1. What is software?

電腦程式及其相關文件。軟體產品可以為特定的客戶或一般大眾市場而發展。

1. Software life cycle

Requirements -> specification -> design -> implementation -> integration -> maintenance -> retirement

需求 -> 規格 -> 設計 -> 執行 -> 建置 -> 維護 -> 退休

1. Process model
2. Build and Fix Model

優點 : 適用於小系統(100 to 200 lines of code)及個人單獨發展。

缺點 : 缺乏制度跟規劃，由錯誤中學習，並且缺乏效率。

1. Waterfall model(適用於有前例可以參考)

每一個階段必須完成之後，才可以轉移到下個階段。

優點 : 建立嚴謹標準的開發程序，清楚的階段劃分，易於分工及責任歸屬。

缺點 : 共識性不足，易生爭議及反悔。人員易閒置。

1. Rapid prototyping model(適用於開發)

開始於需求的蒐集，開發者與使用者會面，定義整個軟體的目標，並進入循環，使用者給予雛型評價，開發者進行改良，直至滿足使用者的需求。

優點 : 使用者可以高度的參與，提早發現需求，並提早修改。

缺點 : 缺乏整體規劃分析及設計，故較難維護，長期而言，系統較易失敗。

1. Incremental model

開發週期可重複往返進行，每個階段的產出都是產品，所以每個階段產出都非常明顯，但完成的產品會一直因為上一階段的產出而有所變動。

優點 : 開發週期反覆進行，且使用者皆有參與，風險較waterfall model低。

缺點 : 若 build錯誤太多,將可能導致所有程序的失去控制，需有明確發展規劃，及控制build的正確性。

1. Synchronize-and-stabilize model

微軟的生命週期模型，每個區塊都由一個小組負責，所有小組是並行的。每天的結束進行測試和偵錯、每個build的結束進行穩定測試。

優點 : 每個組件總是同時進行

缺點 : 需要高成本，但是並不是所有項目都是成功的

1. Spiral model

將瀑布模型的最終結果導回源頭，成為一個往復式的圓圈，並強調風險分析，分析是否有其他方法，同時找出存在風險並加以解決。

優點 : 使用演進式的方法，在每一個演進層次中皆進行風險分析。

缺點 : 進化的過程不容易控制，並需要很多風險評估的專業知識。

1. Object-oriented life cycle models

物件導向程式由一群class組成，程式執行是一群對象交亙運作。

先定義class specification，確認系統正確性，寫代碼時可繼續沿用。

優點 : 沒有浪費精力，易發展增量，迭代開發。

1. Verification and Validation

驗證(Verification): "我們是否正確的開發了產品？" -> 軟體應該與規格相符

Do the thing right.

確認(Validation): "我們是否開發了對的產品？" -> 軟體應該執行使用者的需求Do the right thing.

1. Agile
2. 什麼是Agile? 有效地(快速且適切地)回應改變
3. Agile methods
   1. 重點代碼，不是設計，而且基於一個Iterative的軟件開發方法。
   2. 旨在快速交付可工作的軟件和演變。
   3. 敏捷方法的目的是減少的開銷，因為沒有過多的重複工作。
4. The cost between Agile process and Conventional process



1. Agile Process
2. 針對客戶的需求開發軟件
3. 強調建構的行為
4. 提出多個 software increments
5. 適應變化的發生
6. Extreme Programming (XP)
7. XP Planning

以創造user stories開始，並分配成本，制定交貨日期。

1. XP Design

對於困難的問題，提出”秒殺解決方案”，並鼓勵重新建構內部的Iterative。

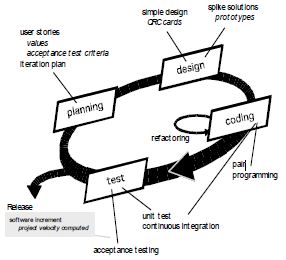
1. XP Coding

成對編程(Pair programming)

是指所有的代碼是由兩個人在一個工作站上編程一個任務生產。

1. XP Testing

每天執行所有單位的測試，最後並由客戶驗收測試。



1. 缺點 : 各項原則都有互補作用，一旦某項原則沒有確實做好，就有可能帶

來危機。

1. 優點 :
2. 省去需求文件化的時間與閱讀需求文件可能產生的錯誤
3. 快速訂出專案的範圍
4. 能在極短時間內更換新版本
5. Design Principle
6. 設計可溯源的需求模型。
7. 始終要考慮系統架構。
8. 設計數據處理功能是重要的。
9. 用戶界面設計時，應配合最終用戶的需求，並強調易用性。
10. 組件級設計上功能是獨立的。
11. 應鬆散耦合組件到另一個和外部環境。
12. 設計交涉（模型）應該很容易理解。
13. 設計會進行迭代式的發展。每次迭代中，設計者應使其更簡單。
14. Quality Management
15. McCall
    1. Product operation quality factors

* 正確性(Correctness) : 在允許的條件下滿足其規格。
* 可靠性(Reliability) : 以維修及故障的時間及頻率去測量。
* 效率性(Efficiency) : 與效能相關。
* 誠信性(Integrity) : 自我安全保護機制之設定。
* 使用性(Usability) :
  1. Product revision quality factors
* 可維護性(Maintainability) : 需要多少努力去修復一個程序錯誤。
* 可測試性(Testability) : 測試所需的努力，以確保其執行其預定功能。
* 可調整性(Flexibility) : 需要多少努力去修改
  1. Product transition quality factors
* 可移植性(Portability) : 轉換環境所需的努力
* 可重用性(Reusability) : 在其他應用上可以重複使用的機率
* 互操作性(Interoperability) : 從一個系統到另一個所需的努力

1. Review techniques
2. Walkthrough

* Participant-driven : 與會者提出了他們的名單和作者回應
* Document-driven : 作者依序遍歷文檔，參與者如果有意見則中斷

1. Inspections

* Overview : 概述文檔是由作者編寫
* Preparation : 參與者閱讀文件和填寫故障類型和頻率
* Inspection : 確保每一個項目範圍內，每一個分支是採取至少一次。
* Rework : 作者解決所有故障和問題
* Follow-up : 主持人確保所有故障和問題已經解決

1. Quality Dilemma

如果生產一套軟體具有很高的Quality那麼需要花費很長的時間，非常大的努力和巨額的金錢，但是也沒有人會想買，而且導致公司倒閉，所以業界人士試圖在中間地帶尋求平衡

1. Good Enough Software

最終用戶的願望是足夠好的軟件提供高品質的功能和特點，但同時它提供了其他更加晦澀難懂或專門的功能包含已知錯誤的功能。

1. Cost and Risk of Qualty
2. 預防成本 內部故障成本 外部失敗成本
3. 人們賭他們的工作，他們的舒適，安全，娛樂，他們的決定，和他們的生活在computer software上。
4. Garvin
5. Performance Quality (效能)
6. Feature Quality (功能)
7. Reliability (可靠性)
8. Conformance (一致性)
9. Durability (耐久性)
10. Serviceability (維修方便性)
11. Aesthetics (美學)
12. Perception (知覺)
13. The law
14. Meskimen : There's never time to do it right, but always time to do it over.

覺得時間不夠，應付過去，可是最後反而要花更多時間去彌補

1. Brook : Adding manpower to a late software project makes it later.

專案如果負擔重，加入新人反而會更慢

1. Murphy : Anything that can go wrong will go wrong

不要它發生，可是它還是會發生

1. Module and Component
2. Module :

module是可以動態加载和卸载的，也就是说它不是編譯到主程序中

的，可以做到想要的时候拿来，不想要的时候卸掉。因為module是預先编譯的，但是 module不可以獨立執行。

1. Component :

component是静態的，它的用於自定義一個組件，只要包含了定義好component的包，它在應用的时候相當於把自己嵌進Application中， 要隨著Application進行編譯。

1. Component-Level Design
2. Which Coupling is best , why?

Data coupling,

Simple arguments or data structures in which all elements are used by the called module.

1. Abstract class <-> Concrete class

Abstract class : 此class的methods只定義其signature ,詳細algorithm 並未定義,當另一class繼承此class時,再定義這些methods的詳細內容,後者為real class

1. White box test(結構測試，每個路徑都要執行一次)
2. Path testing

* 得出一個程序設計的邏輯複雜度和使用這項措施定義一個執行路徑
* 保證執行程序中的每個狀態要測試過至少一次。
* 流程圖
* Cyclomatic number : V(G) = E – N + 2

1. Loop testing
2. Simple Loops
3. 直接跳過迴圈
4. 只進入迴圈一次
5. 進入迴圈兩次
6. 進入迴圈m次 m<n
7. (n-1), n, (n+1)次進入迴圈 ，n 是通過的最大次數
8. Nested Loops

從最內層迴圈開始，設定所有外部迴圈的最低iteration value，測試（最小值＋１），正常值，（最大值－１），最大值，完成後就向外移出一個迴圈

1. Concatenated Loops

如果每個迴圈是互相獨立的，則把每個迴圈當最是一個簡單迴圈，如果不是則將它當作巢狀迴圈

1. Black box testing(功能測試，驗證功能是否與預期相符)
2. 找尋以下幾類錯誤：
3. 不正確或丟失的功能
4. 接口錯誤
5. 數據結構或外部數據的基礎上獲得的錯誤
6. 行為或性能錯誤
7. 初始化和終止錯誤
8. Path testing

* 基於圖的測試方法
* 等價分區
* 臨界值分析
* 矩陣測試

1. Different between class and object

類別(Class) : 可以說是一種型態，但沒有實體。  
物件(Object) : 是用Class實作出來的東西。

1. Compound condition(複合條件) <-> Simple condition(簡單條件)
2. The Classes condition
3. Inheritance : 物件導向的基本特徵語言
4. Aggregation : 一個Class由多個classes組合而成，內部class可以獨立存在
5. Association : 系統由一群互相作用元素執行特定的任務，以滿足用戶需求
6. Composition : 一個Class由多個classes組合而成，內部class不可獨立存在
7. 放大錯誤

10 requirements errors amplified 2:1 to become 20 errors in design

20 new design errors introduced 20 + 20 = 40 and amplified 1:1.5 to become 60 code errors

30 new errors introduced during coding 30 + 60 = 90 at the start of testing

30% of 90 errors uncovered during unit testing leaving 90 – 0.3 \* 90 = 63 errors

30% of 63 errors uncovered during integration testing leaving 63 – 0.3 \* 63 = 45 errors

50% of 45 errors uncovered during validation testing leaving 45 – 0.5 \* 45 = 23 errors

10 requirements errors with 60% caught during reviews reducing the error count to 4 which are

amplified 2:1 to become 8 errors in design

20 new design errors introduced 20 + 8 = 28 with 60% caught during reviews leaving 17 errors which

are amplified 1:1.5 to become 26 errors

30 new errors introduced during coding 30 + 26 = 56 with 60% caught during reviews leaving 34

errors at the start of testing

30% of 34 errors uncovered during unit testing leaving 34 – 0.3 \* 34 = 24 errors

30% of 24 errors uncovered during integration testing leaving 24 – 0.3 \* 24 = 17 errors

50% of 17 errors uncovered during validation testing leaving 17 – 0.5 \* 17 = 9 errors

1. Java code

public class VegTest{

static void binky(Vegetable veg) {veg.eat();}

public static void main(String [] args){

/\*

Vegetable v = new Vegetable();

binky(v);

\*/

/\*

Vegetable t = new Tuber();

binky(t);

\*/

Vegetable p = new Potato();

VegTest.binky(p);

}

1. Testing
2. Unit Testing：驗證模組正確性，測試模組資料流。
3. Integration Testing：將模組組合並驗證其正確性。
4. Validation Testing：驗證模組群是否合乎客戶需求。
5. System Testing：將軟體與其他系統組件整合，進而驗證系統是否合乎需求。