。
- 待處理項目(Action Item)。
- **發現問題的種類**：缺失(Defects)、遺漏(Omissions)、或是互相衝突(Contradictions)。
- **改善的建議**：針對每個待處理項目，記錄負責人、處理動作、問題嚴重程度(Severity：High/Medium/Low)、錯誤發生來源、目前解決狀態(Status : Open/Closed)。
- **問題嚴重程度(Severity)**：可分為高(High)，表示1個或多個系統特性無法使用、違反需求或嚴重影響使用者。例如，系統一啟動就當機，系統接收到訊息就當機；或是中等(Medium)，表示使用者使用時會很不方便、不愉快，但不會影響系統的功能，例如，系統能夠手動建置，但無法使用檔案自動建置等；亦或是低(Low)，表示少許表面錯誤，例如，訊息拼字錯誤、標頭文字錯誤。

審查報告內容－範例

專案名稱	停車場收費系統	日期	2007/12/25
作者	王英明	審查者編號	1
產出項目	交易模組	審查時間	1.5小時
審查者	陳文華	錯誤摘要	高＝1個 中等＝1個 低＝1個
審查目標	檢查交易規則		

編號	位置 (Page/Line)	說明 (Comments)	錯誤嚴重性 (H/M/L)	錯誤來源 (Source)	狀態已處理 (Y/N)
1	Page 1, Line 70	請在此加上PP參考	L	交易模組	N
2	Page 6, Line 30	產品規格特性描述不合標準	M	交易模組	N
3	Page 8, Line 25	此處使用者手冊未說明	H	交易模組	N

「執行審查」工作事項

- 審查會議約為15分鐘至2小時，視審查項目而定，平均約為1小時。
- 所有參與者必須牢記，此為**問題確認會議**，非為問題解決會議。
- 讀者可以是產品作者或其他人，逐句說明審查項目資料。
- 協調者確實根據議程時間主持討論，確認審查者提出的問題，是否為問題或錯誤，並且不要使會議成為「問題解決」會議。
- 記錄者記錄觀察資料(Observations)，包括**錯誤、問題和審查意見**。
- 決定問題處理方式或計畫，確認負責處理人員，最後討論決定此問題後續需要重新調查研究或重做、或是只修護有問題的部分。
- **確認問題的種類與嚴重程度**，決定此產品是否符合品質要求、或者是接受但需要小部分修改，或是需要大幅度的修改（須重新審查）、或取消此產品，重做一次。

「處理與追蹤」工作事項

- 記錄者**分送審查報告**。
- 負責人員解決被分配到的**待處理項目(Action Item)**。
- 若審查結果為僅需小部分修改即可有條件接受，所有待處理項目必須由審查者再次確認問題是否被解決；**若審查結果為必須大幅度修改，則必須準備召開另一次審查會議。**
- 審查者追蹤、判斷並更新待處理項目的處理狀態，若有未處理或未解決的待處理項目，必須適時**提醒負責人員**。

6

審查評估

- **審查度量(Review Metrics)**是透過量化方法，以統計方式應用在大量審查資料上所獲得的指標性數據，組織或團體即可根據這些數據進行分析，以了解成效或改善程序。

- 審查度量中，可以包含的基本量化指標如下：
  - 審查項目的大小(Size)。
  - 審查花費的時間。
  - 缺失發現的數量。
  - 缺失未發現的數量，表示在之後的再次審查、或測試、或使用者使用時出現的缺失數量。
  - 每小時審查發現缺失的數量。
  - 每頁或每行(LOC)發現缺失的數量。

進階審查度量指標 -1

- **錯誤偵測效率(Error Detection Efficiency, EDE)**
  - 計算公式：EDE = DR/AD，其中DR(Defects of Review)為審查紀錄上，特定型態（如：需求、設計或程式碼階段）的錯誤偵測數量；AD(All of Defects)是使用所有測試方法找出的錯誤數量。
  - EDE是一種用來衡量審查程序發現的錯誤效率所使用的度量。當此值愈高時，代表大部分的錯誤都可以在審查的過程中被發現並及時的修正。

- **成本效用(Cost Effectiveness, CE)**
  - 計算公式：CE = CR/AC，其中CR(Cost of Review)是由審查者找出缺失的成本；AC(All of Cost)是此缺失若留到後面修復、測試與除錯所花的成本。
  - 若CE值偏高，則表示此階段所進行的審查在成本上已經偏高，可以把審查的人力再往前一個階段加重。

進階審查度量指標 -2

- **缺失密度(Defect Density, DD)**
  - 計算公式：DD = QD/S，其中QD(Quantity of Defect)是審查報告或缺失報告中缺失的數量；S(Size)是預估程式的大小。
  - DD可以用來預估表示每單位審查標的中（程式可用行數，文件類可用頁數）可能的缺失數量，以衡量之後所產生的審查報告中所發現的缺失數量是否合理。

- **缺失移除比率(Defect Removal Leverage, DRL)**
  - 計算公式：$DRL_{phase\_x} = DH_{phase\_x} / DH_{unit\_test}$，其中$DH_{phase_x}$是階段x每小時缺失移除數量；$DH_{unit_test}$是單元測試階段每小時缺失移除數量。
  - 藉由不同階段所求出的DH_{phase_x}，組織可以發現何種階段的審查效果最大，或針對審查不足的階段，增將審查的工作量，或是將人力移往效益最大的階段進行審查。

缺失移除比率範例

開發階段 (Phase)	每小時缺失移除數量 (DH)	缺失移除比率 (DRL)
範例：25 C++程式碼		
設計審查(Design Review)	3.91	3.91/1.39 = 2.8
程式碼審查(Code Review)	5.01	5.10/1.39 = 3.6
編譯(Compile)	9.43	9.43/1.39 = 6.8
單元測試(Unit Test)	1.39	1.39/1.39 = 1.0
範例：36 Pascal程式碼		
設計審查(Design Review)	3.12	3.12/1.31 =2.4
程式碼審查(Code Review)	3.15	3.15/1.31 = 2.4
編譯(Compile)	7.99	7.99/1.31 = 6.1
單元測試(Unit Test)	1.31	1.31/1.31 = 1.0

軟體動態測試方法

- 軟體的動態測試對象為**實際的軟體系統或其組成元件**，因此必須等待軟體實作開發完成，有了實際的程式碼之後才能執行。

- 任何最終交付給使用者的軟體系統，都必須經過此階段的測試，以確保重大的錯誤或缺失皆已更新，避免交付之後所造成的系統更新成本與違約所造成的賠償。

- 動態測試的主要關鍵在於能夠設計出有效的測試輸入，即測試案例(Test Case)，來對可執行的軟體程式輸入測試資料，將執行後的結果跟預期的結果相比對，以找出程式的缺失。

測試案例設計方法簡介

- 一個測試工作是由許多的測試案例的執行所構成，而測試案例中將記錄所有相關的測試資訊，以輔助測試人員進行測試，找出未發現的錯誤。
- 完整的測試案例可以包含編號、名稱、目標、條件、輸入資料、測試步驟、期望結果等資訊。
- 在軟體系統發展初期就開始設計測試案例(只要有辦法收集到足夠的測試資訊)，並視需要開始進行測試，並非一定要等到所有的軟體元件都開發完畢才開始進行測試案例的設計。
- 設計測試案例的原則為：
  - 1. 能**具有合理的機率偵測到軟體錯誤**，以避免完成所有的測試，卻無法提升軟體品質，浪費執行成本。
  - 2. **不重複出現相同的測試案例**，以避免過多無用的測試案例，無法發現新問題，且亦浪費測試執行與案例管理的成本。 。

白箱測試

- 是種根據程式碼產生測試案例與測試資料的技術，也是種強迫檢視程式碼的做法，但成本比較高。

- 根據虛擬碼或原始程式碼的**可能執行路徑**來設計測試案例，可以發現65%的模組測試階段錯誤，能夠有效地測試程式細節，對程式碼的控制結構相關錯誤較有效。

- 白箱測試方法對以下錯誤不完全有效：
  - (1)完全錯誤或遺漏的功能；
  - (2)介面的錯誤；
  - (3)資料結構與範圍的錯誤；
  - (4)起始值設定的錯誤。

白箱測試原則

- 白箱測試根據程式碼的內部構造，測試是否依照設計規格正確運行，其測試原則為：
  - 1.保證模組中每條獨立路徑在執行時至少都通過一次。
  - 2.所有包含邏輯判斷的語句都要確實執行。
  - 3.所有迴圈的邊際條件及內部操作都得檢驗。
  - 4.檢驗所有內部資料結構的正確性。

白箱測試的主要技術

- **基本路徑測試**：最基本的測試方式，藉由找出所有的獨立路徑，用以確認程式碼中每一行敘述都被測試案例執行過，可以發現簡單的程式敘述撰寫錯誤。

- **邏輯條件測試**：透過檢測程式碼中每個邏輯條件的組合，來設計可能的測試案例資料，確保不會遺漏某個特定的邏輯條件組合，從而發現邏輯條件運算組合上潛在的錯誤。

- **資料流測試**：主要是從程式碼中資料變數的使用上去測試測試案例資料，由資料的定義 (Define)與使用(Use)的追蹤，來發現資料使用上的錯誤。

- **迴圈測試**：主要是針對程式碼片段中若有迴圈的設計時，如何設計測試案例資料，以正確的測試迴圈的敘述，發現可能的問題。

黑箱測試

- 將測試的元件模組或系統，當作一個黑色的箱子，只依照產品表現在外的功能效果，測試是否正確地達成任務，因而著重於功能與性能的需求、介面、初始與結束正確性的測試。

- 黑箱測試的目的在檢驗輸入正確資料時是否會產生正確結果，而依據使用者需求規格設計測試案例的黑箱測試，可以了解程式是否滿足使用者需求。

黑箱測試的優缺點

- 黑箱測試的優點為：
  - 1. 測試案例設計成本低。
  - 2. 從規格及使用者角度考慮問題，對程式外部功能有很好的效果。
  - 3. 不容易受到程式影響而產生偏見。

- 黑箱測試的缺點為：
  - 1. 不容易測到程式細節，需要長時間而大量的測試案例以確保穩定性。
  - 2. 若設計的測試案例太少則測試的效果較差。若設計大量測試案例或以不同方法設計測試案例則容易造成重複設計與浪費測試時間。

黑箱測試技術的案例設計問題

- 1. 功能的有效性如何測試？
- 2. 系統是否對某種輸入值特別敏感？
- 3. 資料型態的邊界值如何切割出來？
- 4. 系統能容納多大的資料量？
- 5. 資料的某種合併對系統操作產生什麼後果？
- 6. 什麼類型的測試案例輸入較好？

黑箱測試的主要技術

- **等價分類法(Equivalence Partition)**：將輸入資料分成幾個等級，然後分別加以測試。

- **邊界值分析法(Boundary Value Analysis)**：查看在極大值或極小值周圍的執行情形是否有錯，因為許多錯誤常發生在輸入或輸出資料條件的邊界值。但此方法只適合數值資料的分析使用。

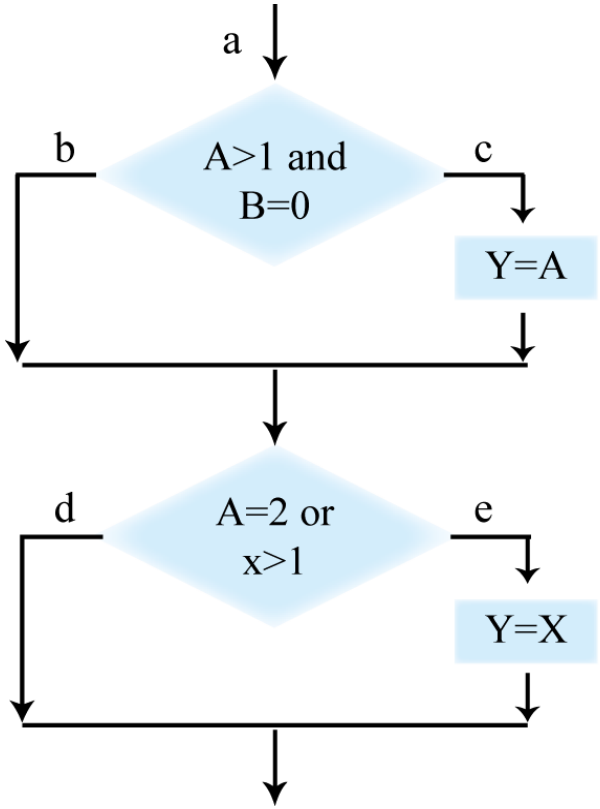
- **因果圖(Cause-Effect Graph)**：當需求規格有比較複雜的邏輯說明時，適合使用此種方法。

6 測試覆蓋性

- 透過「**覆蓋率**」**(Coverage Rate)** 的分析，可以進一步了解所使用的測試方法或是所設計的測試案例，對於所要測試的標的能測試的範圍、程度到何種等級。
  - 測試案例對**需求的測試覆蓋率**，就可以用測試案例所測試的需求個數占全部需求個數的比例，來了解測試案例所涵蓋的範圍。
  - 程式碼的測試，則可能分析如**程式敘述的覆蓋率**等。

- 白箱測試針對程式碼的測試，有以下不同的覆蓋性準則(Coverage Criteria) 可以利用:
  - 敘述覆蓋(Statements Coverage）
  - 分支決策覆蓋(Branches/Decision Coverage)
  - 條件覆蓋（Condition Coverage)
  - 多重條件組合覆蓋(Multiple-Condition Combination Coverage)

覆蓋率計算說明範例




```
10 INPUT A,B,X
20 IF (A>1) AND (B=0) THEN Y=A
30 IF (A=2) OR (X>1) THEN Y=X
40 PRINT Y
```

註：以下覆蓋率的計算說明，將使用上述範例進行說明。

6

敘述覆蓋

- **敘述覆蓋 (Statements Coverage）：**測試個案要能使程式的**每個敘述**至少都執行一次。

- 計算說明: (參考前述範例)
  - 變數輸入值(A,B,X)若為(2,0,3)時，就可覆蓋10, 20, 30, 40行所有可執行指令，結果會得到3。
  - 若把行號20的AND改成OR，或是行號30的X>1改成X>0，亦或是行號20後面的Y=A改成其他的敘述，程式執行結果仍然沒變，所以找不出這一種程式撰寫上的邏輯錯誤。
  - 敘述覆蓋是最弱的邏輯涵蓋準則，因此，白箱測試至少要做到此類測試。

分支決策覆蓋

- **分支決策覆蓋(Branches/Decision Coverage)**：測試案例要使程式的**每個決策點**至少都執行一次。

- 計算說明: (參考前述範例)
  - 輸入值(A, B, X)若為(3,0,3)和(3,1,1)，能使得(A>1) AND (B=0)和(A=2) OR (X>1)這兩個布林運算式均能產生真值和假值。
  - 若將程式碼(A>1)誤寫成(A>2)，這組測試案例無法測出其邏輯錯誤。
  - 決策覆蓋測試方法的嚴密性是比敘述覆蓋高。

條件覆蓋

- **條件覆蓋（Condition Coverage）**：和分支決策覆蓋類似，不過條件覆蓋是以**各別條件**為主，而分支決策覆蓋則以整個判斷所構成的布林運算式為主。條件覆蓋必須使程式中所有邏輯判斷情況都至少執行過一次。

- 計算說明: (參考前述範例)
  - (A>1) AND (B=0) 運算式包含A>1和B=0兩個條件。
  - 前述程式具有下列四個條件A>1、B=0、A=2、X>1，如果要這四個條件都能產生真值與假值，測試案例就必須包含(1)A>1；(2)A<=1；(3)B=0；(4)B<>0；(5)A=2；(6)A<>2；(7)X>1；(8)X<=1。
  - 輸入值為(2,0,3)可以滿足(1)、(3)、(5)、(7)的情況，輸入值為(1,1,1)時，可以滿足(2)、(4)、(6)、(8)的情況，合併這兩個測試案例便可達成設定的目標。

決策／條件覆蓋

- 條件覆蓋嚴密性似乎比分支決策覆蓋高，但並非絕對。
  - 輸入值為(1,0,3)和(2,1,1)這組測試案例，雖然可以使上述四個條件均產生真值與假值，但並未使運算式(A>1) AND (B=0)和(A=2) OR (X>1)具有真值與假值。

- **決策／條件覆蓋(Decision/Condition Coverage)**：需設計足夠的測試案例，使**判斷敘述的布林運算式中的每個條件的所有可能值**都至少執行一次，同時每個判斷敘述的**所有可能判斷結果**都至少執行一次。
  - 例如，輸入值(2,0,3)和(1,1,1)即可達成此目標。

多重條件組合覆蓋

- **多重條件組合覆蓋(Multiple-Condition Combination Coverage)**：也就是**不同組合的判斷情況**都至少執行一次。

- 從決策／條件涵蓋方法延伸，目標是使測試數據涵蓋**每個布林運算式中各種條件組合**。

- 計算說明: (參考前述範例)
  - 第一個布林運算式有下列四種條件組合：(1)A>1，B=0；(2)A>1，B<>0；(3)A<=1，B=0；(4)A<=1，B<>0。
  - 第二個布林運算式也有下列四種條件組合：(5)A=2，X>1；(6)A=2，X<=1；(7)A<>2，X>1；(8)A<>2，X<=1。
  - 測試數據(2,0,4)滿足(1)與(5)的組合，(2,1,1)則滿足(2)與(6)的組合，(1,0,2)則滿足(3)與(7)的組合，和(1,1,1)滿足(4)與(8)的組合，透過此四組測試數據便可涵蓋上述八種條件組合。

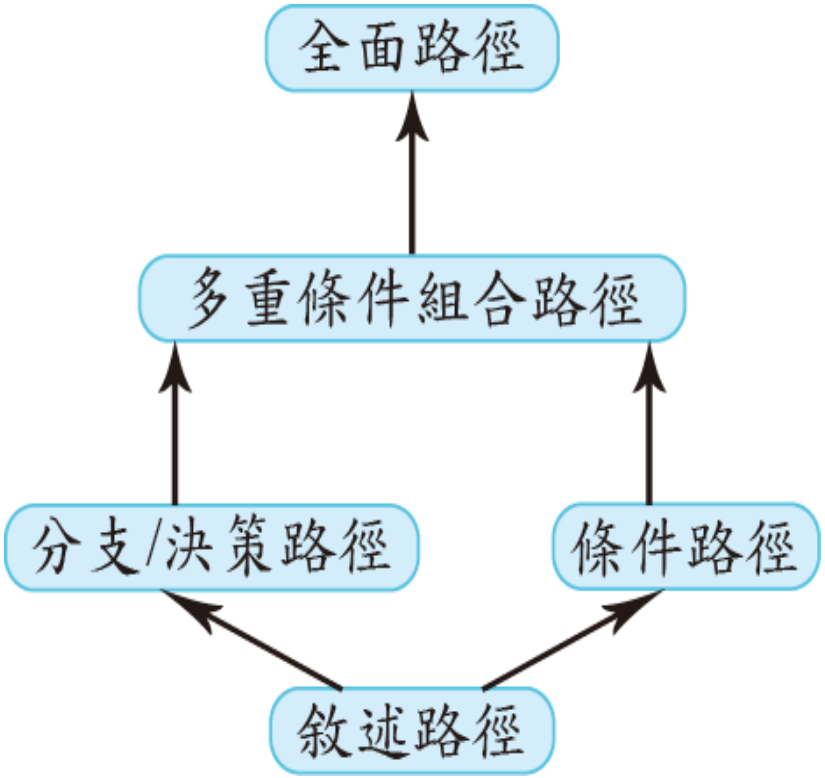
全面路徑覆蓋

- **全面路徑覆蓋（All-Paths Coverage）**：其目標為使測試案例執行**所有程式路徑**，每種路徑皆須有一種測試案例，因此又可稱為窮舉測試。
- 這種測試方法相當嚴密，但所需時間與成本也相當高，實用性不大。
- 除多重條件涵蓋的各種條件組合外，各判斷運算式間也必須加以組合，一個測試案例只測試某種組合。
- 計算說明: (參考前述範例)
  - 兩個布林運算式，每一個布林運算式都有四種條件組合，因此測試路徑（測試數據）有$4 \times 4 = 16$種。
  - 當有100個運算式時，其測試路徑至少便達到$2^{100} = 1.27 \times 10^{30}$種，因此不實用性可見一般。

測試覆蓋性層次關係

- 白箱測試中的五種測試覆蓋性層次高低如右圖顯示，由下而上表示覆蓋性由低而高。

- 黑箱測試的覆蓋準則，因為其應用的方式可以針對非程式碼的部分，因此除了窮舉測試外，還有使用者功能測試覆蓋、效能測試覆蓋、使用案例測試覆蓋、循序圖測試覆蓋及狀態圖設計覆蓋等。



測試覆蓋性層次關係

測試案例的設計方法說明

- 白箱測試常用測試案例設計方法:
  - 基本路徑測試
  - 邏輯條件測試
  - 資料流測試
  - 迴圈測試

- 黑箱測試常用測試案例設計方法:
  - 等價劃分方法
  - 邊界值分析
  - 因果圖方法

基本路徑測試

- **基本路徑測試(Basic Path Testing)**主要分析程式碼的**執行邏輯**，找出所有可能執行路徑的**基本路徑**，設計測試案例以執行測試。

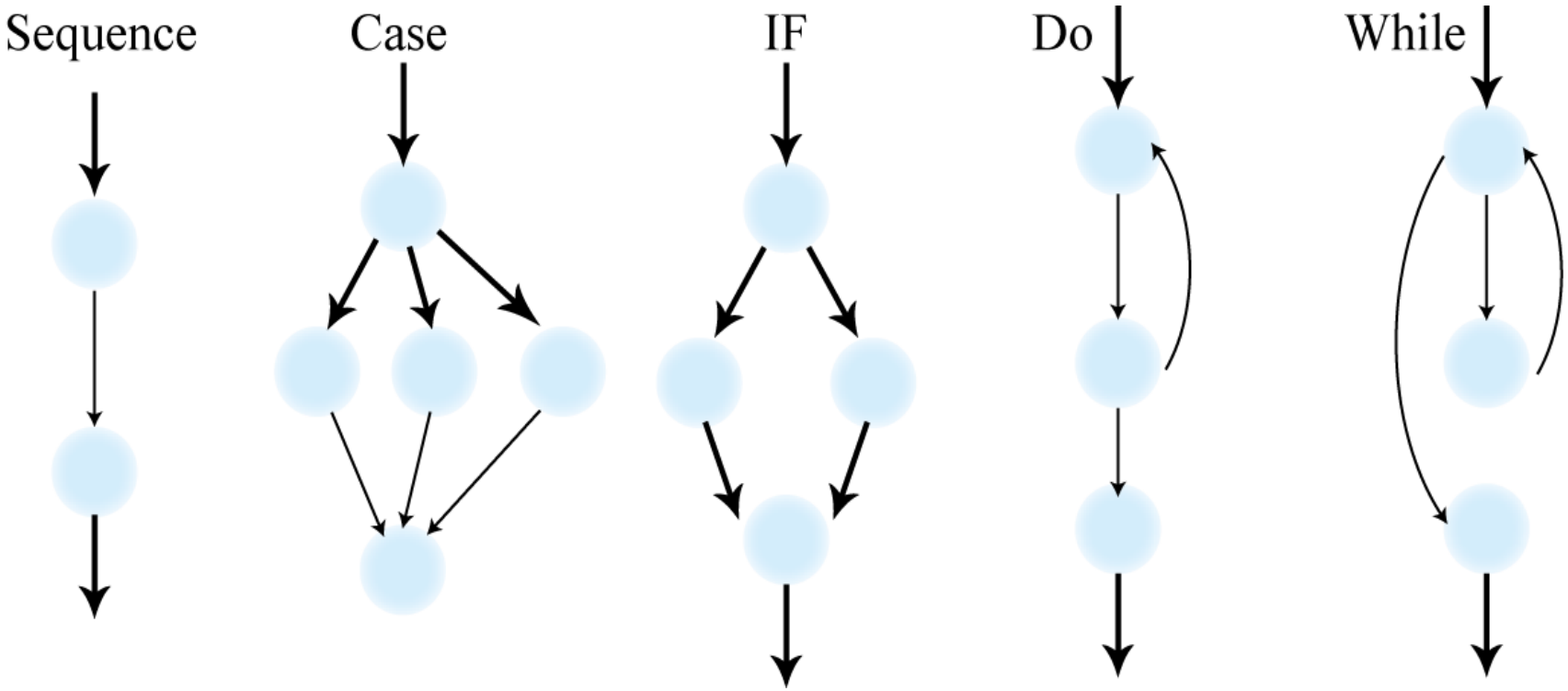
- 將測試次數從無數多次降低為基本路徑次數，如此可以保證程式內每條敘述均至少執行過一次。

- 如何由程式碼轉換成流程圖，進而推算基本路徑則是本測試的主要程序。

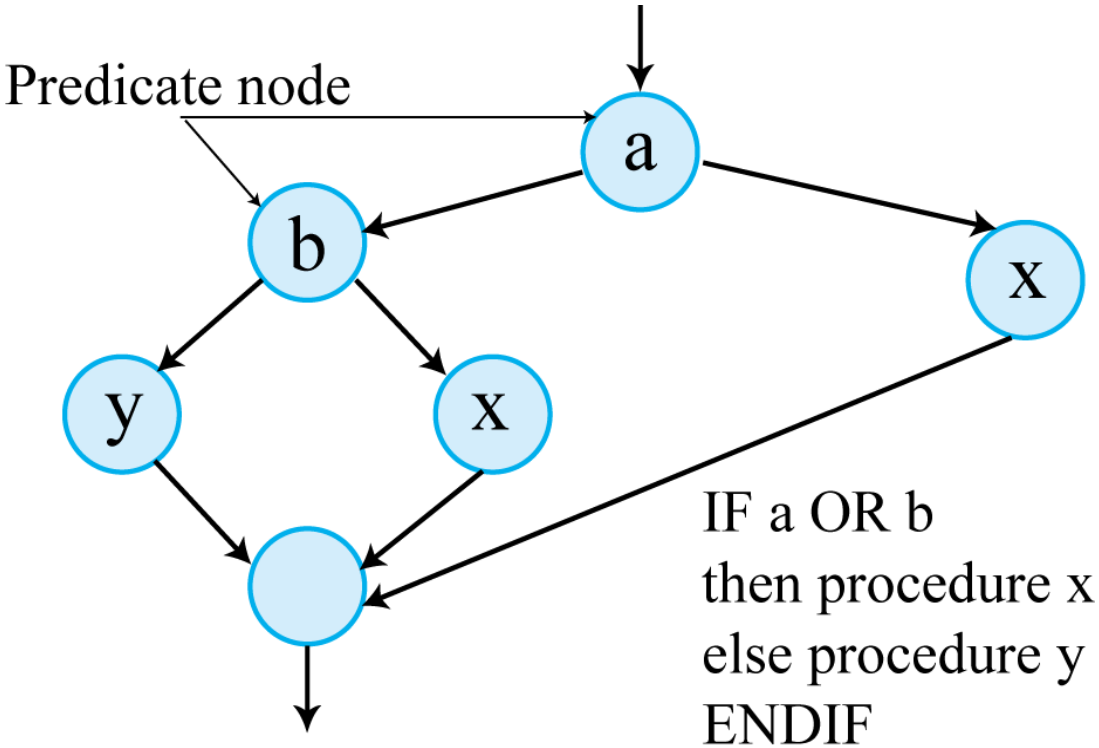
基本路徑的測試程序 -1

- **1. 根據程式碼畫出流程圖(Flow Chart)：**
  - 圓表示流程圖**節點(Node)**，可以為一個或多個程式敘述。
  - 單一條件的節點稱為**判斷節點**，發出兩條以上的邊(Edge)。
  - 循序的節點和判斷節點亦可組成單一節點。
  - 箭頭表示邊，為控制流程，邊結束於節點，即使此節點不表示任何敘述（如：if then else）。
  - 程式敘述一般標有數字，以提供流程圖的繪製，任何程式都能轉換成流程圖，以G表示。
  - 若程式有複合條件（一個或多個布林運算子），則需要分別為條件a和b創建單獨的節點。

流程圖符號



複合條件之流程圖形表示



基本路徑的測試程序 -2

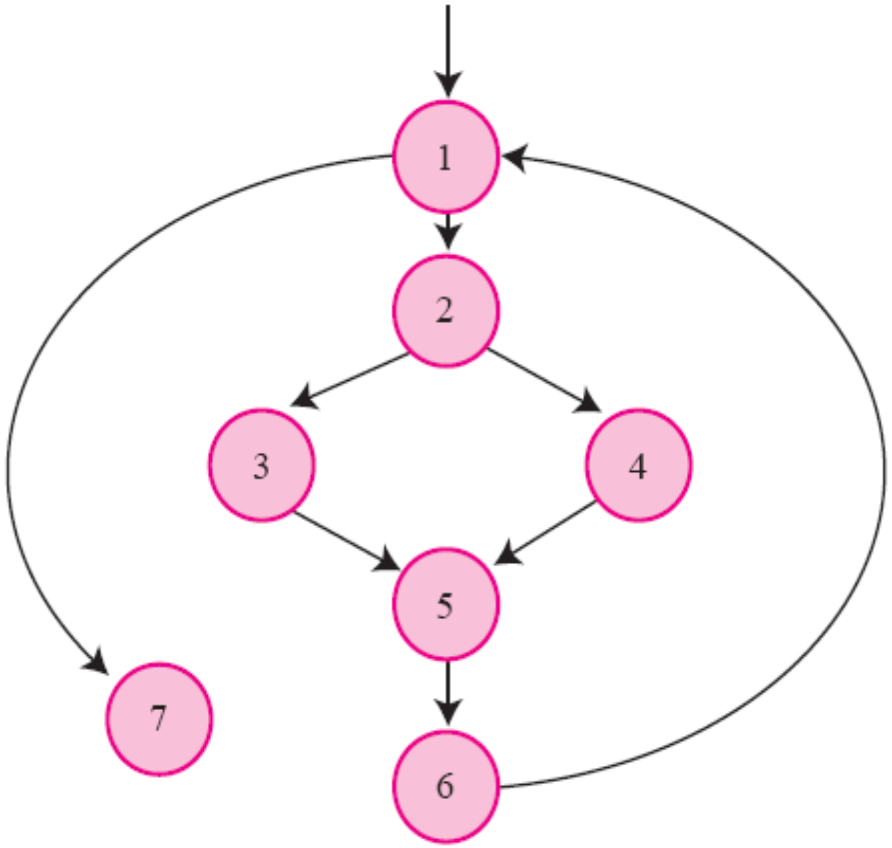
- **2. 根據流程圖G計算迴圈複雜度：**
  - 程式的邏輯複雜度V(G)又稱為迴圈複雜性(Cyclomatic Complexity)，即基本路徑個數的上限。
  - 邊及節點所形成的區域稱為域(Region)，計算域的個數即為迴圈複雜度，計算域時要加上圖形的外部區域。
  - 公式V(G) = E − N + 2，E為邊的數量，N為節點數量；或公式V(G) = P + 1，P是流程圖G中判斷節點的數量，都可以算出迴圈複雜度。

基本路徑的測試程序 -3

- **3. 找出所有基本路徑：**
  - 基本路徑又稱為獨立路徑，係指程式從開始到結束，至少含有一組新的指令敘述或一個新的條件判斷指令的路徑。
  - 以流程圖的角度來看，獨立路徑應至少含有一段未曾走過的路徑。
- **4. 針對每個基本路徑設計測試案例，進行測試。**

流程圖形基本路徑表示

路徑1-7是一條新的路徑；路徑1-2-3-5-6-1-7，則是路徑1-7加入新的一段路徑1-2-3-5-6-1後所產生；路徑1-2-4-5-6-7，加入新的一段路徑2-4-5；以上三條都是獨立路徑。而路徑1-2-3-5-6-1-2-4-5-6-1-7則不是一條獨立路徑，因為只是前述路徑的組合。所有可能的執行路徑都可以由前述三條獨立路徑組合而成。



基本路徑測試範例 -1

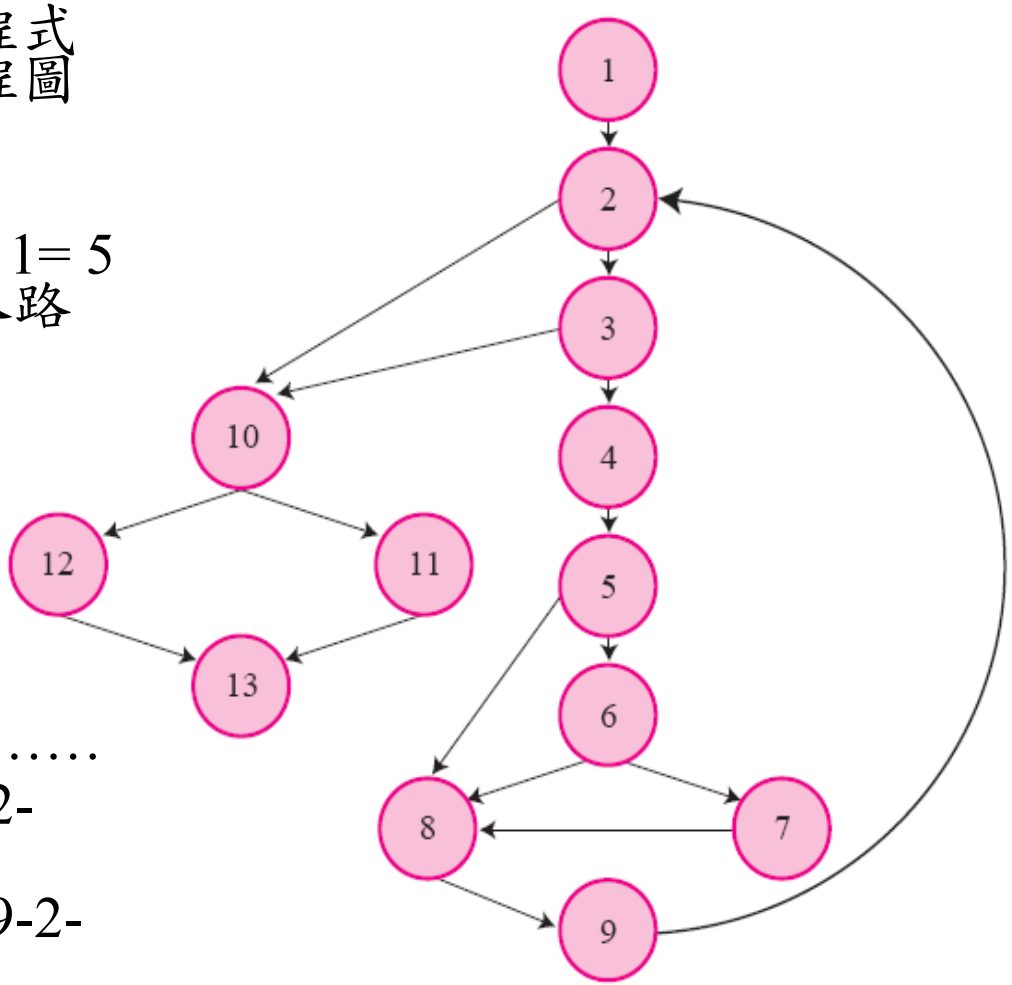
- 輸入100個以下的數字，數字限定在一定範圍內，計算所有數字的總和(sum)、有效輸入個數(valid)，以及平均(average)。而輸入−999表示停止輸入。若valid = 0，則average = −999。

程式碼範例

```
        Interface return average, input, sum, valid
        Interface accept value, minimum, maximum
        Type value[1:100] is Integer Array
        Type average, total, input, valid, minimum, maximum, sum, i is Integer
1.      i =1 ; input = 0; valid = 0;sum = 0;
2,3.    Do While value[i] <> −999  And  input < 100
4.      increment  input by 1;
5,6.    If  value[i] >= minimum  And  value[i] <= maximum
7.          Then increment  valid by 1;
                sum = sum + value[i];
            Else skip
8.      End If
        increment i by 1;
9.      End Do
10.     If valid > 0
11.     Then average = sum / valid;
12.     Else average = −999;
13.     End If
```

基本路徑測試範例 -2

- 根據上面的程式碼，透過程式敘述編號對應節點繪製流程圖（右圖）。
- 計算迴圈複雜性V(G) = P + 1= 5個判斷節點+ 1 = 6，即基本路徑上限個數為 6。
- 找出基本路徑為：
  - Path1：1-2-10-11-13
  - Path2：1-2-10-12-13
  - Path3：1-2-3-10-11-13
  - Path4：1-2-3-4-5-8-9-2-……
  - Path5：1-2-3-4-5-6-8-9-2-……
  - Path6：1-2-3-4-5-6-7-8-9-2-……



基本路徑測試範例 -3

- 針對每個基本路徑**設計測試案例，進而執行測試。**
  - Path 1：1-2-10-11-13，測試案例設計為value[1] = −999，但valid=0無法進入11。所以path1不能單獨測試，只能成為路徑4、5、6測試中的一部分。
  - Path 2：1-2-10-12-13，測試案例設計為value[1] = −999，期望輸出average = −999；sum = 0。
  - Path 3：1-2-3-10-11-13，測試案例設計為嘗試處理第101個或更多個的值，而其中前100個值應該是有效的，期望輸出：average = −999。
  - Path 4：1-2-3-4-5-8-9-2-……，測試案例設計為value[i] =有效輸入且i < l00，value[r] <最小值，r <= i，r是其中一個輸入索引，期望輸出為正確平均值和總數。
  - Path 5：1-2-3-4-5-6-8-9-2-……，測試案例設計為value[i] =有效輸入且i < l00，value[r] >最大值，r <= i，r是其中一個輸入索引，期望輸出為正確平均值和總數。
  - Path 6：1-2-3-4-5-6-7-8-9-2-……，測試案例設計為value[i] =有效輸入且i < l00，期望輸出為基於i個數值的正確平均數和總數。

邏輯條件測試

- **邏輯條件測試(Logic Condition Testing)**主要在於檢查程式的邏輯判斷條件，可分為簡單條件判斷與複雜條件判斷。

- 簡單邏輯條件包括布林變數和關係運算式。
  - 關係運算式表示為「E1 關係運算子 E2 」，其中E1和E2為算術運算式，關係運算子有<, <=, =, <> , >, >=。

- 複雜條件由多個簡單條件、布林運算子、小括號組成。
  - 布林運算子為AND、OR和NOT。

邏輯判斷的條件錯誤類型

- (1)布林運算子錯誤，包括AND, OR, NOT使用不正確、遺失或增加額外的布林運算子；
- (2)布林變數錯誤；
- (3)括號錯誤；
- (4)關係運算子錯誤(<, > , ==, >=, <=)；
- (5)算術運算子錯誤(+, − *, /)。

資料流測試 -1

- **資料流測試(Data Flow Testing)**又稱為DU測試(DU Testing)，針對資料**定義(Define)**與**使用(Use)**進行分析，確保測試案例涵蓋程式中資料的使用情形。

- **資料定義**是針對資料變數指定正確狀態，或初始化資料變數的值，但並非變數宣告。
  - x = 3; x = y + z;，說明了對資料x的定義情形。

- **資料使用**是資料變數使用於計算(Computation Use, C-use)。
  - z = (x × 3)/2，說明了對資料x的計算使用情形。

- 使用於**條件判斷**(Predicate Use, P-use)。
  - If (x > 3)，說明了對資料x的條件判斷使用情形。

資料流測試 -2

- 資料流測試的路徑選擇即根據程式中變數定義及使用的指令敘述，選擇程式測試路徑，又稱為**DU鏈(DU Chain)**。

- **資料變數X的DU鏈定義為[X, S, S']**，表示在指令敘述S中定義變數X，並且在指令敘述S'中使用變數X。
  - $X \in \{ DEF(S) \cup USE(S') \} \Rightarrow [X, S, S']$
  - DEF(S)={X | 敘述S包含X的定義}
  - USE(S) ={X | 敘述S包含X的使用}
  - 若敘述S為if或迴圈敘述，則其DEF集合為空。

- **資料流測試不保證覆蓋程式的所有分支**：例如，if-then-else的敘述結構中，若then部分沒有定義任何變數，且else部分不存在，則DU測試不保證覆蓋某一分支。

資料流測試範例

- 當員工1週工作時數達55小時（含），且遲到次數少於2次時，則加發10%的薪資；而員工1週內沒有遲到和請假，則加發1,000元。

範例程式碼

```
1.  int w=56, d=0, p=0, a=0;
2.  if ((w>=55) && (d<2)
3.  {
4.        p = p * 1.1;
5.  }
6.  if ((d==0)&&(a==0))
7.  {
8.        p = p + 1000;
9.  }
```

針對**資料變數** p 而言，
可以得到以下的資訊:

DEF(1) = {w, d, p, a}
USE(4) = {p}
程式測試路徑 Path = 1,2,3,4,5

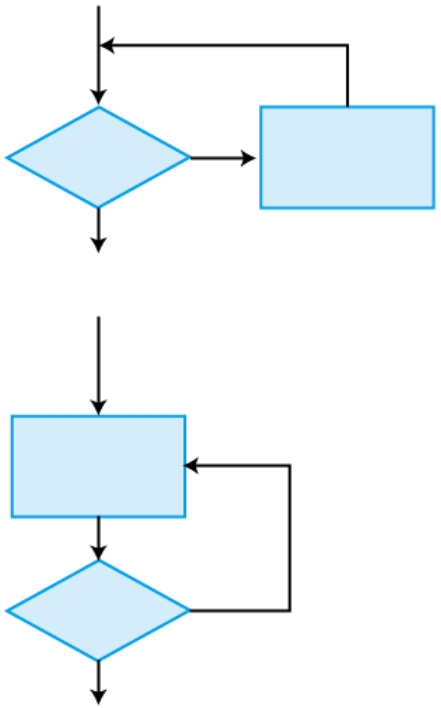
迴圈測試

- 迴圈為常用的程式結構，用以處理在某種條件下不斷重複的執行的敘述。

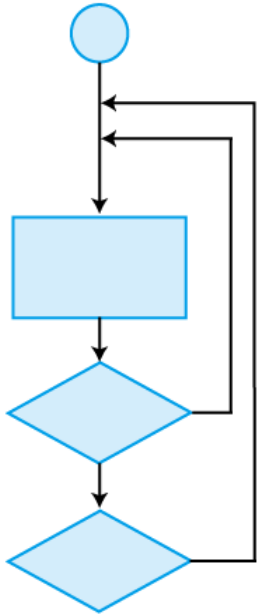
- 由於迴圈的重複特性，常使得錯誤的情形更加難以偵測。
  - 錯誤可能發生在特定第n次的迴圈執行時，如果測試案例無法偵測到第n次的迴圈執行結果不正確，將無法找出程式邏輯上的錯誤。

- 要測試一個程式中所有的迴圈組合亦如窮取測試法一樣不可行。
  - 針對常見的四種迴圈結構：**簡單迴圈**、**巢狀迴圈**、**串聯迴圈**和**非結構化迴圈**，可用不同方式進行測試。

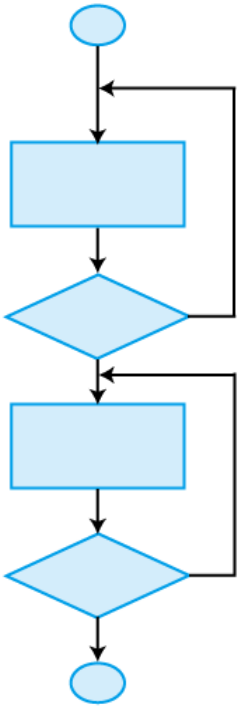
迴圈結構



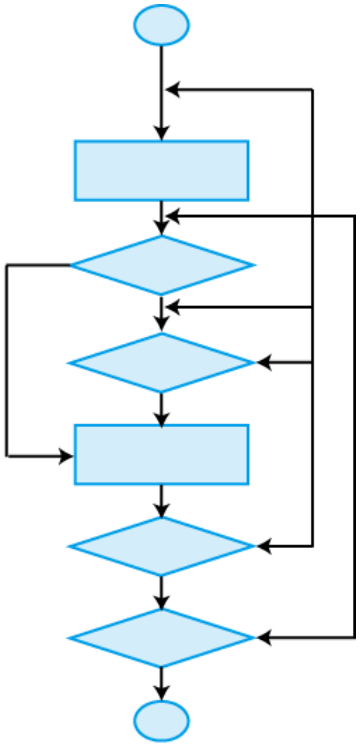
簡單迴圈



巢狀迴圈



串聯迴圈



非結構化迴圈

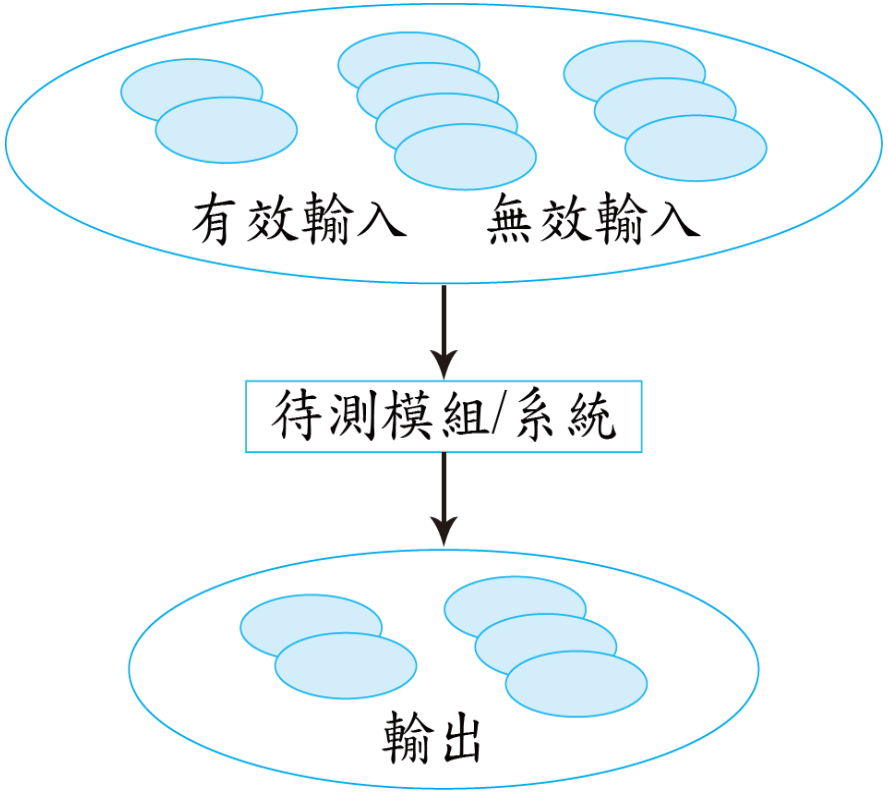
不同迴圈結構的測試方法

- **簡單迴圈**：若n為通過此迴圈的最大次數，則測試案例的設計方法可以是跳過整個迴圈、只通過此迴圈1次、通過此迴圈2次、與通過此迴圈m次(m < n)、通過此迴圈 n－1、n、n＋1次。
- **巢狀迴圈**：簡單迴圈技術的延伸，其測試案例的設計方法可以從最內部的迴圈開始，把其他迴圈設為最小值，對最內部迴圈執行簡單迴圈測試，同時使外部迴圈的索引值為最小。向外擴展，產生下個迴圈的測試，而保持其他更外部迴圈的索引值為最小。重複上述步驟，直到由內而外測試完所有的迴圈。
- **串聯迴圈**：若各個迴圈間相互獨立，則使用簡單迴圈測試方法設計測試案例即可。但若2個迴圈串聯，而迴圈1的計數器用於迴圈2的初始值設定時，則這2個迴圈是不獨立的，此時應使用巢狀迴圈測試方法。
- **非結構化迴圈**：此類型的迴圈在設計實作上較不易理解與較複雜，且不容易設計測試案例，應將此段程式碼重新設計成結構化程式，方便利用上述方式進行測試。

6 等價劃分方法

- **等價劃分方法 (Equivalence Partition)** 是以不同的**資料類別**、劃分規則來切割輸入域，以設計測試案例。

- 所切割的輸入域的子集合中，任何輸入資料對程式中所造成的錯誤都是**等效**的，同樣可以發現某類錯誤。



等價劃分概念

等價劃分規則 -1

- 1. 輸入條件中若規定變數值x的**範圍**介於a和b值之間，則取1個有效等價類別滿足a＜x＜b，和2個無效等價類別分別滿足x＜a和x＞b的值來設計測試案例。例如，輸入x為4＜x＜10，選擇測試案例可為數值：3, 7, 11。
- 2. 輸入條件若規定模組輸入值的**個數為N**，則取1個有效等價類別為個數N，和各1個無效等價類別為個數＞N，與個數＜N的值來設計測試案例資料。例如，輸入班級所有人成績，計算全班平均、班級人數為10 <= N <= 75人。則設計測試案例輸入資料為9, 60, 76人的成績。
- 3. 若輸入資料須遵守某個**規則**，則選擇1個遵守該規則的有效等價類別和若干個違反該規則的無效等價類別。例如，輸入條件為非0非1開始的3位數字，則設計測試資料為011, 100, 300。
- 4. 若輸入條件規定為值的**集合**，則設計1個在該集合中的有效等價類別和1個不在該集合中的無效等價類別。例如，自動提款機(ATM)系統輸入關鍵字命令，命令集合{check, deposit, billPay}，則設計測試資料為「check」和「OK」。

等價劃分規則 -2

- 5. 若輸入條件是**布林值**，則設計1個表示真值的有效等價類別和1個表示假值的無效等價類別。例如，自動提款機的提款是否要換小額鈔票，則設計有效類別{Yes/True}和無效類別{OK}的測試案例。
- 6. 若輸入條件是**1組值（n個）**，程式要對每個輸入值分別處理，則針對該組值設計n個有效等價類別和1個不屬於該組值的無效等價類別。例如，自動提款機提款輸入金額為{1000, 2000, …, 30000}，則設計測試資料：1000, 2000, …, 30000, 31000。
- 7. 利用以上原則劃分出的等價類別裡，其中各元素在程式處理的方式有不同之處，則將該等價類別**再進一步劃分為更小的等價類別**。例如，ATM的提款是否要換小額鈔票，則針對有效類別，再細分兩個等價類別{Yes/True}和{No/False}。

等價劃分方法範例 -1

- 若有一函式Search()，用來搜尋某個輸入值Key是否出現在另一輸入的T陣列當中，並傳回布林值Found告知是否找到，以及找到的的元素位於陣列中的索引L。
  - 介面規格：Procedure Search (Key: in ELEM; T: in ELEM_ARRAY; Found: out BOOLEAN; L: out ELEM_INDEX) ;
  - 前置條件(Pre-Condition)：陣列至少有1個元素（T'FIRST <= T'LAST）
  - 後置條件(Post-Condition)：元素被找到，L是索引(Found and T [L] = Key)或元素不在陣列中(not Found and not (exists i, T'FIRST <= i <= T'LAST, T [i] = Key))

等價劃分方法範例 -2

- 根據等價劃分的原則，將輸入資料等價劃分為兩類：
  - 第一類：
    - 1. 符合前置條件：(1)陣列有1個元素；(2)陣列有多個元素。
    - 2. 不符合前置條件：陣列長度為0，即0個元素。
  - 第二類：
    - 1. Key在陣列被找到的位置有三種：(1)最前面；(2)中間位置；(3)最後位置。
    - 2. Key不在陣列中。
- 經過等價劃分後排列組合，去掉不可能的組合，測試案例中的資料可設計為以下類別：
  - 1. 陣列長度為0。
  - 2. 陣列有1個元素，Key在陣列中。
  - 3. 陣列有1個元素，Key不在陣列中。
  - 4. 陣列有多個元素，Key在陣列的最前面。
  - 5. 陣列有多個元素，Key在陣列的中間位置。
  - 6. 陣列有多個元素，Key在陣列的最後位置。
  - 7. 陣列有多個元素，Key不在陣列中。

等價劃分方法範例 -3

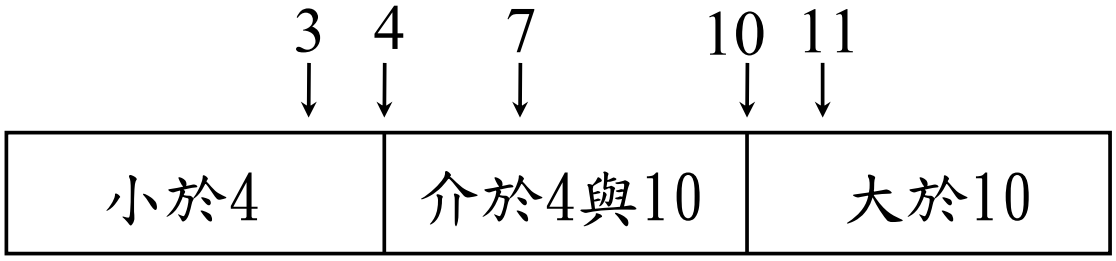
根據以上條件，設計測試案例資料如下：

Input Sequence (T)	Key	Output (Found, L)
17	17	true, 1
17	0	false, ?
17, 29, 21, 23	17	true, 1
41, 18, 9, 31, 30, 16, 45	45	true, 7
17, 18, 21, 23, 29, 41, 38	23	true, 4
21, 23, 29, 33, 38	25	false, ?

邊界值分析方法

- **邊界值分析(Boundary Value Analysis)**為等價劃分的擴展，邊界值通常是**等價類別的界限**，因為根據實務經驗，錯誤發生在輸入域邊界上的機率遠大於在輸入域中間的情況。

- 邊界值分析方法增加選擇等價類別的「邊界」值來設計測試案例，同時也可以根據輸出域的邊界值來設計測試案例。



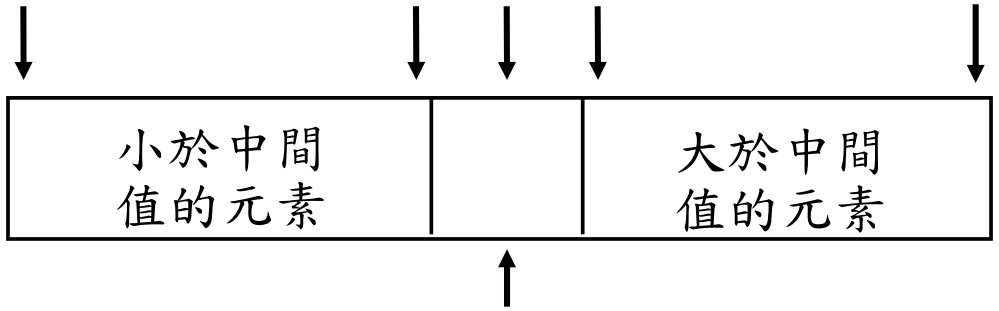
輸入範圍示意圖

邊界值的選擇規範

- 1. 若輸入條件或輸出條件規定值的**範圍**為a和b（如圖6-15），則取等於和剛好大於a，以及等於和剛好小於b的值設計測試案例。例如，輸入x為4 < x < 10，則選擇測試案例為邊界值：3, 4, 7, 10, 11。
- 2. 輸入條件若規定模組輸入值的**個數為N**，則取1個有效等價類別為個數N，和各1個無效等價類別為個數N + 1與個數N – 1的值來設計測試案例資料。例如，輸入班級50人成績，計算全班平均。則設計測試案例輸入資料為49, 50, 51個人的成績。
- 3. 若輸入資料須遵守某個**規則**，則選擇1個有效等價類別和若干個無效等價類別。例如，輸入條件為非0非1開始的3位數字，則設計測試資料為011, 100, 300。
- 4. 若輸入或輸出域是有順序的**集合**，則選取集合的第一個和最後一個元素做為測試資料。例如，自動提款機提款輸入金額為{1000, 2000, …, 30000}，則設計測試案例資料：1000, 30000。
- 5. 如果程式使用1個**內部資料結構**，則選擇資料結構邊界的值為測試資料。如100個陣列大小，設計邊界檢查0個或101個陣列大小。
- 6. **分析需求或設計規格**，找出其他可能的邊界條件。

邊界值分析範例 -1

- 接續等價劃分範例中的Search()函式，假設陣列為已排序過，且每次都是從中間位置開始尋找，則可將測試狀況分類為以下的等價類別：
  - 1. 第一類為關鍵元素(Key Element)在陣列中(K-Class = 1)；和關鍵元素不在陣列中 (K-Class = 2)。
  - 2. 第二類為陣列是奇數 (T-Class = 1)；和陣列是偶數 (T-Class = 2)。



二元搜尋範例

邊界值分析範例 -2

- 根據上述等價類別與邊界值分析，設計出以下的測試案例與預期的測試結果。

等價類別				輸入值 Input		輸出結果 Result	
T-Class	n/b	K-Class	n/b	T Array	K	L	Found
1	n	1	n	{1,2,3}	2	1	TRUE
1	n	1	b	{1,2,3}	1	0	TRUE
1	b	1	b	{1}	1	0	TRUE
1	b	2	b	{1}	2	-	FALSE
2	n	2	n	{3,5,6,9}	4	-	FALSE
2	b	2	n	null	9	無此資料	
2	n	1	b	{3,5,6,9}	9	3	TRUE
n/b: 一般值(normal) or 邊界值(boundary)							

因果圖方法

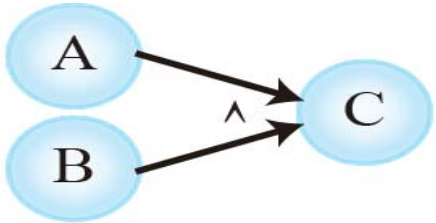
- 當需求規格描述為多種條件組合，因而產生相對應的多種結果，此種複雜邏輯的使用者需求，可以使用**因果圖(Cause Effect Graph)**來設計測試案例。

- 因果圖的設計步驟如下：
  - 1. 根據需求規格，找出所有需求的**因(Cause)**和**果(Effect)**。
  - 2. 將因和果設成**節點(Node)**，編號後置於因果圖的左邊與右邊。
  - 3. 繪製因果的關係為**邊(Edge)**。

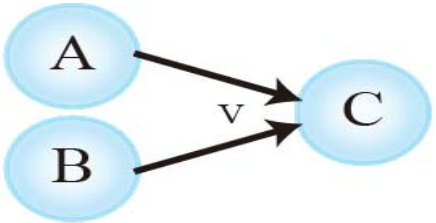
- 因果圖利用**決策表(Decision Table)**為輔助工具，其先決條件為：
  - 1. 條件順序不會影響動作結果。
  - 2. 每個規則都是互斥，只要滿足其中一個，就不需再檢驗其他規則。
  - 3. 若有許多的動作或影響，其發生的順序並不重要。
  - 4. 規則的定義要有一致性，不能彼此衝突。

因果圖符號

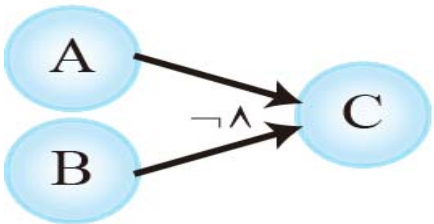
如果 A 則 B



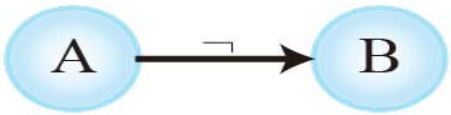
如果 (A 且 B) 則 C



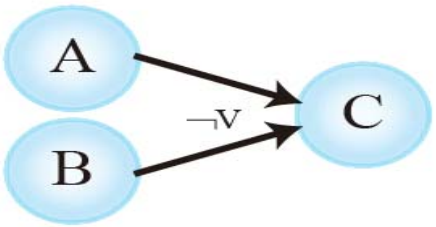
如果 (A 或 B) 則 C



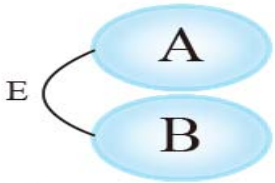
如果 (非(A 且 B)) 則 C



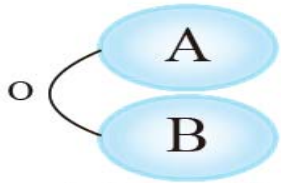
如果 (非 A) 則 B



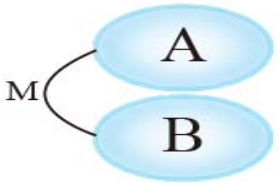
如果 (非 (A 或 B)) 則 C



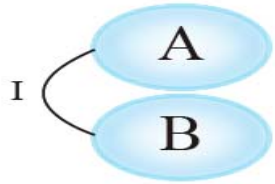
互斥：A, B最多一個成立



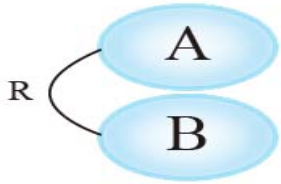
唯一：A, B必須只有一個成立



遮罩：當A出現，
B不一定出現，
A不出現，
B不確定。



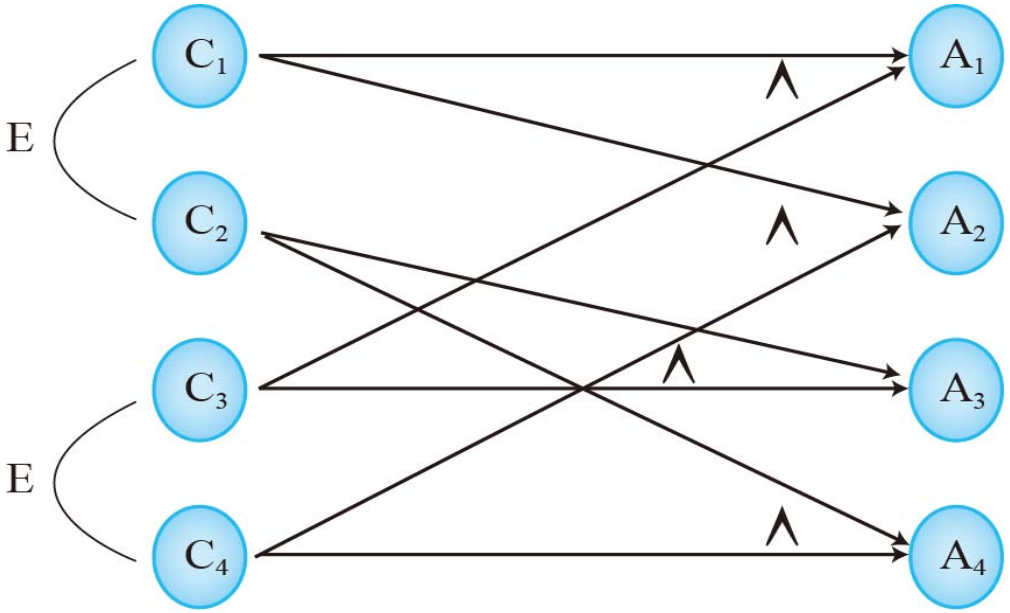
包含：A, B至少一個成立



要求：當A出現B一定出現

因果圖方法範例 -1

- 員工薪水計算範例：年薪制員工犯嚴重過失時，扣年終獎金的4%；犯一般過失時，扣年終獎金的2%；非年薪制員工犯嚴重過失時，扣當月薪資的8%；犯一般過失時，扣當月薪資的4%。



因果圖

原因	結果
C1-年薪制員工	A1-扣年終獎金的4%
C2-非年薪制員工	A2-扣年終獎金的2%
C3-嚴重過失	A3-扣當月薪資的8%
C4-一般過失	A4-扣當月薪資的4%

原因與結果列表

因果圖方法範例 -2

- 決策表（下表）中測試案例 (Test Case, TC) 標記Y表示此為測試案例，灰色表示違反條件不可能出現。第一列的1～16表示16種可能的因／果組合情形，而每一行（除了第一行之外）中的0與1則分別表示該原因或是結果的描述條件是成立或是不成立。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
C2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
C3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
C4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
A1						0	0			0	1					
A2						0	0			1	0					
A3						0	1			0	0					
A4						1	0			0	0					
TC						Y	Y			Y	Y					

6.5 軟體動態測試策略

- 軟體的開發是由許多小的單元模組，經過一連串複雜的整合過程，最後形成整個軟體系統雛型。在這過程中，開發團隊需要不斷的測試、發現錯誤與修正錯誤，直到軟體品質到達一定的程度，才進行接下來的移交階段。

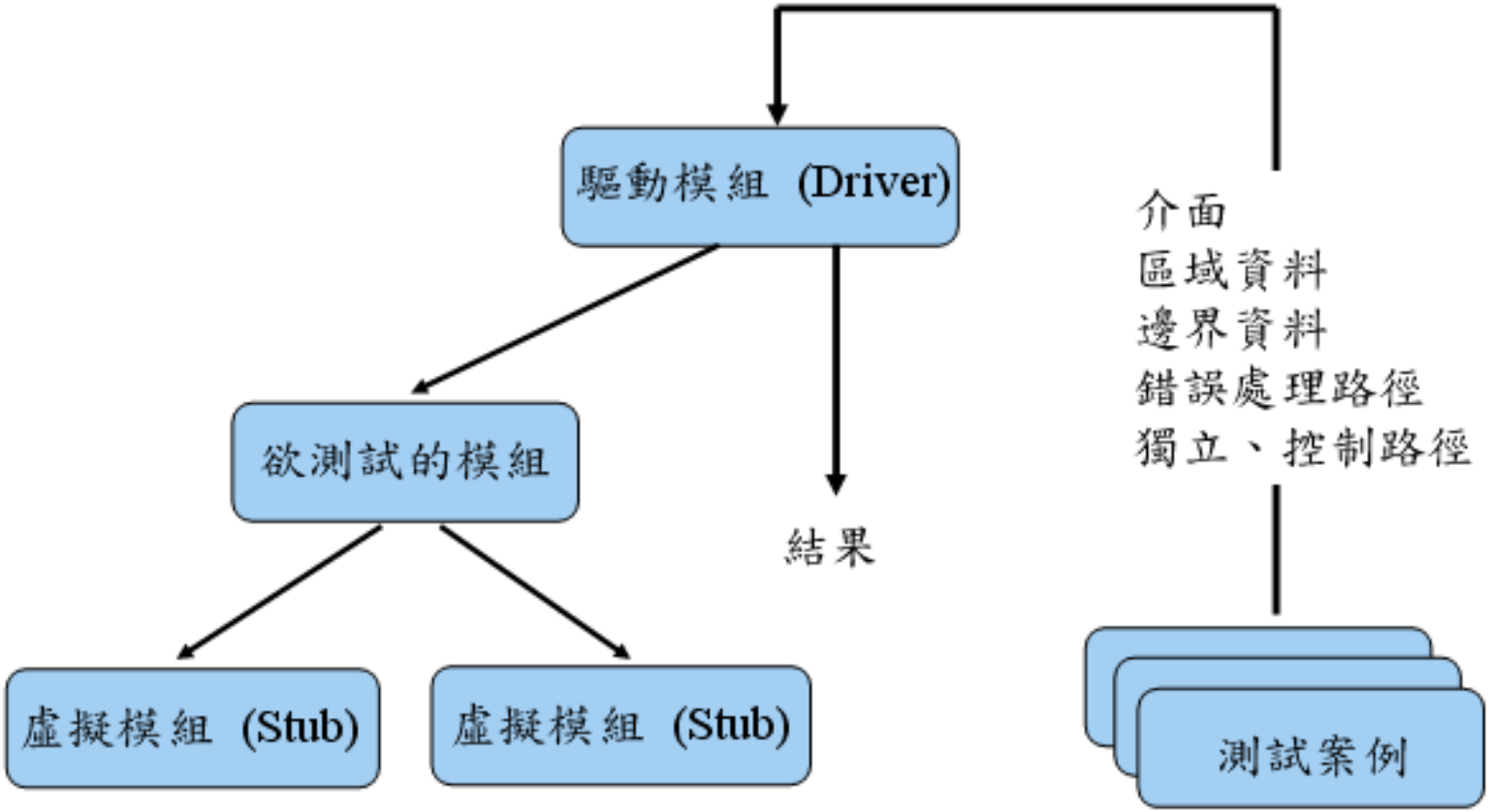
- 移交軟體系統給使用者的過程中，開發團隊亦需要對即將上線的軟體系統進行一系列的測試，確保其符合使用者的要求與滿足將運行的環境資源限制。最後經由正式的驗收步驟，正式將軟體系統於使用者端安裝與上線成功。

- 軟體動態測試策略大致上分成以下多種類型，包含單元測試(Unit Testing)、整合測試(Integration Testing) 、系統測試(System Testing) ，以及驗收測試(Acceptance Testing)。

單元測試

- 由軟體開發人員親自對軟體內部結構執行最小規模的測試。
- 受測程式可以是單獨或僅提供某項功能的程式。
- 此類測試的目的在確保其功能可單獨且正確地運作，每個單元必須能獨立測試而不受其他元件的影響，是屬於所有測試策略中最低層次的測試。
- 主要使用白箱測試方法，並配合黑箱測試方法，來設計測試案例，以檢驗每個單獨的模組是否正確的執行其預期功能。

單元測試架構


驅動模組 (Driver)
介面
區域資料
邊界資料
錯誤處理路徑
獨立、控制路徑
欲測試的模組
結果
虛擬模組 (Stub)
虛擬模組 (Stub)
測試案例

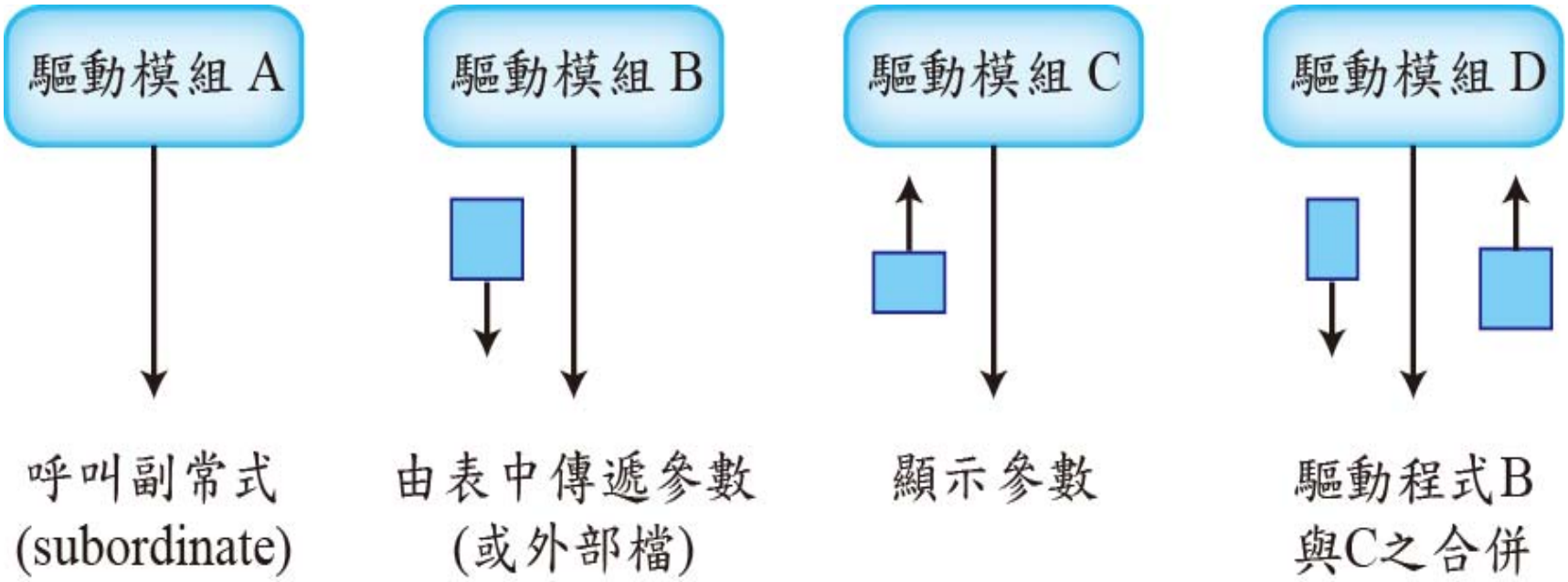
驅動模組與虛擬模組

- **驅動模組(Driver)**可以視為「主程式」，接收測試案例的資料，傳遞到被測試模組，再比較預期結果。

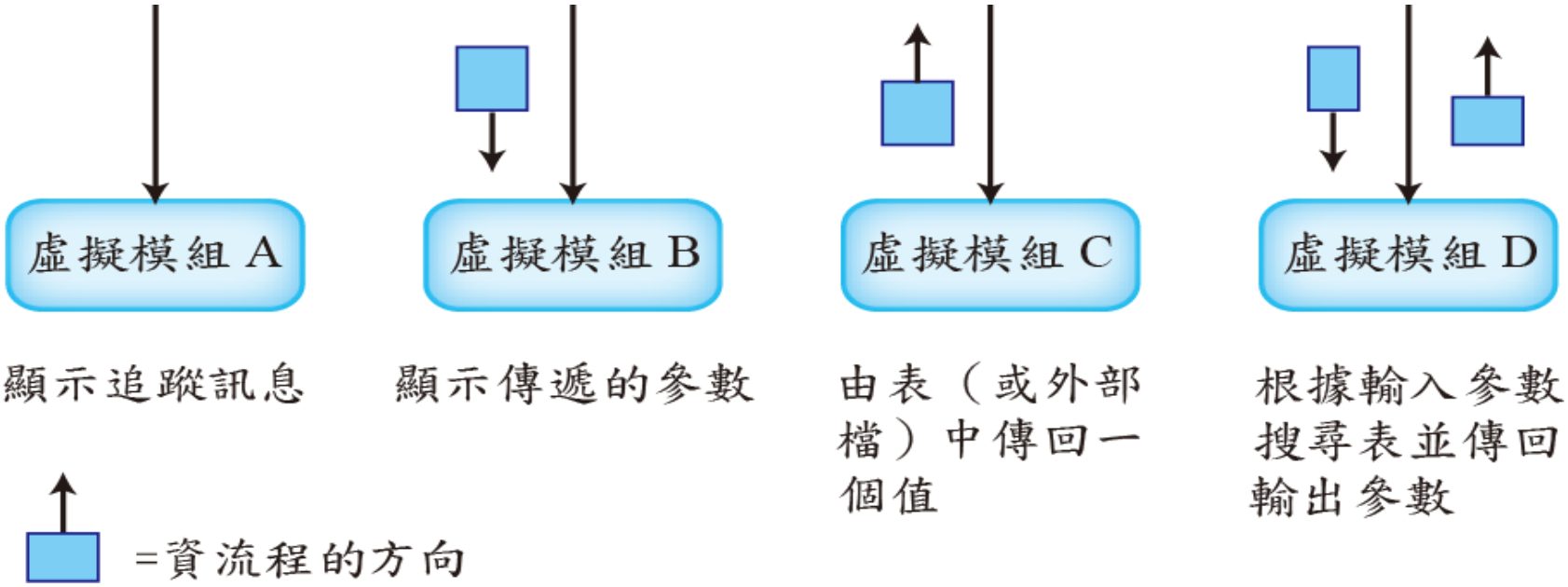
- **虛擬模組(Stub)**替代被測試模組所呼叫的其他副程式，模擬設計其介面，進行最基本的資料輸入、資料操作、輸出呼叫結果與回傳參數到被測試模組。

- 驅動模組和虛擬模組是為了測試而額外開發的程式，並非最終交付給客戶的軟體產品，且最終將被其他真實的單元模組或子系統所取代。

不同類型驅動模組


驅動模組 A
驅動模組 B
驅動模組 C
驅動模組 D
呼叫副常式
(subordinate)
由表中傳遞參數
(或外部檔)
顯示參數
驅動程式B
與C之合併

不同類型的虛擬模組


虛擬模組 A
顯示追蹤訊息
虛擬模組 B
顯示傳遞的參數
虛擬模組 C
由表（或外部檔）中傳回一個值
虛擬模組 D
根據輸入參數搜尋表並傳回輸出參數
=資流程的方向

單元測試的重點項目 -1

- **單元模組介面**：測試流入或流出的資料正確性。根據模組的使用情形，可能有以下的互動情形，主要可以區分為溝通介面與外部I/O介面兩種。個別介面的測試檢查清單介紹如下:
  - 模組呼叫的輸入參數數量是否等於引數數量？
  - 參數與引數的資料型態是否一致？
  - 參數與引數單位系統是否一致？
  - 函數的數值屬性和引數順序是否正確？
  - 輸入的參數是否更改過？
  - 模組中全域變數的定義是否不小心更改過？
  - 傳入模組的參數條件（值的範圍）是否需要限制？
  - 另外，模組執行外部I/O時之介面測試或檢查清單的參考項目為：
  - 檔案屬性是否正確？
  - 檔案的開啟與關閉(OPEN/CLOSE)敘述是否正確？
  - 資料格式宣告與輸出入(I/O)指令敘述是否一致？
  - 緩衝區大小與記錄大小是否一致？
  - 檔案在使用前是否已開啟？
  - 檔案限制條件是否已處理？
  - I/O錯誤是否已處理？
  - 輸出資訊文字是否錯誤？

單元測試的重點項目 -2

- **區域性資料**：測試模組內部的資料結構，或暫時儲存資料是否正確執行演算法。此部分的設計的測試案例主要著重於找出下列問題：
  - 不正確的資料型態宣告。
  - 錯誤的初始值或預設值。
  - 不正確的變數名稱（如：拼錯或截斷）。
  - 不一致的資料型態設定。
  - 陣列宣告造成下溢、上溢或位址出界。
  - 全域變數是否曾不正確的更改過。

- **邊界條件**：測試模組在邊界上是否正確執行。模組控制結構中的獨立路徑至少須執行過一次以上。此部分設計的測試案例主要著重於檢查下列項目：
  - 當一個n維陣列的第n個元素已處理時。
  - 當具有i次迴圈的第i次重複啟動時。
  - 當最大或最小可允許的值被處理時。
  - 檢查資料結構設計是否影響邊界條件。
  - 控制流程條件判斷中，剛好大於、等於或小於最大值或最小值的測試案例。

單元測試的重點項目 -3

- **獨立、控制路徑**：測試模組範圍內路徑控制的錯誤。獨立路徑是指程式的執行路徑中，至少包含一組新的語句或是一個新的條件之路徑。在測試此類的錯誤時，可以參考以下的建議或是準則進行測試案例的設計。
  - 基本路徑、條件和迴圈測試等測試方法，可有效發現路徑錯誤。
  - 控制流程可能會常受到判斷敘述中比較運算式的影響，例如，控制流程的變動常發生在某個判斷敘述之後。
  - 迴圈可能發生的錯誤包括不正確或不存在的迴圈結束，或不正確的更改迴圈變數。
  - 條件比較可能發生的錯誤包括算數計算錯誤、比較運算錯誤。
  - 常見的算數計算錯誤包括不正確的算術運算次序、混合資料型態運算、不正確的初始化、精準度不夠、錯誤的運算符號表示、優先運算等級錯誤。
  - 常見的比較、邏輯運算錯誤包括不同資料型態的比較、不正確的邏輯運算子、優先運算等級錯誤；由於精準度的錯誤，使得等號不成立；不正確的邏輯變數比較。

單元測試的重點項目 -4

- **確認所有錯誤處理路徑均已考慮到**：好的錯誤處理路徑設計能在錯誤發生時，重新開始或正常結束，或是正確顯示錯誤訊息。以下幾點是常見的錯誤：
  - 錯誤敘述可閱讀性不好。
  - 錯誤描述與實際錯誤不一致。
  - 錯誤條件導致系統干預優先於錯誤處理。
  - 異常條件處理不當。
  - 錯誤描述資訊不足，無法有效幫助除錯。

整合測試

- 整合測試(Integration Testing)是指在元件通過單元測試之後，開始整合單元模組，以檢查元件之間的一致性。
- 若僅部分元件完成單元測試，其餘尚未發展完畢，則可先整合已完成單元測試的元件並進行整合測試，此時開發人員亦應加入整合測試工作。
- 整合測試的目的在於找出系統整合時的錯誤。
  - 雖然每個單元模組都通過個別的單元測試，但組合後的各模組未必能順利執行，因此須檢驗此組合過程與系統架構是否正確，通常用黑箱測試進行此種檢驗。
- **最佳的整合測試是先使用由上而下的整合測試，並在確定較具關鍵性的綜合模組後，再進行由下而上的整合測試。**

整合測試的層次

- 1. **模組測試**是指針對單一功能將相關元件集合起來，進行元件與元件間互相合作的測試。測試的目的主要是驗證任意數個元件間的介面及互相呼叫服務的過程及結果是否正確，因此各**元件之間的溝通介面設計**為測試重點。

- 2. **子系統**是由數個模組集合成為提供完整服務的大型模組，為確保子系統內部的功能正確，除在模組測試中先確定各模組的功能外，還必須測試模組與模組間的通訊介面及結果是否正確，因此各**模組間的通訊機制及溝通介面設計**為子系統測試的重點。

- 3. **整合子系統成為整套系統**，此階段的測試著重於驗證**子系統之間的介面溝通**是否正確，並驗證整合後的全系統是否仍能符合其該有的**功能性與非功能性需求**。

整合測試時可能發生的問題

- 資料經過介面傳遞時可能忽略了另一個模組。
- 或是某個模組可能透過資料的傳遞，無意間對其他模組產生不良影響。
- 可個別接受的不精確性資料，可能被放大到不可接收。
- 全域性資料結構可能造成副作用，帶來模組間的整合問題。

整合測試的策略

- **非漸增整合(Non-Increasing)**：所有模組同時合併測試，缺點是龐大的程式容易讓錯誤原因複雜化，增加錯誤修正的困難。

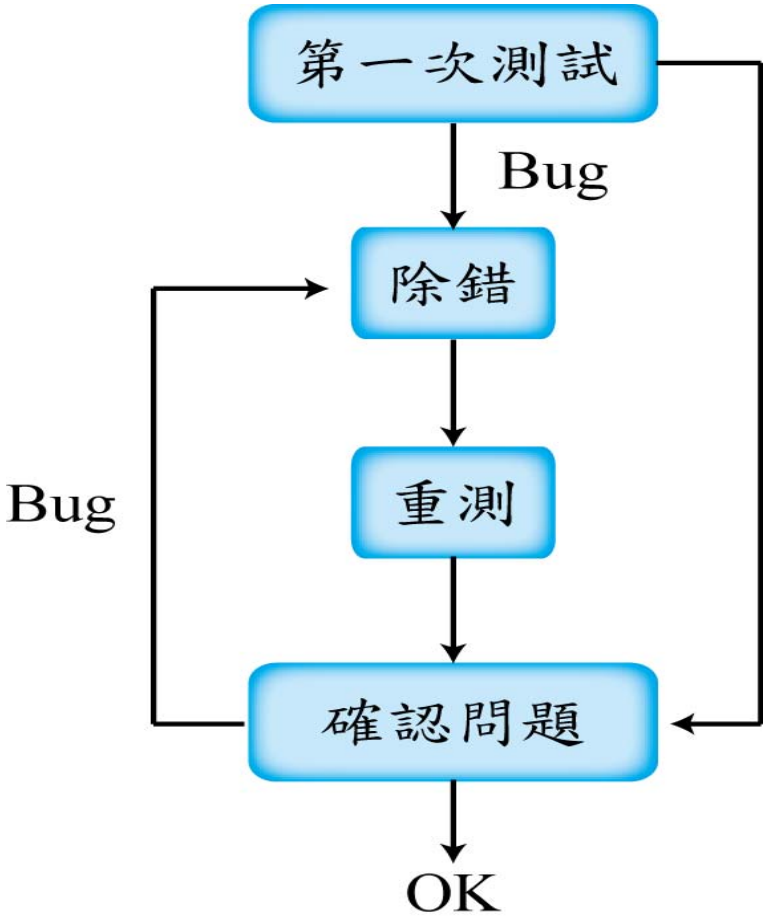
- **漸增整合測試**：以較小的程式區段落進行測試，錯誤較容易被隔離、修正，也比較能應用系統化測試方法，按照步驟工作，能完整測試介面。又可以分為**由上而下**、和**由下而上**兩種方法。

由上而下的漸增整合程序

- 1. 模組由主控模組（主程式）開始，按照控制層次由上而下整合。
- 2. 主控模組為測試的驅動模組，所有子模組當做虛擬模組。
- 3. 子模組以**深度優先**或**廣度優先**，用實際模組逐一替代虛擬模組整合。
- 4. 整合每個模組進行**迴歸測試**（進行全部或部分的測試），確保不會引入新的錯誤。
- 5. 重複第2步驟直到建立整個程式結構。

迴歸測試程序

- 迴歸測試是指在除錯之後，必須重複之前所做過的相同測試，並比較結果是否與前次相符。
- 迴歸測試可以確保修正（新增）的功能，不會影響原來已經完成並測試通過的功能，或確保問題（錯誤）已經被修正。



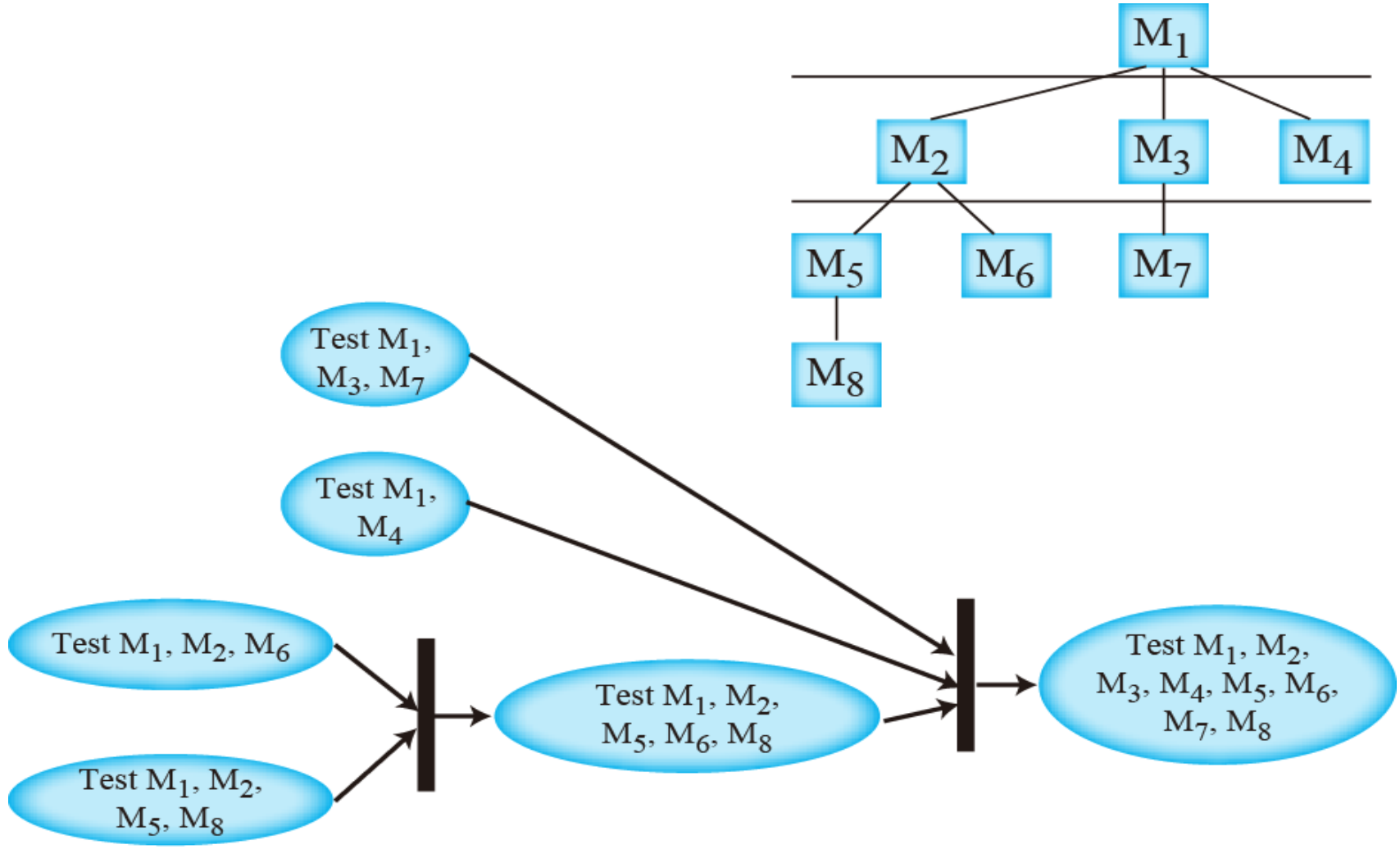
深度優先整合與廣度優先整合

- 由上而下的整合程序又可分為「深度優先整合 」與「廣度優先整合 」兩種整合策略。

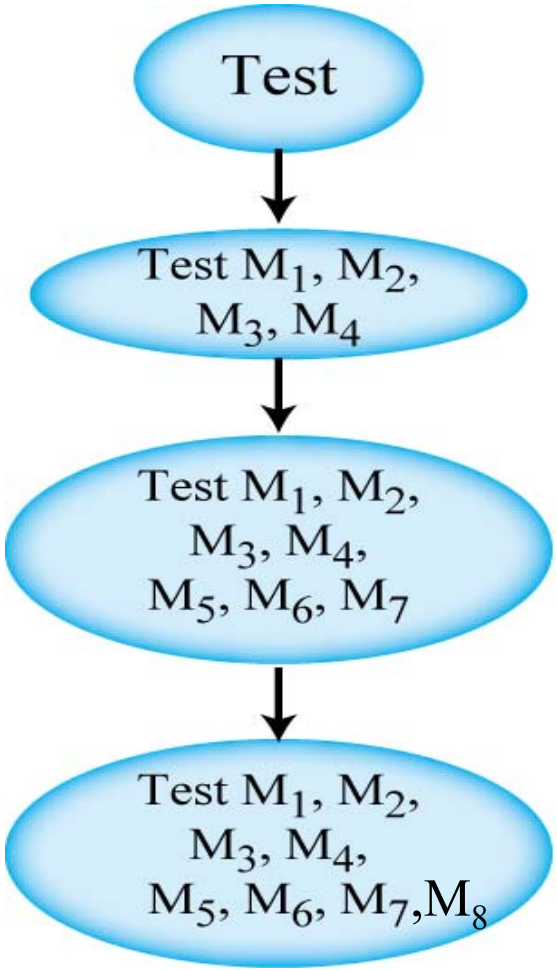
- **深度優先整合策略**：針對**主要控制路徑的模組**先進行整合，主要路徑的選擇取決於特定應用程式的性質。
  - 優點：在整合測試程序初期，就可驗證主要控制與決策點，證實軟體的完整功能。

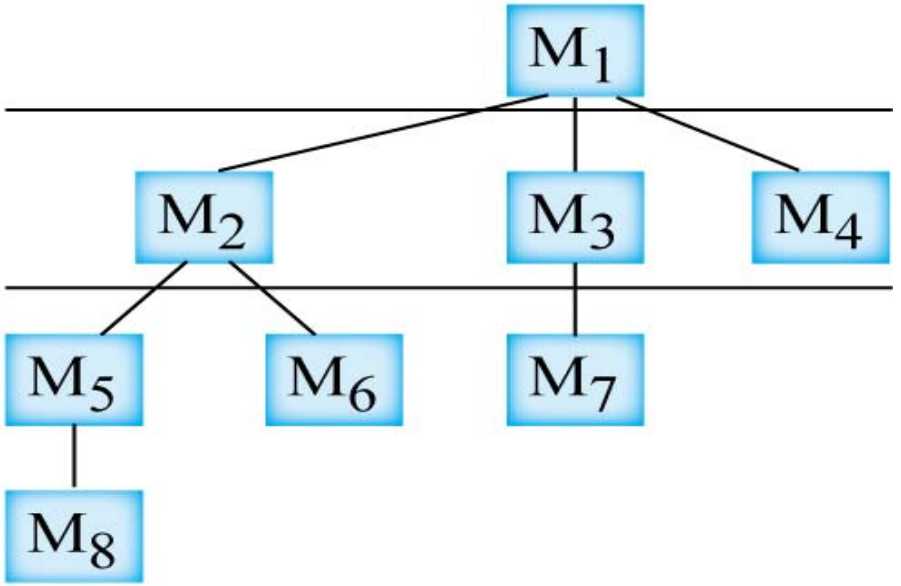
- **廣度優先整合策略**：針對每層次直接處理的所有子模組，水平掃描整個結構，由較**高層次的模組先整合**完畢後，再依次整合下一層次的模組，直到整合所有的模組為止。

深度優先整合範例


M1
M2
M3
M4
M5
M6
M7
M8
Test M1, M3, M7
Test M1, M4
Test M1, M2, M6
Test M1, M2, M5, M8
Test M1, M2, M5, M6, M8
Test M1, M2, M3, M4, M5, M6, M7, M8

廣度優先整合範例

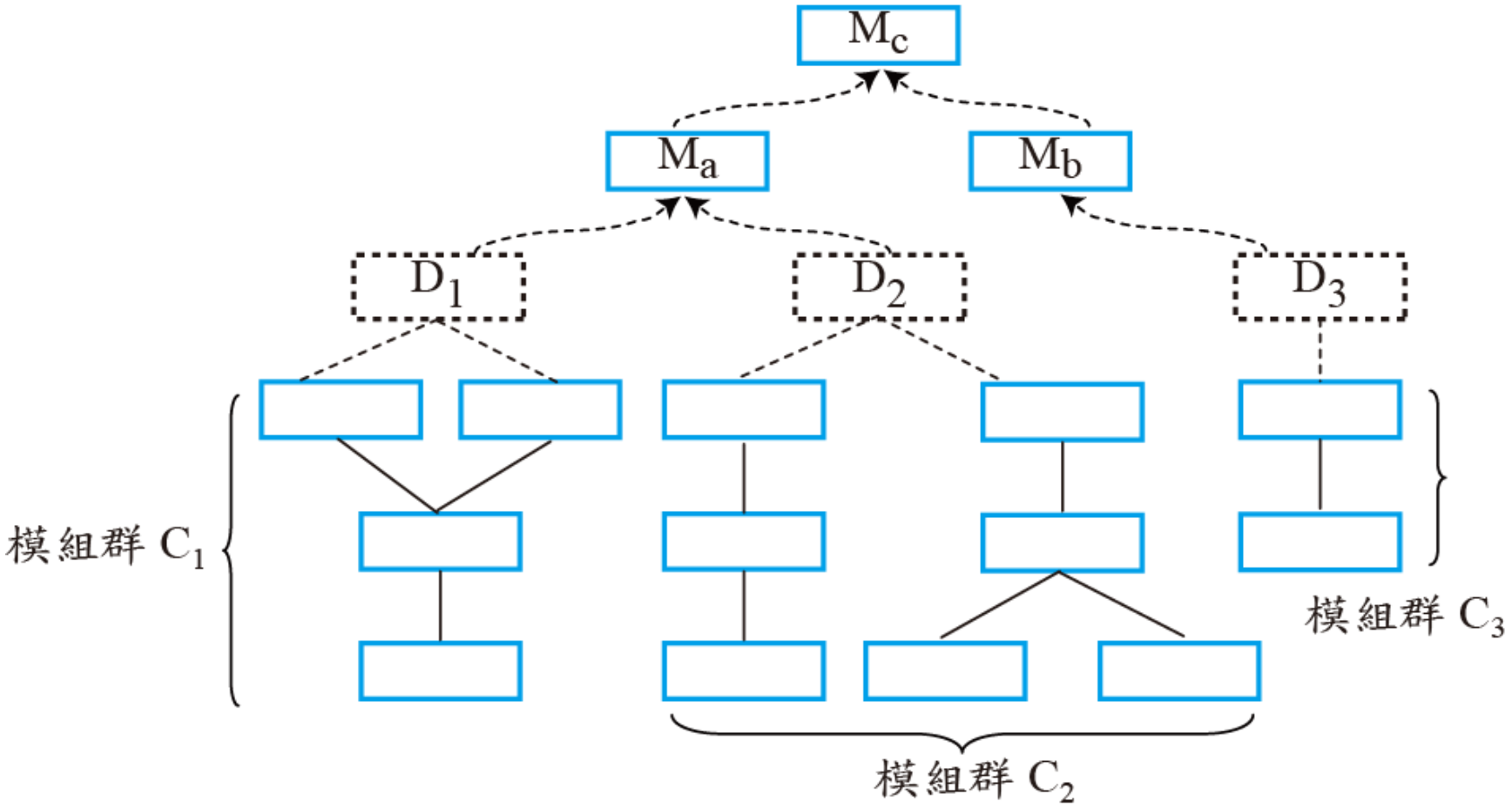

Test
Test M_1, M_2, M_3, M_4
Test M_1, M_2, M_3, M_4, M_5, M_6, M_7
Test M_1, M_2, M_3, M_4, M_5, M_6, M_7,M_8


M_1
M_2
M_3
M_4
M_5
M_6
M_7
M_8

由下而上的漸增整合程序

- 由下而上的整合原則是從子模組（程式結構中最低層模組）開始進行整合測試，模組由下而上整合，不需透過虛擬模組的建置。其整合程序如下：
  - 1. 合併低層模組成1組模組，執行特定系統子功能。
  - 2. 撰寫驅動模組（進行測試的控制程式），協調測試案例的輸入和輸出。
  - 3. 測試每組模組。
  - 4. 刪除驅動模組，在程式結構中向上合併，形成更大的模組群(Cluster)。

由下而上的整合範例



整合策略的比較

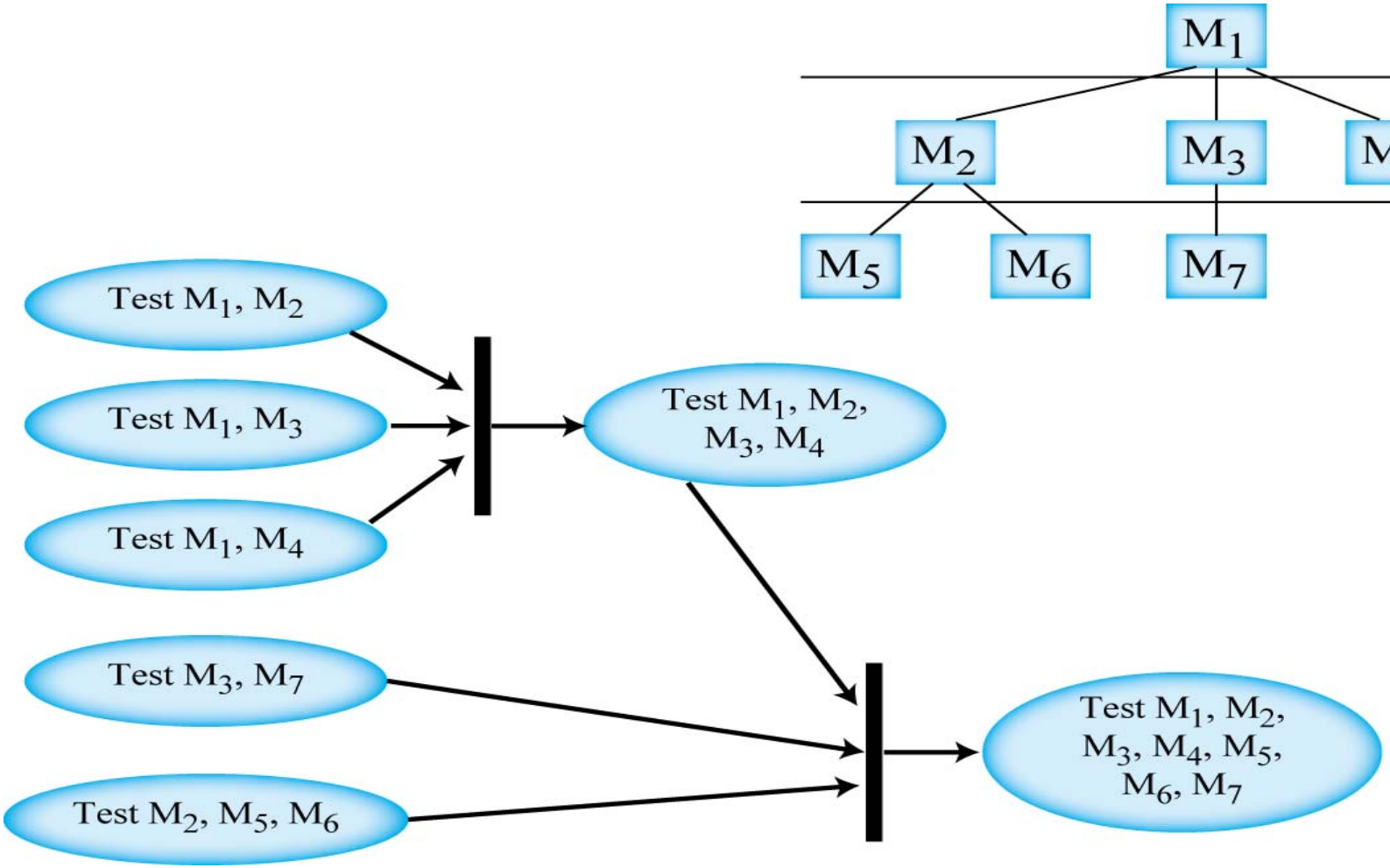
- 由上而下整合策略的缺點：
  - 使用虛擬模組替代較低層次模組時，真正有效的資料無法向上流入較上層的程式結構中，將可能延後許多測試，直到虛擬模組被實際模組替代，如此會喪失模組的回應控制測試，增加判斷錯誤原因的困難度。
  - 虛擬模組的開發若要盡量接近真實模組，卻又可能導致較多的額外開發成本。

- 由下而上整合的缺點：
  - 「直到最後一個模組加入，程式才完整」，對於重要功能或是系統行為的確認將無法提早進行。

夾心測試策略

- 高層次程式結構使用由上而下策略，較低層次則使用由下而上策略，兩種方法結合。
  - 驅動模組的數量將大幅減少且減化模組群的整合。
- 測試人員進行整合測試時，對於重要模組應儘可能較早測試，而且必須使用迴歸測試。
- 判斷重要模組時，可以利用以下的特徵進行：
  - 1. 具有多個軟體需求功能。
  - 2. 具有較高控制層次（位於程式結構中相對較高的位置）。
  - 3. 較複雜或容易出錯（迴圈複雜性可以做為指標）。

夾心測試範例


M1
M2
M3
M4
M5
M6
M7
Test M1, M2
Test M1, M3
Test M1, M4
Test M1, M2, M3, M4
Test M3, M7
Test M2, M5, M6
Test M1, M2, M3, M4, M5, M6, M7

整合測試文件的重要資訊

- 1. 需要測試的功能：
  - 介面一致性：測試每個模組置入系統結構時，內外部的介面。
  - 整體功能的有效性是否錯誤或遺漏。
  - 資訊內容：與局部或整體資料結構有關的錯誤。
  - 效能(Performance)：驗證軟體效能是否受到限制。
- 2. 測試項目(Test Items)
  - 參考單元測試報告，定義整合測試的模組功能。
  - 模組的測試描述、測試案例、問題、預期結果與實際結果。
- 3. 測試工作項目(Testing Tasks)
  - 測試整合程序與整合策略。
  - 測試階段和結構組成：由上而下的測試系統結構，依據整合策略中的各整合群組，應制訂需要測試的軟體功能和行為特徵。
- 4. 測試環境(Environment)
  - 額外開發的軟體：簡要的描述虛擬模組和驅動模組。
  - 環境和資源：例如，系統運行的環境或必須使用特定的工具。
- 5. 測試時程(Schedule)：參考軟體開發計畫，根據元件模組的完成與整合時程，適當的建立每個測試階段的開始日期和結束日期。

6 系統測試

- **整個資訊系統**的測試(System Testing)，檢驗系統中各子系統之間的整合，與整個系統軟硬體功能及執行績效是否符合需求，主要是由**硬體工程師**與**軟體測試人員**進行系統測試。

- 測試環境應儘量與**實際環境相似**，而測試案例的設計方法以黑箱測試技術為主，並參考系統的需求、規格與效能要求，來進行設計。

- 此階段軟體測試工程師的任務為:
  - (1)分析軟體內部介面問題、參與系統測試規劃與設計，以確保充分測試軟體；
  - (2)以模擬資料測試軟體系統介面的錯誤，並設計錯誤處理路徑，測試來自其他硬體系統或外部軟體的錯誤資訊。

系統測試的測試項目種類 -1

- **儲存(Storage)測試**：測試是否滿足主記憶體及輔助記憶體的使用限制。
- **相容(Compatibility)測試及轉換(Conversion)測試**：測試與其他系統的相容性及系統轉換的程序的正確性。
- **設備(Facility)測試**：確認所有設備均已整合納入並且能正常運作。
- **文件測試**：驗證使用者文件的正確性。
- **維護測試**：驗證可維護性，如：系統提供的協助、偵錯的平均時間及維護程序等。

系統測試的測試項目種類 -2

- **耐久(Durability)測試**：在極端操作條件下進行測試，例如，高溫、低溫、高溼度、高轉速（中央處理器、光碟機或磁碟機），以及隨意的操作。典型軟體包括武器系統或長程交通工具中的即時系統(Real-Time)韌體，都需要進行此種測試。

- **安裝測試**：驗證軟體系統已妥善安裝且能正確的運作，確認所有參數與運作條件均已依需要適當設定與記錄。軟體開發或測試單位應針對軟體特性，建立適當的軟體安裝檢核表或安裝測試軟體，以利安裝單位據以執行測試。

- **容量測試(Volume Testing)**：測試系統所能承受的最大資料容量，包括實體或邏輯限制，並且驗證是否滿足專案或組織需求。「資料容量」可能為特定資料庫大小或特定介面檔案大小。

系統測試的測試項目種類 -3

- **可用(Usability)測試**：測試是否有人為因素或使用性問題所導致的錯誤。其進行方式為設計一套測試劇本，找不同程度的使用者進行測試，並收集數據，繪製成測試圖形模型。
  - (1)此資訊系統可以提供多少個使用者服務；
  - (2)完成工作所需時間(Time on Task)：使用者完成基本工作所需花費的時間；
  - (3)正確率(Accuracy)：使用者執行某項工作時所犯錯誤的數量；
  - (4)回想率(Recall)：使用者多久沒有使用系統？是否還記得如何使用；
  - (5)情緒感受(Emotional Response)：使用者完成工作時的感覺，以及使用者在使用上的信心與壓力程度。

系統測試的測試項目種類 -4

- **可靠(Reliability)測試**：測試軟體在規範的**條件**與**時間**範圍內完成規範的**功能**。
  - 規範時間包括失效平均間隔時間(Mean Time between Failure, MTBF)與每月平均休息時間(Average downtime per month)。其中 **MTBF = MTTF + MTTR**，MTTF是指系統平均失效時間(Mean Time to Failure)，MTTR是指系統失效後的平均修復時間(Mean Time to Repair)。
  - 規範條件因素包括硬體、作業系統、資料輸入範圍、資料儲存與傳輸、操作程序。
  - 規範的功能包括不同的任務功能。

- **可取得(Availability)測試**：測試獲取資源所需要的反應時間，通常在正常或最大工作負載(Workload)情況下都不同，此類測試對於常使用的線上(On-Line)應用資訊系統尤其重要。其度量可定義為：
  - Availability = [MTTF / (MTTF + MTTR)] × 100%

系統測試的測試項目種類 -5

- **建構(Configuration)測試**：測試作業系統、資料庫管理系統等支援各種不同硬體架構的狀況。
  - (1)哪種印表機與所測試的程式相容？
  - (2)哪種螢幕顯示卡或晶片組，或螢幕顯示器模組（包括解析度、顏色）或螢幕類型最適當？
  - (3)哪種麥克風或滑鼠驅動程式（包括製造廠商、版本等）可以支援？
  - (4)程式執行的最適當範圍（最大值、最小值，溫度、溼度記憶體容量等）為何？
  - (5)程式是否需要特殊型態的記憶體？快速記憶體？便宜記憶體？基本記憶體或擴充記憶體？
  - (6)程式需要在哪種等級的電腦執行？伺服器？哪種廠牌？是否需要特殊的工業電腦？
  - (7)系統軟體為何？版本？作業系統版本？視窗軟體？記憶體管理軟體？安全控管軟體？嵌入ROM的應用程式？

- **回復(Recovery)測試**：用各種方法使系統中的軟體失效，並檢驗是否有回復到正常狀態的能力。若須以人工來回復，那麼就必須測量修復的平均時間是否符合可接受的標準。

6

安全測試

- 安全測試(Security Testing)的目的在檢驗系統是否足以保護其不受到不當或非法的使用與入侵，包括外來不友善入侵及組織內部的違法或不道德行為。

- 需要安全性測試的系統一般都是含有敏感性資訊或是會對個人造成嚴重傷害或獲得好處的系統。

- 為了預防系統受到攻擊，必須時常執行資料庫備份、經常檢查並修補安全性漏洞、檢查稽核資料庫或系統紀錄(Log)。稽核資料庫必須保留從上次安全測試至今的資料。

安全性測試的種類

- 交由第三方安全測試顧問公司執行測試。
- 邀請駭客(Hacker)加入測試團隊。
- 模擬駭客使用各種方法取得密碼，進入破壞系統，或測試未知安全性漏洞，竄改稽核資料庫。
- 設計特殊軟體攻擊系統，破壞系統防禦功能，使其無法正常服務其他使用者；或故意造成系統錯誤，利用修復期間進入系統；或透過瀏覽沒有安全保護的資料，找到系統密碼。
- 偵測非法授權或滲透的系統存取。例如，使用工具監聽所有網路連線埠號(FTP, HTTP, SMTP, POP3 Ports)，執行應用程式緩衝區過載(Buffer Overflow)檢查，掃描已知安全性漏洞(Holes)，如阻斷服務(Denial-of-Service, DoS)攻擊，並分析其風險。

效能測試

- 所有的測試階段都可實施效能測試(Performance Testing)。
- 要到所有系統模組完全整合後，才能確定系統的真正效能。
- 效能測試是以**嚴謹的方式度量系統資源的效用**，例如，處理器周期，檢驗軟體執行的效能是否滿足系統的需求。
- 效能測試通常用於多人使用的系統（如：網路系統），或是具有時間要求性的系統（如：即時系統或嵌入系統）。
- 有時需要特殊的硬體或工具軟體來協助測量系統執行時的動態性質或數據。
- 效能測試一般可分為負載測試(Load Testing)和壓力測試(Stress Testing)

效能測試的程序

- 1. 分析應用程式，每分鐘需要處理多少數量的要求(Request)，每次要求或期望的反應時間(Response Time)為何？
- 2. 設定正常／最大(Normal/Maximal)案例的負載需求。
- 3. 規劃硬體、網路、資料庫、中央處理器(CPU)、記憶體的配置及設定測試停止條件。
- 4. 以多執行緒(Multi-Thread)模擬多使用者狀態，執行系統資源要求與監控。
- 5. 分析測試結果，包括連線數量(Count)、反應時間、連線失敗(Failure)數量、執行緒(Thread)數量和失敗之間的關係以及成功／失敗(Success/Fail)比率？
- 6. 若不符合測試目標，應適度調整系統配置參數，如：中央處理器、記憶體／快取記憶體(Memory/Cache memory)、磁碟機(Disk)、輸出／輸入(I/O)、網路，可允許連線會談(Sessions)數量等相關參數。
- 7. 完成測試報告。

6

負載測試

- 主要在測試電腦、周邊設備、伺服器以及網路的規格工作限制，藉以找出元件的限制、瓶頸或錯誤。
  - 1. 長時間(Longevity)：評估系統在長時間中的一般工作負載。
  - 2. 高容量(Volume)：評估系統在限制的時間內高負載情況。

- 通常的做法是在實驗室控制環境下比較不同系統的能力，或精確量測單一系統的能力，或量測系統在真實世界的功能理想值。典型的**環境控制條件**情形為：
  - 1. 從網路上下載大量的檔案。
  - 2. 在同一部電腦或伺服器上同時執行多個應用程式。
  - 3. 指定多項列印工作到印表機中。
  - 4. 給予伺服器大量的電子郵件。
  - 5. 從硬碟中持續的存取資料。
  - 6. 測試具有高速處理器的電腦系統，但卻使用有限的記憶體(RAM)

6

壓力測試

- 主要是測試電腦、網路、程式或設備在不適當的狀態下仍具有一定的效用。
  - 1. 量化評估：量測系統發生錯誤的頻率或在何種情況或時間下會當機。
  - 2. 質性評估：測試是否可以抵擋阻斷服務(Denial-of-Service, DoS)攻擊。

- 壓力測試必須控制在不利的執行環境下進行，直到系統當機或性能低於一定程度，包括：
  - 1. 在同一部電腦同時執行許多高資源要求的應用程式。
  - 2. 針對同一個網站同時製造大量的服務要求。
  - 3. 嘗試使用病毒、木馬或間諜程式感染系統。
  - 4. 測試發生異常數量、頻率或容量資源要求時的反應。
  - ……

- 壓力測試很耗時而且繁雜瑣碎，進行時可以一次僅改變一個**控制環境變數**，並做為**系統效能調校**的參考。

煙霧測試

- 煙霧測試(Smoke Testing)源自於硬體業用語，硬體元件經過修復後，開啟設備的電源，如果沒有冒煙，就表示基本功能沒有問題，元件便通過測試。

- 軟體中，煙霧測試意指在將變更簽入(**Check-in**)至產品的原始碼結構樹之前，確認程式碼是否變更的一種程序。

- 在審查過程式碼之後，煙霧測試是用來識別和修正軟體缺失最符合經濟效益的方法，以確認程式碼函式中的變更是否與預期相同，而且不會使整個元件不穩定。

煙霧測試的原則

- **與開發人員一起合作**：由於煙霧測試著重於**已變更的程式碼**，因此必須與撰寫程式碼的開發人員一起合作。
- **進行程式碼檢視**：執行煙霧測試之前，應先針對變更的程式碼進行檢視，以驗證程式碼品質，確保不會出現錯誤或功能錯誤。
- **使用正確的更新版本**：煙霧測試著重於**確認所更新的可執行程式之功能變更**，能正確的在測試環境中執行。
- **建立每日組建(Daily Build)**：每日組建需要每位開發人員配合，將程式碼保持同步。將建立每日組建設定為開發小組的最高優先順序，確實執行每日組建以及煙霧測試，以確保更新的可執行程式的品質。
- **Web測試和負載測試**：建置Web測試和負載測試時，建議在執行任何長時間且負載過重的測試之前，先執行煙霧測試。

6

驗收測試

- 透過**軟體系統操作者**來進行測試，並**使用實際完成的軟體系統為測試標的**，與**實際系統的操作數據**來進行測試，而非使用模擬的測試資料與系統。

- 根據需求分析規格中所定義的驗收準則來檢驗軟體的功能與效能是否滿足顧客的需求，一般採用黑箱測試方法進行。

- 驗收測試根據其測試的方式與測試環境的不同，又可以進一步細分為以下兩種測試。
  - 1. **阿法(α)測試**：由軟體開發人員指導使用者在系統開發所在地進行操作，開發者可同時記錄錯誤和使用時所產生的問題。
  - 2. **貝塔(β)測試**：在終端用戶處由使用者自行進行測試，開發者並不在現場，使用者須自行記錄問題並回復給開發者。

- 此階段發現的錯誤幾乎不太可能在期限內完成更改，因此須根據錯誤的嚴重性與軟體使用合約等限制，訂出雙方皆能接受的補救辦法。

6

本章總結

- 介紹軟體測試的實務方法，包括軟體驗證與確認的技術，以期所開發的軟體產品有一定的品質。
- 說明驗證與確認的意義以及實施原則和方法，包括軟體靜態分析與動態測試。
- 討論軟體測試的重要性以及軟體測試的V模型，之後根據軟體測試的流程逐一探討測試規劃與測試的執行。
- 測試的執行程序，以及IEEE測試計畫書有詳細的說明。
- 介紹各種不同的靜態分析方法與應用，對審查會議進行方式與審查評估有詳細的說明。
- 探討各種不同的動態測試方法，包括白箱測試與黑箱測試技術。
- 討論動態測試策略，包括單元測試、整合測試與各種不同的系統測試。

作業練習 (Exercises)1

1. 請根據軟體測試的V模型，說明各個軟體開發階段，可以使用什麼測試方法實施軟體驗證與確認。

2. 請根據過去的一個軟體開發經驗，依循IEEE軟體計畫書格式撰寫一份軟體計畫書。

3. 若組織或團隊希望導入軟體靜態分析流程，請針對各個軟體開發階段，設計各階段相關的審查檢驗清單。

4. 請根據過去的一個軟體開發經驗，選擇一個軟體靜態分析方法審查軟體工作產品，並說明選擇與審查的理由。

作業練習 (Exercises)2

5. 有一程式如下所示，其功能為輸入三角形三個邊，以判斷是否為正三角形、等腰三角形、直角三角形、一般三角形或不成為三角形。請設計測試案例，滿足白箱測試的敘述覆蓋。

```
public static String testTriangle (int a, int b, int c) {
    String result = "";
      if ((a<=0) || (b<=0) || (c<=0)) {
            result = "Not a Triangle \n";
      }
      else {
            result = "Be a Triangle \n";
            if (((a+b)<c) || ((a+c)<b) || ((b+c)<a)) {
                  result = "Not a Triangle \n";
            }
            else {
                  if ((a==b) ||(b==c) || (a==c))    {
                        result = "Isosceles Triangle \n";
                  }
                  if ((a==b) && (b==c))    {
                        result = "Regular Triangle \n";
                  }
                  if (((a*a+b*b)==(c*c)) || ((a*a+c*c)==(b*b))
                     || ((b*b+c*c)==(a*a))) {
                        result = "Right Triangle";
                  }
            }
      }
      return result;
}
```

作業練習 (Exercises)3

6. 有個二元搜尋法程式碼如下，請繪出程式流程圖形、計算複雜度，及找出所有獨立路徑，並設計測試個案。

```
1 int Binary_search (elem key, elem* T, int  size) {
2    int bott=0, top=size-1, mid, found=-1 ;
3    while (bott <=top) && (found==-1))    {
4        mid = top + bott / 2 ;
5        if ( T [mid] == key  )
6            found = mid ;
7        else if (T [mid] < key )
8            bott = mid + 1 ;
9        else
10           top = mid-1 ;
11   }  // while
12   return found;
13 }  //binary_search
```

作業練習 (Exercises)4

7. 請根據以下程式，畫出流程圖形，以及求出DEF和USE之集合，以導出DU測試路徑。

```
1. s = 0;
2. x = 0;
3. while (x < y) {
4.   x = x + 3;
5.   y = y + 2;
6.   if (x + y < 10)
7.     s = s + x + y;
8.   else
9.     s = s + x - y;
10.}
```

作業練習 (Exercises)5

8. 若有一程式模組可以輸入三角形三個邊長，判斷此三角形是否不成為三角形，或是等腰三角形、直角三角形、正三角形。試以黑箱測試的等價劃分技術設計測試案例資料。

9. 請根據以下功能描述，試以黑箱測試的邊界值分析技術設計測試案例資料：汽車保險公司保險費計算方式為投保金額乘以保險費率，保險費率依駕駛人狀況不同，其點數有差異。20點以上費率為5%，16點以下費率為 4%，12點（含）以下費率為 3%，點數算法如下：

   年齡：18～39歲者8點；40～59歲者5點；60歲以上者2點。

   性別：男性6點，女性 3點。

   婚姻：已婚者4點，未婚者7點。

   車齡：每滿一年加0.5點，最多加5點（四捨五入取整數）。

作業練習 (Exercises)6

10. 請根據以下規格，以黑箱測試的邊界值分析技術設計測試案例：計算N（15<N<60）個學生成績，包括國文、英文、計概，成績範圍從0～100分，成績以一維陣列儲存，學號為8位整數，姓名在4個字元以內；計算所有N個學生三科總分與平均分數，加以排序後依名次印出；並請列出計概不及格的學生學號、姓名與分數。

11. 請根據以下簡單的自動提款機系統的部分需求規格描述，以黑箱測試的因果圖技術設計測試案例。
    - 執行『查詢餘額』指令時，輸入錯誤的帳號，則印出錯誤訊息。
    - 執行『查詢餘額』指令時，輸入正確的帳號，以及錯誤的密碼，則印出錯誤訊息。
    - 執行『提款』指令時，輸入正確的帳號、密碼，但交易金額輸入錯誤，則產生錯誤訊息。
    - 執行『提款』指令時，輸入正確的帳號、密碼，以及正確的交易金額，則進行後續提款動作。
    - 執行『轉帳』指令時，輸入正確的帳號、密碼，則進行後續轉帳動作。

作業練習 (Exercises)7

12. 請根據以下的需求規格，以黑箱測試的因果圖技術設計測試案例：飲料單價為5元的自動販賣機，投入5元或10元，並按下「可樂」或「果汁」按鈕後，所選取的飲料會掉出來。如果販賣機沒有零錢，則顯示「零錢找完」的紅燈，此時投入10元後按下按鈕，會掉出10元，不會掉出飲料。如果有零錢可找時，「零錢找完」的紅燈會熄滅，投入10元後按下按鈕，會掉出所選取的飲料，以及5元。

13. 請說明單元測試與整合測試的區別，包括執行的人員與工作內容。

14. 以下系統應該考量哪些系統測試？請簡要說明其理由。
    - 數位照片網路沖洗系統。
    - 捷運換幣系統。
    - 自動提款機。
    - 手機線上遊戲。
    - 桌上型線上遊戲系統。
    - 大學聯考電腦查榜系統。

15. 甲金融銀行打算建置網路銀行系統，提供客戶網路帳戶查詢與轉帳的功能。請替該銀行設計效能測試計畫。（假定測試目標為交易平均處理時間不超過5毫秒，該銀行有100萬名客戶，與該銀行系統有3台資料庫伺服器和4台應用程式伺服器。）