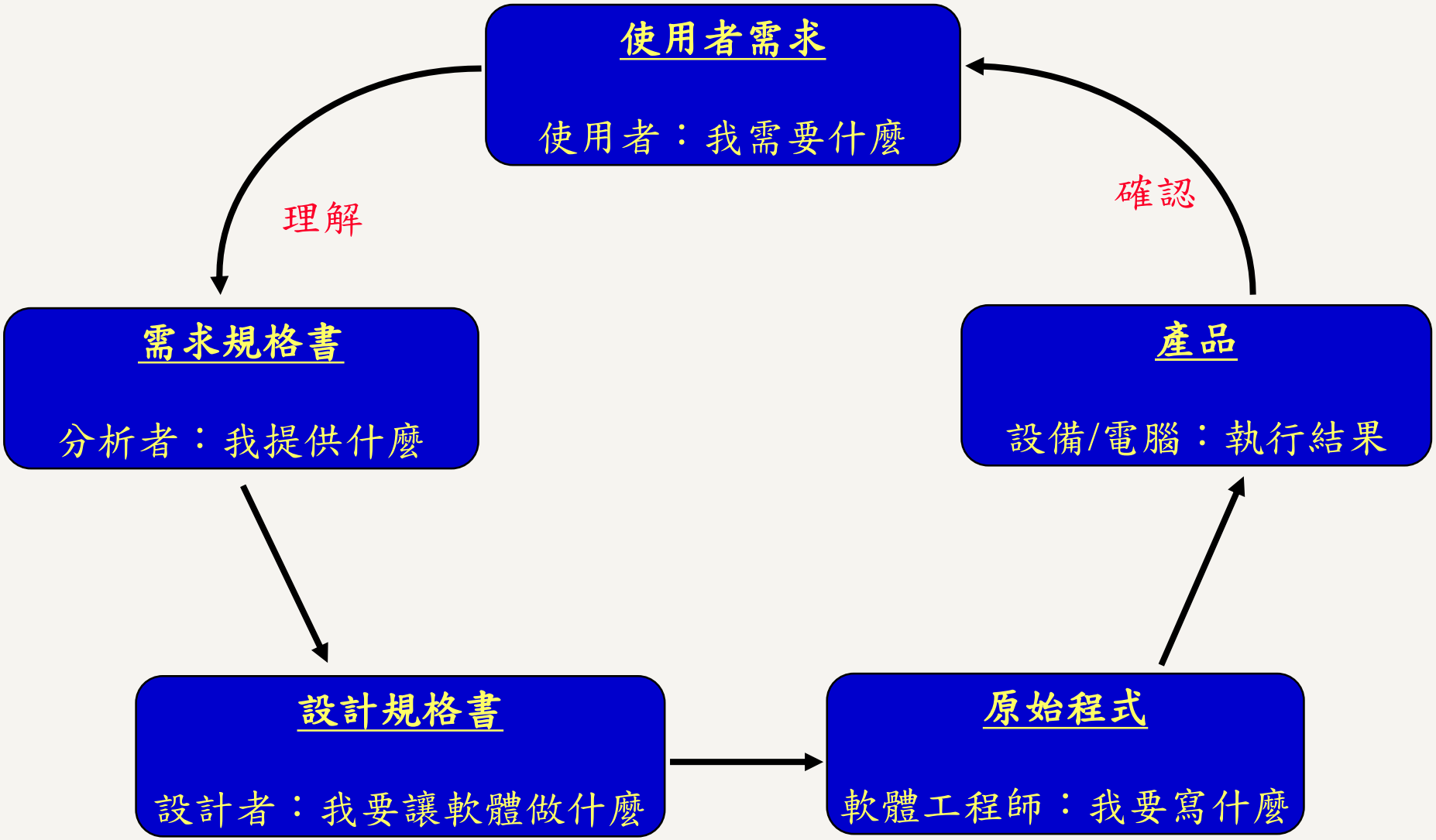
軟體測試

長庚大學資訊工程系
林仲志博士
cclin@mail.cgu.edu.tw

重點

- 何謂V&V
- 測試流程
- 何謂黑箱、白箱、動態與靜態測試
- 何謂單元、整合、系統測試
- 單元測試工作重點
- 整合測試進行方式
- 列舉不同測試項目與目的 (page 55 - 56)
- 軟體缺陷分類

軟體開發生命週期


使用者需求
使用者：我需要什麼
理解
確認
需求規格書
分析者：我提供什麼
產品
設備/電腦：執行結果
設計規格書
設計者：我要讓軟體做什麼
原始程式
軟體工程師：我要寫什麼

IEEE 830 軟體需求規格書建議

- 簡介
  - 系統目的
  - 系統範圍
  - 名詞定義、縮寫
  - 現有系統限制
  - 參考
  - 系統概觀
- 系統整體性描述
  - 整體系統概述
  - 系統操作環境平台
  - 產品角度
  - 產品功能
  - 系統使用者
  - 系統限制
  - 系統假設
  - 使用者作業活動
  - 遵循及參考標準
  - 驗收前應準備之文件

IEEE 830 軟體需求規格書建議

- 需求詳述說明
  - 外部介面需求
    - 操作概念與腳本
    - 使用者介面
    - 硬體介面
    - 軟體介面
    - 溝通介面
  - 功能性需求
    - 分類1（例如：子系統）
      - 功能性需求 R101
      - 功能性需求 R109
  - 效能需求
  - 設計限制
  - 軟體系統屬性（特性）
  - 其他需求
- 附錄

非功能性需求

- 效能、安全性、可靠性與可維護性
- 交付、安裝與環境需求
- 設計與實做限制
- 測試需求與驗收標準
- 技術/標準限制
- 風險控管

需求品質準則

- 明確且適當地陳述(Clearly and properly stated)
- 完整性 (Complete)
- 一致性 (Consistent)
- 能個別界定 (Uniquely Identified)
- 能適當地執行 (Appropriately implement)
- 能驗證 (Verifiable)

Software Design Specification(軟體設計規格)

- SRS與SDS之間的關係：SRS → 描述軟體設備將可以作些什麼；SDS → 這些對於軟體設備的需求，要如何被實作出來
- 內容的呈現必須要充分，確保軟體工程師開發軟體設備時，可以清楚了解並使用最少的設計決策實作
  - 設計方法與工具
  - 組織架構
  - 系統流程
  - 軟體元件規格
  - 界面設計規格
  - 資料結構設計規格

Traceability Analysis(可溯性分析)

- Traceability Analysis文件內容可利用表格的呈現方式，將設計的需求、規格以及測試需求連結在一起，也可將鑑定出的危險、實作以及防治危險方面的測試聯繫在一起，有助於內容的連貫性，提升檢視文件效率。

設計需求	對應之設計規格	對應之V&V測試	軟體缺陷風險
SAS004	SDS101	V&V200	FMEA002

Software Development Environment Description (軟體開發環境描述)

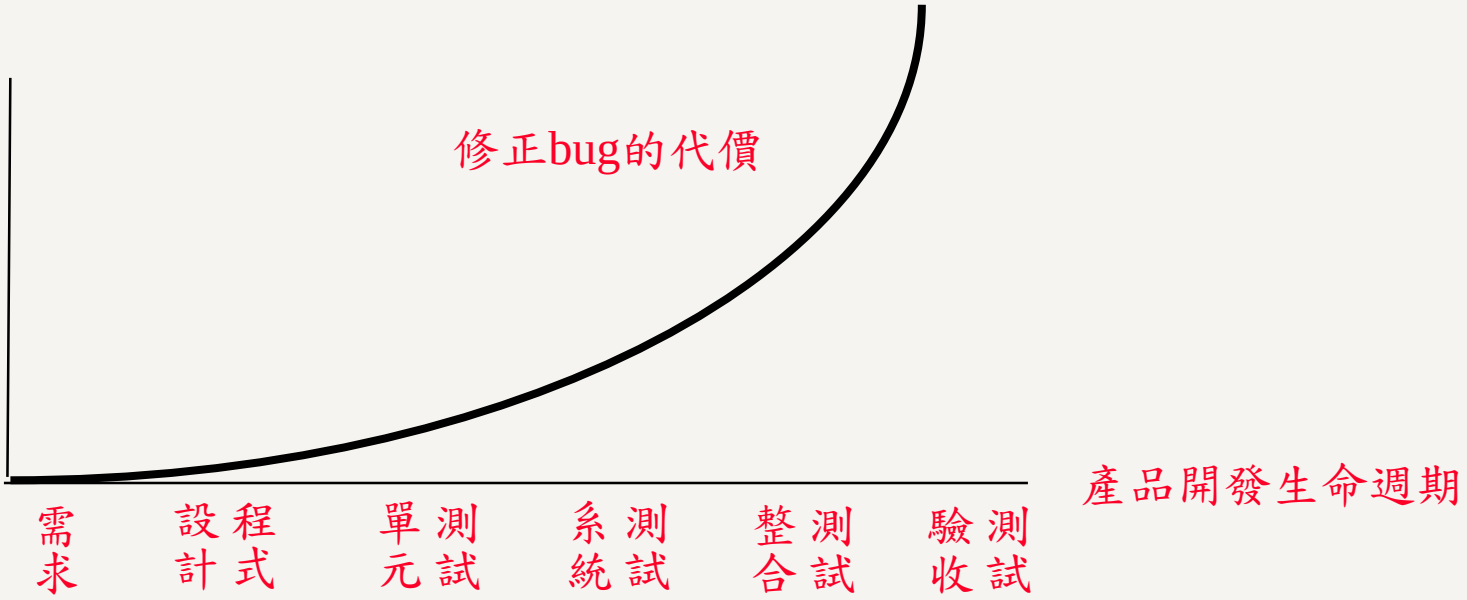
- 軟體發展：軟體發展生命週期計畫之摘要，軟體發展生命週期如Water fall模式、V型模式、螺旋模式等，訂定流程與時程計畫
- 軟體專案品質管理
  - 專案管理計畫：說明對於整體專案的管理方式，如時程規劃(可使用甘特圖表示)、人力資源、管理流程/策略、check point(管制點)等
  - 軟體測試計畫：說明軟體測試計畫，以確認其符合規格及客戶需求
  - 建構與維護管理計畫 ：說明如何管理相關系統標準與文件，並記錄軟體系統發展之改變狀況、各元件間之相互關係，以及評估、追蹤、記錄軟體元件之改變，區分不同版本之演變。

軟體測試的迷思

- 由於大多數企業缺乏軟體測試與實踐之知識，所以對軟體測試工作容易有以下幾點誤解
  - 軟體品質有問題，全部是軟體工程師的錯
  - 文件化過程太浪費時間，口頭說說就好
  - 軟體測試技術要求不高，隨便找一個人就能做
  - 等到產品開發最後階段才進行測試

關於軟體測試

- 依據過去經驗每千行程式碼大約有60個缺陷，2/3缺陷在需求與設計階段，在這個階段發現問題的修正費用最少，如果到系統測試才發現，要花10倍以上經費，若到產品驗收時期，則需花費100倍以上
- 美國國防部要求每千行0.01以下的錯誤，電信/銀行之系統平均每千行0.05個錯誤，一般企業軟體為每千行0.5個錯誤



軟體測試人員 Vs 軟體開發人員

- 微軟為例
  - 2000年全球52,000員工，10,000開發人員，15,000測試人員
  - 測試費用佔60%
  - Exchange研發：開發人員140人，測試人員350人
  - Windows 2000研發：開發人員1,700人，測試人員3,200人
  - IE4產品研發：開發時間6個月，測試時間8個月

測試需要花費龐大的人力、物力與時間

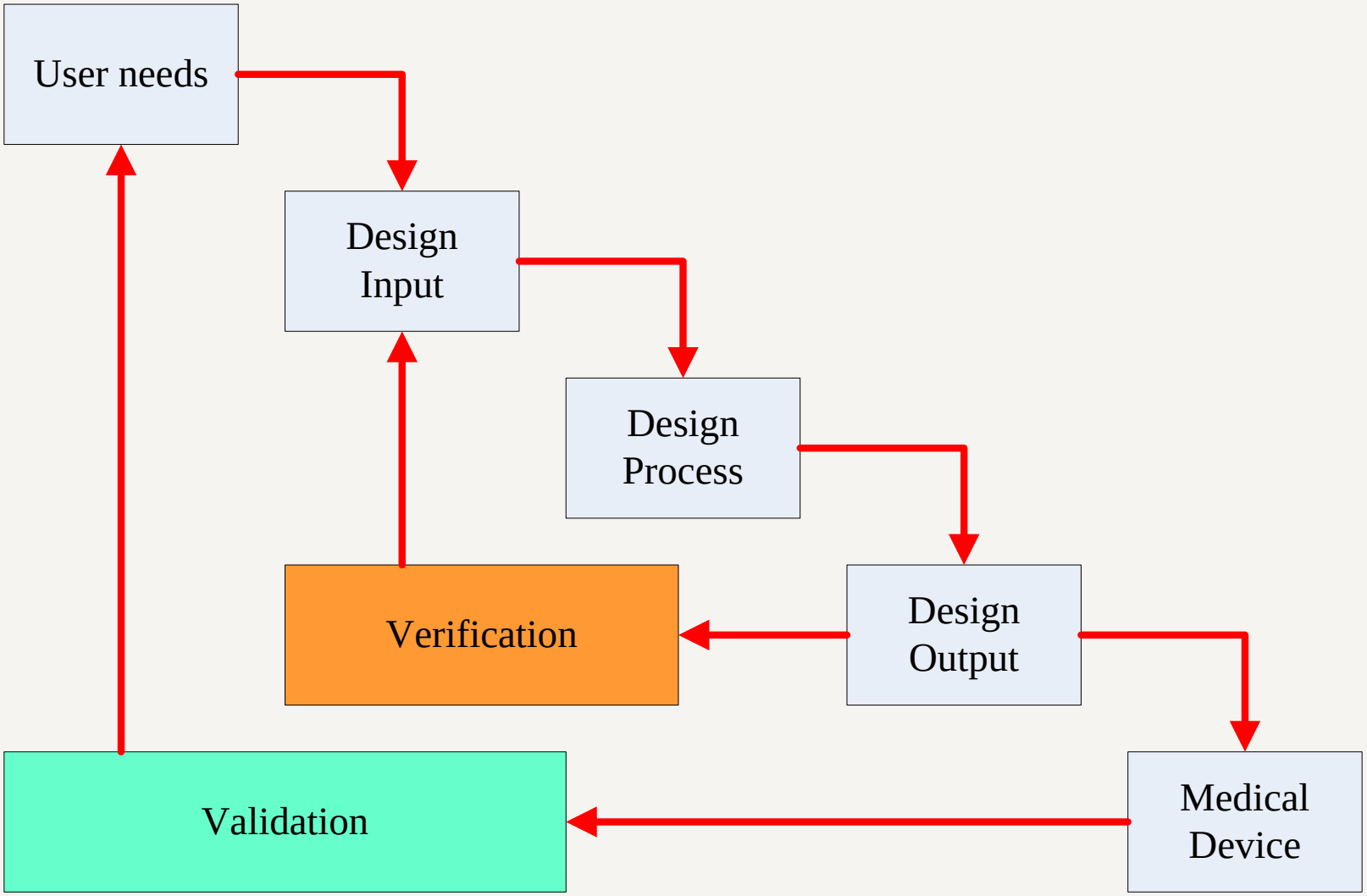
何謂軟體驗證

- 軟體驗證是透過產品發展生命週期活動來評估軟體產品品質的嚴謹方法
- 軟體驗證的工作將努力確保品質被導入軟體產品之中，並讓軟體滿足所需達到的功能與使用者之需求
- 軟體驗證的工作將包括產品與發展程序之分析、評估、審核、檢視、測試等
- 實務運作必須將軟體工程(Software Engineering)與品質管理系統相結合

Verification and Validation

- 驗證(Verification)
  - 保證軟體產品可以正確實現某一功能
  - 軟體開發生命周期中每個階段的正確性與完備性
  - Are we building the product right?
    - 我們是否正確地開發了產品
- 確認(Validation)
  - 保證軟體符合功能需求
  - 需求規格的確認，軟體邏輯性的確認
  - Are we building the right product
    - 我們是否開發了正確的產品

Verification and Validation



Verification and Validation Documentation (確認與驗證文件)

- 使用列表的方式，摘要說明如何針對單元/整合/系統三種levels進行V&V之相關測試，該測試須與危險分析相互對照。
- 報告內容
  - 測試項目編號
  - 軟體版本
  - 測試人員、檢視人員
  - 測試項目名稱、測試目的、設備與使用工具
  - 測試方法、輸入規格、輸出規格、環境需求
  - 測試通過標準、測試步驟描述、測試結果

Revision Level History(校訂版本歷史紀錄)

- 內容須包含產品發展過程中，軟體修正的歷史
- 可利用條列式表格的方式呈現發展週期中軟體更改的主要內容，包含日期、版本編號，以及簡短描述此版本的更動與前一版本之間的關係

修訂編號	修改日期	軟體版本	主要更改內容	與前一版本之關係

軟體測試與驗證項目

測試流程

- 測試計劃 (test plans)
  - 測試的範圍、方式、資源與排程
- 測試設計 (test design)
  - 設想要測試的功能與方式
  - 要能夠完成回歸測試(regression test)
- 測試案例 (test cases)
  - 執行最精簡的測試案例
- 測試程序 (test procedure)
  - 測試系統之執行步驟
- 測試執行 (test execution)
  - 從元件測試開始，然後進入整合、系統與驗收測試
- 測試報告 (test report)
  - 摘要所有測試結果

軟體測試的分類方法

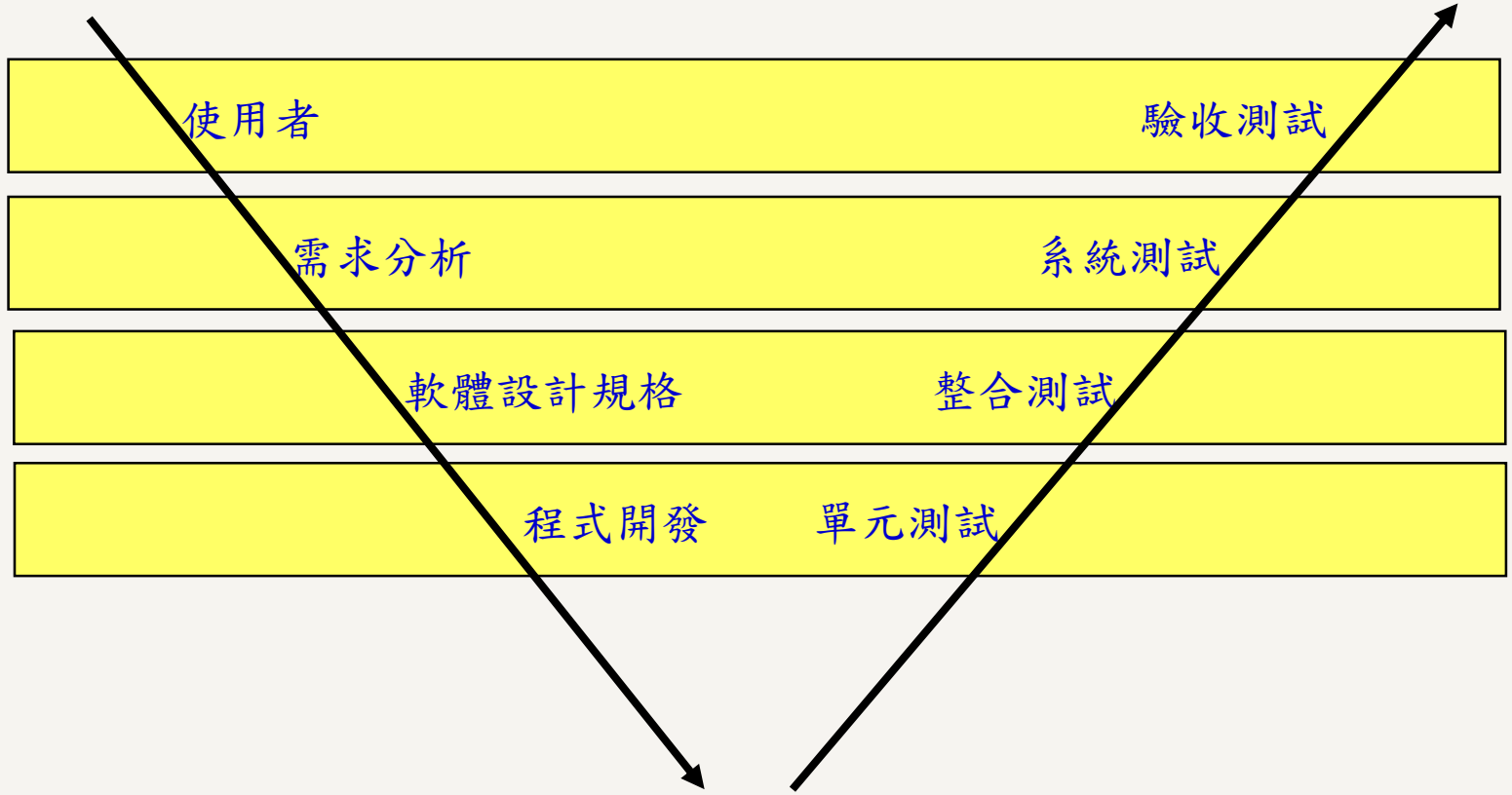
- 白箱測試：著重結構測試
  - 動態：使用測試資料進行測試
    - 測試邊界
    - 結構測試 (路徑涵蓋)
  - 靜態：不用執行軟體
    - 程式證明
    - 異常分析
- 黑箱測試：注重功能測試
  - 動態
    - 決策表格架構測試
    - 因果圖
  - 靜態
    - 規格證明

軟體缺陷分類屬性

- 缺陷嚴重度 (Severity) ：對軟體的嚴重程度
- 優先順序(Priority)：緊急修復順序
- 缺陷狀態(Status)：處裡狀況
- 缺陷起源(Origin) ：事件如何被發現
- 缺陷來源(Source) ：缺陷的起因
- 缺陷根源(Root Cause) ：根本因素

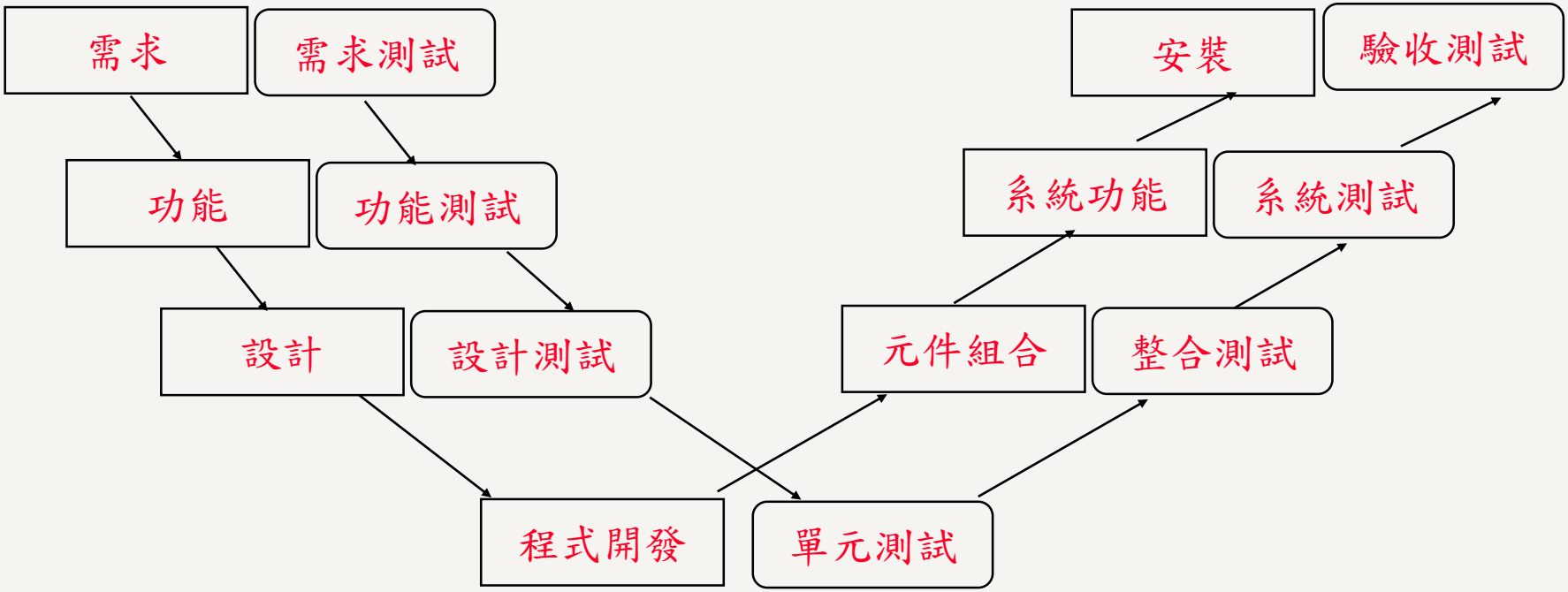
測試模型 – V Model

- 需求分析、軟體設計、程式開發等步驟隨時間依序進行，而測試的順序正好相反



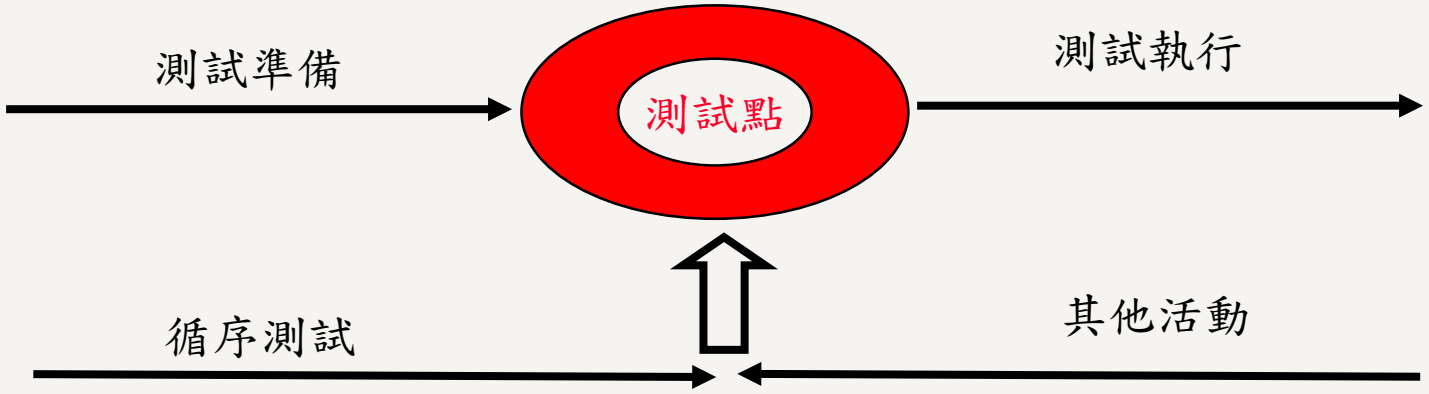
測試模型 – W Model

- 修正V model兩個缺點
  - 測試是軟體開發後期的工作
  - 要測試的對象只有軟體
- 伴隨開發週期同步進行測試



測試模型 – H Model

- W模式將測試視為一個獨立的活動問題，H模式則將測試視為一個系統流程
- 測試活動貫穿整個產品週期
- 測試活動可以依序先後執行，但也可能反覆的執行



軟體測試指引

- 判斷何時停止測試
- 清楚了解使用者需求、操作流程、系統設計與元件介面等規格
- 指定測試者測試的責任
- 描述每種測試情況的預期結果
- 避免無法重複產生,或是無規律的測試
- 寫出有效與無效輸入條件的測試案例(test case)
- 進行測試

軟體測試的分類方法

- 白箱測試：著重結構測試
  - 動態：使用測試資料進行測試
    - 測試邊界
    - 結構測試 (路徑涵蓋)
  - 靜態：不用執行軟體
    - 程式證明
    - 異常分析
- 黑箱測試：注重功能測試
  - 動態
    - 決策表格架構測試
    - 因果圖
  - 靜態
    - 規格證明

窮舉測試

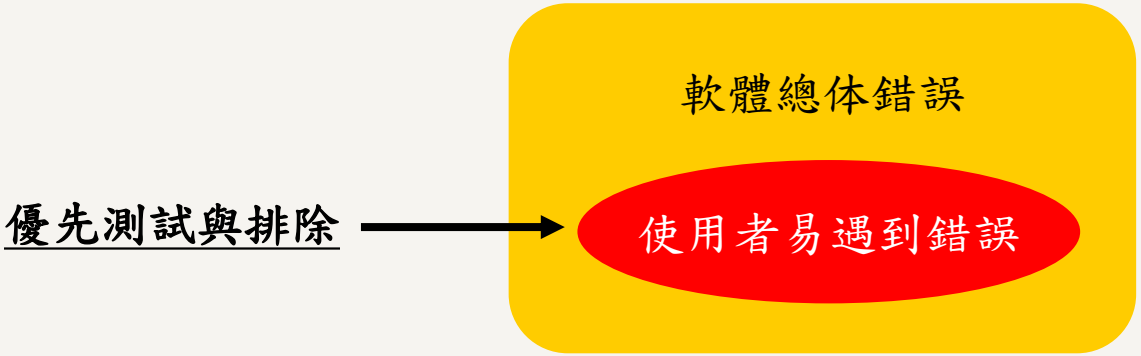
- 窮舉測試定義：將所有可能的輸入資料全部拿來做測試，或是覆蓋所有程式可能執行的路徑
- 測試範例將是天文數字
- 說明：假設一個軟體有兩個輸入、一個輸出。如果兩個輸入為32位元的整數，利用窮舉法共有$2^{32} * 2^{32} = 2^{64}$情況，如果測試一次需要1毫秒，共需5億年

靜態分析

- 靜態分析：不實際執行程式，而是透過檢查與閱讀等方式來發現錯誤的軟體測試技術，以及評估是否確實依照規劃執行邏輯順序
- 方式
  - Walk Through (快速閱讀)：經驗豐富的開發人員，經由開發者的解釋，檢視軟體的邏輯錯誤、程式碼規範，將人腦充當電腦
  - Inspection (檢閱)：以會議形式進行，明定會議目標、流程與規定、採用check list方式進行錯誤檢視
  - Review(複審)：比inspection更嚴謹，第一步提供相關文件，以及常見錯誤清單。第二步召開審查會議，開發者透過講解發現程式中的錯誤。

測試範例

- 測試範例定義：由一對滿足測試情境的輸入/輸出資料所組成，透過這個資料範例，可以執行某個層面的軟體測試
- 測試範例隨測試方法不同而不同
- 軟體中的錯誤很多，考量時間與經費不可能全部修正，所以應該對使用者容易發生的錯誤進行測試
- 測試範例是為了提高有效的測試比率



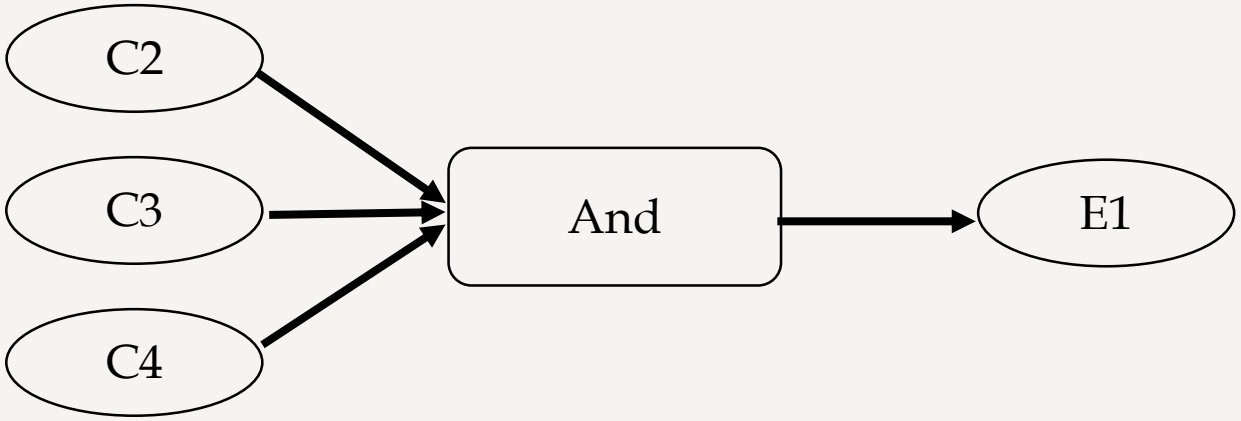
常用的黑箱測試方法 -決策表格

- 基本原理：這種測試特別適用於軟體需求是以”if – then”為敘述的系統架構
- 例如： if cond(1) then action(A) else action(B)
- 決策表：包含了所有測試情況的欄位，上半部為必須滿足的條件，下半部為產生動作
- 缺點：分開考慮所有輸入內容，會出現一些沒有必要的組合

條件	
條件1	1
條件2	1
條件3	0
條件4	1
條件5	0
條件N	…
條件N+1	0
動作	
動作1	0
動作2	0
動作3	1
動作4	0
動作5	0

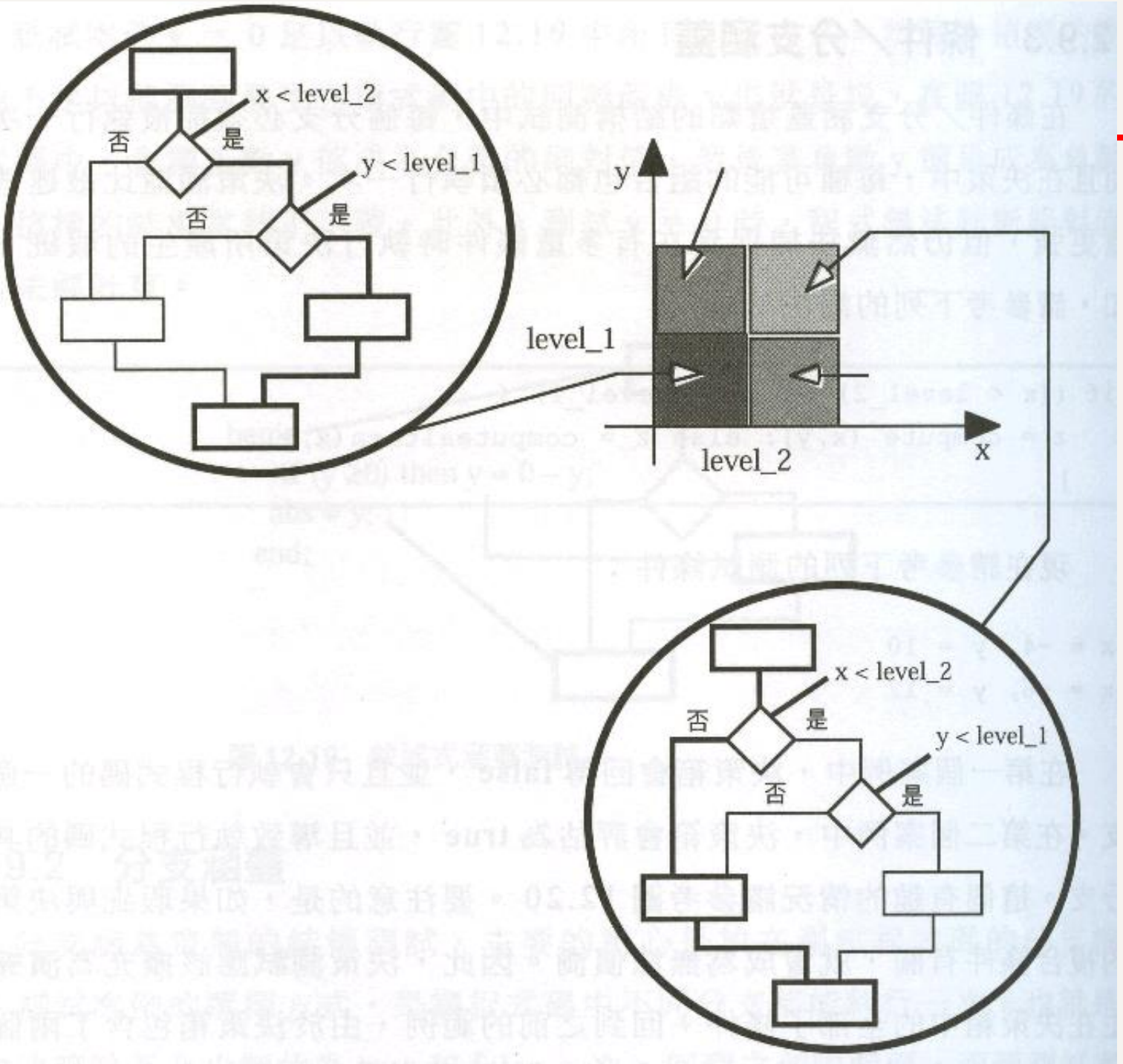
常用的黑箱測試方法－因果圖

- 指定輸入(因)與輸出(果)的組合，圖形包含了相互有關係的結點，用以表示邏輯關係
- 因果圖在刪除沒有關係的輸入點後，可以產生一個精簡的決策表
- 符號
  - 利用"C"表示原因、以"E"表示結果
  - 例如有四個原因C1、C2、C3、C4，但E1僅與C2、C3、C4有關，則可以畫出以下因果圖

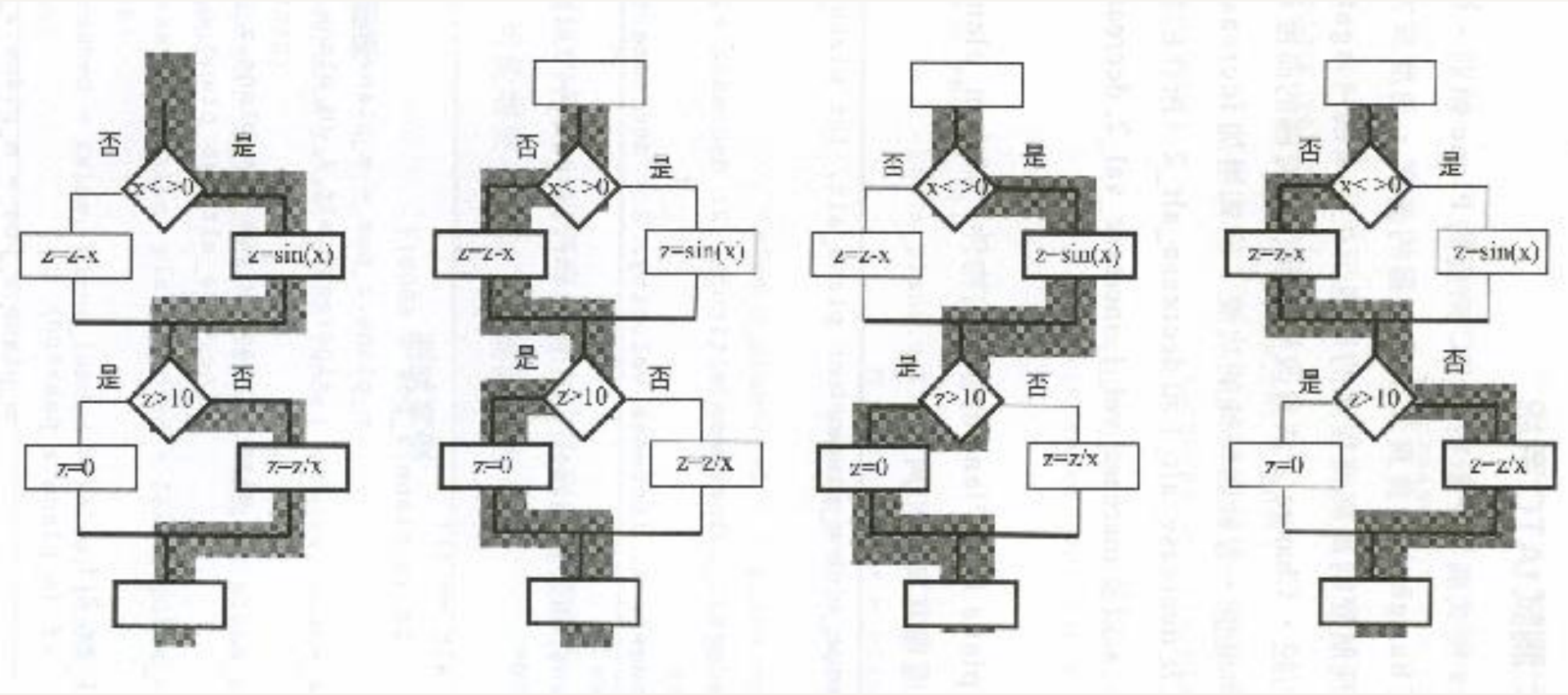


常用的白箱測試－路徑覆蓋

- 路徑覆蓋測試就是設計足夠的案例，覆蓋程式中所有可能的路徑，真實世界很難做到，必須將路徑數目壓到一定限度
- 路徑覆蓋的複雜度
  - N+1 < 路徑數目 < 2^n


x < level_2
否
是
y < level_1
否
是
y
level_1
level_2
x
x < level_2
否
是
y < level_1
否
是

路徑覆蓋



常用的白箱測試－邊界值測試

- 從經驗得知，錯誤常發生在輸入或是輸出範圍的邊界上，因此利用邊界值的測試，可以找到更多的錯誤。
- 邊界值測試是白箱測試一個有效的方法
- 例如三角型三邊(A、B、C)，兩邊和大於第三邊，如果在邊界值出現”A+B=C“，就出現錯誤
- 方法：
  - 確定邊界值
  - 測試值的選取：等於、略小於、略大於邊界值
  - 如果輸入條件規定了參數個數，則用最大參數個數、最小參數個數、最大參數個數加1、最小參數個數-1
  - 如果測試資料為一集合，測試時選擇集合中第一個與最後一個作為測試範例

單元測試

- 單元測試是對軟體基本組成單元進行測試，可以是一個函式、功能等具有基本屬性之"單元"
- 重點在於發現程式實作的邏輯錯誤，也就是要發現程式模組內部的各種錯誤
- 單元測試不僅要白箱測試，同時也要有黑箱測試
- 單元測試必須是可重複的，因為程式碼修改、升級與維護都需要反覆執行
- 單元測試與撰寫程式所花的時間與精力大致相同。雖然會花費不少時間與成本，但卻對整個產品的測試有重大意義，因為bug越晚發現，所有修正的成本越高

單元測試的內容

- 測試者要依據詳細的設計規格書，了解模組的IO條件和邏輯架構，主要採用白箱測試，黑箱測試之案例為輔，使之對於合理與不合理之輸入都能鑑別與回應
- 測試內容
  - 模組介面：檢查進出模組的資料是否正確
  - 區域資料結構測試：檢查資料結構是否保持完整性
  - 執行路徑測試：檢查計算錯誤、判斷錯誤、流程控制錯誤產生的程式錯誤
  - 錯誤處理測試：內部錯誤處理是否有效
  - 邊界測試：檢查臨界資料是否正確

單元測試內容 - 模組介面

- 參數的數入個數、型態、順序
- 所測的模組，如果有呼叫到其他子模組，需測試這些參數是否匹配
- 是否修改到僅作為輸入用的參數
- 輸出的參數個數、型態、順序是否正確
- 全域變數的定義在各模組間是否一致

單元測試內容 -區域資料結構測試

- 檢查不一致或不正確的資料類型
- 使用尚未初始化的變數
- 錯誤的初始值或是預設值
- 變數名稱拼寫錯誤
- 檢查不一致的說明資料

單元測試內容 -執行路徑測試

- 運算的優先順序不正確
- 物件的資料型態彼此不相容
- 演算法錯誤
- 運算精度不夠
- 運算符號表示錯誤，邏輯符號表示不正確
- “差1的錯誤”，迴圈數少一次，或是多一次
- 錯誤或不可能的終止迴圈
- 不小心修改迴圈的變數

單元測試內容 -錯誤處理測試

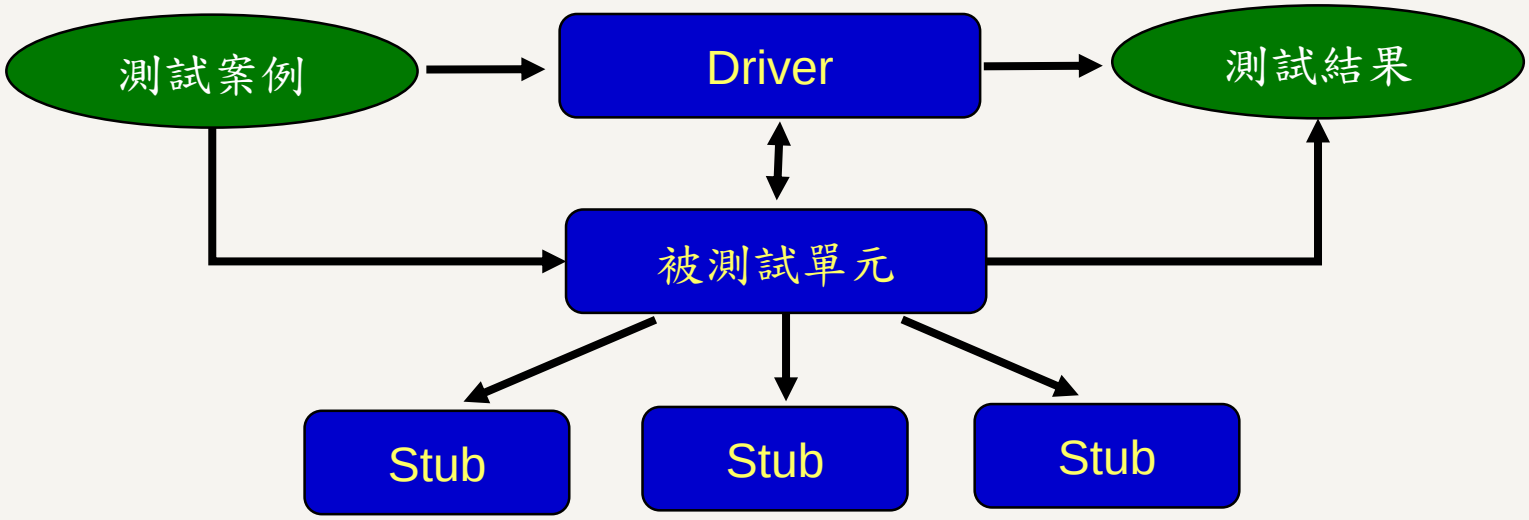
- 出錯的描述不足以對錯誤定位和確定錯誤的原因
- 顯示的錯誤與實際的錯誤不符
- 對錯誤的處理不當
- 在錯誤處理之前，錯誤已經引起其他系統單元的錯誤

單元測試內容 –邊界測試

- 在n次迴圈中測試第0次,第1次,第n-1次,第n次,第n+1次
- 邏輯判斷中取條件的最大或是最小值是否有錯誤
- 資料流程中在等於、小於、大於這些比較值條件下執行是否正確

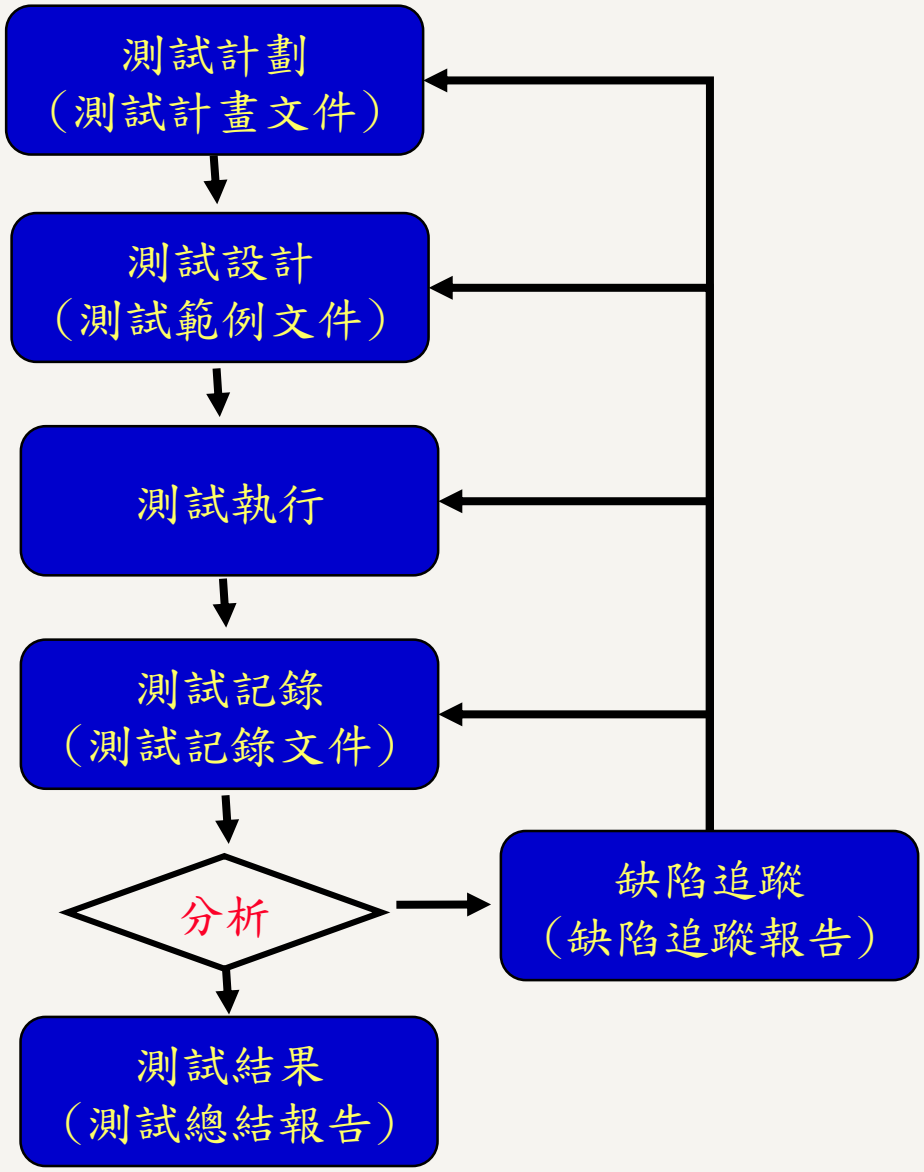
單元測試的環境

- 在單元測試時，如果該測試模組不是獨立完整的程式，需要兩個輔助的模組
- 驅動模組(Driver)：接收資料後轉傳給測試單元，再接收輸出結果
- 樁模組(stub)：代替測試單元所呼叫的子模組



單元測試實施步驟與文件

- 測試計劃：針對測試目標，規定測試範圍、環境、人員分工與時間表
- 測試設計：根據測試計劃，設計測試範例包括測試步驟、測試場景、測試程式與資料、預期結果
- 測試執行：執行測試設計
- 測試記錄：記錄執行過程與結果
- 缺陷追蹤：記錄、評估與分發缺陷報告
- 測試結果：評估最後的測試品質與效果



單元測試文件範例

- 測試計劃基本資料
  - 測試單元名稱
  - 完成測試日期
  - 測試摘要
  - 測試目的
  - 測試範圍
  - 測試類型
  - 測試環境
  - 被測元件原始程式碼位置
- 測試範例
  - 範例編號
  - 範例情境描述
  - 測試步驟
  - 測試資料說明
  - 期望輸出結果
- 測試記錄
  - 測試人員
  - 測試時間
  - 測試內容：如路徑測試、介面測試、宣告測試..等
  - 使用測試範例編號
  - 輸出結果
  - 測試觀察符不符合期望結果
- 缺陷追蹤報告
  - 嚴重程度
  - 錯誤詳細描述與重現方式
  - 建議修改方式
  - 修改人員與修正時間
  - 執行狀況：提交、複審、修改中、修正完畢
- 測試總結
  - 缺陷統計
  - 是否通過單元測試

整合測試

- 整合測試：依照設計規格，對所有需要組裝的單元模組進行整合測試，又稱為組裝測試或是聯合測試
- 測試考量
  - 模組組裝時，穿越模組介面的資料是否正確
  - 每個功能組裝起來後，會不會造成另一個模組不良的結果
  - 各個子模組整合起來，是否達成總體功能要求
  - 全域變數是否有問題
  - 單個模組的錯誤是否有累積的效應

整合測試與單元測試關聯與區別

- 整合測試對象是模組間的整合關係，找出與軟體設計相關的程式結構、模組呼叫關係、模組介面方面的問題
- 單元測試主要是模組內部的白箱測試
- 整合測試以程式結構測試為主，發現軟體與系統定義不符合或是與之矛盾的地方，測試方式結合黑箱與白箱測試，但以黑箱測試為主

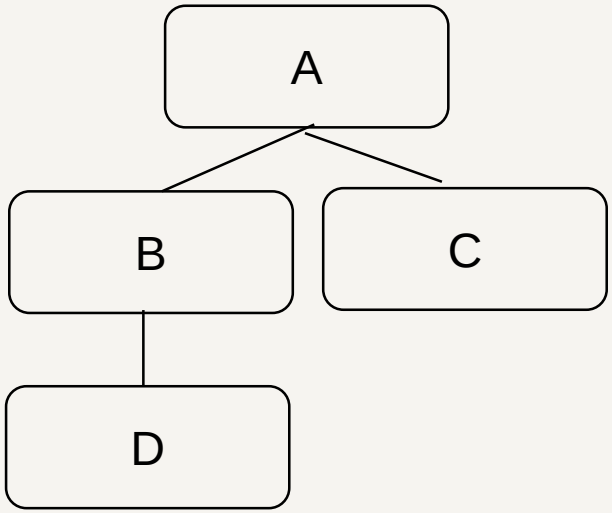
整合測試步驟

- 首先確定子系統有哪些模組，保證這些模組都通過單元測試
  - 主要依據"軟體設計說明書"
- 由開發人員組裝這些模組，生成一個子系統，並保證各模組都能正常發揮功能
- 設計測試範例，以一個關鍵模組為核心展開測試。測試重點以功能與性能為主軸
  - 擬定測試計劃：進度安排、測試範例、人員分配
- 搭建必要的測試環境，按照所寫的測試範例，進行測試
- 記錄測試結果
  - 問題記錄、問題解決追蹤報告、測試總結

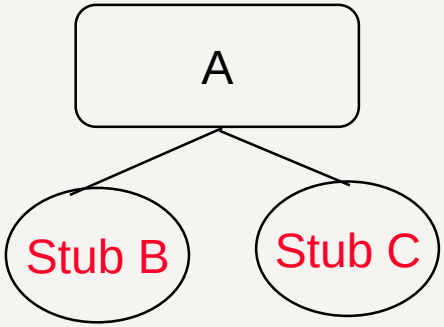
整合測試系統建置方式

- 一次性整合方式：又稱整體組裝，首先將每個模組分別進行模組測試，然後再把所有模組裝在一起，達成所要的測試。
  - 缺點：實務上一次整合成功的機會不大，即便發現錯誤，有時很難判定出問題的模組為何
- 由上而下的增殖方式：將模組沿控制層次由上而下進行整合
  - 優點：較早驗證了主要控制流程與邏輯判斷點
  - 缺點：需要模擬一些Stub，這點在實務上很困難
- 由下而上的增殖方式：從程式最底層模組開始組裝，意即先完成子模組，好處是不需要開發模擬的stub
  - 優點：一些複雜的IO以及演算法都是底層模組，可以優先找出這部份的問題
  - 缺點：程式一直沒有實際的主體，直到最後一個模組加上去才形成實體。對主要的控制流程最後才接觸到

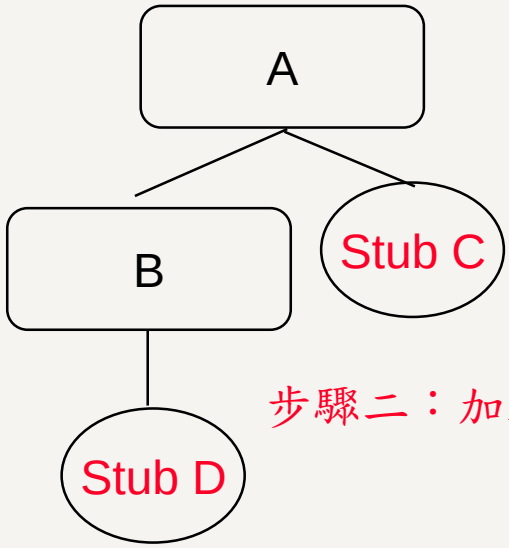
由上而下的增值方式示意圖



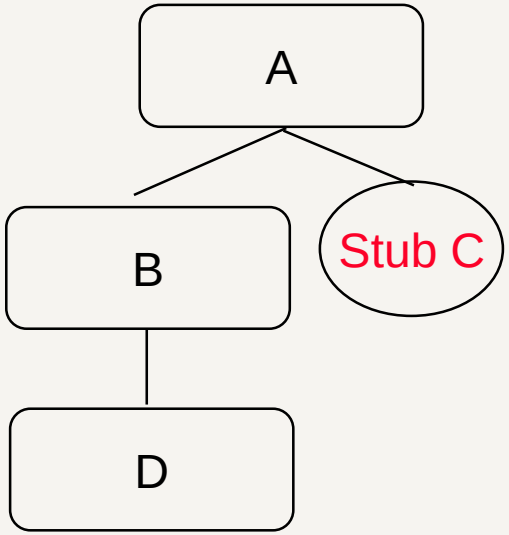
期望測試



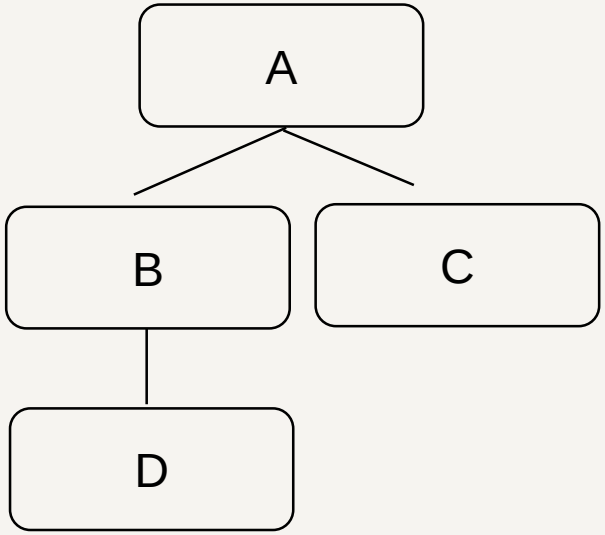
步驟一：加入A



步驟二：加入B



步驟三：加入D



步驟四：加入C

混合增殖式測試

- 目的：整合”由上而下增殖方式”與”由下而上的增殖方式”兩方法的優點
- 常見的方式
  - 衍變的由上而下的增殖方式：加強對IO模組以及重要演算法模組之測試，並由下而上整合功能完整且獨立的子系統，然後由主模組開始由上而下進行增殖測試
  - 由上而下 & 由下而上增殖測試：首先對資料”讀取”這部份子系統由下而上進行模組整合測試，然後再對資料”寫入”的子系統做由上而下的整合測試

整合測試之關鍵模組

- 測試者需確定關鍵性模組，對這些模組及早進行測試
- 關鍵性模組的特徵
  - 明確定義為軟體功能需求
  - 在程式結構中屬於類似控制模組這種較高層次的模組
  - 較複雜易發生錯誤的模組

系統測試

- 主要目的驗證整體系統是否滿足當初所擬之系統需求規格之定義
- 整個測試的範圍除了軟體外，還包括硬體、資料、操作人員和其他支援軟體
- 以黑箱測試為主
  - 規格測試、輸入/輸出測試、功能測試

系統測試與單元、整合測試區別

- 測試方法：系統測試完全以黑箱測試為主
- 測試範圍：單元測試主要測試模組內的介面、資料結構與邏輯等。整合測試主要測試模組之間的介面與異常。系統測試主要測試系統是否滿足使用者的需求
- 評估基準不同：系統測試主要評估基準是測試範例對需求規格的覆蓋率，單元與整合測試主要的評估是程式碼的覆蓋率

系統測試的內容 (I)

- 功能測試：
  - 對於產品功能進行測試，驗證符合需求規格書
- 效能測試
  - 驗收系統的整體效能是否達成目標，一般與負載測試結合
- 負載測試
  - 在大量資料、大量存取情況下，評量系統的效能與功能穩定度
- 壓力測試
  - 在人為資源緊缺的環境下(如CPU佔用、記憶體減少、網路頻寬限制)，檢查系統是否會發生問題
- 疲勞測試
  - 連續保持長時間的測試，檢查系統是否會出現問題
- 易用性測試
  - 操作介面是否簡易、操作流程是否一致性
- 安裝測試
  - 檢查安裝時是否正確安裝所有檔案，是否會破壞其他檔案
- 配置測試
  - 在不同環境下(如作業系統、軟體環境)，驗證系統的功能
- 文件測試
  - 文件是否齊全，內容格式是否一致

系統測試的內容 (II)

- 安全測試
  - 檢查系統是否有病毒，系統資料是否正確加密，並具有權限管理機制
- 恢復測試
  - 系統發生災難時，檢查系統是否能夠回復破壞的資料與環境
- 回歸測試
  - 當系統更新部份程式碼時，是否會引入新的錯誤或者舊的錯誤重新出現
- 演練測試
  - 交付使用者之前，利用相似的環境進行測試
- Back-to-back測試
  - 設置一組測試，在不告知任何事情的狀況下，獨立進行測試，用來評估測試團隊的效果
- 度量測試
  - 人為放入錯誤，並根據被發現的比例來確定遺留的錯誤數量
- 比較測試
  - 與競爭產品或是舊版產品做比較，來確定系統的優勢與劣勢

系統測試活動過程

活動名稱	輸入	輸出
制定系統測試計劃	軟體需求文件 軟體專案計畫	系統測試計畫
設計系統測試	系統測試計畫 軟體需求	系統測試範例 系統測試過程
實施系統測試	系統測試計畫	系統測試腳本
執行系統測試	系統測試計畫 系統測試範例 系統測試過程 系統測試腳本	測試結果
評估系統測試	測試結果	分析報告 修改變更請求

軟體測試是一個藝術與工程的結合

- 20世紀60年代開始有測試任務，80年代開始有測試職業，90年代開始有測試科學，21世紀開始有測試專業
- 測試的目標是發現問題
- 測試在理論與方法都不是很成熟，效果取決於測試資源、團隊能力，所以說軟體測試是一場戰爭

軟體測試的改進方法

- 外聘更多的測試人員
- 將原有開發人員轉任為專責的測試人員
- 加強所有人對於軟體測試的專業知識
- 購買或自行開發軟體測試工具
- 將測試工作外包

Thanks For Your Attention

長庚大學資訊工程系 林仲志
cclin@mail.cgu.edu.tw
(03)2118800-5964
0922743818