ch8

**Good software design should exhibit：堅固性、商品、喜悅**

**Design and Qualitythe design must：** 設計，品質設計必須：

1.實現所有含明確要求的分析模型

2.必須是可讀的是可以理解的指南

3.提供該軟體的完整picture

**Quality attributes品質屬性**

1. 功能性
2. 可用性
3. 可靠性
4. 效能
5. Supportability可支持性

**Design Principles 10取5**

1. 不應目光窄小
2. 應可追溯到分析model
3. 不應該推倒重來
4. 應被構造來適應變化
5. 具uniformity and integration(均勻性、集成)

**Quality Guidelines 8取5**

設計應 1. 模組化

2.包含資料、體系結構、介面和元件的不同的表示。

3.讓資料使用適當的資料結構

4. 讓元件表現出獨立的功能特性

5. 使用有效地傳達其意義的符號表示

**(設計)基本概念 \*12**：抽象、架構、pattern、分割相關性、模組性、隱藏性、功能獨立、改善、aspect(方面)、重構、OO設計、設計classes

(\*12 各別解釋請看ppt)

5 types of **design classes**

1. User interface class
2. Business class:
3. Process class:
4. Persistent class: 持久
5. System class:

**characteristics of well-formed class**

1. Complete and sufficient: 完整和充分
2. Primitiveness: 原始性
3. High cohesion:
4. And single-mindedly attributes and methods to implement responsibilities

專心於屬性和方法實作責任

1. Low coupling:

**Design Model Elements**

1. Data elements
   * Data model --> data structures
   * Data model --> database architecture
2. Architectural elements
   * 有關應用程式域的資訊
   * 具體要求模型元素
   * 體系結構模式和風格的可用性
3. Interface elements
4. Component elements



1. Deployment elements



ch9

**Why Architecture?**

使到一名軟體工程師：

1. 分析 設計的有效性，滿足其明示的要求
2. 考慮建築備案，進行設計更改時仍相對容易
3. 減少與軟體建設相關的風險。

**Why is Architecture Important?**

1. 軟體體系結構的表示是enabler
2. 突出了早期的設計決定
3. 構成了一個相對較小，智力理解的模式

**Architectural Descriptions定義**

1. 建立一個概念框架和詞彙的使用
2. 為表示架構的描述，提供詳細的guideline
3. 鼓勵健全的建築設計實踐。

**Architectural Genres**

Genres=意味著特定類別在overall software domain，每個類別中很多子類別

**Architectural Styles**

1. a set of components 元件
2. a set of connectors 連接器
3. constraints 限制
4. semantic models 語意模型

**Architectural Patterns：併發性 、持久性、分佈**

**Architectural Design**

1. 必須放入上下文
2. 確定一套建築原型
3. 通過定義和細化執行每個原型來指定系統的結構

**Analyzing Architectural Design**

1. 收集方案。
2. 激發要求、 約束和的環境描述。
3. 請描述體系結構樣式/模式，選定的解決的方案和要求：

• 模組視圖

• 流程視圖

• 資料流程程視圖

1. 評估品質屬性
2. 確定品質屬性的敏感性。
3. 批判候選架構 （在步驟 3 中開發） 採用敏感性分析進行第 5 步中。

**Architectural Complexity複雜性評估通過考慮體系結構內的元件之間依賴性**

1. 共用依賴
2. 資料流程
3. 約束的依賴

**Architectural description language (ADL)為設計者提供：**

1. 分解體系結構元件
2. 成大建築塊組成的單個元件和
3. 表示元件之間的介面 的能力

**架構分區：**

**水平**：定義為每個主要功能模組的獨立分支，使用控制模組來協調之間的通訊

**垂直**：分層決策和工作，decision making modules(決定制作模組)應該位於頂部的體系結構

**Why Partitioned Architecture?**

1. 容易測試
2. 易於維護
3. 副作用更少
4. 易於擴展

ch10

**What is a Component?：a modular, deployable, and replaceable part of a system封裝執行，並公開一組介面。**

OO觀點：一個元件包含 一個協作class的集合

傳統觀點：一個元件包含logic、data structures，實作process和一interface可以用來傳遞資料

**Basic Design Principles**

1. The Open-Closed Principle (OCP).
2. The Liskov Substitution Principle (LSP).
3. Dependency Inversion Principle (DIP).
4. The Interface Segregation Principle (ISP)
5. The Release Reuse Equivalency Principle (REP).
6. The Common Closure Principle (CCP).
7. The Common Reuse Principle (CRP).

**Design Guidelines \*3：元件；介面；相依與繼承**

**Cohesion內部的 coupling元件間的**

**Component Level Design**

1. 定義問題領域
2. 定義infrastructure domain(基礎設施領域)
3. 闡述哪些可重用的元件不可獲得
4. 描述資料來源 和 定義需要管理的class
5. 開發和擬訂元件的行為表現
6. 描述配置圖提供額外實現細節
7. 考慮替代方案

**Component Design for WebApps**

1. 明確界定提供給end user的功能，計算或資料處理
2. 凝聚內容和功能，為最終使用者提供一些所需的能力

**Functional Design for WebApps**

1. 執行當地語系化的處理生成內容，動態的方式執行導航能力， ；
2. 提供計算或資料處理的能力，是適當的 web 應用程式的business domain;
3. 提供完善的資料庫查詢和訪問
4. 建立與外部企業系統的資料介面。

**Algorithm Design**

1. 審查該元件的設計說明
2. 使用逐步求精來開發演算法
3. 使用結構化程式設計來實現程式邏輯
4. 使用 '正式方法' 來證明邏輯

**Algorithm Design Model：**

圖、虛擬碼、程式語言、決策表

**Program Design Language (PDL)：**

易結合來源碼、機器可閱讀性、可以生成圖、可聲明data、易維護

**Why Design Language?**

機器可閱讀性、結合來源碼、易維護、易review

**Domain Engineering**

調查定義域、分類、收集代表性app、分析app樣品、建立物件分析模型

**Identifying Reusable Components考慮可重用： 12取5**

1. 是在未來實現上所需的元件功能?
2. 有重複的的元件功能?
3. 該元件依賴于硬體嗎?
4. 細節可以移至另一個元件?
5. 足夠為下一個執行優化的設計嗎?

**Qualification 資格 須先考慮：7取5**

1. application programming interface API
2. 開發和集成工具所需元件
3. 嵌入式設計
4. 安全功能
5. 異常處理

**easy integration"易實現集成化"的含義是：**

1. 為所有的元件庫，實行統一的資源管理的方法 ；
2. 常見的活動，如資料管理
3. 介面體系結構內以及與外部環境已以一致的方式。

**Composition = 組成**

必須建立基礎設施，以將元件綁定在一起。 組成的建築元素包括：

1. 資料交換模型
2. 自動化
3. 結構化的存儲
4. 基礎物件模型

ch11 User Interface Design

**Interface Design：Golden rules:易學、易用、易了解**

**Interface Design：Typical Design Errors：**

1. 缺乏一致性
2. 太多的記憶
3. 沒有指導説明
4. 沒有上下文敏感性
5. 反應較差
6. 不友好

**User Interface Design：Golden rules：**

1. 使用者在控制中
2. 減少uesr記憶體負載
3. 介面一致

**使用者在控制中：**靈活的互動。不會強制使用者，變成不必要的或不受歡迎的行動

**減少uesr記憶體負載：**建立有意義的預設值。定義是直觀的快捷方式。

**介面一致：**整個系列的應用程式保持一致性。不要隨意更改，除非有令人信服的理由這樣做。

**User Interface Design Models：**使用者模型、設計模型、心智模型、實現模型。

**Interface Analysis：**

1. 人 （最終使用者）通過介面將與系統交互；
2. end-users工作必須執行的任務
3. 提出了作為介面的一部分的內容
4. 環境將進行這些任務。

**使用者分析：13取5**

1. 使用者是訓練有素的專業人員、 技術人員、 辦事員、 或製造業工人？
2. 平均使用者有什麼級別的正規教育？
3. 是使用者能夠學習?
4. 使用者社區的年齡範圍是什麼？
5. 如果使用者使用的系統錯誤，後果是什麼？

**Analysis of Display Content9取5**

1. 使用者可以自訂內容的螢幕位置嗎？
2. 是適當的螢幕識別分配給所有內容嗎？
3. 如果一個大報告提出，如何應該它進行分區為了便於理解？
4. 如何將顏色用於增進瞭解嗎？
5. 如何將錯誤訊息和警告提交給使用者呢？

**Design Issues**

1. 回應時間
2. 説明設施
3. 錯誤處理
4. 功能表和命令標籤
5. 應用程式可訪問性
6. 國際化

**WebApp Interface Design**

**Where am I?** 向user說明已被瀏覽過的web、和告知層次結構中的位置

**What can I do now?** 介面應該總是能夠説明使用者理解他當前的選項

1. 提供了哪些功能？
2. 什麼環節的生活？
3. 哪些內容是有關？

**Where have I been, where am I going?** 該介面必須便於導航。

提供地圖使用者的位置和可能採取什麼路徑移動到別處在 web 應用程式內。

**Effective WebApp Interfaces**

有效介面：

1. 可視並容錯，
2. 不涉及使用者與系統的內部運作
3. 服務提供最大的工作，需要最少user的資訊

**Interface Design Principles：14取5** 預期、溝通、一致性、控制自治、效率

**Interface Design Workflow：12取8**

1. 查看分析模型的資訊和要求
2. 布局粗略草圖
3. 定義操作相關聯的任務
4. 優化界面布局
5. 確定實現介面所需的物件
6. 開發介面行為的表示
7. 描述每個狀態的介面布局
8. 完善

**Aesthetic Design 審美設計 6取4**

1. 不要害怕空白。
2. 強調內容。
3. 組織布局元素從左上角到底部右。
4. 設計佈局時，請考慮解析度和瀏覽器視窗大小。

ch14 Quality Concepts 品質概念

對於軟體來說，可能會遇到**兩種品質**:

**設計品質Quality of design**包括要求、 規範和系統的設計。

**品質的一致性Quality of conformance**是主要側重于執行的問題。

**使用者滿意度 = 符合產品 + 優質交付符合預算和時程表**

**Quality—A Pragmatic View 一個務實的視圖**

1. 超然的view
2. 使用者view
3. 製造商view
4. 產品view
5. 基於value的view

**軟體品質可以被定義為：**

有效的軟體process的實施方式，create一種有用的產品，提供可度量的價值，對於那些產生它和那些使用它的人

**effective software process：**有效的軟體process建立基礎設施，支援任何努力在建立高品質的軟體產品。。

**useful product：**一個有用的產品提供內容、 功能和最終使用者的欲望的特點

**Adding Value：**通過添加值的生產者和使用者的軟體產品，高品質的軟體提供了軟體組織和最終使用者好處。

**Quality Dimensions** 品質維度

1. 性能品質。
2. 功能品質。
3. 可靠性。
4. 一致性。
5. 耐久性。
6. 可維護性。
7. 美學。
8. 認知。

**McCall’s Quality factors 麥考爾的品質因素**

**Product operation view**

1. 正確性： 何種程度的程式滿足它規範和滿足客戶的使命目標
2. 可靠性： 多大程度上，一個程式可以預計執行預期的功能和精度要求。
3. 效率： 金額的計算資源和代碼執行其功能所必需的程式。
4. 完整性： 哪些訪問軟體或資料的未經授權的人多大程度上可以控制
5. 可用性： 努力需要學習、 操作、 準備輸入，並解釋程式的輸出

**Product revision view**

1. 可維護性： 查找並修復錯誤，在程式中所需的工作量。
2. 靈活性： 修改業務程式所需的工作量。
3. 可測試性：： 測試一個程式所需的努力，確保執行預期的功能

**Product transition view**

1. 可攜性：從一個硬體和/或軟體系統內容到另一個傳輸程式的所需
2. 再使用性：多大程度上屬於一種程式或部分作業系統程式可以重用在另一個應用
3. 互通性：溝通到另一個系統所需的

**Quality factors**

1. 功能： 哪種軟體的程度滿足明確的需要： 適用性、 準確性、 互通性、 法規遵從性、 安全
2. 可靠性： 的軟體是可供使用的時間量。成熟、 容錯能力、 可恢復性：
3. 可用性： 哪種軟體的程度是很容易使用。可理解性，易學性、 可操作性
4. 可維護性： 可以對軟體進行修復的易用性。習語可分析性、 可變性穩定性
5. 可攜性： 哪種軟體可以轉移到另一個環境的易用性。適應性，替換。

**Targeted quality factors**

1. 直覺：學位的介面應該遵循預期的模式這樣一個新手可以使用它沒有重大意義的培訓
2. 效率：操作和資訊可以是位於的公關發起的程度
3. 穩健性：軟體錯誤的輸入的資料或不恰當的使用者交互的處理程度
4. 豐富：該介面提供豐富的功能集的程度

**The Software Quality Dilemma困境**：所以業內的人士試著去那神奇的中間地帶的產品在哪裡不夠好，不是馬上拒絕，如期間評價，但也不是那麼多的完美主義和這麼多的工作物件，它會花很長時間或成本太高來完成。

**Cost of Quality**

預防成本包括

1. 品質先期策劃
2. 正式的技術審查
3. 測試設備
4. 培訓

內部缺陷成本包括

1. 重做
2. 修復
3. 失效模式分析

外部缺陷成本是

1. 解決客戶投訴
2. 退貨和更換
3. 説明線支援
4. 保修工作

**Quality and Security**

必須考慮安全性、 可靠性、 可用性和可靠性，人們太多精力關注的 bug 和不夠的缺陷。

**Achieving Software Quality**實現軟體品質

成功的關鍵因素：

1. 軟體工程的方法
2. 專案管理技術
3. 品質控制
4. 品質保證

ch15 Review Techniques 審查技術

**What Are Reviews?**

1. 由技術人員對技術人員進行的一次會議
2. 在軟體工程過程中創建的工作產品的技術評估
3. 軟體品質保證機制
4. 訓練場

**What Reviews Are Not**

1. 專案摘要或進展評估
2. 會議目的只是為了傳遞資訊
3. 一種機制為政治或個人報復的 ！

**What Do We Look For? 錯誤和缺陷**

1. 錯誤 — —在軟體發佈給最終使用者之前發現的品質問題
2. 缺陷 — — 已向最終使用者發佈了軟體後才發現的品質問題

**Defect Amplification**

缺陷放大模型可以用於說明在軟體過程設計和代碼生成操作期間的生成和檢測的錯誤。

**Metrics 度量標準 (計算題的樣子)**



**Informal Reviews分正式reviews**

1. 和一位同事簡單的桌面檢查，軟體工程的工作產品
2. pair programming的review-oriented方面

🡺結對程式設計鼓勵不斷審查作為創建一個工作產品 （設計或代碼）。

優點是立即發現的錯誤，並且更好的結果是工作產品的品質。

**Formal Technical Reviews正式的審查技術**

目標：

1. 發現功能、羅及or任何執行的錯誤
2. 驗證正在審查的軟體，達到它的要求
3. 確保已根據代表了軟體預定義的標準
4. 實現以統一的方式開發軟體
5. 要使專案更易於管理

**The Review Meeting審查會議**

1. 三至五人 （通常） 應參與審查。
2. 提前準備應該發生，但應該不會要求每一個人超過兩個小時的工作。
3. 審查會議的時間應不少於兩個小時。
4. 焦點位於一個工作產品

**參與者：**生產者，審查領導者，檢閱者(reviewer)，錄音機

**Conducting the Review 進行review 10取5**

1. 審查產品，不是製作者。
2. 設置議程並進行維護。
3. 書面筆記。
4. 限制參與者的數目，並堅持要求提前準備。
5. 審查您最初的評論。

ch16 Software Quality Assurance軟體品質保證

**Comment on Quality品質上的評論**

1. 每個人都是為它。（在一定條件下，當然。）
2. 每個人都覺得他們理解它。（即使他們不想解釋它。）
3. 每個人都認為執行只是自然傾向。（畢竟，我們做相處得不知何故。）

**Elements of SQA**

1. 標準
2. 檢查和審核部門
3. 測試
4. 錯誤/缺陷收集和分析
5. 變更管理
6. 教育
7. 供應商管理
8. 安全管理
9. 安全
10. 風險管理

**SQA Goals**

1.要求品質。

2.設計品質。

3.代碼品質。

4.品質控制的有效性。

**Statistical SQA 統計品質保證**

1. 有關軟體錯誤和缺陷資訊收集和分類。
2. 試圖追蹤每個錯誤和缺陷對其背後的原因
3. 使用Pareto principle原則
4. 一旦已經確定的重要的幾個原因，動議修正已造成的錯誤和缺陷的問題。

**Six-Sigma for Software Engineering**

六西格瑪方法定義了核心步驟：

1. 定義客戶需求和可交付成果和專案的目標
2. 衡量現有的過程和它的輸出
3. 分析缺陷的度量標準
4. 改進流程
5. 控制process

**軟體可靠性 (計算的樣子)**

MTBF平均故障之間的時間=MTTF平均故障時間+MTTR平均修復時間

Availability = [MTTF/(MTTF + MTTR)] x 100%

**Software Safety軟體安全**

1. 軟體安全是側重于確定和評估潛在的危險，可能會產生負面影響的軟體，還會造成整個系統發生故障的軟體品質保證活動。
2. 如果危害可以在軟體過程的早期識別，可以指定軟體設計的特點，這將消除或控制潛在的危害。

**ISO 9001: 2000 是適用于軟體工程的品質保證標準。**

**主題劃定的要求：**

1. 管理責任
2. 品質體系
3. 合同評審
4. 設計控制
5. 文檔和資料控制
6. 產品標識和可追溯性
7. 程序控制
8. 核對總和測試
9. 糾正和預防措施
10. 控制的品質記錄
11. 內部品質審核
12. 培訓
13. 服務
14. 和統計技術

ch17 Software Testing Strategies 軟體測試策略

**Software Testing**

測試是行使具有特定意圖，包括發現在交付給最終使用者之前的錯誤的程式的過程。

**Testing Shows：**

1. Errors 錯誤
2. requirements conformance 要求一致性
3. performance 效能
4. indication of quality 品質的指示

**Strategic Approach策略方針**

1. 應進行有效的技術審評
2. 不同的測試技術是適合於不同的軟體工程方法和不同時間點時間。
3. 通過軟體的開發者進行了測試和 獨立的測試組。
4. 測試和調試是不同的活動，但調試必須安置在任何測試的策略。

**兩種V**

**Verification**核查是指的任務，以確保軟體正確地實現特定的功能集。

**Validation**驗證是指一組不同的任務，以確保軟體，建立了可追溯到客戶的要求。

核查："我們正確地構建產品嗎?"

驗證："我們正在正確的產品嗎?"

**Who Tests the Software?**

Developer：瞭解系統、但是，將"輕輕"測試、driven by "delivery"

independent tester(獨立測試儀)：必須瞭解系統，但是，將嘗試打破它，以品質驅動

**Testing Strategy測試策略圖**

測試由小到大

**Integration Testing Strategies\*3**

  

1. Top Down Integration：更換stubs，深度第一，當新的模組被整合，一些測試的子集是重新運行
2. Bottom-Up Integration：更換drivers，深度第一，工作模組分組到構建和集成
3. Sandwich Testing：頂級模組進行測試with stubs，工作模組分組到構建和集成

**Testing the CRC Model**

1. 重新拜訪CRC Model和物件關係模型。
2. 檢查每個 CRC 索引卡，確定委派的責任是否合作者定義的描述部分。
3. 反相連接，確保每個合作者被要求服務接收請求是從一個合理的來源。
4. 使用反向的連接檢查在步驟 3 中，確定是否其他類可能需要或是否責任適當地分為幾個class之間。
5. 決定是否廣泛要求的責任，可能組合成一個單獨的責任。
6. 步驟 1 至 5 反復，每個class透過每次分析模型的演化。

**High Order Testing高階測試**

1. 驗證測試：重點是軟體要求
2. 系統測試：重點是系統集成
3. Alpha/Beta 測試：重點是客戶的使用方式
4. 恢復測試：部隊軟體無法在以各種方式，並驗證正確執行恢復
5. 安全測試：驗證系統中內置的保護機制事實上，將保護它免受不當滲透
6. 壓力測試：執行系統要求中異常的數量、 頻率或卷資源的方式
7. 效能測試：測試軟體的集成的系統範圍內的運行時性能

**OO Testing Strategy**

1. Class test相當於unit test
2. 集成應用三種不同的策略
   1. thread-based testing — — 集成的回應一個輸入或事件所需的類集。
   2. use-based testing — — 集成的回應一個用例所需的類集。
   3. cluster testing — — 集成的演示一個協作所需的類集。

**WebApp Testing 10取5**

1. 審查 web app的content model內容模型，發現錯誤。
2. 審查 web app的interface model介面模型，確保可以容納所有的用例。
3. 審查 web app的design model設計模型，以揭露導航誤差
4. 每個功能的元件進行了單元測試。
5. 進行了效能測試。

**Debugging Techniques**

1. brute force / testing 暴力法: 利用memory dump，測試每一個步驟、條件的正確性。
2. Backtracking 依錯誤的發生，往回追蹤，找出錯誤的地方。
3. Induction 歸納法: 假設可能發生錯誤的原因，再用data去驗證。
4. Deduction 演譯法: 列出所有可能的原因，依不同測試結果逐一刪除不可能的原因，留下正確的。

**Correcting the Error 糾正error**

1. Bug 的根源轉載于該程式的另一部分嗎？
2. 下一個可能會引入由正要使此修復程式 bug是什麼 ？
3. 我們做了什麼來防止這種錯誤的可能？

**Final Thoughts**

1. Think
2. 使用工具
3. 從別人那裡得到幫助
4. 一旦更改，使用迴歸測試來找出任何副作用

ch18 Testing Conventional Applications測試傳統app

**Testability可測試性**

1. 可操作性 — — 它乾淨地運作
2. 可觀測性 — — 每個測試案例的結果很容易觀察到
3. 可控性 — — 度的測試可以自動化並優化
4. 分解 — — 測試可以有針對性地
5. 簡單 — — 減少複雜的體系結構和邏輯，以簡化測試
6. 穩定性 — — 在測試過程中的幾個變更
7. 可理解性 — — 的設計

**Good Test：**

1. 一個好的測試已找到出錯的可能性很大
2. 一個好的測試不是多餘的。
3. 一個好的測試應該是"同類最佳"
4. 一個好的測試應該是既不是太簡單了，也不是太複雜了

**white-box testing** 是一種使用描述作為元件層級設計一部分的控制層級，以得到功能完善的設計哲學。

1. 一個可執行所有軟體功能的測試
2. 在True or false上兩邊執行
3. 在迴圈邊界上和元件界限上測試

**black-box testing** 焦距在軟體功能需求上，可找出錯誤

1. 非正確或遺漏的錯誤
2. 介面錯誤
3. 資料結構或程式外部資料庫存取錯誤
4. 功能錯誤
5. 初始和結束錯誤

ch19 Testing Object-Oriented Applications

**若要充分測試 OO 系統，必須做三件事：**

1. 必須擴大測試的定義
2. 單元測試和集成測試的策略必須改變顯著
3. 測試案例的設計必須考慮物件導向軟體的獨特特點。

**測試OO模組**

1. OO analysis and design models的review明顯有效，因為相同語義構造現在分析、 設計和編碼level。
2. 因此，在class屬性的定義中，是在分析過程中發現的問題將會繞過，如果問題沒有發現，直到設計或code （或甚至是分析下一次反覆運算） 可能會出現的副作用。

**OO Testing Strategies**

* 1. Unit testing
  2. Integration Testing
  3. Validation Testing

**Testing Methods**

* 1. 基於故障的測試
  2. 測試class和class層次結構

繼承並不能排除徹底測試的所有派生類的需要。事實上，它實際上可以使測試過程的複雜化。

* 1. 基於方案(Scenario)的測試設計

**OOT 方法\*4**

1. Random Testing
2. Partition Testing
3. Inter-Class Testing
4. Behavior Testing