Chapter 1

軟體工程概觀

資科系
林偉川

1

問題

- 84 % of all software projects do not finish on time and within budget (Survey conducted by Standish Group)
  - 8000 projects in US in 1995
  - More than 30 % of all projects were cancelled
  - 189 % over budget
- Key issues:
  - Software firms are always pressured to perform under unrealistic deadlines.
  - The clients ask for new features, just before the end of the project, and unclear requirements.
  - Software itself is awfully complex.
  - Uncertainties throughout the development project.

2

軟體工程簡介

- 軟體危機－Y2K、軟體成本、BUG不滅、開發進度、時間成本、膨脹的軟體管理、外包
- 軟體工程是學習一些工程開發方法，建構出切實可行的框架，來協助軟體發展的計劃、概念化與設計
- 工程方法和技術應用，用來為軟體系統進行
  - 計劃、規格制定、設計、實作、確認、測試、計量、維護與發展

3

軟體工程簡介

- 增量式發展：逐步遞增的方式發展部份功能及特徵，構成所謂軟體增量
- 軟體增量為系統的部份功能及特徵
- 一旦確定軟體增量與需求是一致且穩定的，接著便可進一步精鍊並加入新功能，Microsoft即是採行這種同步穩定的方法，並發揮相當大的成效

4

軟體工程簡介

- 開發機構的成熟度之評估：藉由能力成熟度模型(CMM)來達成，此模型針對軟體開發機構的持續改善程度，與相對應的『軟體程序成熟度等級』之鑑定，所建立出一套規範
- CMM是由美國CMU之軟體工程學院於1995所發展出來的，主要目的是幫助軟體機構改善其軟體開發程序的成熟度，CMM能促進軟體程序不斷地改善，由一開始混亂、雜亂無章的等級，進化至嚴密、精確且有紀律的等級

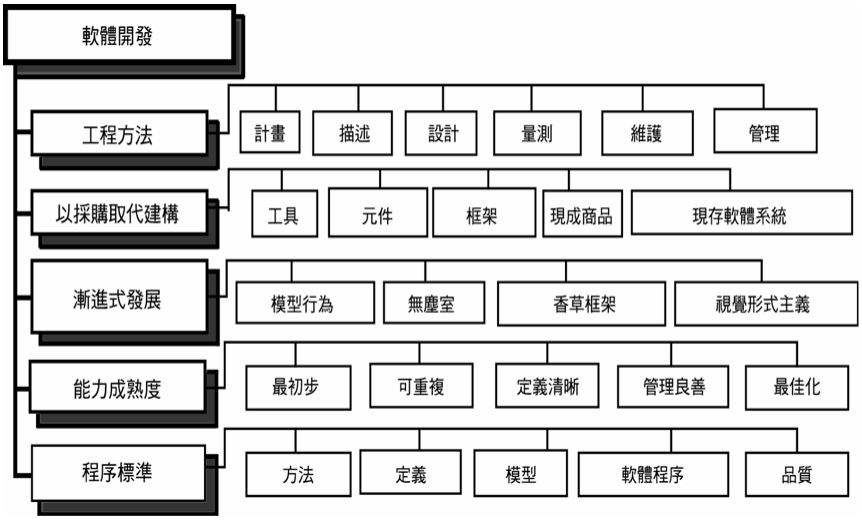
5

軟體工程簡介

- 軟體標準是規範開發軟體時應該遵循的方法、規則、需求及實作，有標準才能計量軟體實體的內容、價值或品質
- 軟體程序是由任何有助於開發軟體產品的活動、方法與實作所組合而成，包括：
  - 描述及準備藍圖：指明系統的資料與控制元件之結構
  - 編寫程式
  - 軟體的檢驗
  - 軟體的部署

6

軟體工程簡介



7

名詞簡介

- 軟體程序
  - 任何有助於開發軟體產品的活動、方法與實作所組合而成
  - 將使用者的需求轉換成軟體產品的所有相關活動
- COTS ➔購買取代建構
- 專案參與者 ➔包括顧客、外包廠商及承包商等
- 資訊隱藏
  - 開發者不必事必恭親地了解每一個細節
- 模組化結構
- 細節經練 V. S. 軟體增量性發展
- 標記語言 ➔ xml or html
- 軟體能力成熟度模型(CMM) ➔軟體標準

8

採購取代建構

- 以購買取代建構的方法為將成本用來購買現成商品(COTS)，會比閉門造車從新開發要來得有效率
- 可藉由購買部份甚至完整的軟體套件，可以加速軟體的開發形成

9

軟體工程的開發方法：元件

- **元件(Java Class)**
  - **經過測試、具有特殊用途的軟體單元**
  - **元件式軟體開發技術 ➔ 提供使用者再利用的程式碼**
  - **元件式程式設計(mega-programming) ➔ 元件可來自重複使用的函式庫 or component-based programming**
  - **Component-ware(CW)技術 ➔ 設計出可輕易掛入特性的個別軟體單元，這些單元可輕易掛入任何應用程式，以增強其功能，Fig. 1.2為元件式軟體與元件式服務所創造的營收預測圖表**

10

軟體工程的開發方法：元件

- 框架
  - 元件的組合體(類別函式庫)，可用來簡化程式的架構，且亦可外掛於程式
  - 軟體系統分解成一個個可重複使用的元件
  - 要發展出能使元件與程式銜接的介面

11

軟體工程的開發方法：元件

- 軟體介面
  - 讓不同元件能和彼此互動與溝通的程式
  - 元件介面技術
    - JavaBean/SUN不受限於平台介面，讓開發者創造及使用Java元件
      - 協助軟體開發者撰寫JAVA類別的元件架構
      - Bean為可重複使用元件，可匯出屬性、產生事件並執行Java方法
    - DCOM/Microsoft
    - AcitveX/Microsoft
      - 將現存的軟體元件轉變為內嵌於瀏覽器執行的元件

12

軟體工程的開發方法：元件

- 軟體產品是由元件所組成的電腦程式
  - 可理解、可使用、可擴充
- 軟體資源
  - 為了開發軟體所需的人力、設備及工作場所
- 軟體實體(software entity)
  - 軟體工程領域中的軟體程序、需求、產品和資源

13

工程方法

- 工程方法可以避免開發軟體過程中的混亂
- 實用的(Practical)
  - 因為它構築在許多經過證明的方法及實作基礎上
- 循序的(Orderly)
  - 活動及產品之先後順序與定義，可透過「軟體程序模型」來安排
- 計量的(Measured)
  - 每一個階段，都必須應用軟體度量(metric)去量測已完成的產品，量測結果是一種指標，判斷出軟體的品質、成本及可靠度(MTBF、MTTF)
- 應用系統化有紀律及可量化的方法，來進行軟體開發與維護。

14

工程方法

- 將科學原理與方法應用在軟體開發 ➔1968 NATO
- 有效節省成本並可產生優良的軟體產品
- 軟體開發需建立可量測的目標，將軟體品質進行量化
- 測量結果將某件事物與某項標準做比較以測定出其大小程度
- 計量軟體程序的工作產品時，重點在於產品之成本、可靠度與品質
- Bug不滅定律

15

軟體測量

- 代表軟體實質的屬性與某些數字或符號之間的關聯
- 軟體實質的屬性
  - 效能 (cpu時間、產能) 、複雜度(multithreading)、隱密字元(Unix shell cryptic)、可靠度(作業系統)等
  - Applet是可在網頁上執行的小程式，一旦被使用者下載後，Applet便能在不耗用伺服器資源的情況，執行工作並透過網頁與使用者互動

16

軟體測量

- 測量的動機來自於我們需要對軟體的內部結構及行為獲得更多了解
- 測量的活動從早期的軟體程序問題定義及專案規劃，一直到軟體維護等
- 測量的最終目的在於了解某的特定的軟體程序，是否有幫助減少軟體的本質性及附屬性問題

17

軟體開發的問題

- **概念性**的問題(**概念架構**、**本質性**的問題)
  - 如何制定、設計與測試一個軟體系統的概念結構
- **表現性**的問題(附屬特性)
  - 如何描述、表現軟體及測試此表現方式的精確度
- **概念性的問題**較為困難，因軟體的**實體本質**是由錯綜複雜的概念架構而成
- 一個程式內的資料集、資料項的關聯、演算法、與函數呼叫中都可見概念的存在

18

軟體開發的問題

- 表現性的問題關切的是軟體的附屬特性
- 附屬特性(某特性存在不影響某事物存在與否)
  - 使用的程式語言(高階或機器語言)
  - 程式組成方式(模組化或單一區塊)
  - 是否以物件導向來開發，軟體的功能不會改變

19

軟體開發的問題

- 軟體本質的屬性
  - 複雜度 → 愈複雜愈難理解 O(n)
    - 一個程式中條件式的數量、疊套階層、資訊流及敘述式計量來量測或可用資源(人員或設施數量)、時間(程式平均開發或運算時間)、空間(工作或硬碟空間)
  - 抽象化
    - 呈現軟體需求功能、使用方式、或是架構規格
    - 隱藏在軟體程序模型下的邏輯概念

20

表1.1 軟體工程的革新技術解決問題

對付本質性問題的方法	解決附屬性問題的技術
1. 購買取代建構（現成商品方法）	8. 高階程式語言
2. 透過以原型（prototype）與客戶進行溝通的方式，反覆不斷地精鍊需求規格	9. 物件導向（Object-oriented；OO）程式設計
3. 漸進式軟體開發	10. 人工智慧（Artificial intelligence；AI）
4. 培養有天份的系統設計師	11. 專家系統（Expert systems）
5. 香草框架	12. 自動化程式設計
6. 將軟體系統模組化	13. 視覺化程式設計
7. 軟體系統的分析	14. 程式驗證（verification）
	15. 硬體改良

MTBF

- **What is MTBF?**
  MTBF is an abbreviation for Mean Time Between Failures.
  MTBF is a measure of how reliable a product is. MTBF is usually given in units of hours; the higher the MTBF, the more reliable the product is.
  For electronic products, it is commonly assumed that during the useful operating life period the parts have constant failure rates, and part failure rates follow an exponential law of distribution. In this case, the MTBF of the product can be calculated as:

22

MTBF

- MTBF = 1/(sum of all the part failure rates)

  and the probability that the product will work for some time T without failure is given by:

  R(T) = exp(-T/MTBF)

  Thus, for a product with an MTBF of 250,000 hours, and an operating time of interest of 5 years (43,800 hours):

  R = exp(-43800/250000) = 0.839289

  which says that there is an 83.9% probability that the product will operate for the 5 years without a failure, or that 83.9% of the units in the field will still be working at the 5 year point.

23

MTBF

- MTBF專有名詞解釋：(Mean Time Between Failure)平均無故障時間,平均故障間隔時間
  說明：它是衡量PC產品(尤其是電器產品)的可靠性、穩定性最重要的一項指標，屬於國際行業標準，單位為"小時"，是產品在規定時間內保持功能的一種能力。
  具體來說，是指相鄰兩次故障之間的平均工作時間，也稱為平均故障間隔。它僅適用於可維修產品。同時也規定產品在總使用階段累計工作時間與故障次數的比值。

24

MTBF

- 電子產品的MTBF一般不能低於：
  電腦：4000小時
  磁碟陣列：50000小時
  MO光碟機：100000小時以上
  聯想電腦：45000小時
  UPS：50000小時
  CD：100000小時
  LG液晶顯示器:50000小時

25

MTBF

- MTBF平均無故障工作時間的規定大陸的MTBF的大約15000小時左右，國際大廠，基本上是40000小時。
- 試驗測試並不會把一台電腦真的放在那裏不停地運行45000小時，以聯想來說共使用196台電腦進行測試，結果在30天內每天連續24小時的連續測試中，全部電腦都無一故障全部通過，上述方式完全符合MTBF測試的標準和換算方式，如果在測試過程中，出現一起故障，那整個測試時間就自動延長7天，如果出現3起故障，那就意味著本次測試失敗。

26

MTBF

- 大多數廠家是根據產品的平均無故障時間是按照產品實際使用情況的紀錄進行計算的。
- 產品的實際MTBF公式
  平均無故障時間（MTBF）的計算方法首先我們確認每個月安裝的產品的平均數(A)，並計算產品發售的時間(B)，從而計算出產品使用的總月數。
  C = (Ax1)+(Ax2)+…(AxB) 然後我們認定賣出的產品有70%在使用中，並且每天工作12小時。從C計算出產品使用的總小時數D。
  D = Cx70%x12x30最後用產品使用的總小時數D除以產品出現故障的次數E，得出該產品的平均無故障時間。MTBF = D / E

27

物件導向方法引入之技術

- 資料抽象化、模組化、資訊隱藏、資料封裝
- 資訊隱藏是藉由將軟體分解為模組的方式，把程式設計細節隱藏在其中
- 模組在邏輯上是程式中的獨立元件，每個模組皆隱藏其所含資料結構的特徵，及內容之設計細節，並提供必要的操作介面，讓使用者得以正確地使用該模組之功能，但無法取其內部資料
- 繼承 ➔ 再利用率
- 同型異構

28

物件導向方法引入之技術

- 一個類別可以想程式一個具有直觀操作與封裝資料結構的模組
- 一個物件(object)則是一個類別(class)的實例(instance)
- 運用物件導向技術可將注意力由如何設計類別，轉為如何呼叫現存類別所提供的操作介面(method)

29

物件導向方法引入之技術

- 類別與物件提供功效顯著的資訊隱藏功能，進而簡化軟體設計
- 物件導向技術不僅解決軟體表現性的問題，同時也解決構成軟體基礎的概念架構的問題
- 軟體工具或開發環境成為商品、開發新程式的類別庫(class library)和應用程式框架(application framework)都可購買取得

30

處理軟體本質性的難題

- 快速塑型(rapid prototyping)對需求進行反覆及互動式的精鍊是對抗軟體概念性本質問題最好方法之一(善用圖形人機介面)
- 軟體原型是模擬某特定的操作介面，並執行一個或數個系統的主要功能
- 塑造軟體原型的好處是可以很容易地呈現出特定的概念結構，使客戶得以測試其一致性與可用性

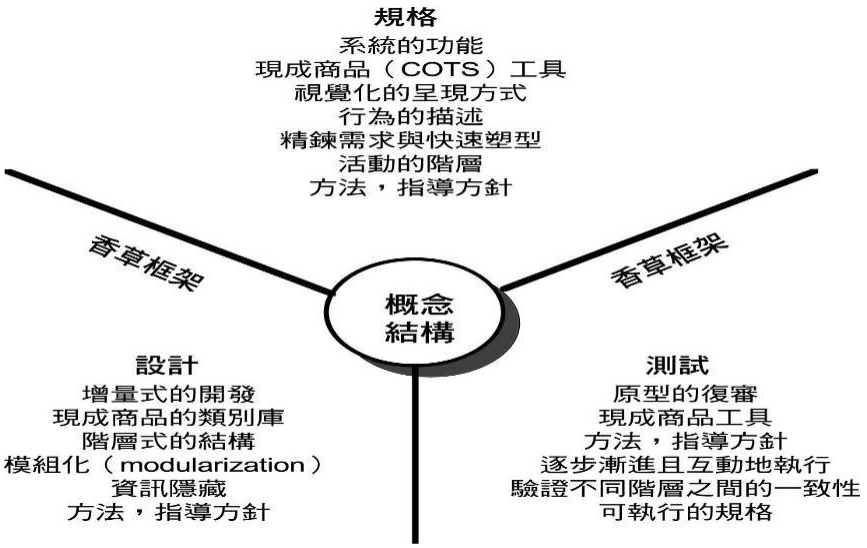
31

處理軟體本質性的難題

- 另一種方法為逐步漸進地開發軟體(Incremental Development)
- 軟體原型加逐步漸進地開發軟體，此兩種方法合併起來就是香草框架(vanilla framework)

32

香草框架



香草框架

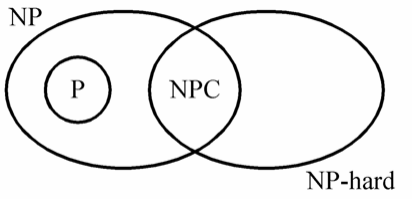
- 解決軟體本質性問題的方法，是一種演進式、通用的概念架構，可協助縮短設計軟體時高階概念與實作間的隔閡
- 是一種由上到下較為自然的方式去掌握軟體系統的概念結構，可以控制及管理系統
- 制定系統規格時是使用分層的方法，其構想是建立一功能性的階層關係，不同階層是由許多控制活動交織而成
- 藉由視覺的方式展現階層的概念架構，良好的視覺符號有助於表達良好的演算法

34

演算法原則

- 利用什麼方法，由原始的概念或資料，推導出結論?此推論是否堅實(soundness)?完整(completeness)?
  - 堅實(soundness)：每一推論步驟是否均正確無誤?或者可能潛藏錯誤(如歸納法之結論)?
  - 完整(completeness)：所有的答案，是否均被推導出來?

35



- **P**: the class of problems which can be solved by a deterministic polynomial algorithm.
- **NP** : the class of decision problem which can be solved by a non-deterministic polynomial algorithm.
- **NP-hard**: the class of problems to which every NP problem reduces.
- **NP-complete (NPC)**: the class of problems which are NP-hard and belong to NP.

36

Decision problems

- The solution is simply "Yes" or "No".
- Optimization problems are more difficult.
- e.g. Traveling Salesperson Problem (TSP)
  - Optimization version:

    Find the shortest tour
  - Decision version:

    Is there a tour whose total length is less than or equal to a constant *c* ?

37

香草框架

- 以階層的方式看待軟體系統的模型，每個階層中的活動及捕捉了系統的功能
- 不同的系統元件有可能是重疊的
- 資料元素與資料儲存藉由輸出及輸入(fan-in、fan-out)，在不同層級的活動間流動
- 軟體的控制活動即構成行為，對行為描述需求來自反應式系統(event-driven)
- 反應式系統等候來自外在環境觸發的事件，並能立即地對輸入作出回應 ➔ 人機使用者介面、即時處理控制器

38

軟體開發的工程引出技術

- 需求工程
- 軟體工廠
- 無塵室工程(cleanroom engineering)
- IEEE軟體工程標準
- 軟體程序成熟度框架(SPMF)➔ CMMI

39

軟體開發的工程技術

- 需求工程指的是有系統地運用可驗證的原理、方法、語言及工具，去分析與描述使用者的需要，同時描述軟體系統中行為與非行為的特徵，這些特徵可滿足使用者的需要

40

軟體開發的工程技術

- 需求工程的副產品主要為：軟體需求規格(SRS)與軟體產品之非行為需求
- 軟體需求規格為人們對軟體計畫中提出的問題作深入詳盡的分析(包括所須配合硬體的需求)
- 需求工程主要分為兩個階段
  - 檢測(問題解決與分析階段)
  - 行為的及非行為的規格制定(軟體描述階段)
    - 行為的描述➔軟體程序之資料流進入與離開條件是可被驗證
    - 非行為的描述➔產品品質須達到某計量的屬性

41

軟體開發的工程技術

- 軟體產品的屬性指的是軟體產品可計量特性
- 軟體產品可計量特性包括：人機工程程度(GUI的擺設規範)、可靠性及可維護性(OS穩定性考量)，軟體品質量測計量則依據某些準則來決定
- 軟體量測是針對其屬性，以階層的方式進行

42

軟體開發的工程技術

- 整體軟體品質評估就高階屬性稱為因子，因子的計量是根據低階屬性稱為準則
- 軟體因子是以使用者導向的觀點來看待產品品質，準則是以軟體導向的特性來表示產品品質
- 軟體品質因子(高階屬性)與準則(低階屬性)的因果關係如下表

43

品質因子與準則

品質因子(因)	準則(果)
正確性	可追蹤性、完整性、一致性
可靠性	容錯能力、一致性、準確性、容易度
可維護性	一致性、簡潔度、簡易度、模組化 等
可測性	簡易度、模組化、測試儀器 等
效率	執行、儲存效率
完整性	存取控制、存取稽核
可使用性	操作性、訓練、親切度 等
可攜性	軟體系統獨立性、軟體獨立性 等
互用性	通訊、資料的共通性
可重複使用性	自我描述能力、模組化、可攜性 等

44

容錯能力

- 容錯能力(**Fault-tolerance) by** Duplication can give fault-tolerance in three ways:
- Replication: Providing multiple identical instances of the same system, directing tasks or requests to all of them in parallel, and choosing the correct result on the basis of a quorum;
- Redundancy: Providing multiple identical instances of the same system and switching to one of the remaining instances in case of a failure (fall-back or backup);

45

容錯能力

- Diversity: Providing multiple *different* implementations of the same specification, and using them like replicated systems to cope with errors in a specific implementation.
- A redundant array of independent disks (RAID) is an example of a fault-tolerant storage device that uses redundancy.

46

容錯能力

- A lockstep fault-tolerant machine uses replicated elements operating in parallel. At any time, all the replications of each element should be in the same state. The same inputs are provided to each replication, and the same outputs are expected. The outputs of the replications are compared using a voting circuit.

47

容錯能力

- A machine with two replications of each element is termed **dual modular redundant** (DMR). The voting circuit can then only detect a mismatch and recovery relies on other methods.

48

容錯能力

- A machine with three replications of each element is termed **triple modular redundant** (TMR). The voting circuit can determine which replication is in error when a two-to-one vote is observed. In this case, the voting circuit can output the correct result, and discard the erroneous version. After this, the internal state of the erroneous replication is assumed to be different from that of the other two, and the voting circuit can switch to a DMR mode. This model can be applied to any larger number of replications.

49

軟體品質的計量

- 軟體品質是根據準則量測結果的加權總合來評估
- 測量軟體品質的方法(品質因子)
  - 步驟1：選定用來測量某個軟體因子(software factor)的準則
  - 步驟2：決定每個準則的權重(weight)w(通常0≦w≦1)
  - 步驟3：決定每個準則分數(criteria score)的尺度(例如：0≦準則分數≦10)
  - 步驟4：選定每個準則分數的最大及最小目標值

50

軟體品質的計量

- 步驟5：選定每個因子分數(factor score)的最大及最小目標值
- 步驟6：為每個準則給定一個分數
- 步驟7：計算加權後的總和
- 步驟8：將加權總和與之前設定的因子分數之最大－最小範圍作比較
- 步驟9：若加權總和落在最大－最小分數的範圍以外，則將每個準則的分數單獨與設定的準則分數範圍作比較，作為軟體改進活動的指引

51

測量軟體的可重覆使用性

- 有兩個軟體產品分別為Steersman與Ucontrol，我們打算量測其軟體實體的可重覆使用性
- 軟體品質量測方法的步驟
  - 自我描述能力(SD)
    - 軟體藉由自然語言的使用，得以自我解讀及理解
    - Pascal、COBOL (better) V.S. Java、Fortran
  - 模組化(M)
    - 一系統或電腦程式分解為彼此獨立的元件，使每一元件的改變，對其他元件造成的衝擊減到最小(C++、Java、Occam)

52

測量軟體的可重覆使用性

- 可攜性(P)
  - 軟體由某個硬體或軟體環境，轉移至其他環境的容易程度
- 平台獨立性(PI)
  - 軟體不受限於某一特定類型的電腦平台，而能在多種不同平台上執行的能力(Java Applet [better] V.S. COBOL)

- 針對特定專案，計畫團隊須制定適當的準則及準則的權重，以作為軟體開發的指引
- 由資深的軟體開發師指定各項分數，可訂定軟體品質因子的目標分數，以提供軟體開發團隊一個測量的尺度，這些尺度提供判斷軟體實體的接受度基礎

$$reusabilit\ y = W_1(SD) + W_2(M) + W_3(P) + W_4(PI)$$

53

可重覆使用性的準則方法的前3個步驟

可重覆使用的準則	Steersman分數	Ucontrol分數	權重
自我描述能力（SD）	5	1	$w_1 = 0.8$
模組化（M）	5	7	$w_2 = 0.9$
可攜性（P）	9	3	$w_3 = 0.2$
平台獨立性（PI）	1	9	$w_4 = 1$

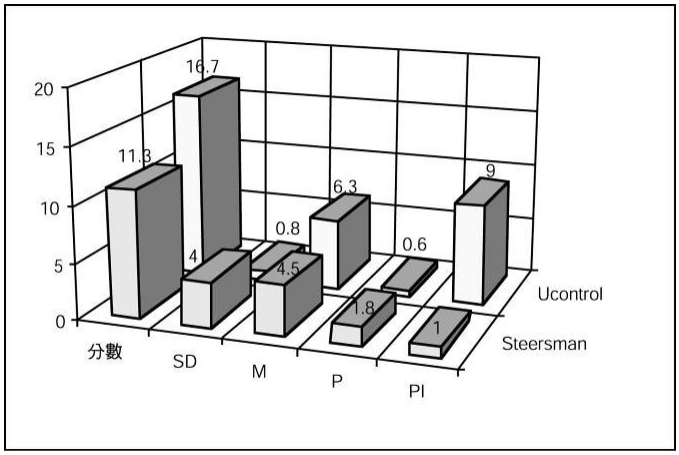
54

可重覆使用性的評估Ucontrol 較好

Steersman 的可重覆使用性	Ucontrol 的可重覆使用性
reusability $= w_1(SD) + w_2(M) + w_3(P) + w_4(PI)$ $= (.8)(5) + (.9)(5) + (.2)(9) + 1$ $= 4.0 + 4.5 + 1.8 + 1$ $= 11.3$	$w_1(SD) + w_2(M) + w_3(P) + w_4(PI)$ $= (.8)(1) + (.9)(7) + (.2)(3) + 9$ $= 0.8 + 6.3 + 0.6 + 9$ $= 16.7$

55

軟體產品的加權分數



56

比較說明

- Ucontrol 之16.7比Steersman之11.3較好
- 若提高Steersman之平台獨立性由1變為6則計算結果由11.3提高至16.3
- 藉此提供開發者一個明確的方向，去改良軟體產品以臻專案目標
- 先前設定的最大－最小因子分數範圍，則點出軟體實體的品質不足之處
- 可假設每個加權準則的最小目標值為5，最大目標值為15可描出相對的kiviat圖(雷達圖)

57

統計圖表結果

- 藉由kiviat圖描繪出測量結果與最大值最小值之間的關係，可訂定其標準範圍值
- kiviat圖是畫出一個帶有輪幅的車輪，以車輪內部的輪轂半徑代表一個選定的最小值，以車輪外框的半徑代表一個選定的最大值，每條輪幅則代表每一項待測的準則
- 理想狀況是每一項準則的量測結果應該落在kiviat圖的外框(最大值)與輪轂(最小值)之間

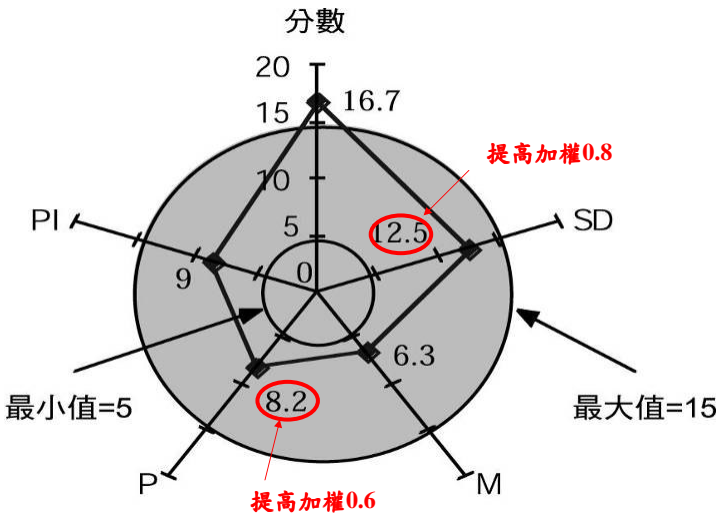
58

統計圖表結果

- 軟體品質因子分數的目標水準，可經由事前與客戶的協調而作調整
- 若將**可重覆使用性**目標分數的範圍設為[15,18]，則Ucontrol軟體的分數已達可接受的水準(16.7)而Steersman軟體的分數(11.3)則得再做加強
- 軟體品質因子的分數評估，可透過與『軟體品質控管團隊』所指定的最大及最小因子分數值做比較而決定

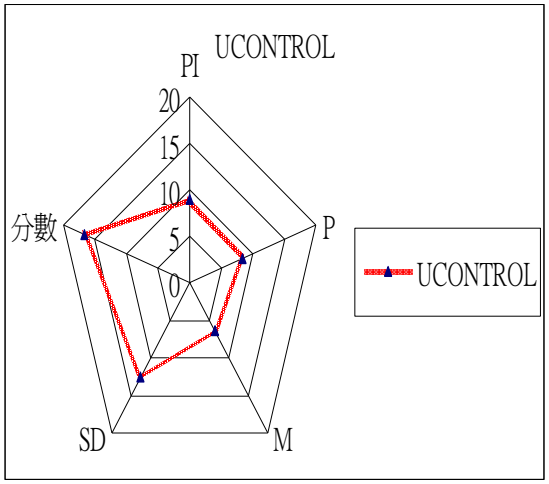
59

Kiviat圖形



60

以Excel畫出Kiviat圖



	A	B
1		UCONTROL
2	PI	9
3	P	8.2
4	M	6.3
5	SD	12.5
6	分數	16.7

61

採購取代建構

- 軟體開發常是重複使用、擴展或精鍊現存的軟體產品
- 購買現成的軟體比重新打造要有利得多
- 開發一個可執行於網頁瀏覽器之跨平台圖形人機介面的使用者介面➔Java Applet使用Swing套件，絕對比重新開發這些技術節省時間及人力成本

62

採購取代建構

- 軟體工廠
  - 如何將現存的軟體整合到一個新的產品中，或是如何修改他們以產生新的產品
  - 將軟體工程的工具及技術，與產品、程序及組織開發的管理技術結合在一起
  - 可適用於一系列性質相似的軟體專案
- 軟體工廠可帶來的效益有：可重複使用性質相似的軟體程序計劃及軟體工程指引、品質的分析與控管、技術的標準化、開發相似產品的軟體實體之可重複使用性、漸進式地改進產品設計與效能

63

增量式開發

- 面對開發軟體時，系統需求及資源不斷改變的狀況，漸進式發展技術則提供主要解決之道
- 每一個增量不宜改變太大，並且審慎地選定，為將來可能的增量鋪路，共有兩種方法：
  - Humphrey法則
  - 無塵室工程

64

Humphrey法則

- 用一系列的規則將增量式軟體開發技術公式化，以利軟體經營團隊在軟體程序的需求與設計階段有方法可循
- 表1.5列出適當的軟體開發階段與**Humphrey**法則，以及觸發我們引用這些法則的事件

65

表1.5 增量式開發方法

- **需求階段 ➔ 維持穩定性，軟體經營進化模型**
  - **觸發事件：軟體需求的增量步驟已完成**
  - **回應方式：凍結需求(凍結漸進步驟)**
  - **編號：1**
  - **Humphrey法則：在開始設計前，凍結每個增量步驟的需求，使開發人員確保他們在實作階段的增量步驟，藉由設計階段的回饋，引發需求的變更➔軟體經營漸進化模型，此為無塵室軟體開發技術的奧秘**

66

表1.5 增量式開發方法（續）

- 需求與設計階段 ➔有助於對未來增量的理解
  - 觸發事件：準備增量
  - 回應方式：選定增量以支持將來的增量
  - 編號：2
  - Humphrey法則：適合於軟體開發的需求與設計階段，選定每個增量以支持未來的增量，並且（或者）增進對需求的了解

67

表1.5 增量式開發方法（續）

- 設計階段 ➔每次軟體增量的幅度不宜過大
  - 觸發事件：產品的需求定義已趨穩定
  - 回應方式：選擇規格較小的增量進行實現
  - 編號：3
  - Humphrey法則：以小幅度、增量式的步驟製作軟體產品

68

表1.5 增量式開發方法（續）

- 設計階段 ➔ 每個增量需求在實作階段都不會被改變
  - 觸發事件：需求改變的發生
  - 回應方式：將改變延期
  - 編號：4
  - Humphrey法則：在製作期間若需求發生改變，設法將這些改變延至未來的增量再作實現，每個增量步驟稱為建構，每個建構的目標是儘可能產生可執行的程式碼。

69

表1.5 增量式開發方法（續）

- 設計階段
  - 觸發事件：急迫的需求改變發生
  - 回應方式：停止開發，重新審視及修訂軟體需求規格（SRS）
  - 編號：5
  - Humphrey法則：若改變相當急迫，停止開發工作，修改需求，修訂計劃，然後再由設計階段開始

70

無塵室工程

- 由科學化的方法及工具構成，可對遞增開發中的軟體產品，進行品質的評估與控管，並保證軟體產品在遞交給客戶時的適用性。
- 主要特色為增量式發展及透過對軟體增量以驗證為基礎的檢測、與統計測試方式，對品質進行獨立的評估(Kiviat圖)
- 在軟體開發過程中嚴格地將設計與測試階段分隔開來，以便有效達到品質的控管

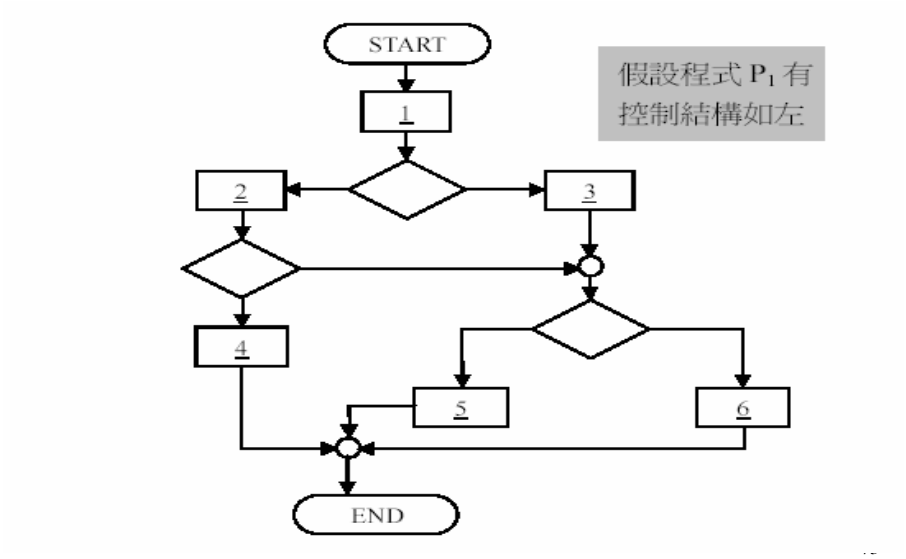
71

無塵室工程

- 軟體設計及測試相關的活動都屬於管線的一部份(pipeline)
- 用管線技術控制軟體開發活動，可提高開發速度。當設計者一旦完成手頭上的增量描述時，下一個增量的描述工作即可立即展開
- 採用增量式發展使得軟體程序較容易被控制
- 每個新的軟體增量得接受個別測試
- 測試可以簡單的分為：單元測試(unit test)→模組測試→整合測試→子系統測試→系統測試→驗證測試

72

單元測試範例



單元測試範例

- P1 至少須執行三個測試案例
- 例一：
  - Case 1：1，2，4、Case 2：1，2，5、Case 3：1，3，6
- 例二：
  - Case 1：1，2，4、Case 2：1，3，5、Case 3：1，2，6
- 路徑測試（path testing）➔
  - 每一條路徑在測試案例中，要至少執行一次。
  - P1 至少須執行五個測試案例
    - Case 1：1，2，4、 Case 2：1，3，5、 Case 3：1，3，6 、Case 4：1，2，5、Case 5：1，2，6

74

驗證測試

- 又稱為先期測試(pilot test)
- 驗證開發出來的系統是否滿足使用者的需求規格

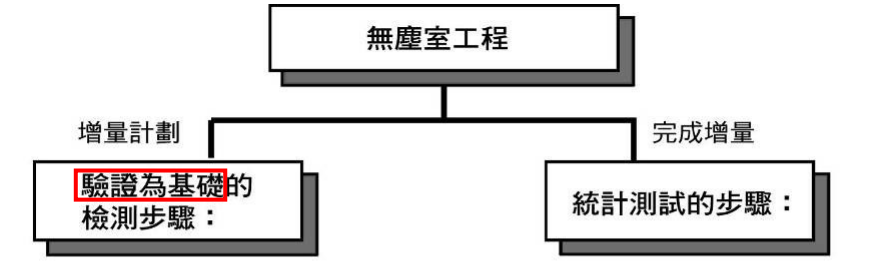
75

驗證測試

- 又可區分為
  - α test：
    - 由使用者在開發者端(軟體公司)進行的測試
    - 軟體公司以本身的成員擔任使用者，先期進行的測試
  - β test：
    - 由客戶在使用者端，進行的先期測試
    - 裝機測試(installation test)
    - 檢查系統的完整性(completeness)
    - 檢查系統的特性(功能性與非功能性)是否受到工作環境的影響？

76

無塵室工程的基本步驟



1.制定軟體增量應符合的條件（功能）。
2.開發軟體增量（例如：需求、架構、模組程式碼）。
3.證明該增量確實符合指定的功能及特性。

1.找出須測試的軟體功能，以及執行該功能所須的輸入資料。
2.根據使用者實際使用狀況，描述軟體輸入資料的特性。
3.根據步驟2找出的操作特性，隨機地選擇測試資料。
4.判定實現該增量的功能是否通過測試。

無塵室工程

- 軟體增量以驗證為基礎的檢測取決於規格中所指定該增量必須達到的正確條件➔開發前三思而後行
- 正確條件能幫助開發團隊聚焦，致立於正確的方向，同時提供衡量準繩，讓我們得以檢驗一個增量符合其條件的程度
- 可有效消弭缺點並促進品質的提昇
- V & V (Verification & Validation) : do the right thing & do the thing right!!!

78

Verification

- **Verification:** Am I building the product right, Determining if the system complies with the requirements and performs functions for which it is intended and meets the organization's goals and user needs. It is traditional and is performed at the end of the project, Am I accessing the right data (in terms of the data required to satisfy the requirement)➔按圖施工
- Demonstration of consistency, completeness, and correctness of the software at each stage and between each stage of the development life cycle.

79

Verification

- Performed during development on key artifacts, like walkthroughs, reviews and inspections, mentor feedback, training, checklists and standards
- Low level activity

80

Validation

- **Validation:** Am I building the right product, The review of interim work steps and interim deliverables during a project to ensure they are acceptable. To determine if the system is consistent, adheres to standards, uses reliable techniques and prudent practices, and performs the selected functions in the correct manner, Am I accessing the right data (in the right place; in the right way)➔開發出來的是對的產品，是使用者要的東西
- Determination of correctness of the final software product by a development project with respect to the user needs and requirements.

81

V & V

- Performed after a work product is produced against established criteria ensuring that the product integrates correctly into the environment
- High level activity

82

無塵室工程

- 驗證過的軟體增量遞交至檢定團隊時，會進行統計測試，統計測試是創造一個使用模型，定義所有可能的軟體使用狀況，以找出壓力狀態並根據使用模型，隨機地產生測試資料。
- 每個測試案例的輸入資料，可以檢驗出軟體可能的使用情形，使開發者確定軟體可在使用模型所指出的壓力狀態下，正確無誤地執行

83

無塵室工程的14個程序準則

- 程序類型：管理(存在於軟體開發流程的所有階段，需與所有其餘的無塵室工程程序同步實施)
- 程序包括：
  1. 專案計劃
  2. 專案管理
  3. 效能提昇
  4. 工程變更程序
- **無塵室工程**主要由**14**個程序組成，用來量測軟體團隊的**開發效能**，表**1.6**於1-26頁列出每個程序產出的工作產品

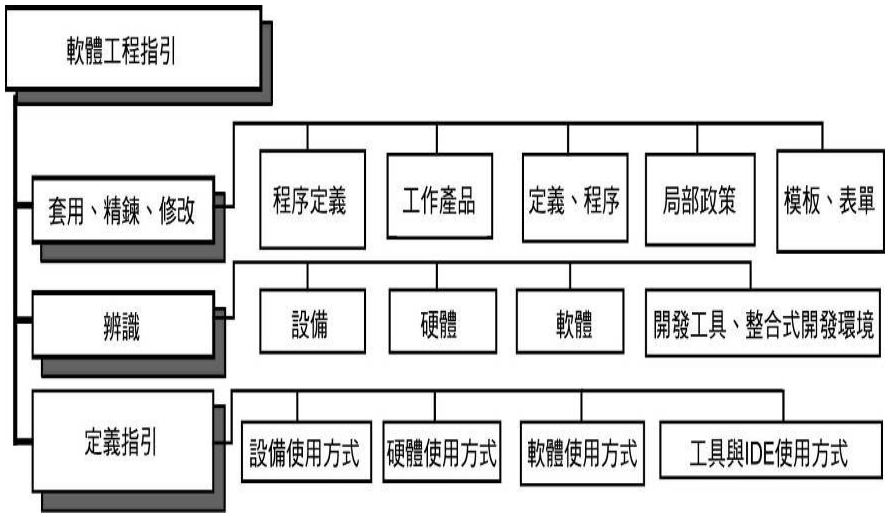
84

無塵室工程的準則

- 工作產品包括：
  - 無塵室工程指引(CEG) 圖1.8 於1-26頁
  - 軟體開發計劃(SDP)
  - 專案記錄
  - 效能提昇計劃
  - 工作變更日誌

85

無塵室工程的指引結構(CEG)



86

CEG & SDP

- CEG必須不斷經鍊以供無塵室團隊、生產線與特定專案使用
- CEG也應被修改以適合標準(ISO or IEEE) 、特定技術 (Windows NT OS)以及特定程式語言 (Java or Ada)
- SDP主要目的是促進效能的追蹤、量化的程序管理以及展開如需求分析等工作
- SDP定義出軟體程序管理的基礎，以及軟體程序之工作產品的組織、排程、追蹤、評量及控制

87

無塵室工程的14個程序準則

- 程序類型：規格製定(需求階段)
- 程序包括：

5. 需求分析
6. 功能規格制定程序
7. 使用規格制定程序
8. 架構規格制定
9. 增量計劃程序

88

無塵室工程的14個程序準則

- 工作產品包括：
  - 軟體需求
  - 功能規格
  - 使用規格
  - 軟體架構
  - 增量建構計劃

89

無塵室工程的14個程序準則

- 程序類型：開發(設計階段)
- 程序包括：

10. 軟體工程
11. 增量設計程序
12. 正確性驗證
13. 建立使用模型及測試計劃程序

90

無塵室工程的14個程序準則

- 工作產品包括：
  - 再造計劃 (re-engineering)
  - 再造軟體
  - 增量設計
  - 增量驗證報告
  - 使用模型
  - 增量測試計劃
  - 統計測試案例

91

無塵室工程的14個程序準則

- 程序類型：檢定(統計控制階段)
- 程序包括：

  14. 統計測試及檢定程序
- 工作產品包括：
  - 可執行系統
  - 統計測試報告
  - 增量檢定報告

92

能力成熟度模型(CMM)

- Software process *capability* describes the range of expected results that can be achieved by following a software process.
  - Predicting outcomes for the next project.
- Software process *performance* represents the actual results achieved by following a software process.
  - Focusing on the result achieved.
- Software process *maturity* is the degree to which a specific process is explicitly defined, managed, controlled and effective.
- *Infrastructure* is the underlying framework of an organization.

93

能力成熟度模型(CMM)

- *Culture* is the way doing thing.
- *Institutionalization* is the building of infrastructure and culture to support the methods, practices and procedures.
- Therefore, a mature software organization needs an infrastructure and culture to support its methods, practices and procedures so that they endure forever.

94

能力成熟度模型(CMM)

- Benefits and Risks of CMM
  - Maturity level, key process areas, common features and key practices.
  - Disciplined, consistent, predictable.
  - Sound judgment is necessary to use the CMM correctly.
  - Intelligence, experience and knowledge must shape an appropriate interpretation of the CMM in a specific environment.

95

能力成熟度整合模型(CMMI)的演化

- Nov, 1986, Carnegie Mellon University' SEI (http://www.sei.cmu.edu/) and MITRE (http://www.mitre.org) developed a process maturity of framework for assessing the capability of federal government's software contractors.
- Sep, 1987, Watts Humphrey reported a brief description of software process maturity framework.
- 1991: CMM V. 1.0 for software published
- 1993/1994, CMM V. 1.1 for software published, Personal Software Process (PSP developed by the SEI)

96

能力成熟度整合模型(CMMI)的演化

- 1995, New specialized CMM published by the SEI, including
  - CMM's for acquisition (SA-CMM)
  - System engineering (SE-CMM)
  - Integrated product development (IPD-CMM)
  - Human resource management (People-CMM)
- 1996: Team Software Process (TSP) developed by the SEI
- 1997, To emerge EIA/IS 731, Capability Maturity Model Integration Project by the DOD

97

能力成熟度整合模型(CMMI)的演化

- 12/2000: CMMI V. 1.02 Published (III Translation of Chinese Version)
- 12/2001,CMMI V. 1.1 Published
- 1/2002: Release V 1.1 Training Materials to Transition Partners
- 3/2002: Publish CMMI-SE/SW/IPPD/SS V1.1
- 12/2003: Complete sunset period for precursor models

98

能力成熟度整合模型(CMMI)的演化

- Two methods:
  - Software process assessment, to determine the state of an organization's current software process. Basically, it is for an internal process improvement program.
  - Software capability evaluation, to identify contractors qualified to perform the software work.
- Consistency and predictability will result in visible software process. ➔ Visibility

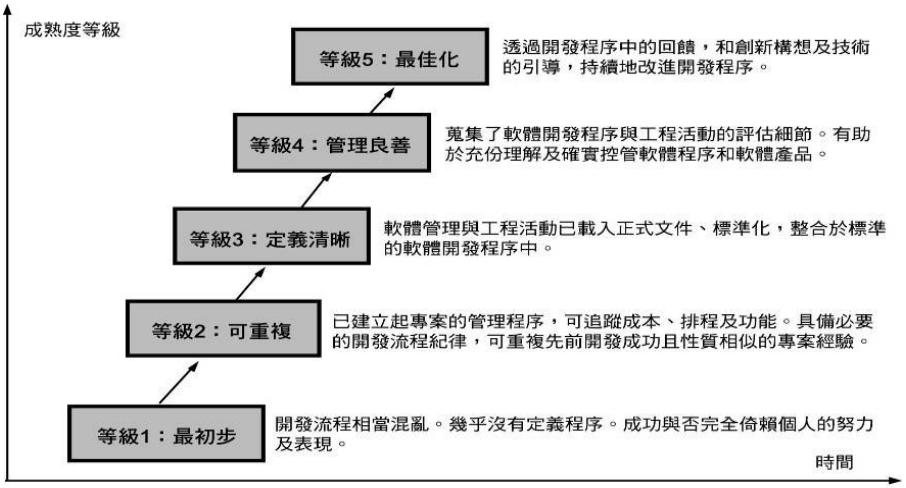
99

能力成熟度模型(CMM)

- 測量成熟度等級共分為五個等級
  - 每一等級都是可以具體衡量
- 建立清單(checklist)，列出描述每個成熟度等級特性的項目，接著估量軟體開發組織在每個項目上達到的程度，並計算估算程度的總和
- 依據總和的結果即可判斷出此組織所達到的成熟度等級
- 整個測量流程可被自動化，甚至可能模擬一個組織改變了成熟度等級後的情形

100

軟體程序管理的成熟度



能力成熟度模型最初的等級

- 混亂(chaos)
- 處於這個等級的組織為雜亂無章，儘依靠個人的努力完成軟體專案
- 缺乏計劃及明確的指導方針，使得軟體開發團隊無所適從
- 軟體開發程序幾乎完全沒有定義，若能創造出好的產品，簡直是奇蹟
- One man project!!!

102

能力成熟度模型最初的等級

- 表1.7的清單可用來判斷一個組織是否處於最初步等級
- 若獲得最初步等級中的較高分數，表示該軟體組織尚處於萌芽階段，開發流程會有潛在的混亂
- 清單中每一項目的分數指示出軟體機構的缺點(高分項目)或長處(低分項目)，進而得知改進的方向

103

能力成熟度模型最初的等級

- 運用相同的方法，計算其他成熟度等級清單的得分，就可以明確地評估軟體開發機構的能力成熟度
- 若需改進，則應由SDP與CEG開始著手進行基本的改進

104

能力成熟度等級的內部結構

- 能力成熟度等級的測量可藉由關鍵程序區(KPA)來實施
- 每一個關鍵程序區定義了一系列相關的活動，執行這些活動可以加強程序的成熟度
- 表1.8於1-29頁提供每個成熟度等級的關鍵程序區之概要
- 由於能力成熟度等級1之特徵為缺少定義明確的軟體程序管理秩序，該等級並無相對應的關鍵程序區，所以表1.8沒有列出等級1之關鍵程序區

105

等級2(可重複程序)關鍵程序區

- 其特色為具有一定開發紀律，以完成軟體開發專案
- 紀律是靠建立起專案規劃、推估(人月、資源、技術)、複審及追蹤系統的共識而達成的
- 適當運用這些開發準則，就有可能保證軟體產品的品質，並有效率地重複軟體開發程序

106

等級2(可重複程序)關鍵程序區

- 等級2共有6個關鍵程序區

1. 需求管理(RM)
   - 開發團隊對於客戶需求的共識及了解
2. 軟體專案規劃(PP)
   - 制定軟體專案與軟體程序管理的計劃
3. 軟體專案追蹤與監督(PT)
   - 建立軟體程序活動的能見度(visibility)，有助於察覺開發流程是否偏離專案計劃所訂的方向

107

等級2(可重複程序)關鍵程序區

4. 軟體轉包管理(SM)
   - 對軟體承包廠商的選擇與管理
5. 軟體品質保證(QA)
   - 對於技術性及計劃性的工作提供獨立的審查以確保產品完全根據計劃而實現
6. 軟體組態管理(CM)
   - 建立並監督任何工程上的改變，以維護軟體程序的產品 ➔ version control VB之VSS

108

等級3(定義清晰程序)關鍵程序區

- 由能力成熟度模型的等級3開始，正式步入標準的階段，此時軟體專案的建構及評估均有標準作為指導
- 軟體標準是由許多規則組成，這些規則應該在軟體開發過程中，以有紀律且一致的方法實施
- 軟體標準也提供一個基準，讓我們得以評估一個軟體實體或活動的體積、內容、價值及品質

109

等級3(定義清晰程序)關鍵程序區

- 等級3共有7個關鍵程序區

1. 組織程序聚焦(PF)
   - 建立組織的責任感，以提昇軟體程序的活動及能力
2. 組織程序定義(PD)
   - 發展並維護軟體程序的資產(工具、測量儀器、標準），以提昇程序的效能
3. 訓練計劃(TP)
   - 目的在於發展、提昇團隊成員所需的技術及知識

110

等級3(定義清晰程序)關鍵程序區

4. 整合式軟體管理(SM)
   - 統整軟體工程與管理活動，這些活動經過適當調整，以符合軟體程序標準
5. 軟體產品工程(PE)
   - 整合所有軟體工程活動，制定出定義明確的軟體程序，以便實際且有效率地生產出正確、一致的軟體
6. 群組協調(IC)
   - 建立不同軟體專案團隊之間的共同合作
7. 同儕複審(PR)
   - 提供軟體審查機會，儘早地在軟體開發週期中找出錯誤

111

等級3(定義清晰程序)關鍵程序區

- 軟體檢測的實施，與清單及軟體標準有關，有時必須修改這些清單與標準，以符合某個特定的軟體專案
- 清單涵括之內容為軟體開發工作的：進入與離開條件、驗證團隊、軟體品質、可靠性及成本測量
- 清單亦涵蓋軟體檢驗計畫，與必要的準備和報告程序

112

等級4(管理良善程序)關鍵程序區

- 能力成熟度模型的等級4共有兩個主要的活動：資料的蒐集與分析及軟體品質的控管
- 等級4共有2個關鍵程序區

1. 量化的程序管理(QP)
   - 根據軟體資料的蒐集及分析，量化地控制軟體程序的效能 ➔ taskmgr
2. 軟體品質管理(QM)
   - 將軟體品質與其目標作比較，以提供量化的評估結果 **MTBF** (the average time a device will function before failing) **/ MTTF** (mean time expected until the first failure)

113

等級5(最佳化程序)關鍵程序區

- 能力成熟度模型的等級5為經營軟體程序的最高層次，與最佳化程序相關的活動有：缺點預防、盡可能將軟體程序自動化、提昇軟體品質與團隊生產力的方法以及縮短開發時間

114

等級5(最佳化程序)關鍵程序區

- 等級5共有3個關鍵程序區

1. 缺點預防(DP)
   - 找出軟體缺點的起因，並設法預防缺點再發生
2. 技術變更管理(TM)
   - 找出有助於軟體開發的技術(工具、方法、程序)，並循序地將它們轉移至軟體開發流程中
3. 程序變更管理(PC)
   - 目標在於持續地改進軟體品質與生產力，同時減少軟體產品開發的週期

115

組織提供之軟體標準

- 電子電機工程師協會(IEEE) ➔ 良好軟體需求規格應具備的內容及品質(IEEE 380-1993)
- ISO/NATO制定軟體開發及組態管理的國際標準
- NATO標準4591，制定硬體方面的標準
- 國際標準組織(ISO) ➔ 設計與描述(ISO6593)、文件(ISO9127)、軟體品質管理(ISO 9000 series)
- 美國國家標準組織(ANSI)與IEEE致力發展工業的軟體開發標準

116

組織提供之軟體標準

- 美國國防部(DoD)發行軍事上的軟體標準 ➔ 嵌入式系統之生命週期(DoD 2167)
- 英國標準組織(BS)涵蓋軟體開發各個層面的標準 ➔ BS7799資訊安全
- 英國電子工程師協會(IEE)
- 物件管理組織(OMG) ➔ 軟體運作標準，允許每個應用程式能互相溝通(CORBA)
- 標準有助於軟體程序的管理，因為標準提供準則，使我們用一致的方法去審查軟體開發團隊的成效

117