


軟體工程實務
Class 4: Software Testing

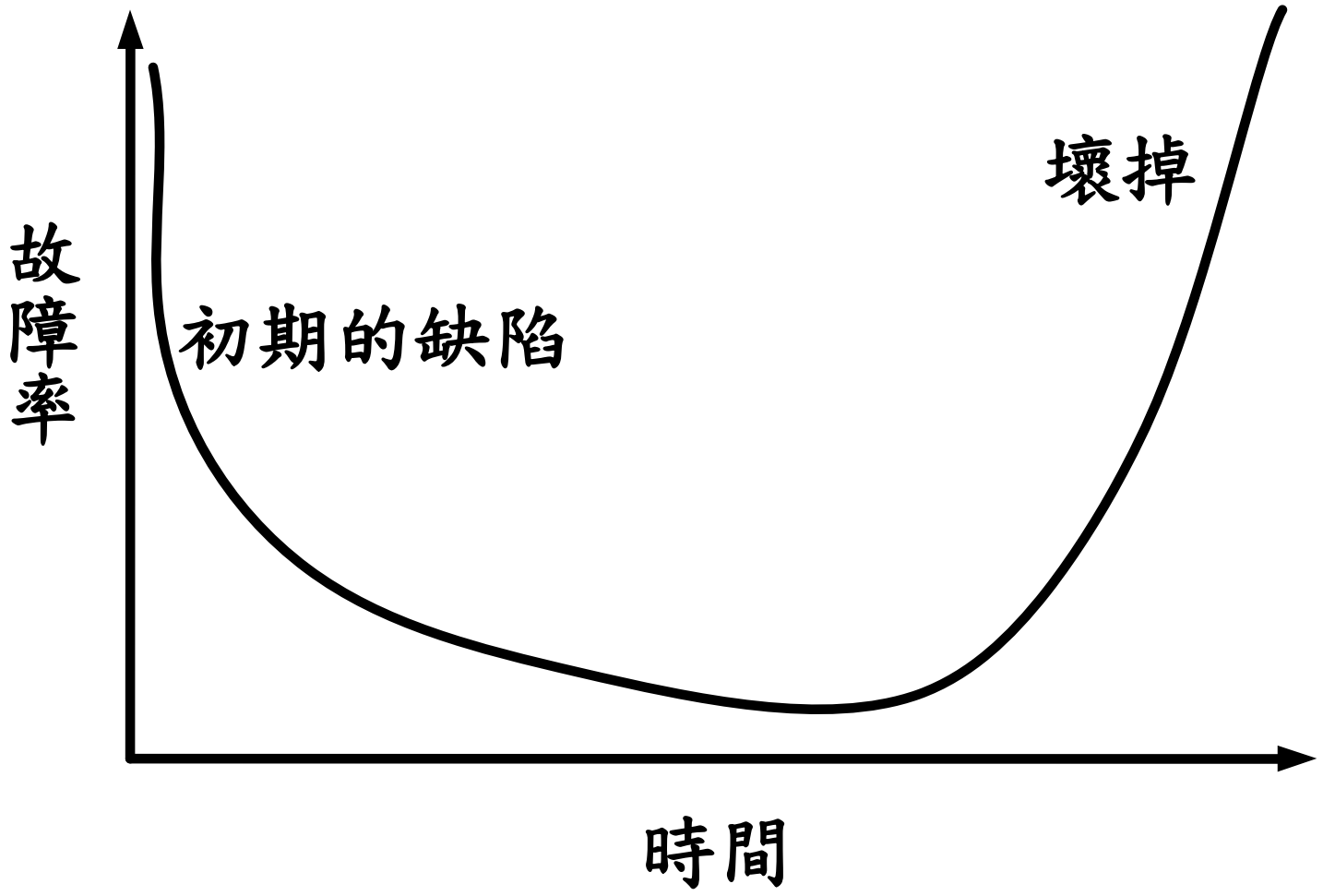
逢甲大學 人工智慧研究中心
資 訊 工 程 學 系

許懷中

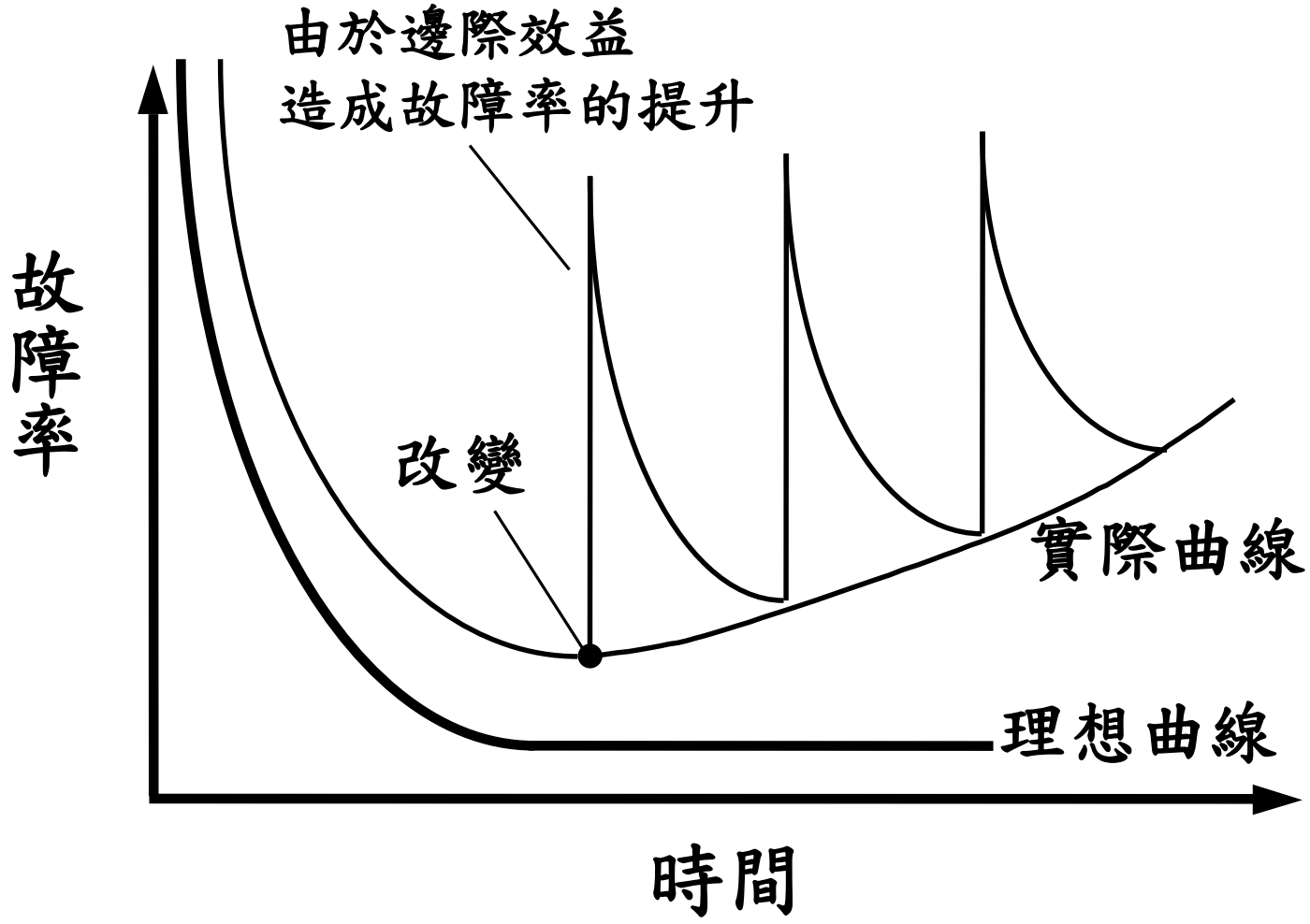
何謂測試？

軟體測試與硬體測試

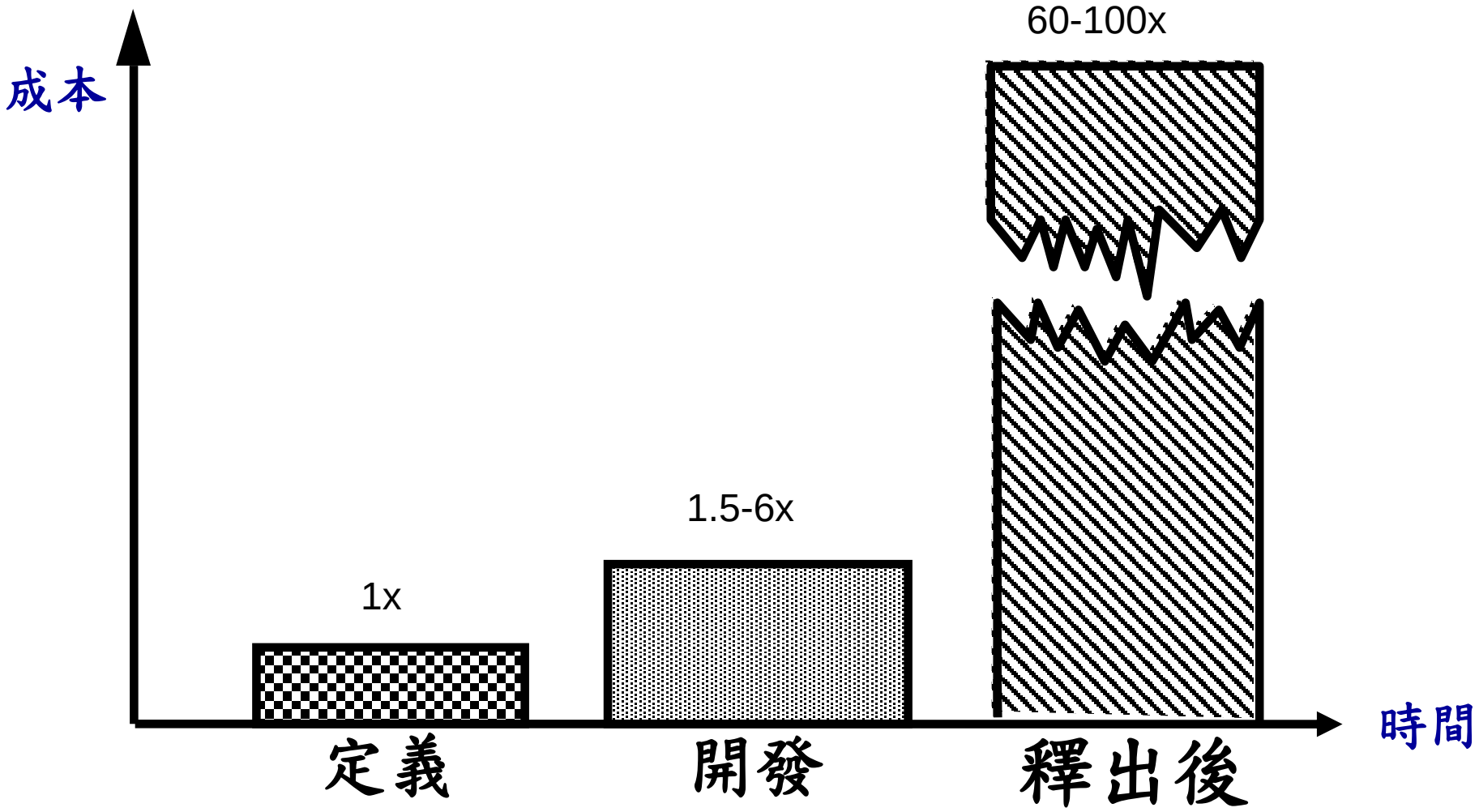
經過初期的除錯後，硬體就不太容易再出現問題



軟體測試與硬體測試



改變的成本



軟體的錯誤會造成巨大損失

- 2007
  - 台灣高鐵售票系統上線後當機連連，系統無法應付突然湧現的購票人潮（no stress testing)
- 2007
  - 台灣彩卷系統：十億獎金引爆人潮，當機連連。
- 2000-2005
  - 巴拿馬國家癌症中心，5個病人接受過量迦瑪射線照射死亡，15人引發嚴重併發症。
- 2003
  - 軟體錯誤造成美國東北部及加拿大停電，5000萬人受影響，3人喪生。

軟體的錯誤可能致命

- 2000
  - 控制軟體問題，美國海軍飛機墜落，4人喪生。
- 1997
  - 韓國空難，225人喪生（雷達控制軟體問題）
- 1995
  - 美國航空在哥倫比亞撞山159人喪生（導航軟體問題）
- 1994
  - 華航名古屋空難：自動駕駛軟體(Auto-Pilot)重飛模式和手操作的控制桿之昇降舵相互對抗。

一個小錯也可能引發重大問題

- 1963年美國太空總署，一個FORTRAN 程式迴圈敘述案例

  DO 5 I=1,3 ------(正確)

  被人為錯誤打成

  DO 5 I=1.3 -------(錯誤)

  FORTRAN編譯器視為

  DO5I=1.3 --------(缺陷)

- 導致飛往火星的火箭爆炸(失效)，造成一千萬美元的損失。

軟體測試的目的

- 確定軟體的行為與想要的一樣
- 尋找軟體中的缺陷

如何測試DVD撥放器

如何測試DVD撥放器 (cont.)

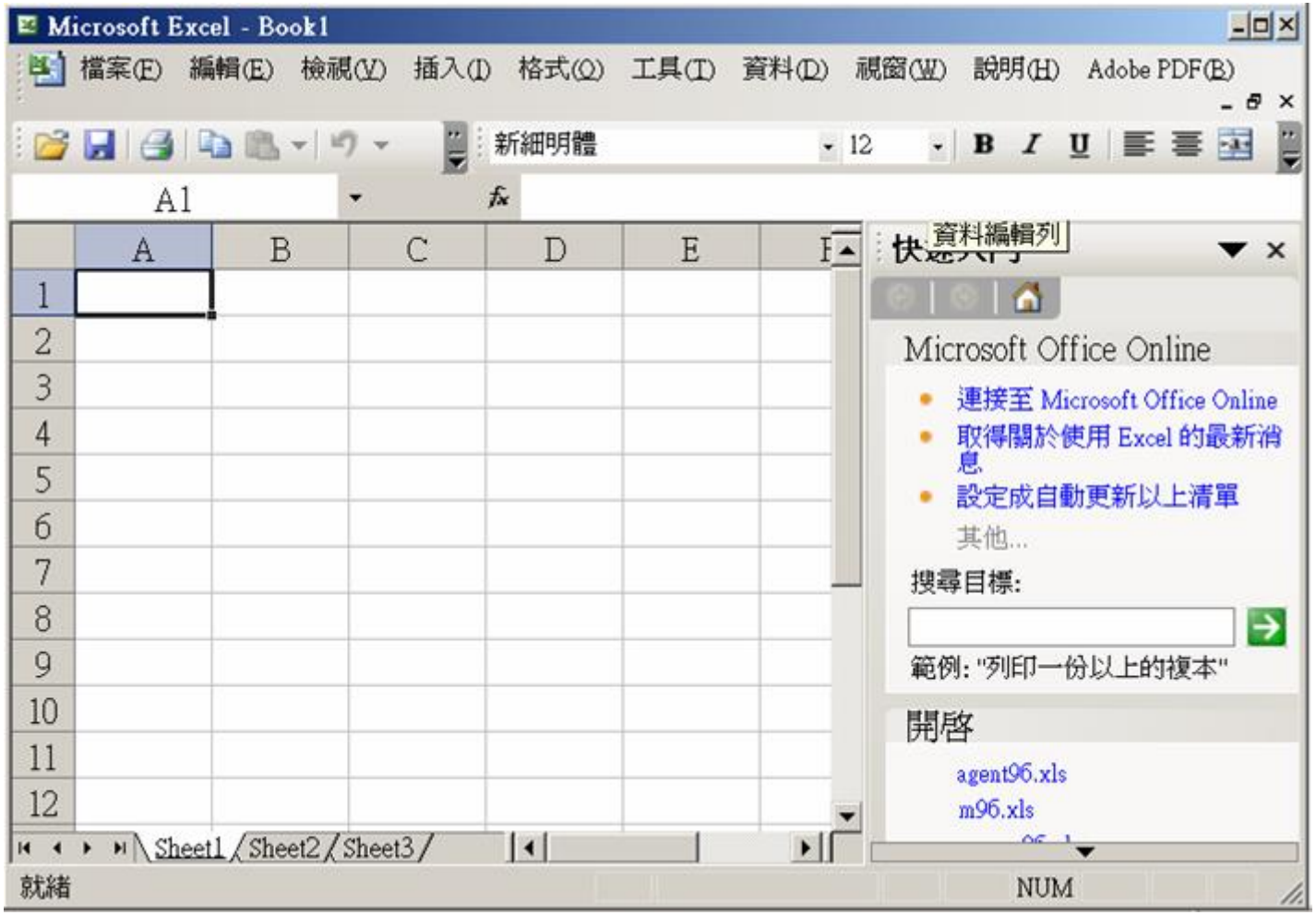
- DIVX/MPEG4/DVD/VCD/CD/CD-R/CD-RW/HDCD/MP3/WMA/JPEG
- 電視制式：NTSC/PAL/AUTO
- 螢幕顯示比例：4:3/ 16:9 可供選擇
- 多重 OSD 語言及字幕語言
- 孩童智能安全鎖定
- 電源供應：AC 110V ， 60Hz
- 自動電源短路保護
- 視頻
  - 1組色差輸出 (Y ， Cb/Pb ， Cr/Pr)
  - 1組 S 端子視頻輸出
  - 1組 CVBS 一般視頻輸出
- 音頻
  - 杜比解碼輸出：2 聲道 1 組
  - 光纖數位輸出端子

如何測試MP4 撥放器

如何測試MP4 撥放器 (cont.)

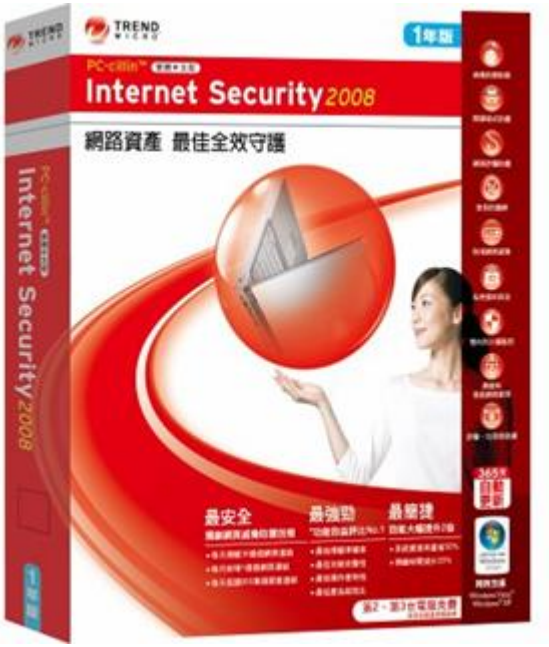
- 3.5吋大螢幕多重觸控介面320 x 480 像素
- 可播放音樂、電影、視訊、有聲書、podcast、相片幻燈片
- Wi-Fi(802.11b/g)
- 容量：16GB
- 最多可儲存 3,500 首歌曲
- 最多可儲存 20,000 張照片
- 最多可儲存長達 20 小時視訊影片
- 音樂播放的電池電力： 長達 20 小時
- 影片播放的電池電力： 長達 4.5 小時

如何測試文書處理軟體？


Microsoft Excel - Book1
檔案(F) 編輯(E) 檢視(V) 插入(I) 格式(O) 工具(T) 資料(D) 視窗(W) 說明(H) Adobe PDF(B)
新細明體
12
A1
資料編輯列
Microsoft Office Online
連接至 Microsoft Office Online
取得關於使用 Excel 的最新消息
設定成自動更新以上清單
其他...
搜尋目標:
範例: "列印一份以上的複本"
開啓
agent96.xls
m96.xls
Sheet1 Sheet2 Sheet3
就緒
NUM

如何測試防毒軟體？


Kaspersky
Version 7.0
Anti-Virus
中文版
三重防護
基礎防禦
KASPERSKY
盒裝續約
2年包


TREND MICRO
1年版
PC-cillin
Internet Security 2008
網路資產 最佳全效守護
最安全
最強勁
最簡捷

如何測試線上遊戲？

什麼時候進行測試？

- 一座橋蓋好了，需要大量的測試嗎？

軟體測試測些測什麼

- 完整性(completeness)
  - 軟體是否具備軟體規格書，設計文件中所描述的功能與性能
- 正確性(correctness)
- 可靠性(reliability)
- 相容性(compatibility)
- 效率(efficiency)
- 可使用性(usability)
- 可攜性(portability)
- 可變比例性(scalability)
- 易測性(testability)

軟體測試的複雜性

- 軟體有複雜的介面，
  - 包括使用者介面、網路介面、檔案介面
- 軟體測試必須考慮更多的情境
  - 正確的執行路徑
  - 不正確的路徑
- 不同應用軟體，測試的理論，技術，實做，程式能力的要求都不低
  - 安全性測試
  - 嵌入式軟體系統測試

軟體測試的迷思

- 軟體測試目的在於證明程式沒有問題
  - 測試無法證明程式是無誤的。
  - 軟體測試只能展現程式錯誤的存在。
- 軟體有問題是測試人員的錯
  - 軟體測試只是一種有效的提高軟體品質手段，但無法百分之百解決軟體品質的問題。
- 軟體測試技術要求不高，比程式設計容易
  - 兩種人無法類比，程式設計能力好的人，可能可以更勝任軟體測試
  - 自動化測試常需要程式設計高手

軟體測試

測試設計

測試的意義

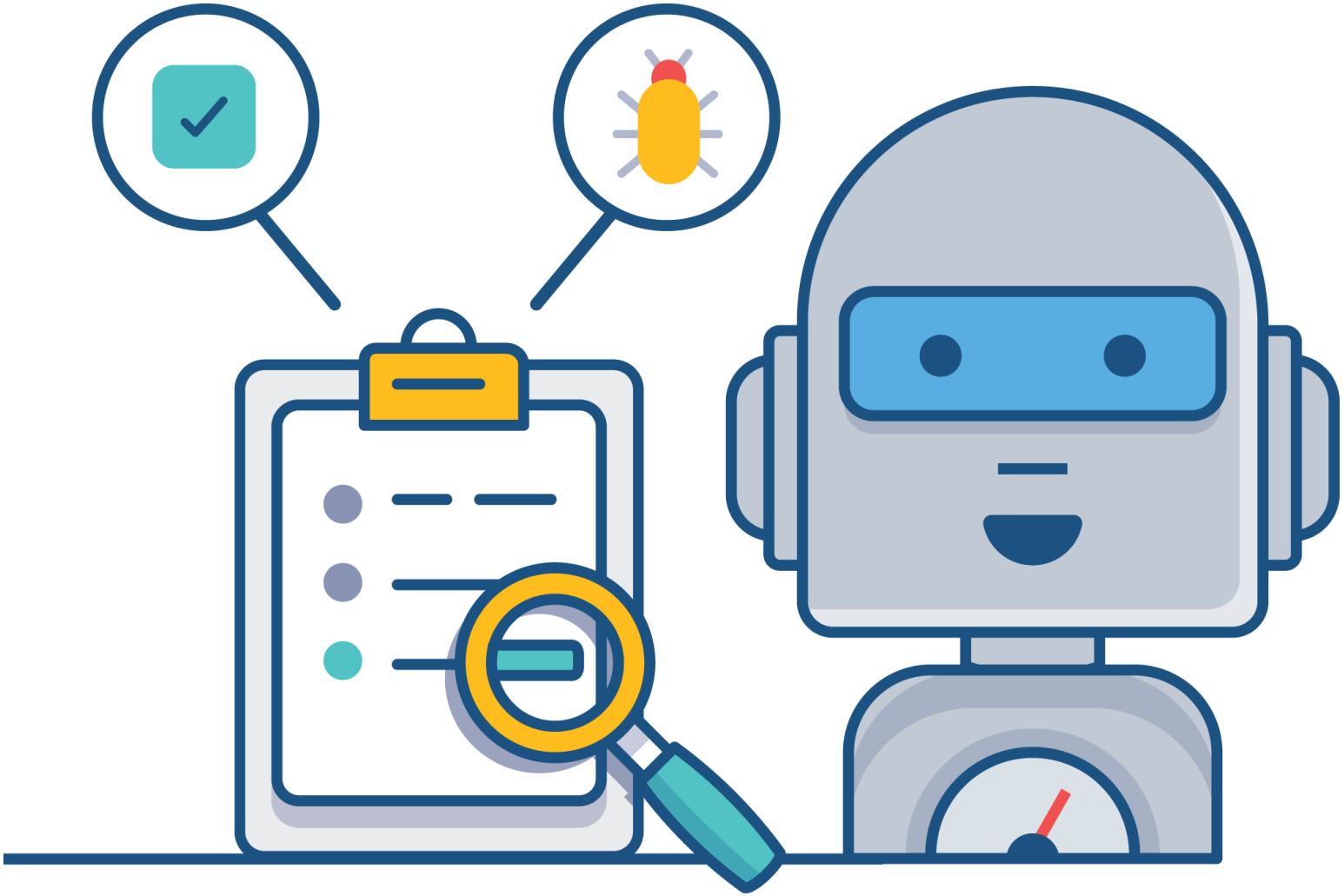
- 測試定義
  - 以一套系統方法執行與檢查軟體，以發現錯誤。
- 測試目的
  - 發現軟體錯誤並進而修改軟體，以提昇軟體品質，避免錯誤造成嚴重損失。
  - 花費最小精力及時間設計能發現最多錯誤的測試
- 錯誤原因
  - 軟體發展過程中，可能因溝通不良造成規格不符或設計錯誤、或程式撰寫時發生疏漏

軟體測試時機

- 越早發現軟體問題，開發費用越低
  - 撰寫程式後修改軟體錯誤的成本是撰寫程式前的10倍
  - 產品交付後修改軟體錯誤的成本是交付前的10倍
  - 軟體品質越高，軟體發行後的維護費用越低
  - 軟體開始規畫時即應考慮測試的規畫時程及人力。
    - 測試花費，占整個軟體發展工作至少30%以上時間及成本。
    - 需求分析與設計發生錯誤約占65%，程式撰寫約占35%
- 早期觀點：Analysis → Design → Building →Testing
- 現在觀點：全程Testing

測試的種類

Validation Testing **Defect Testing**

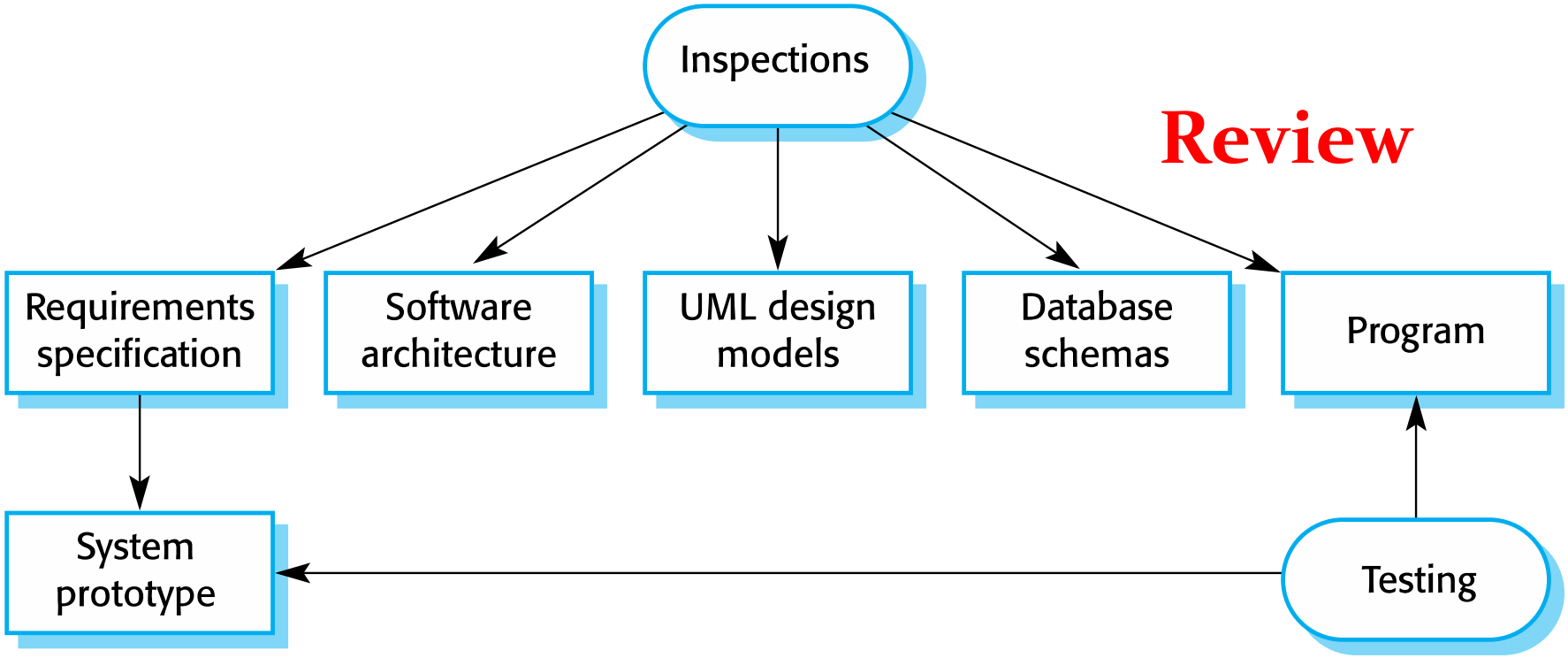


Verification & Validation

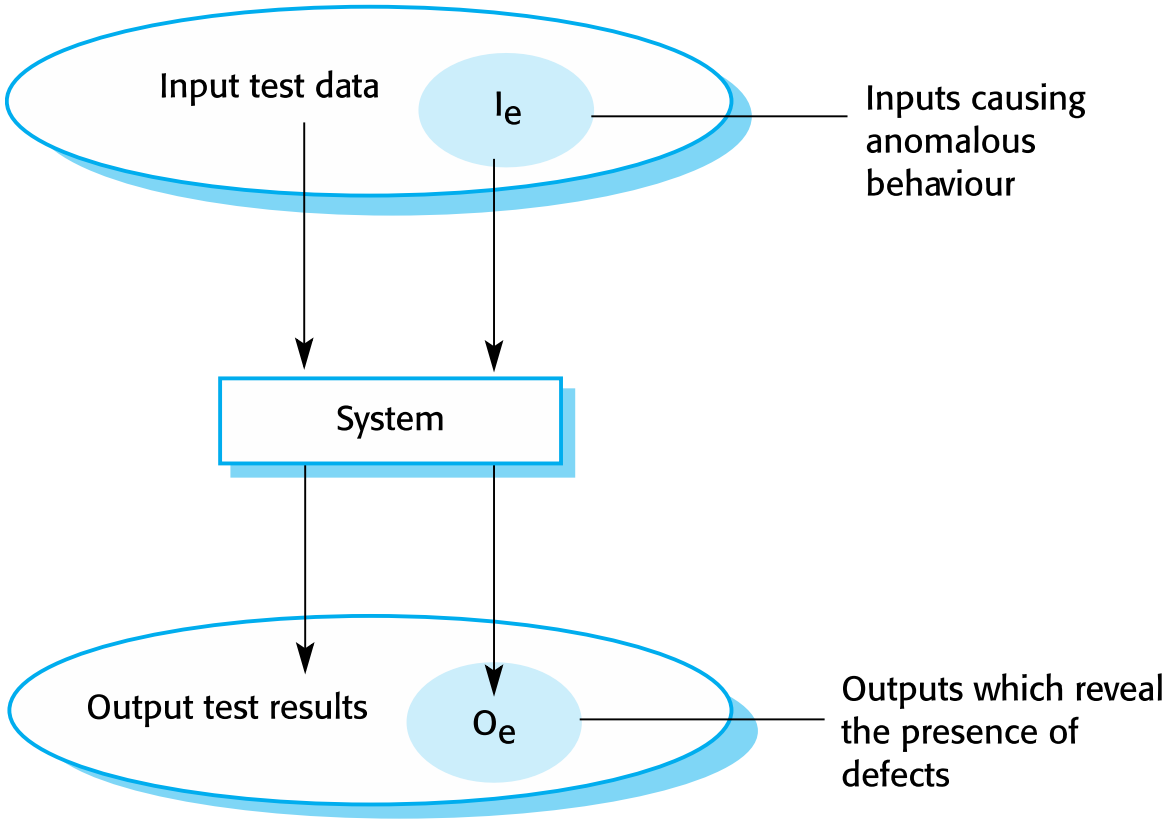
- Verification
  - Do the thing right
  - 確認軟體行為與預期相同 (符合規格)
- Validation
  - Do the right thing
  - 確認軟體符合利害關係人的需求

靜態的軟體測試 – 審查

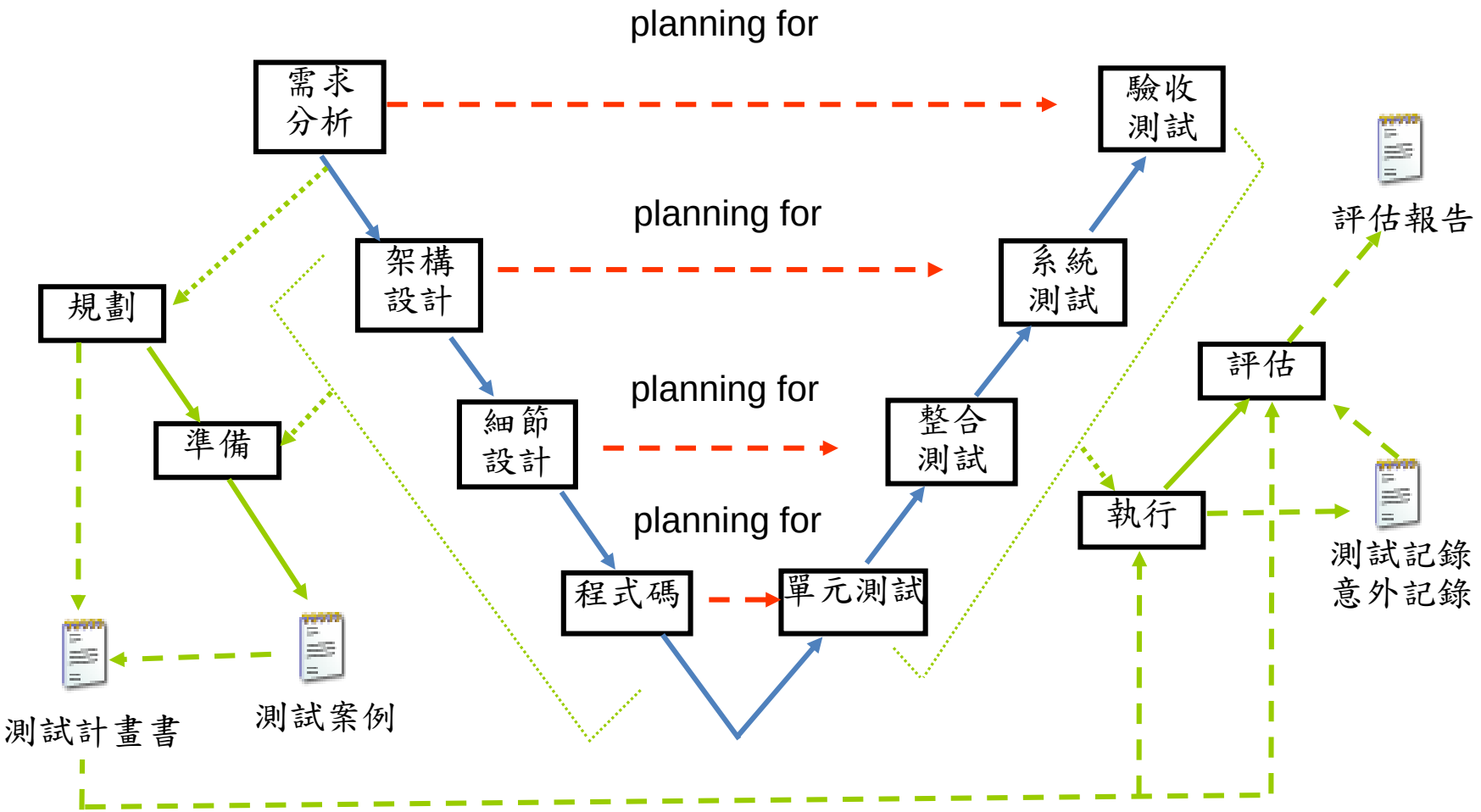
- 軟體審查 (Inspection)



動態軟體測試


Input test data
I_e
Inputs causing anomalous behaviour
System
Output test results
O_e
Outputs which reveal the presence of defects

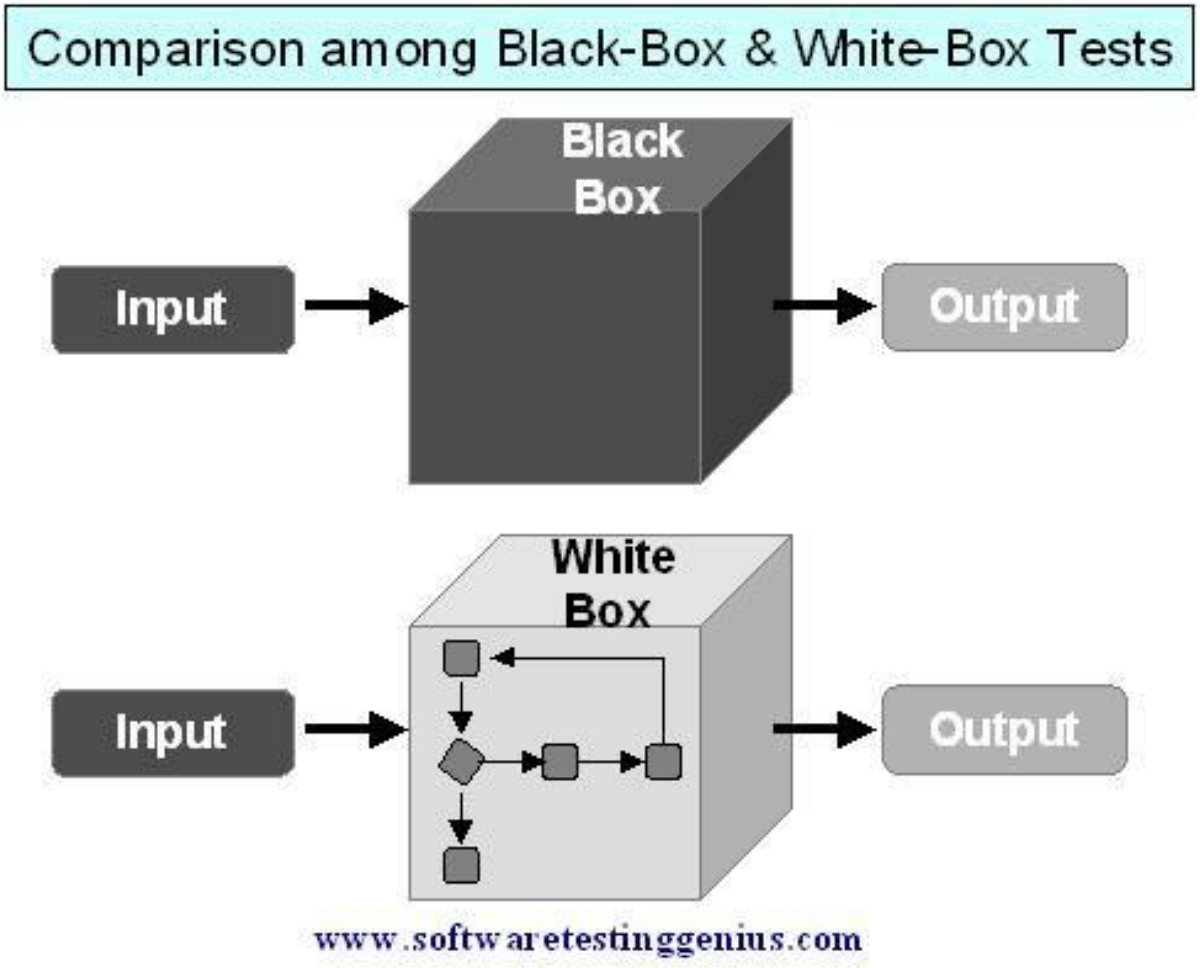
動態軟體測試流程


planning for
需求分析
驗收測試
planning for
架構設計
系統測試
規劃
準備
細節設計
planning for
整合測試
評估
評估報告
執行
planning for
程式碼
單元測試
測試記錄
意外記錄
測試計畫書
測試案例

軟體測試

測試方法概論

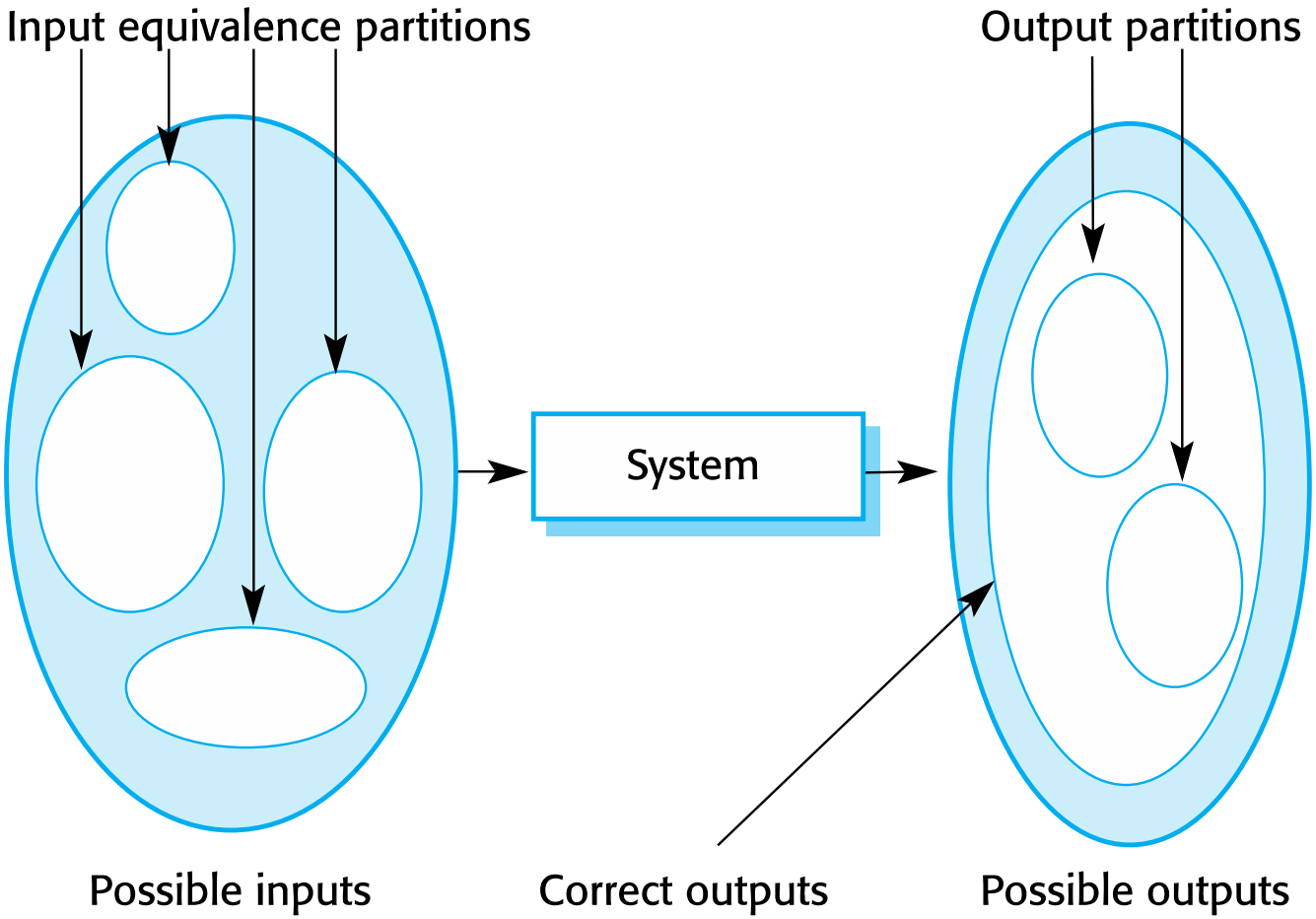
黑箱與白箱測試


Comparison among Black-Box & White-Box Tests
Black Box
Input
Output
White Box
Input
Output
www.softwaretestinggenius.com

黑箱測試 - 亂數測試

- 亂數測試(Random Testing)
  - 亂數測試屬黑箱測試，主要是從一值域中隨機選擇測試案例
  - 每一項輸入測試皆是隨機產生的
  - 無法做精準的測試
  - 但亂數測試是有效率的一種方式

黑箱測試 - 輸入劃分


Input equivalence partitions
Output partitions
System
Possible inputs
Correct outputs
Possible outputs

等價劃分

- 等價劃分(Equivalence Partitioning)
  - 程式被分割成數個等價類別
  - 一個等價類別包含一群資料
- 測試案例的設計是基於一個評價等價類輸入條件
  - 可以減少總測試案例的開發
  - 等價類別可識別一個合法及非法的輸入狀態

等價劃分 (cont.)

- 輸入條件可以為以下型態
  - 定義的數化值
  - 區間的值
  - 值的集合
  - 布林式
- 等價類別的劃分有二種不同情形
  - 有效等價類別
    - 對程式而是合理的，可用來驗證其功能
  - 無效等效類別
    - 對程式而是不合理的，可用來驗證有無不符合規格的地方

等價劃分 (cont.)

- 等價劃分法有以下幾個原則
  - 如果輸入條件指定了一個範圍，則可確立一個有效的和兩個無效等價類別
  - 如果輸入條件需要特定的值，則可確立一個有效的和兩個無效等價類別
  - 如果輸入條件定義了一個集合的值，則可確立一個有效和一個無效等價類別
  - 如果輸入條件為布林，則可確立一個有效和一個無效等價類別

等價劃分 (cont.)

以每個月所得判定薪資所得所屬階級

薪資	分類
0~5000	遊民
5001~15000	貧民
15001~20000	一般
20001~40000	小康
40001~60000	小富
60001以上	富有

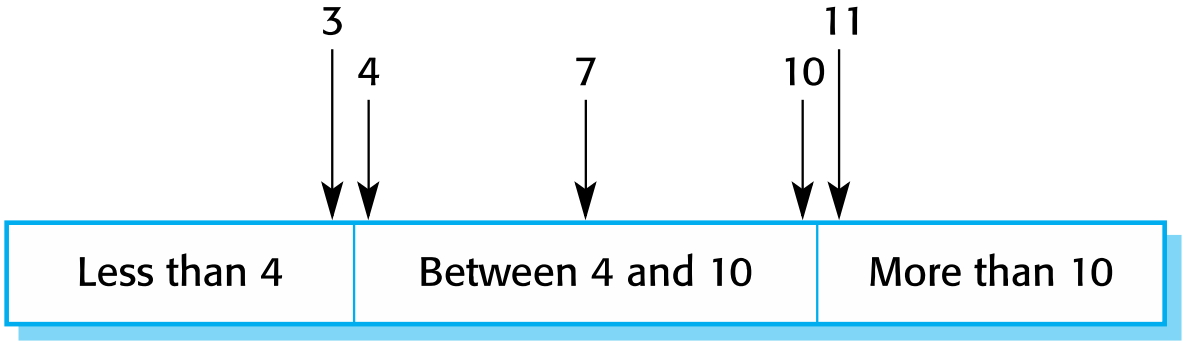
等價劃分 (cont.)

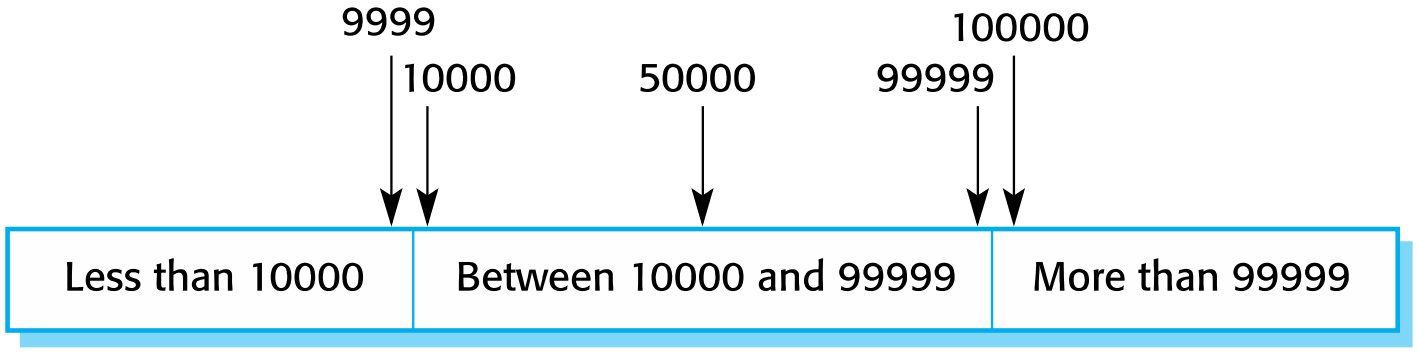
- 輸入域的分數可以劃分為6個有效等價類和1個無效等價類
  - 合法價類: 0~5000, 5001~15000, 15001~20000, 20001~40000, 40001~60000, 60001以上
  - 非法價類:0元以下
- 在類別內任何數據值在被認為是等價的測試條件

黑箱測試 – 決定測試輸入

- 邊界值測試(Boundary Value Testing)
  - 根據經驗，大多的錯誤發生在邊界上
  - 除了在值域內的值需要測試之外，邊界上的值更須要測試
- 決策表測試(Decision Table-Based Testing)
  - 合併所有可能的輸入
  - 所有可能的行為及其對應的輸入條件
  - 指明在何種條件下將執行一項行動
- 錯誤猜測(Error Guessing)
  - 用直覺和經驗找出潛在的錯誤和設計測試案例
  - 測試案例可以得出的一個可能出現的錯誤或容易出錯的情況名單
  - 從過去的開發記錄上猜測，以及記錄已發生的錯誤

黑箱測試 – 決定測試輸入 (cont.)





黑箱測試 – 決定測試輸入 (cont.)

- 決策表

	I	II	III	IV
Con1	False	True	True	True
Con2		False	True	True
Con3			True	False
Act1			X	
Act2				X
Act3	X	X		

白箱測試

- 以開發人員為主
- 測試每個設計是符合要求
- 又稱為結構化測試或玻璃箱測試
- 對程式內部有相當的了解
- 以結構化控制(Control Structures)的方式設計測試案例

白箱測試 (cont.)

- 白箱測試通常集中在幾個方面:
  - 控制結構(Control Structures)
  - 邏輯路徑(Logical Path)
  - 邏輯條件(Logical Conditions)
  - 資料流(Data Flow)
  - 資料結構(Data Structures)
  - 迴圈(Loops)

白箱測試 (cont.)

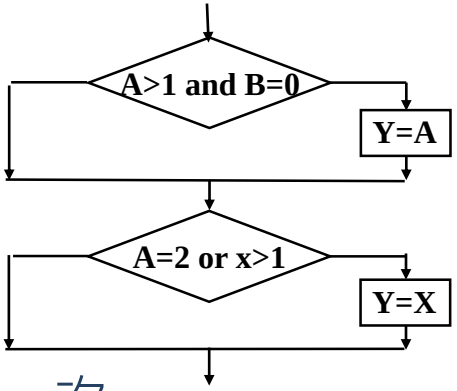
- 使用白箱測試，有以下特性
  - 檢查資料結構是否正確
  - 在迴圈的邊界(Boundaries)及執行邊界(Operational Bounds)上執行迴圈
  - 所有模組至少會被測試一次
  - 在邏輯判定的地方執行True及False的情況

測試覆蓋準則(Coverage Criteria)

- 敘述覆蓋 (Statements Coverage)
  - 測試案例要能使程式每一敘述至少執行一次
- 分支決策覆蓋 (Branches/Decision Coverage)
  - 測試案例要使程式每一決策點至少執行一次
- 條件覆蓋 (Condition Coverage)
  - 所有邏輯判斷情況都至少執行過一次
- 多重條件組合覆蓋 (Multiple-condition Combination Coverage)
  - 不同組合的判斷情況都至少被執行一次
- 全面路徑覆蓋（ All-Paths Coverage ）
  - 測試案例要使軟體每一路徑至少執行一次

Statement Coverage

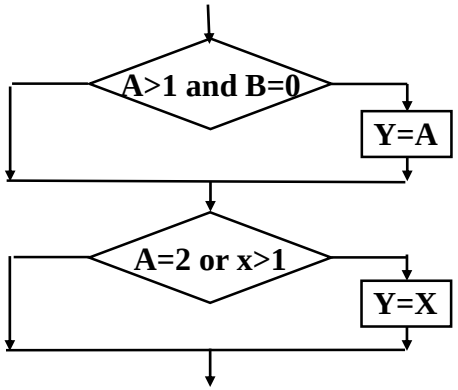
```
10 INPUT A,B,X
20 IF (A>1) AND (B=0) THEN Y=A
30 IF (A=2) OR (X>1) THEN Y=X
40 PRINT Y
```



- 設計測試案例，使每一條指令敘述至少執行一次
  - input為(2,0,3)就可覆蓋所有可執行指令，結果會得到3
- 敘述覆蓋測試方法較不嚴謹，例如若把：
  - 行號20的AND改成OR
  - 行號30的X>1改成X>0
  - 行號20後面的Y=A改成其他的敘述
- 程式執行結果沒變，所以找不出這種錯誤。
- 敘述覆蓋最弱邏輯涵蓋準則，白箱測試至少要做到此測試。

Branches Coverage

```
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
```

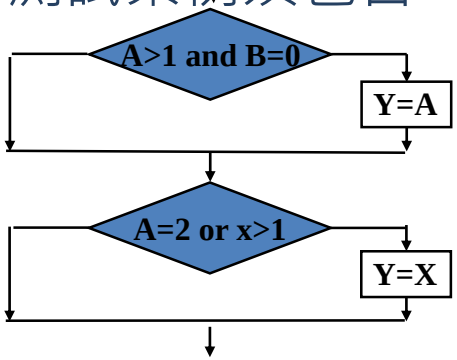


- 分支決策覆蓋目標是設計測試案例，使程式每個判斷取真分支和取假分支至少執行一次。
  - A, B, X若為(3, 0, 3)和(3, 1, 1)能使得（A>1）AND（B=0）和(A=2）OR（X>1)這兩個布林運算式均能產生真和假的值。
  - 若將（A>1)寫成(A>2)卻是無法測出的錯誤。
  - 決策涵蓋測試方法嚴密性比敘述涵蓋高。

Condition Coverage

- 條件覆蓋和分支覆蓋類似，不過條件覆蓋是以某項條件為主，而決策涵蓋則以整個布林運算式為主。使程式中每個判斷的每個條件至少執行一次。

- 例如(A>1) AND (B=0)運算式包含A>1和B=0兩條件。
  - 前述程式具有下列四條件 A>1、B=0、A=2、X>1
  - 欲使這四個條件都能產生真與假的值，測試案例須包含
    - (1) A>1, (2) A<=1
    - (3) B=0, (4) B<>0
    - (5) A=2, (6) A<>2
    - (7) X>1, (8) X<=1



  - (2,0,3)滿足1、3、5、7
  - (1,1,1)滿足2、4、6、8便可達成這個目標。

Condition Coverage (cont.)

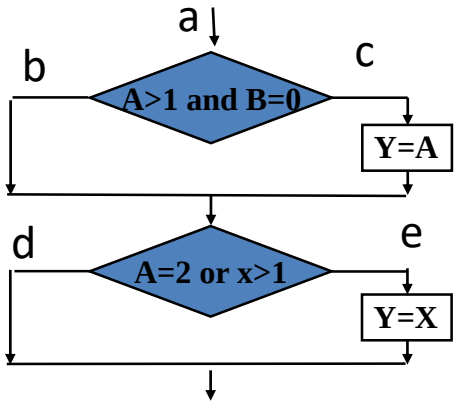
- 條件覆蓋嚴密性通常比決策覆蓋高，但非絕對。
  - 例如 (1,0,3) 和 (2,1,1) 這組數據，雖然使上述四個條件均產生真與假的值，但並未使運算式(A>1) AND (B=0)和(A=2) OR (X>1) 具有真與假的值
- Decision/Condition Coverage
  - 結合決策涵蓋與條件涵蓋 兩種方法。
  - 設計足夠的測試案例，使判斷中每個條件的所有可能值至少執行一次，同時每個判斷的所有可能判斷結果至少執行一次。
  - 例如上例 (2,0,3) 和(1,1,1)可達成此目標

Multiple-condition Combination Coverage

- 多重條件涵蓋的方法是從決策/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。
- 例如前述程式，第一個布林運算式有下列4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4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、5和(2,1,1)滿足2、6和(1,0,2)滿足3、7和(1,1,1)滿足4、8便可涵蓋上述8種條件組合。

Multiple-condition Combination Coverage (cont.)

- 測試數據
  - (2,0,4) 滿足 1 & 5 => a-c-e
  - (2,1,1) 滿足 2 & 6 => a-b-e
  - (1,0,3) 滿足 3 & 7 => a-b-e
  - (1,1,1) 滿足 4 & 8 => a-b-d

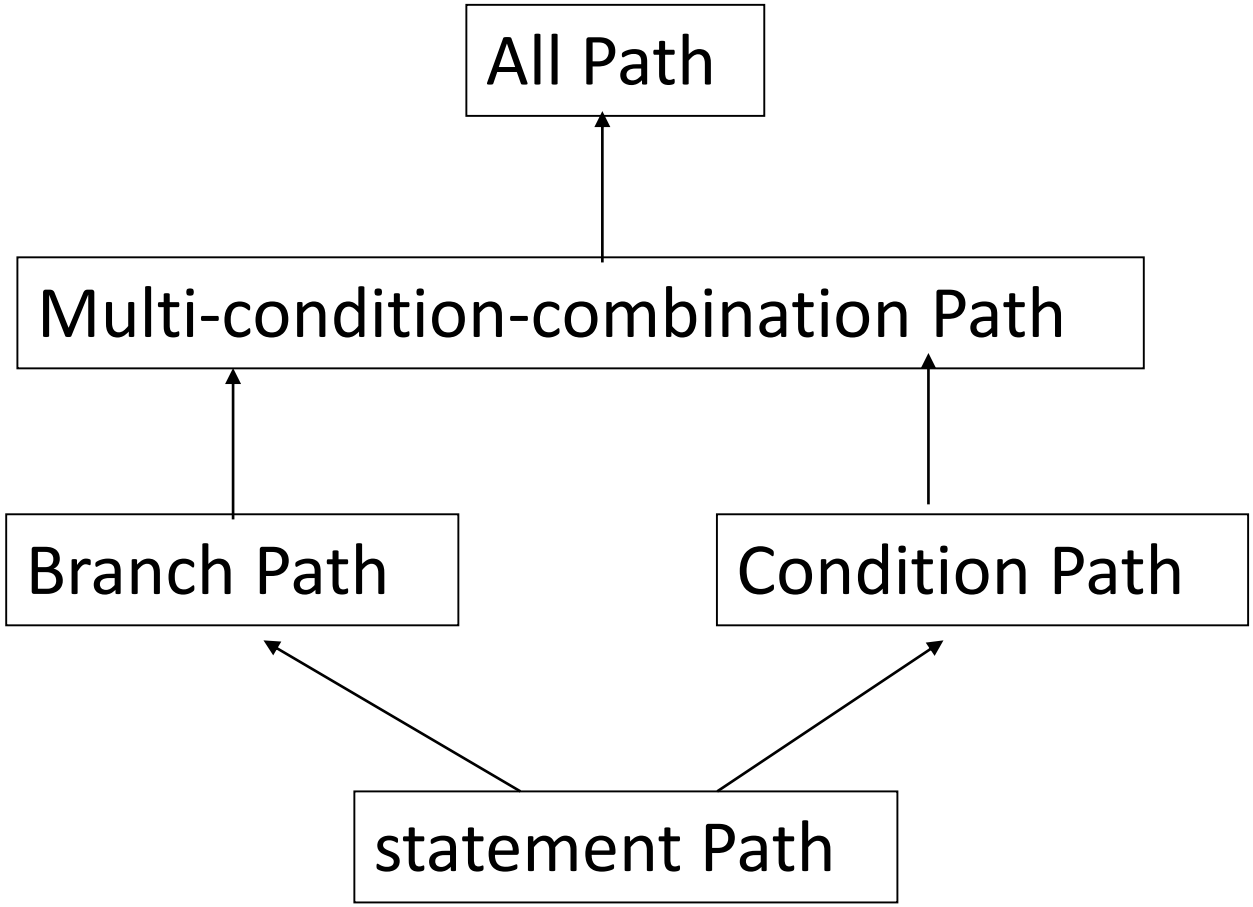


- 沒有覆蓋 a-c-d，所以測試還不完全

All-path Coverage

- 全面路徑涵蓋是設計測試案例使程式執行所有路徑，每種路徑須有一種測試案例測試。
  - 此測試方法相當嚴密，但所需時間與成本相當高，實用性不大。
  - 除多重條件涵蓋的各種條件組合外，各運算式間也須加以組合，一個測試案例只測試某種組合。
  - 如前述程式，共有2個布林運算式，每個布林運算式都有4種條件組合，因此測試路徑（測試數據）有4*4=16種。
  - 當布林運算式增多時，測試路徑呈指數般急速增加。例如有100個運算式時，其測試路徑至少便達到2^100=1.27*10^30種，因此不實用性可見一般。

測試覆蓋率層次


All Path
Multi-condition-combination Path
Branch Path
Condition Path
statement Path

資料流測試

- 資料流測試(Data Flow Testing, DFT)
- 資料流測試為一個資料的生命週期，檢查任何用法或語法導到程式內變數失敗或異常，例如:
  - 資料未定義
  - 資料無法到達某個程式區段
  - 不正常的宣告

資料流測試 (cont.)

- 針對**資料變數**定義與使用進行分析，確保測試案例涵蓋程式中資料使用情形。
- 資料操作種類
  - 定義(Define, d)：指定變數狀態，定義、建立或初始化變數的值，x=3; x=y+z; 。非變數宣告。
  - 使用(Use, u)
    - 資料使用於計算(c-use, computation use), z=(x*3)/2 。
    - 資料使用於預測(p-use, predicate use), If (x>3) 。
- DU chain/DU testing
  - 根據程式中變數定義及使用指令敘述，選擇程式測試路徑。

資料流測試 (cont.)

- X的DU chain
  - 變數X，在第S行敘述中定義，在S'敘述中使用，路徑
  - $X \in \{DEF(S) \cap USE(S')\} \Rightarrow [X, S, S']$
  - $\underline{D}EF(S)=\{X|$敘述S包含X的定義$\}$
  - $\underline{U}SE(S) =\{X|$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
- DU測試不保證覆蓋程式的所有分支
  - 例如if-then-else，then部份沒有定義任何變數，而else部份不存在，此D測試不保證覆蓋某一分支

資料流測試 (cont.)

```
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. }
```

DEF(1)={s} USE(1)=∅
DEF(2)={x} USE(2)=∅
DEF(3)=∅ USE(3)={x,y}
DEF(4)={x} USE(4)={x}
DEF(5)={y} USE(5)={y}
DEF(6)=∅ USE(6)={x,y}
DEF(7)={s} USE(7)={s, x, y}
DEF(8)=∅ USE(8)=∅
DEF(9)={s} USE(9)={s, x, y}
DEF(10)=∅ USE(10)=∅

資料流測試 (cont.)

涵蓋準則包括

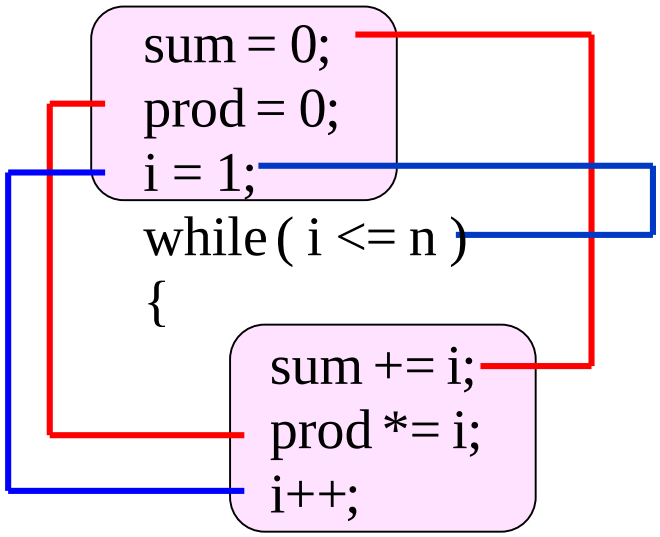
P-use：路徑從變數定義開始，在條件敘述式結束。

i=1; ..... while(i<=n)

C-use：路徑從變數定義開始，在使用於計算時結束。(Red line)

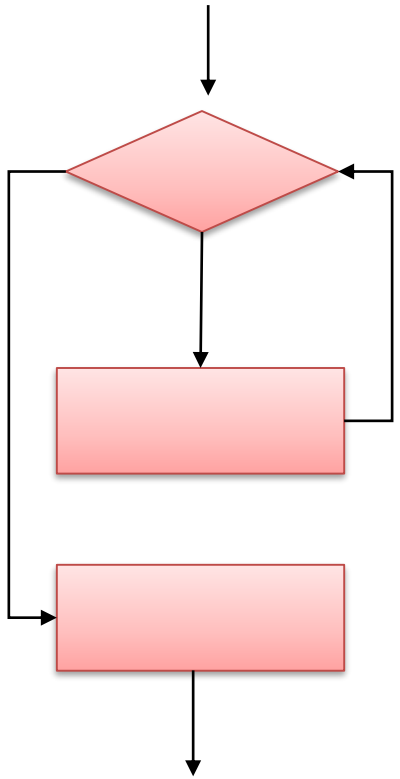
prod=o; .... irod=prod*i;

- All-use/Du-path：路徑從變數定義開始，在被使用時結束。



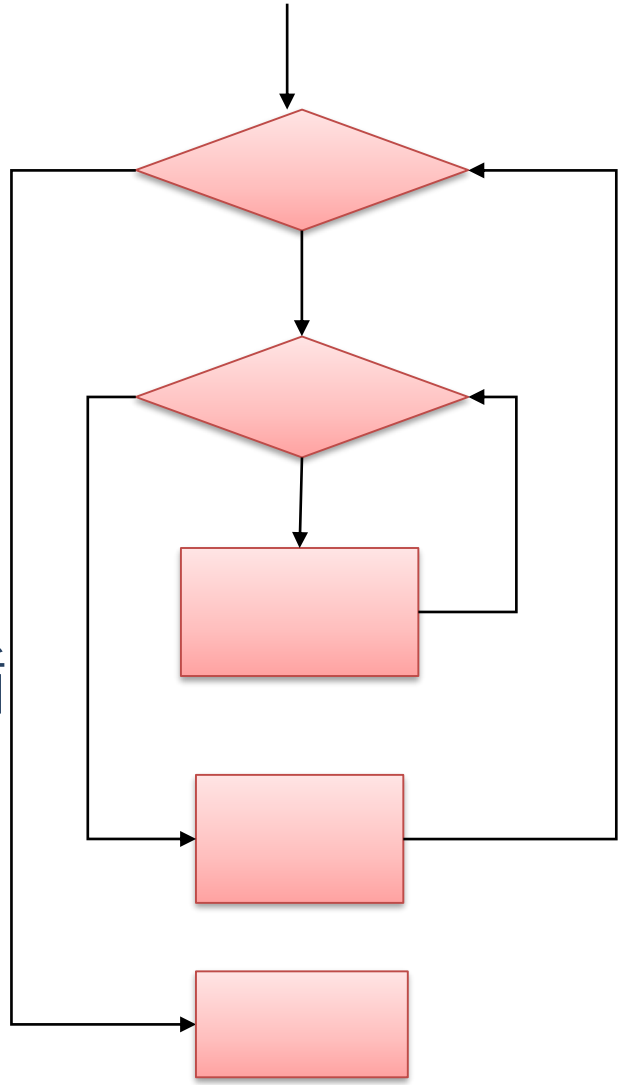
迴圈測試1

- 迴圈測試(Loop Testing)
- 簡易迴圈測試
  - 跳過迴圈入口
  - 只經過迴圈一次
  - 通過迴圈二次
  - 通過迴圈m次, m<n
    - n: max number of loop
  - 通過迴圈n次



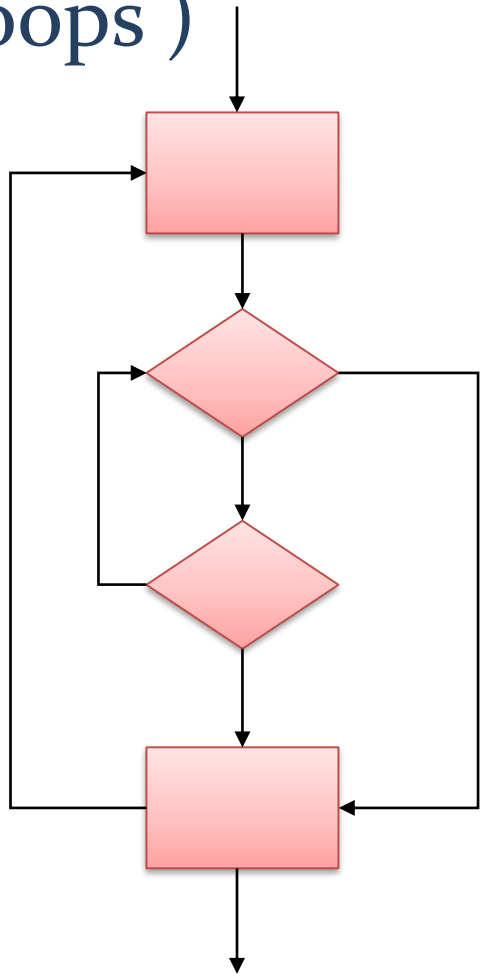
迴圈測試2

- 巢狀迴圈(Nested Loops)
- 巢狀迴圈是簡易迴圈的一種延伸
- 以下是巢狀迴圈的幾個方針:
  - 進行簡單的循環測試內層循環，同時鎖定外迴圈
  - 漸漸向外圈進測試。
  - 持續進行一直到迴圈完結



迴圈測試3

- 非結構式迴圈 (Unstructured Loops)
- 此類迴圈無法測試
- 重新制定架構



設計測試準則

- 設立一套測試標準
  - 如何判定已測試完成
  - 如何針對需求進行測試案例設計
- 製定測試範圍，諸如之前所提:
  - 條件
  - 路徑
  - 節點

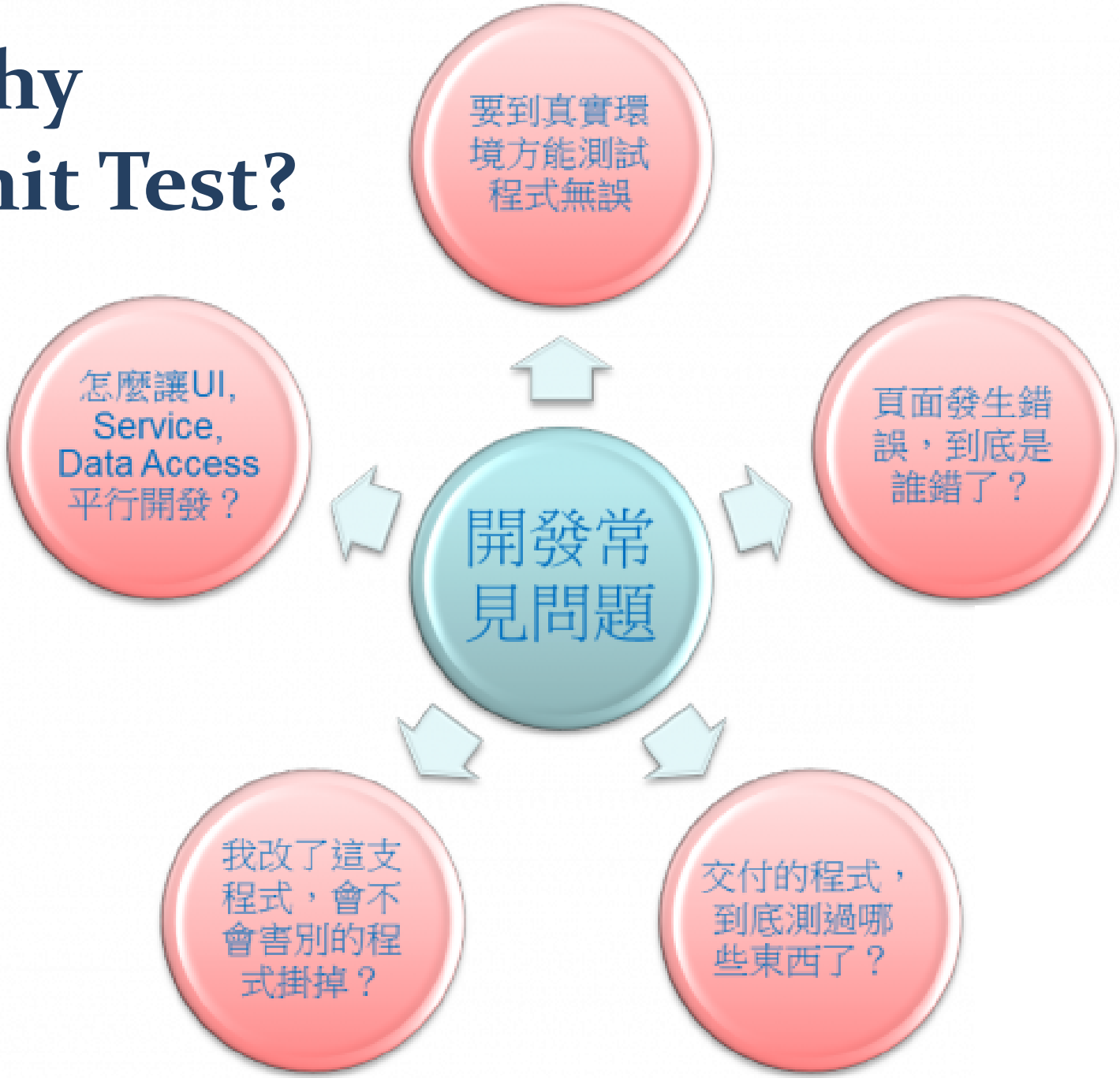
軟體測試

單元測試 (UNIT TEST)

單元測試

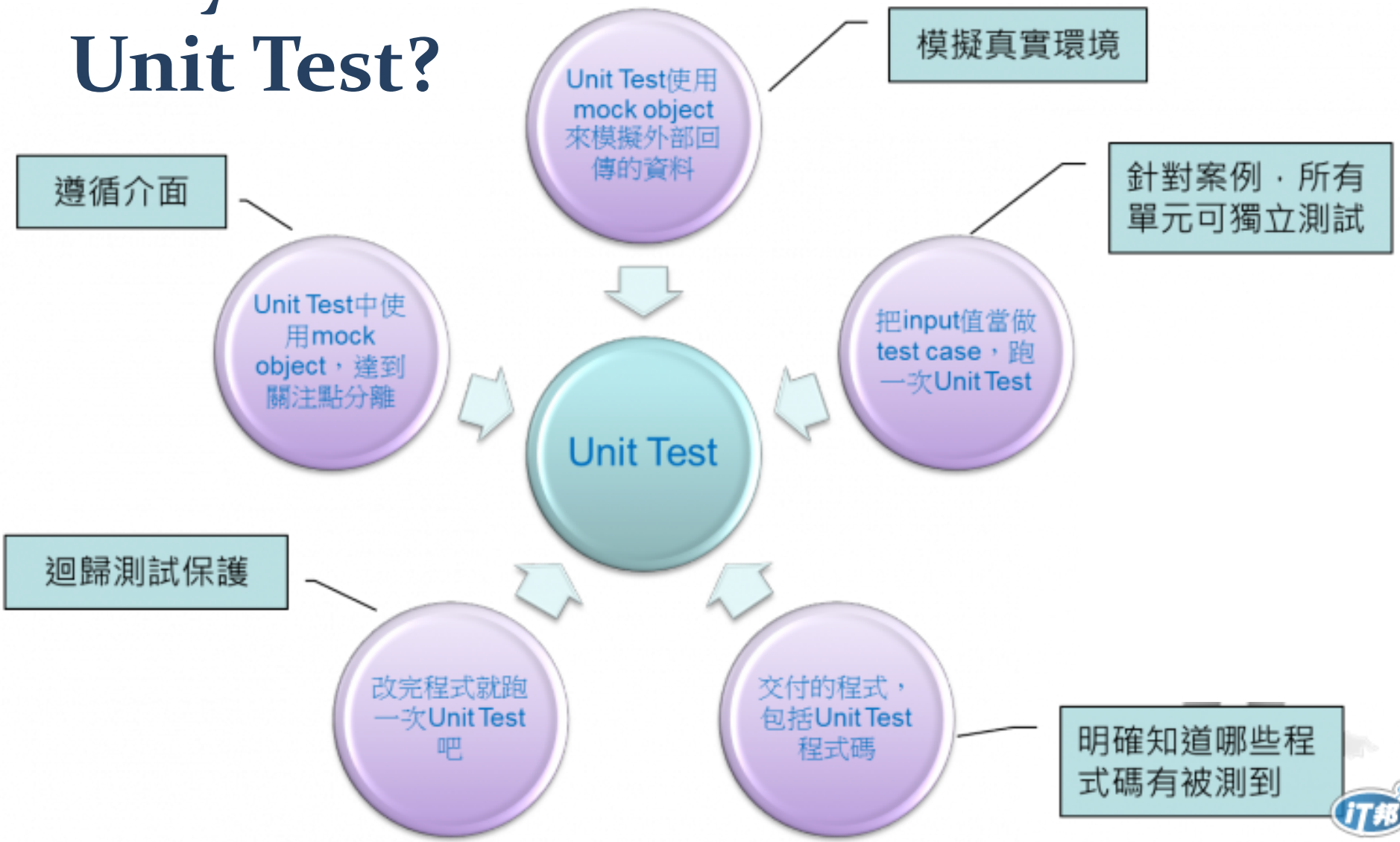
- 低階測試
- 獨立測試
- 細節容易被看清楚
- 通常由程式設計師自行測試
- 由外部使用物件的角度來設計單元測試

Why Unit Test?

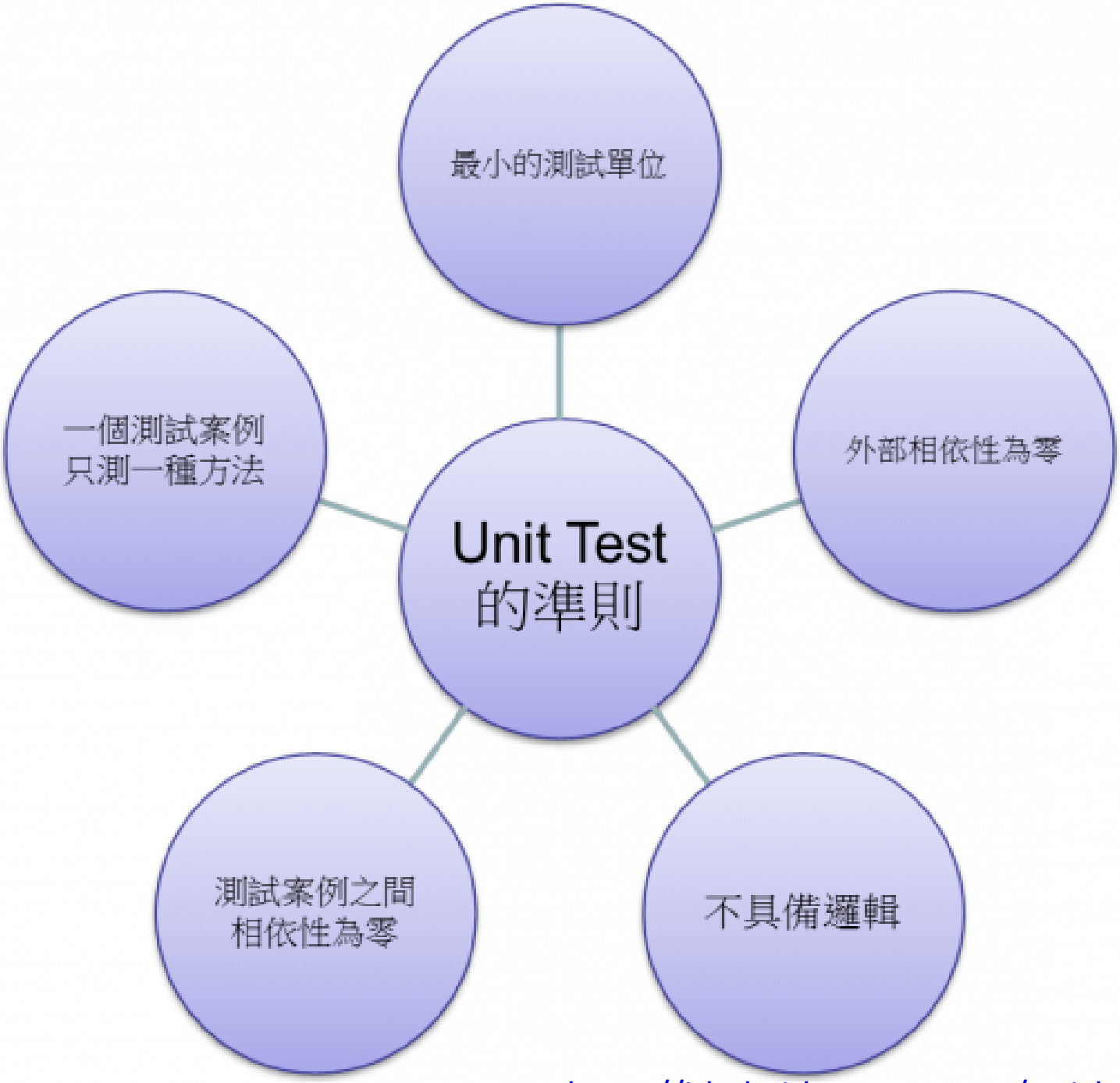


https://ithelp.ithome.com.tw/articles/10102264

Why Unit Test?



https://ithelp.ithome.com.tw/articles/10102264

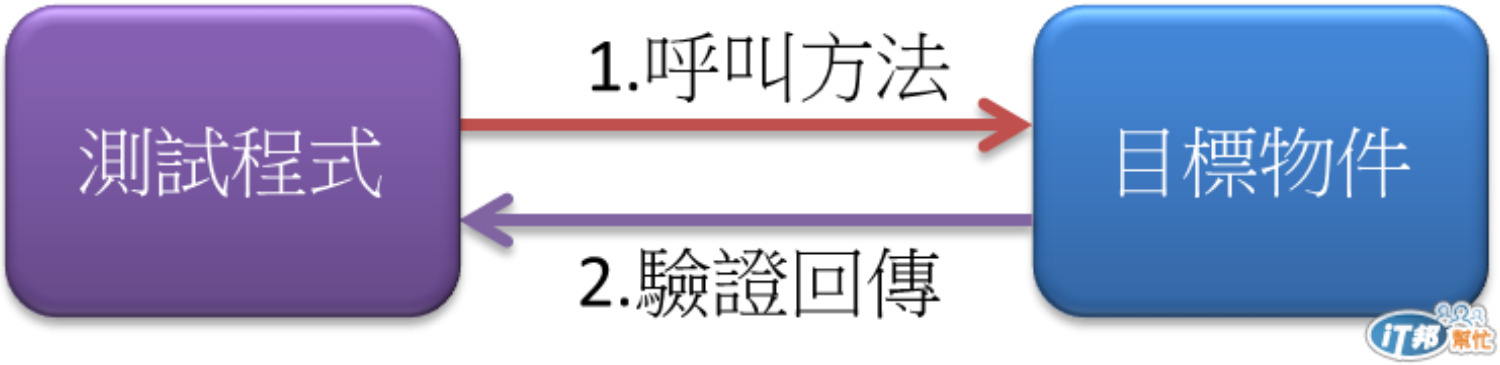


https://ithelp.ithome.com.tw/articles/10102264

如何進行 Unit Test

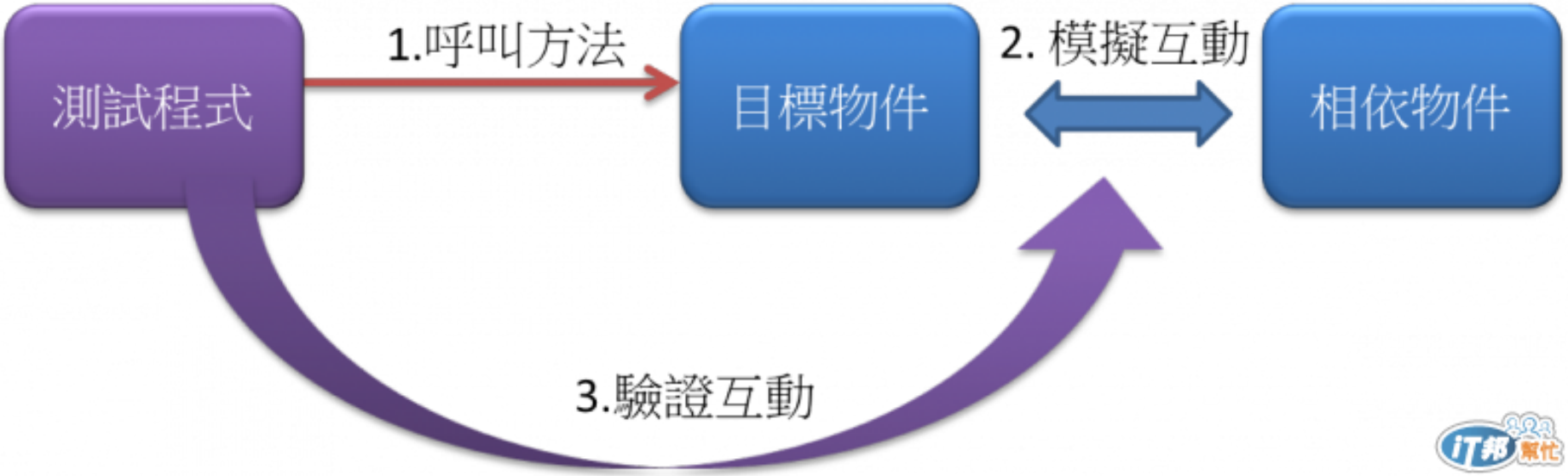
- 界定何謂最小單位
  - 一個函式
  - 物件中的一個方法
- 只關注目標物件
- 模擬與目標物件/函式進行互動

如何進行 Unit Test (cont.)

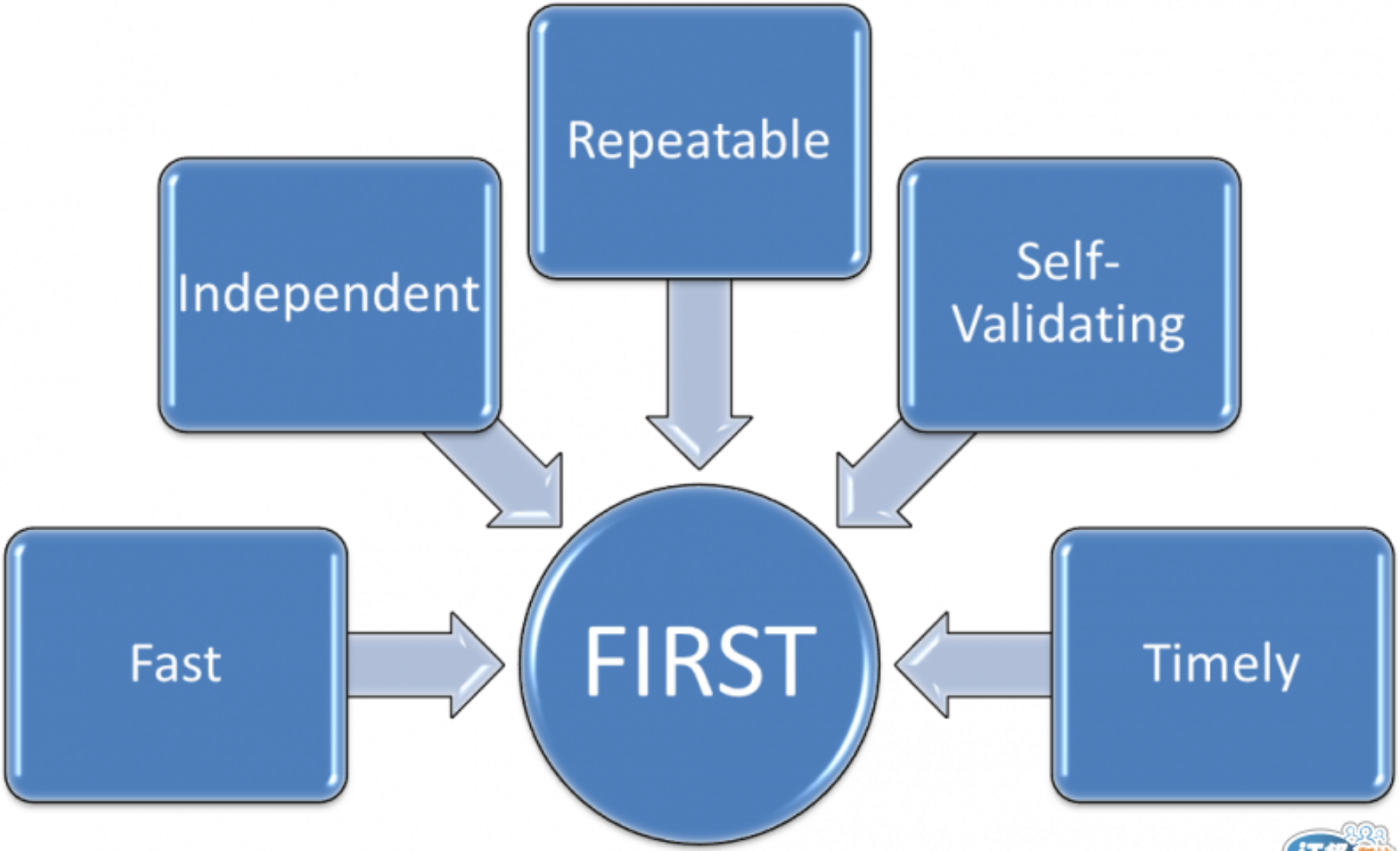


如何進行 Unit Test (cont.)

如何進行 Unit Test (cont.)



Unit Test 特性



https://ithelp.ithome.com.tw/articles/10102264

Unit Test 開發時機

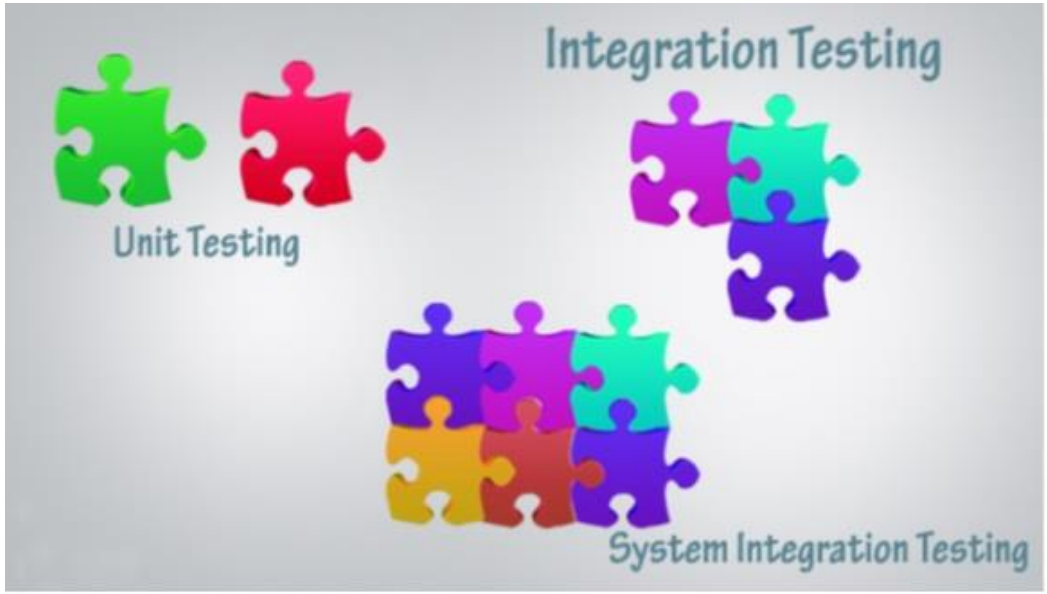
- 在開發功能的同時，開發 Unit Test
  - 測試物件是否滿足外部需求
  - 測試物件是否符合『預期』
- 與預期不符合
  - 需求改變 / 增加 => 開發新的測試案例
- 出現非預期執行結果時
  - Debug, bug fixing

軟體測試

整合測試

整合測試

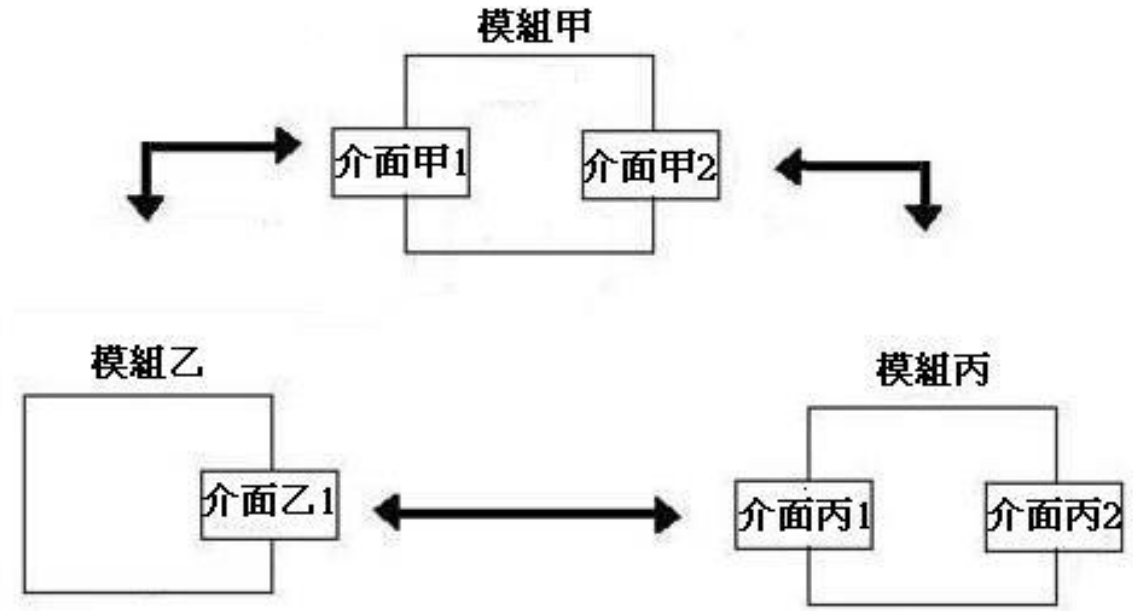
- 多個單元測試
- 可用來測試無法單一測試的群組
- 單元間的溝通
- 非功能性觀點
- 測試策略
  - Top-down
  - Bottom-up
  - Functional



整合測試 (cont.)

- 測試時機
  - 通過單元測試的模組元件
  - 全部或部分單元
- 參與人員
  - 開發人員協助進行整合測試
  - 當軟體整合結構完成，測試小組便可獨立工作
- 測試目的
  - 整合模組間介面一致性問題，測試系統整合功能
  - 資料經過模組界面，可能被忽略
  - 一個模組可能對其它模組產生無意的不良影響
  - 個別模組可接受的不精確性資料，經過其他模組，可能被放大到不可接受的地步
  - 全域性資料結構可能被其他模組無意修改，造成整合問題

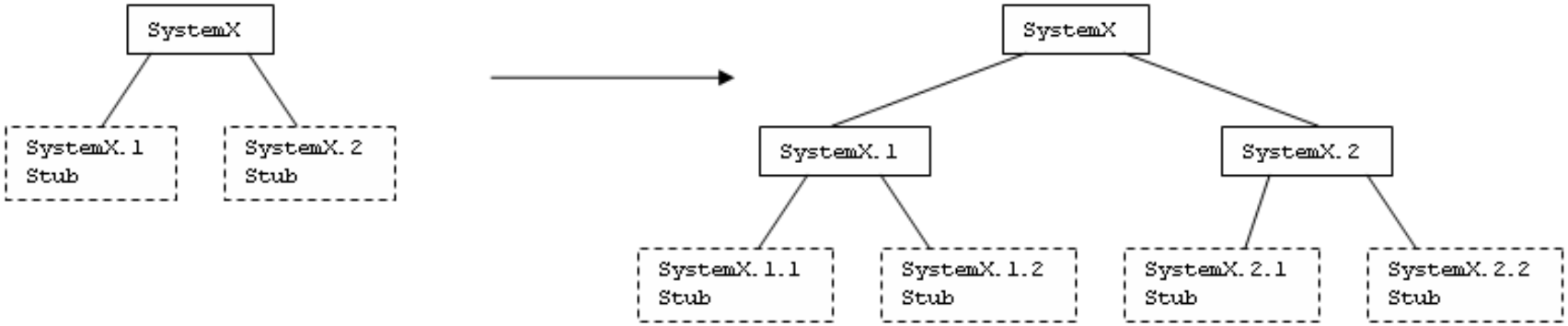
整合測試介紹(Cont.)



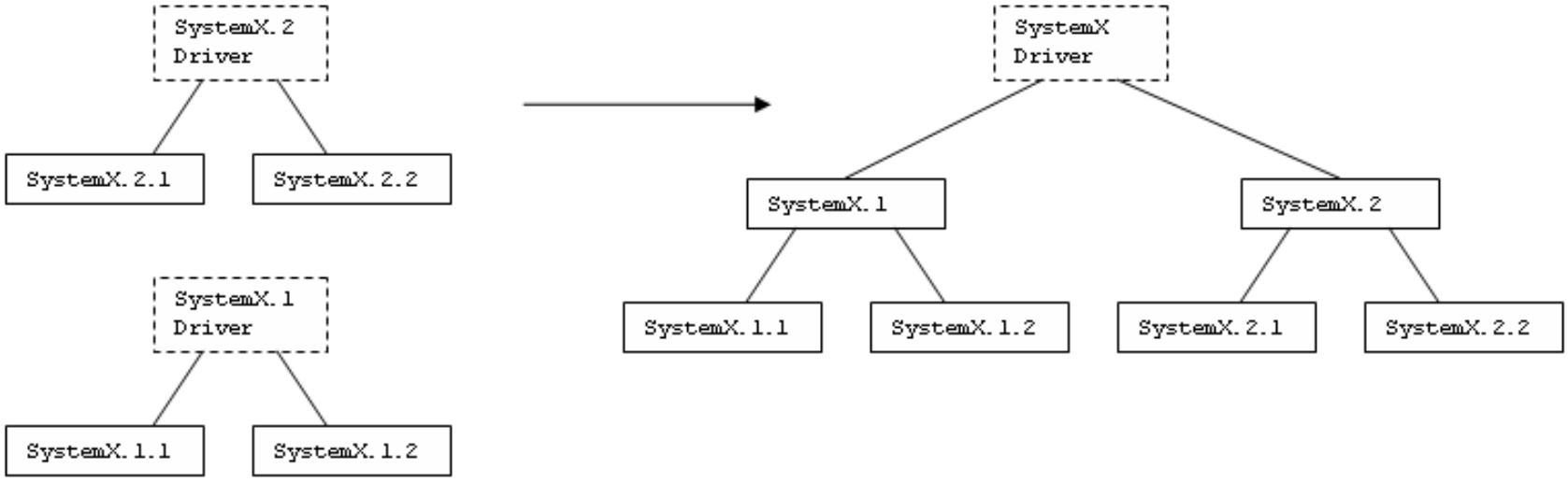
整合測試策略

整合方法	特性	優點	缺點
Top-Down	1.由控制程式開始測，各模組逐次整合。 2.著重於模組間的介面問題。	1.不需撰寫驅動程式。 2.控制程式與部份模組可先整合。 3.早期便出現可執行系統。 4.模組介面間的錯誤可在早期發現。	1.需要殘根(Stub)程式。 2.整合初期只能容許少數人測試，較不易平行。 3.重要模組的問題在測試末期才被發現。
Buttom-Up	1.依序小模組大模組，最後整合為完整系統。 2.每次整合模組可依需要畫分。 3.著重模組功能與效能測試。	1.不需殘根(Stub)程式。 2.可平行工作。 3.重要模組錯誤可早期發現。	1.需驅動程式。 2.整合許多模組後才有第一版的執行系統。 3.模組間的介面錯誤較晚發現。

Top-Down Integration Testing



Bottom-Up Integration Testing



整合測試策略 (cont.)

整合方法	特性	優點	缺點
Sandwich	1.重要模組太多時使用。 2.先將所有重要模組以Bottom-up方式整合，再以Top-down方式整合其他模組。	1.兼具Bottom-up與Top-down優點。	1.需小心規劃，否則殘根(Stub)程式與驅動程式總和，會比Bottom-up與Top-down多。
Big-Bang	1.將所有模組一次整合。 2.用於模組化程度甚高者，各模組間介面簡單、關連性低。	1.啟動成本低，不要殘根(Stub)程式與驅動程式。 2.可平行工作。	1.模組間關係複雜時，測試效果差。

軟體測試

撰寫測試案例

測試案例

- 測試的可預期性與可重複性
- 可以被複製的問題才能被解決
- 盡可能暴露最多的問題
- 測試案例描述測試內容

測試案例 (cont.)

- 內容
  - 編號
  - 標題
  - 前置條件：測試案例執行前的系統狀態
  - 測試步驟
  - 預期結果：判斷測試通過與否的準則
  - 優先程度

測試案例範例

編號	標題	前置條件	測試步驟	預期結果	優先度
TC_001	使用者可撥打內線電話與另一內線分機通話	電話已連接線路	輸入電話號碼 `666`	1. 撥打內線電話至分機 `666` 2. 接通後可與對方通話	高

來自需求的測試案例

- 需求
  - 當使用者撥打電話後，系統會連接使用者與另一方如此雙方便可以使用語音快速溝通

編號	標題	前置條件	測試步驟	預期結果	優先權
TC_001	使用者可撥打內線電話與另一內線分機通話	電話已連接線路	輸入電話號碼 `666`	1. 撥打內線電話至分機 `666` 2. 接通後可與對方通話	高
TC_002	使用者可撥打國內電話與其他號碼通話	電話已連接線路	輸入電話號碼 `(02)2655-7557`	1. 撥打國內電話號碼 `(02)2655-7557` 2. 接通後可與對方通話	高
TC_003	系統會提示使用者撥打了無效的電話號碼	電話已連接線路	輸入電話號碼 `948794`	系統回覆該號碼不存在	中

測試案例與輸入劃分

- 以一個僅接受 1 ~ 1000 的可輸入欄位為例，會需要設計幾個測試案例?

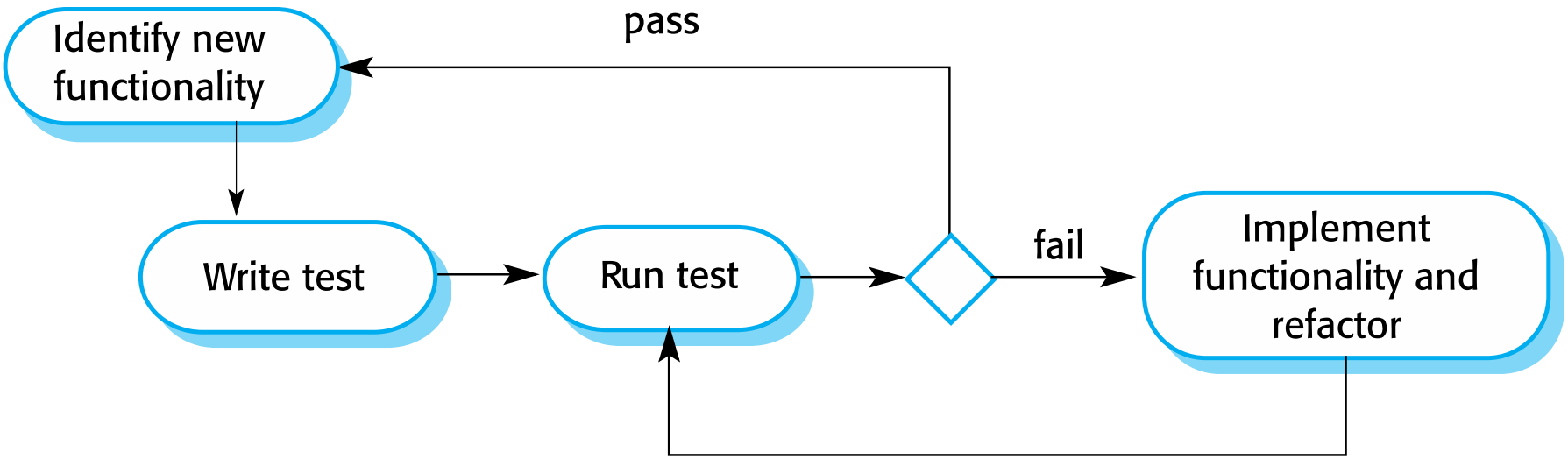
編號	輸入數字	預期結果	備註
1.	0	失敗	最小值 - 1
2.	1	成功	最小值
3.	2	成功	最小值 + 1
4.	666	成功	界於最小值與最大值間的數字
5.	999	成功	最大值 - 1
6.	1000	成功	最大值
7.	1001	失敗	最大值 + 1

編號	輸入數字	預期結果	備註
1.	0	失敗	最小值 - 1
2.	1	成功	最小值
3.	666	成功	界於最小值與最大值間的數字
4.	1000	成功	最大值
5.	1001	失敗	最大值 + 1

軟體測試

測試驅動開發

測試驅動開發 (TDD)


Identify new functionality
pass
Write test
Run test
fail
Implement functionality and refactor

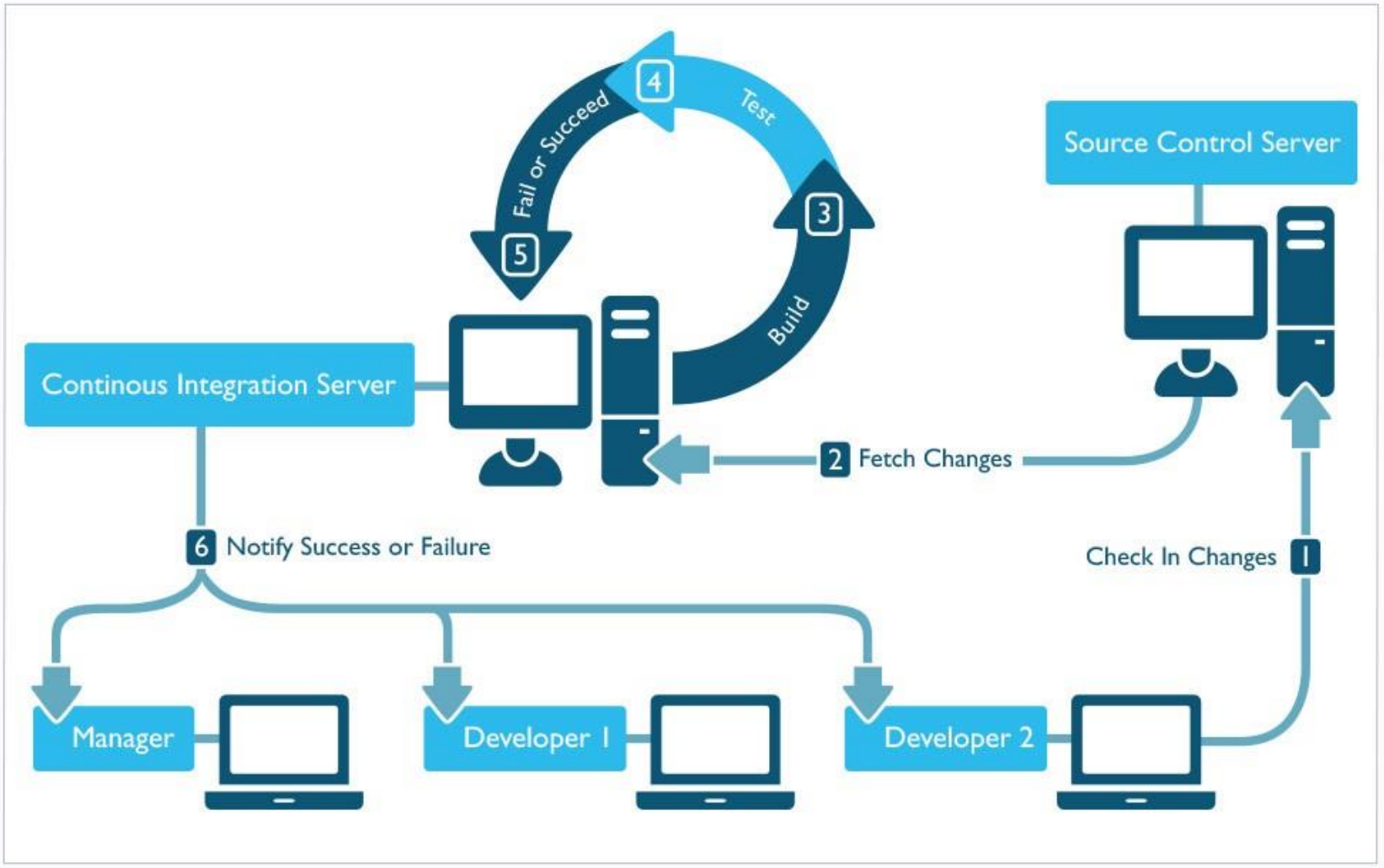
測試驅動開發 (cont.)

- 測試案例即需求，通過測試即滿足需求
- 以測試因應需求改變
  - 需求發生變動 =>
  - 重新撰寫/修改 測試案例 =>
  - 修改程式已通過更新後的測試案例
- 以程式驗證程式是否滿足需求
  - 測試案例也是程式碼
  - 可以自動執行的測試案例

迴歸測試 (Regression Testing)

- 確保系統的改變不會影響過去正常運作的功能
- 需要自動化環境的支持
- 在任何改變真正生效之前，必須先通過迴歸測試

持續整合 (Continuous Integration)



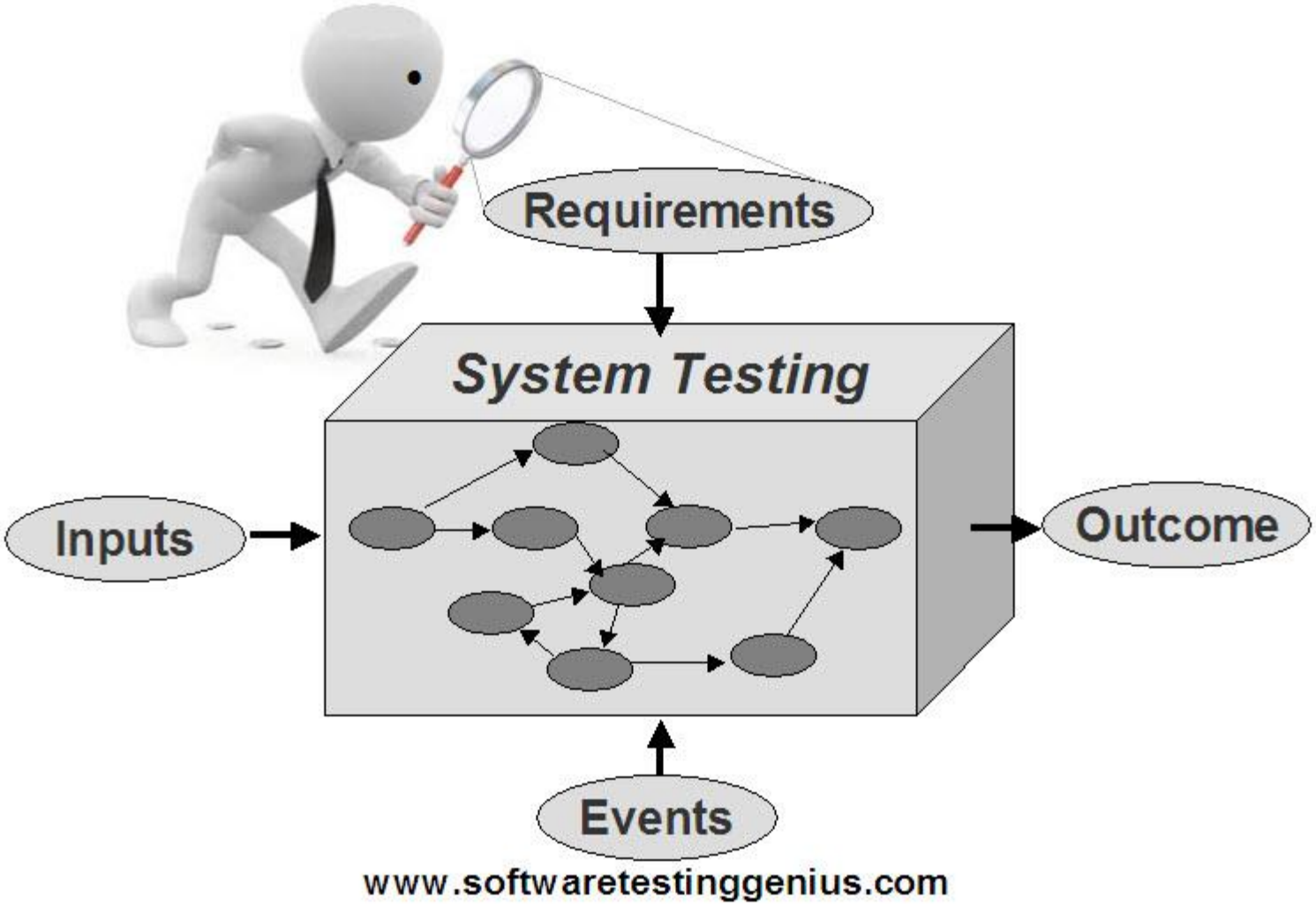
軟體測試

系統測試、驗收測試與其他

系統測試 (System Testing)

- 系統測試是最後的整合步驟
- 功能性
  - 以功能或需求為基底做測試
  - 商業流程為基底做測試
- 非功能性
- 與功能性測試一樣重要
  - 通常不被重視
  - 但此測試是必須被執行的
- 通常由測試人員進行測試

系統測試


Requirements
System Testing
Inputs
Outcome
Events
www.softwaretestinggenius.com

驗收測試 (Acceptance Testing)

- 確認所被實現的功能是否符合當初使用者所要求，解決使用者想解決的事
- 使用真實的使用者資料進行測試
  - Alpha Testing
    - 開發團隊
  - Beta Testing
    - 不特定使用者
- 測試軟體是否已經可以部署於真實的運作環境

與非功能需求有關的測試

- 易用性測試 (Usability Testing)

與非功能需求有關的測試 (cont.)

- 效能測試

Load Testing

Focus: "Response Time"

Stress Testing

Focus: "Response Time" and "Throughput"

Volume/Capacity Testing

Focus: "Response Time"

Endurance Testing

Focus: "Memory"

Scalable Testing

Focus: "Response Time" and "Throughput"

Summary

- 測試可以顯現錯誤，但是無法回答是否仍有錯誤
- 適當地運用過去開發其他系統的經驗來計畫測試
- 盡可能將測試自動化