7 空姐 8 婷雅never 9 阿柔 10 女神(威靈) 11 溫七 12 (我是 金 城 武!!!!!!!!!) 13 真美 14 陸陸 15 宗諺 16 帥哥 17 美麗姊姊 18 威伶 19 振皓 **樓上高調**

**CH7**

**Requirements Modeling: Flow, Behavior, Patterns, and WebApps**

**需求模型 ： 流量，行為，模式和 網路應用**

**有兩種方法來分析模型 :**

structured analysis(結構化分析),

object-oriented analysis(物件導向分析)

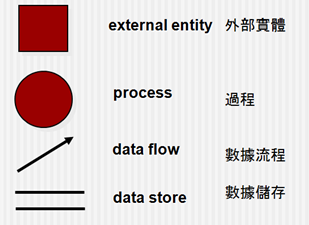
**Flow-Oriented Modeling (流程導向模型 )**

代表數據的物件通過該系統是如何被轉換的

圖解形式使用 data flow diagram (DFD)

**The Flow Model**





生產者或消費者data

例子：一個人，一個裝置

Data must always originate somewhere

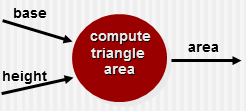
and must always be sent to something

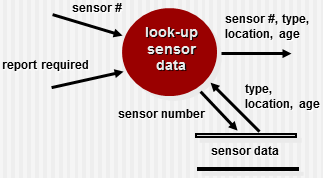
數據轉換（轉換輸入到輸出）

例如：計算稅款，確定面積，格式報告，顯示圖

數據必須以某種方式來達成系統的功能處理

數據流經一個系統中，開始作為輸入，並轉換成輸出

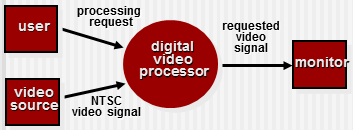


數據通常被存儲供以後使用。

**構造DFD（數據流圖）**

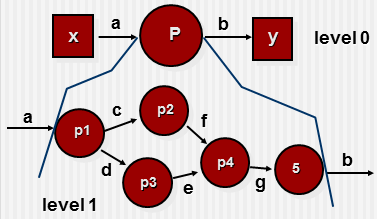
審核用戶場景或data mosel來隔離data object和使用語法解析來確定“操作”

確定外部實體

創建一個0級DFD

寫記敘文，描述變換

分析來確定一個新的水平轉換

“平衡”的流程，以保持data flow的連續性

開發一個1級DFD

使用1：5膨脹比

**Process Specification (PSPEC) 過程規範 (PSPEC)**

1. narrative 敘述

2. pseudocode (PDL) 偽代碼

3. equations 方程式

4. tables 表格

5. diagrams and/or charts 圖或圖表

**Control Specification (CSPEC) 控制規範（CSPEC）**

狀態圖

狀態轉移表

決策表

活性表 左 統稱組合規格(combinatorial spec)

**behavioral modeling表徵 (行為模型)**

每類作為系統的狀態執行其功能和

該系統從外部作為系統中觀察到的狀態執行其功能

**The States of a System**

1.state : a set of 觀察 circum-stances

2.state transition : 從一個狀態轉移到另一個

3.event : 系統表現行為

4.action : process that occurs 結果 of making a transition

**Requirements Modeling for WebApps**

內容分析

交互分析

功能分析

配置分析

**When Do We Perform Analysis?**

Web應用程序建造龐大或複雜

利益相關者數量大

網絡工程師和其他捐助者數量很大

目標Web應用程序將影響企業的底線

關係到企業的成敗

**The Content Model (內容模型)**

Content objects : 內容從使用者範例中摘錄

Attributes : 確定每一個內容的屬性

relationship : 網路應用程序維護中的內容和內容的層次結構的關係

**The Interaction Model (交互模型)**

use-cases

時序圖

狀態圖

用戶界面原型

**The Functional Model (功能模型)**

使用者發現的功能，將運用網路應用程式被交付給最終使用者

操作包含分析類別

活動圖可用於表示處理流程

**The Configuration Model (配置模式)**

服務器端 :

必須指定服務器硬體和操作系統環境

必須考慮在服務器端的相互操作性的考慮

必須指定相應的接口，通信協議和相關協作信息

客戶端 :

必須確定瀏覽器配置問題、定義測試要求

**Navigation Modeling (導航模型)**

沒有重點...

-----------------------------------------------------ch7 end

CH8

1.Dr. Dobbs Journal

Good software design should exhibit:

Firmness : 不能有任何錯誤

Commodity : 符合大多數目的

Delight :使用愉快

2.Analysis model>>Design model

class-based elements

scenario-based elements

flow-oriented elements

behavioral elements

================================

component-level Design :組件

interface Design :介面

architectural Design :架構

data/calss Design :資料/類

3.Quality attributes:

Functionality :功能

Usability :可用性

Reliability :可靠性

Performance :性能

Supportability :可支持

4.Why Information Hiding?

減少"side effects"的可能性

限制了當地的設計決策對全球的影響

通過控制介面強調溝通

鼓勵使用global data

產生更高品質的軟件

5.Functional Independence

Cohesion 是一個模塊的"功能相對強度"的指標。

Coupling 模塊之間的"相對相互依賴"的指標。

6.Aspects :方面

一個cross-cutting concern 的表示

7.Refactoring : 重構

改變一個軟件系統

以不改變code外部行為的過程方式，提高了它的內部結構

8.OO Design Concepts

Design classes:設計class:Entity classes 實體類別、Boundary classes 邊界類別、Controller classes 控制類別

Inheritance:繼承

Messages:訊息

Polymorphism:多態性

9.5 types of design classes

User interface class:使用者介面

Business class:業務

Process class:流程

Persistent class:持久化

System class:系統

10.4 characteristics of well-formed class

Complete and sufficient :完整和充分的

Primitiveness :原始性

High cohesion :高凝聚

Low coupling :低耦合

11.Design Model Elements

Data elements:資料

Architectural elements:架構

Interface elements:介面

Component elements:組件

Deployment elements:配置

12.Architectural Elements

有關應用程式域的信息

具體要求模型元素

建築圖案和款式的可用性

**CH09 Architectural Design**

**Why Architecture?**

**（1）分析設計有效性 : 滿足既定要求**

**（2）考慮架構替代 : 在一個階段，做出設計變更是容易的**

**（3）降低風險 : 與軟件的結構有關**

**Why is Architecture Important?**

**1.軟件體系結構的表示是一個推動者**

**各利益相關者感興趣的是基於計算機系統發展之間的通信**

**2.該架構凸顯早期設計**

**該系統最終成功作為一個經營實體**

**3.建築“構成比較小，思想上抓握模式**

**如何系統的結構，各組件協同工作“**

**Architectural Styles**

1. **數據中心架構**
2. **數據流程架構**
3. **Call and return架構**
4. **物件導向的架構**
5. **分層體系結構**

**Architectural Patterns**

**並發性：應用程序必須處理多個任務的模擬並行**

**持久性：數據仍然存在**

**分配：其中系統或部件與另一個在分佈式環境中進行通信的方式**

**Architectural Complexity**

**共享依賴性 : 生產者與消費者之間的依賴關係**

**流程依賴性 : 代表生產者和消費者資源之間的依賴關係**

**約束依賴性 : 相對流量限制**

**An Architectural Design Method**

**1.衍生程序結構**

**2.分割架構: (1)水平分割**

**(2)垂直分割:Factoring**

**Why Partitioned Architecture?**

**1.軟件容易測試**

**2.軟件易於維護**

**3.副作用小的傳播**

**4.軟件更易於延伸**

**Structured Design**

**目標：推導得出的分區程序架構**

**做法：一個DFD被映射到一個程序架構**

**表示法：結構圖**

**Ch10**

**What is a Component?(什麼是組件)**

**OO觀點：組件包含了一組合作class**

**傳統觀點：一個組件包含處理邏輯，來實現處理邏輯所需的內部數據結構，和一個界面。**

**Basic Design Principles(基本設計原則)**

**開放封閉原則（OCP） : 模塊成分應該對擴展開放，但是對修改關閉。**

**Liskov 替換原則（LSP） : 子類應該替代基類。**

**依賴逆原則（DIP） : 依靠抽象。**

**界面分離原則（ISP） : 特定的界面比通用界面要好。**

**發行再利用等價原則（REP） : 重複使用釋放。**

**共同封閉原則（CCP） : class改變屬於彼此。**

**公共再利用原則（CRP） : class不重新使用不應該放在一起。**

**Design Guidelines(設計指南)**

**組件**

**界面**

**依賴關係和繼承**

**Cohesion(凝聚力)**

**傳統觀點：從一而終的一個模塊**

**OO觀點：凝聚力意味著一個元件與組件本身的屬性和操作是密切相關的**

**Levels of cohesion(凝聚力層級)**

**功能**

**層**

**通訊**

**順序**

**過程**

**時間**

**效用**

**Coupling(耦合)**

**傳統觀點：其中一個組件連接到其他組件的程度**

**OO觀點：測量class彼此連接的程度**

**Level of coupling(耦合層級)**

**內容**

**通用**

**控制**

**郵票**

**數據**

**子程序調用**

**使用類型**

**列入或輸入**

**外部**

**Algorithm Design(演算法設計)**

**審查設計組件說明書**

**使用逐步細化開發的算法**

**使用結構化編程來實現過程邏輯**

**使用“形式化方法”來證明邏輯**

**Algorithm Design Model :可以對質量進行審查**

**圖形化**

**偽代碼**

**程式語言**

**決策表**

**Why Design Language?**

**可以選擇的HOL的衍生物**

**機器的可讀和可處理**

**可以嵌入來源碼，因此更容易保持**

**可以詳細地表示，如果設計者和編碼器是不同的**

**輕鬆查看**

**Domain Engineering(工程結構)**

**1.定義域進行調查。**

**2.歸類從域提出的項目。**

**3.收集域中一個代表性的樣品。**

**4.分析樣品中的每個應用程序。**

**5.制定一個分析模型中的對象。**

**CBSE Activities(活動)**

**組件資格**

**組件適應**

**組件組成**

**組件更新**

**Composition (成分): 基礎結構必須建立結合組件一起**

**架構成分組件包括：**

**數據交換模型**

**自動化**

**結構化儲存**

**基礎對象模型**

**Classification(分類)**

**列舉分類：class的成分和不同的軟件組件的子類的水平分層結構描述**

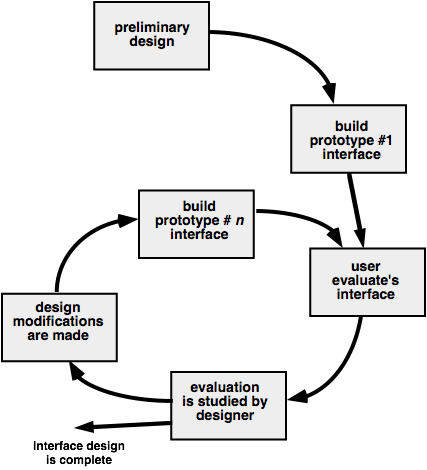
**面分類：區域分析，一組基本的描述性特徵**

**屬性值分類：在結構域定義的所有組件屬性**

**CH11.用戶界面設計(UserInterface Design)**

* **Golden Rules**
  + **控制用戶**
  + **減少用戶的記憶體負載**
  + **使界面維持一致**
* **User Interface Design Models 用戶界面設計模型**
  + **用戶模式：系統所有最終用戶的個人資料**
  + **設計模式：用戶模型設計的實現**
  + **心智模式(系統感知)：用戶的心理意象中的界面**
  + **實現模型：該界面中的“外觀和感覺”再加上描述界面語法的輔助訊息**
* **Interface Analysis 分析界面要知道..**
  + **最終用戶會透過界面與系統互動**
  + **最終用戶必須做好自己的工作**
  + **內容會被當成界面的一部份呈現**
  + **任務進行的環境**
* **Interface Design Steps 界面設計步驟**
  + **界面對象和操作的定義 : 在界面的分析中利用信息開發**
  + **定義事件：會導致改變用戶界面狀態，模擬這種行為**
  + **描繪每個界面的狀態 : 就像最終用戶實際上會看到的**
  + **指示用戶：如何從界面提供的資訊解釋系統的狀態**
* **Interface Design Principles 界面設計原則**
  + **預測：WebApp應被設計能預期下一步行動。**
  + **通信：界面要能與任何由用戶發起的活動溝通**
  + **一致性：導覽控制、菜單和圖標的顏色，形狀，佈局….**
  + **自主控制：界面應要用已建立好的導航約定來促進用戶在整個WebApp上的活動。**
  + **效率：Web應用程序及其界面的設計應該要優化了用戶的工作效率**
  + **重點：Web的界面和其內容應該專注於用戶手頭上的任務**
  + **FITT定律：“捕獲目標的時間是目標距離和大小的函數。”**
  + **人性化界面對象：可重複使用龐大的圖書館已被開發給WebApp。**
  + **減少延遲：應該使用多工的方式。**
  + **可學習性：WebAPP界面的設計應最小化學習所需時間，在重新學習時，也要最小化其時間。**
  + **產品的完整性：須自動保存，發生錯誤，也不會丟失。**
  + **可讀性：經過界面的所有信息應讓年輕人和老年人都可讀。**
  + **追蹤狀態：在適當的時候，用戶的互動應追蹤和儲存，使用戶在登出後仍可以回來拿取。**
  + **可見的瀏覽：讓用戶有工作帶給他們在同一個地方的錯覺。**

**Design Evaluation Cycle 設計評估週期**



**---ch11 end.**

CH12.基層模型設計(Pattern-Based Design)

**\* 設計模型 (pattern design)**

為了描述問題而編篡的方法，而設計模型解決方式能使軟體工程區間抓取設計知識並使它能夠再次使用

\* Generative Patterns

生成模式描述了一個重要可重複的系統方面，然後提供我們一個方式來建立特有的給定上下文。

**\* 模式種類**

Architectural patterns(架構模式)

-描述基礎廣泛設計的問題是用結構的方式來解決

Data patterns (數據模式)

-描述資料經常性循環的問題，而資料模型可以解決此問題

Component patterns (組件模式)

-組件模式描述地址的問題與子系統和部件發展,相互通信的方式

Interface design patterns (介面設計模式)

-界面設計模式描述了通用的用戶界面問題

WebApp patterns(Web 應用程式模式)

-Web應用程序模式解決建築的WebApp時所遇到的問題

Creational patterns(創建模式)

-創建模式焦點在“創造，組合，和對象的代表性”

Structural patterns(結構模式)

-結構模式注重如何將class和對象的組織整合，構建一個更大的

結構

Behavioral patterns(行為模式)

-行為模式解決與物體和方式之間的責任

**\*** Describing a Pattern

1.名稱

2.問題(當模式發表時)

3.動機

4.內容

5.force(力量)

6.解決方法

7.目標

8.相互合作(pattern 之間)

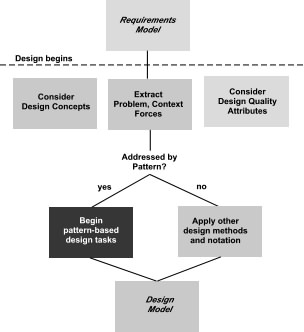
9.結果

10.履行

11.用途( 應給予舉例)

12.相關模式(在設計模式間互相的參照)

\* Pattern-Based Design

 (有需要翻譯請告知我)

**\* WebApp Patterns**

信息架構模式

導航模式

互動模式

呈現模式

功能模式

**\* Design Granularity**

Architectural patterns　　 架構

Design patterns　　　 　設計

Component patterns　　 組件

Chapter 13

1.When should we emphasize WebApp design?

· 當內容和功能是複雜的

· 當Web應用程序的大小包括非常多的內容對象，功能，和分析class

· 當Web應用程序的成功將會對企業的成功有直接的影響

2.Design & WebApp Quality(品質)

**安全**

拒絕外部攻擊

排除未經授權的訪問

確保用戶/客戶的隱私

**可用性**

**可擴展性**

**上市時間**

3.Quality Dimensions for End-Users

· 時間

· 結構

· 內容

· 準確性和一致性

· 反應時間和等待時間

· 性能

4.WebApp Design Goals

· 一致性

· 識別性

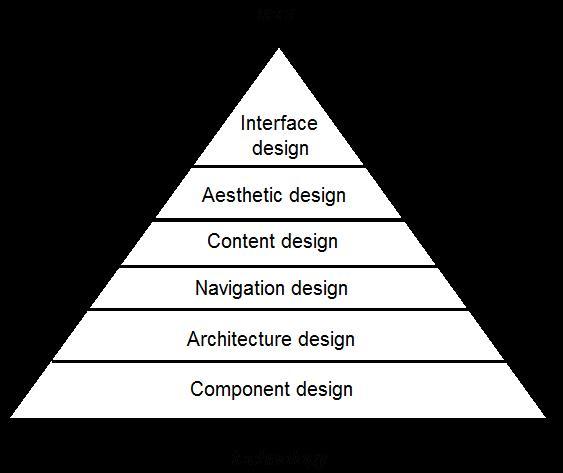
· 穩健性

· 導航性

· 視覺吸引力

· 相容性

5.WebE Design Pyramid



6.Interface Design Principles

· 預測

· 通訊

· 一致性

· 自主控制

· 效率

· 焦點

· Fitt’s定律

· 人性化的物件

· 減少延遲

· 可學習性

· 維護工作產品的完整性

· 可讀性

· 追踪狀態

· 可見導航

7.Navigation Syntax導航語法

· 個人導航鏈接

· 橫向導航欄

· 垂直導航欄

· 標籤

· 網站地圖

Ch14

在設計軟體時有兩個品質問題是有可能會遇到的:

(For software, two kinds of quality may be encountered)

1.Quality of design encompasses requirements, specifications, and the design of the system.

2.Quality of conformance is an issue focused primarily on implementation.

軟體品質的定義為以下:

(Software quality can be defined as)

An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.

一個有效去生產軟體的過程，創造了有用的軟體，並帶給生產者或消費者許多的價值。

“Good Enough” Software

解釋:

Good enough software delivers high quality functions and features that end-users desire, but at the same time it delivers other more obscure or specialized functions and features that contain known bugs

好的軟體帶來高品質和多項特性是使用者所渴望的，但在提供特定功能時同時也帶來更多的Bug。

軟體安全跟整個品質息息相關

(Software security relates entirely and completely to quality. )

1.security

2.reliability

3.availability

4.dependability

達成軟體品質的幾個因素

(Achieving Software Quality)

1.Software Engineering Methods

2.Project Management Techniques

3.Quality Control

4.Quality Assurance

CH15 Review 審查

pair programming:

兩個程式設計師在同一個電腦上共同工作，一個寫程式 另一個幫忙檢查，因為一個人寫程式時常會卡住，兩個人則可以看到對方遺漏的地方

What Are Reviews:

a meeting conducted by technical people for technical people

透過技術人員對技術人員進行會議

a technical assessment of a work product created during the software engineering process

評估產品在軟體工程中的製造過程

a software quality assurance mechanism

保證軟體品質的機制

Defect Amplification:

A *defect amplification model* can be used to illustrate the generation and detection of errors during the design and code generation actions of a software process.

缺陷放大模型可以用來說明當在設計軟體和產生code時，所產生和發現的錯誤。

Formal Technical Reviews(FTR)

includes *walkthroughs* and *inspections*

目標:

to uncover errors in function, logic, or implementation for any representation of the software

發現軟體錯誤的執行、邏輯、功能

to verify that the software meets its requirements

確認軟體符合要求

to ensure that the software represented according to predefined standards

保證軟體根據預定的標準呈現

to achieve software that is developed in a uniform manner

實現軟體以統一方式開發

to make projects more manageable

計畫容易管理

What review look for --> Errors and defects

*Error*—a quality problem found *before* the software is released to end users

Error:在軟體發布給最終用戶前發現品質有問題

*Defect—*a quality problem found only *after* the software has been released to end-users

Defect:在軟體發布給最終用戶後發現品質有問題



軟體品質保證(SQA)

軟體品質保證的元素:

* 標準
* 檢查和審計
* 測試
* 錯誤/缺陷的收集和分析
* 變更管理
* 教育
* 供應管理
* 防護管理
* 安全
* 風險管理

SQA group I:

* 定義SQA的策略
* 準備一個SQA的計畫
* 參與軟體process描述的開發

SQA Group-II:

* 評測軟件工程活動，以驗證是否符合該定義的軟件過程。
* 指定的軟件工作產品的審核，以驗證是否符合這些定義為軟件過程的一部分。
* 確保軟件工作及工作產品的偏差根據文件化的程序被記錄並處理。
* 記錄任何違規並回報高級管理人員。

SQA 的目標:

* 需求品質
* 設計品質
* 程式碼品質
* 品質控制的有效性

Software Safety:

* 是一個SQA活動，著重於鑑定和評估對軟體負面影響和使系統故障的潛在風險。
* 如果風險可以在軟體process中就很早的識別到，軟體設計就可以指定要消除或控制潛在危險。

ISO 9001:2000 :是適用於軟體工程的品質保證標準。

**CH17軟體測試策略 Software Testing Strategies**

**Software Testing**測試運用在送交給使用者之前程式的各個項目的錯誤處理

**What Testing Shows** Error> requirements conformance>Performance>an indication of quality

**V&V** verification(驗證) and validation(確認) 是指檢查軟體是否符合規格及其預期目的的程式。

**驗證（verification）：**是在特定開發階段中，評估軟體是否符合階段開始前所定義條件的程式。驗證程式確保「以正確的方式製造產品」（you built it right）

**確認（validation）:** 是在開發階段後，評估軟體是否符合規格需求的程式。確認程式確保「製造出正確的產品」（you built the right thing）

**Testing Strategy**

單元測試Unit test(代碼生成)

對象主要是副程式、模組或物件方法，其重點在於將該程式當作一個獨立個體加以測試，因此往往需要驅動程式（driver program）的配合，藉著呼叫該程式來執行特定的任務。

軟體工程師

測試模組

測試的項目

**results**

**local data structures**

**boundary conditions**

**independent paths**

**error handling paths**

結果

整合測試Integration test(設計建模 )

是驗證單元之間的介面是否正確，彼此間的訊息傳遞是否無誤。

整合方法:

大霹靂big bang(一次性整合)

漸進式

Top-down（由上而下）對問題先有一個整體的概念，然後再逐步加上設計細節，最後讓整體的輪廓越來越清楚。

Bottom-up（由下而上）先將解決問題可能所需的基本元件、方案給準備好，然後再將這些基本元件組合起來，由小而大最後得到整體。

混和測試法(Sandwich Testing) 結合以上兩種測試策略，從頂層與底層模組向中間整合。

確認測試Validation test(分析建模)

指確認一系統是否符合設計規格或契約之需求內容的測試。

系統測試System test(系統工程)

對象是完整的系統，此時的目的是要確認系統是否達到原始的設計目標。

回歸測試Regression Testing

檢驗軟體原有功能在修改後是否保持完整。

煙霧式測試Smoke Testing

用來識別和修正軟體缺失最符合經濟效益的方法，它是設計用來確認程式碼函式中的變更是否與預期相同，而且不會使整個組建不穩定。

OO testing

首先通過評估的分析和設計模型的正確性和一致性測試改變的策略，以單元的概念擴大，封裝整合class across ‘thread’，使用黑箱測試(測試應用程式的功能，而不是其內部結構或運作)。此這是方法參考了傳統的功能，還包括了特殊的功能

WebApp Testing

對內容、介面、結構進行審查，瀏覽測試，對每個功能的組件進行單元測試，在不同的環境兼容性測試，有無漏洞，效能測試。終端用戶控制、監測人口測試。可用性問題、兼容性問題以及可靠性、性能。

High Order Testing

Validation testing確認測試: 注重軟體的需求功能

System testing 系統測試: 注重系統整合

Alpha/Beta testing α/β測試: 注重客戶使用

Recovery testing 恢復測試: 迫使軟體以各種方式出現故障，並驗證以正確恢復執行

Security testing 安全測試: 保護系統不受到不適當的滲透

Stress testing 壓力測試: 執行系統要求資源在異常數量、頻率、容量都能執行

Performance testing 效能測試: 測試一個整合的系統的軟體效能運行時間

OOTESTING整合應用三種不同的策略

thread-based testing—整合了set of classes 以輸入或事件作回應

use-based testing—整合了set of classes 以使用案例作回應

cluster testing—整合了set of classes 以demonstrate one collaboration作回應

**Bug Categories:** function-related bugs, system-related bugs, data bugs, coding bugs, design bugs, documentation bugs, standards violations, etc.

## CH18 (陳威伶)

=========================請先讀我====================================

原投影片的p24~p27我沒有做整理, 因為那邊我看不懂@@

關於本章重點在於:

何謂test , 好的test , 各種test

ing & SExhaustive Testelective Testing(全部和選擇兩種相對的概念)

White-Box Testing & Black-Box Testing(好奇寶寶和只在乎功能兩種相對的概念)

White-Box Testing包括了:

n Basis Path Testing (自己畫圖看有幾種路徑)

n Control Structure Testing (又分為Condition testing和Data flow testing )

n Loop Testing (各種Loop下面自己看)

Black-Box Testing包括了:

n Comparison Testing (只在乎有可能的情況)

n Orthogonal Array Testing (狀況比較少的情況 )

==========================分隔線==================================

==========================以下正文================================

所謂可測試性Testability

l Operability(可操作性)—運作明確

l Observability(可觀察性)—各種可能的結果能很快的觀察

l Controllability(可控制性)—各階層的測試都是自動化和最佳化的

l Decomposability(可分解性)—測試時可以鎖定小部分

l Simplicity(簡單性)—減少複雜的演算法和邏輯運算

l Stability(穩定性)—測試時沒有太多的變化

l Understandability(可理解性)—便於理解

所謂好的測試 Good Test

n 找到錯誤的可能性高

n 不冗長

n “best of breed” (這我真的不知道什麼意思…最好的種類??)

n 不能太簡單也不能太複雜

內部和外部觀點 Internal and External Views

n 任何產品在測試時分為兩大部分:

n 已了解某產品的各種功能要如何展現,測試時要証明這些各功能能夠完全地操作,並且從各功能中尋找錯誤

n 已了解了某產品內部的運作方式, 測試裡頭各種元件(component)可以一起正常的運作

測試案件的設計 Test Case Design

n 目標: 接露錯誤

n 標準: 在一個完整的方式

n 限制: 用最少的時間和精力

窮舉測試 Exhaustive Testing



這圖有1014種可能的路徑, 如果一毫秒跑一條路徑, 需要3170年的時間

選擇性測試Selective Testing



軟體的測試 Software Testing

n 分為**白箱測試**和**黑箱測試**:

n 白箱測試: 保證各種陳述條件和狀況都至少跑過一次

Basis Path Testing

首先要計算**循環複雜度**(cyclomatic complexity)



若有要做抉擇(就是有必ㄘㄝ的地方), +1

若有封閉區域(就是必ㄘㄝ何起來的地方), +1

右圖做完後是 +4, V(G)=4



關於循環複雜度, 右圖中灰色區域的部分有更高的出錯的可能性



接著導出各種獨立的路徑

因為V(G)=4, 所以有四個路徑

1: 123678

2: 123578

3: 12478

4: 124724…….78

Basis Path Testing總結

n 程式碼轉換成圖的方式

n 算循環複雜度

n 導出種路徑

n 準備各種case使得每個路徑都有測試到

不同的畫圖方式 陣列 Graphic Matrix

6n-graph2.svg\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0\\ 1 & 0 & 1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0 & 1 & 1\\ 1 & 1 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ \end{pmatrix}

若在1的state可以往1,2,5這三個state去

則在陣列中將(1,1),(1,2),(1,5)這三個位置用1表示 其他以此類推

(在此陣列的行與列皆為1~6)

Control Structure Testing

n 條件測試Condition testing —設計一些方法去測試程式中邏輯條件的部分

n 資料流程測試Data flow testing — 根據在程式中使用的變數或定義的位置去選擇測試的路徑

DEF(S) = {X | statement S contains a definition of X}

USE(S) = {X | statement S contains a use of X}

迴圈測試 Loop Testing (各種不同的迴圈)



1. Simple Loops

n 先跳過迴圈走一次

n 走一次迴圈

n 走兩次迴圈

n 走m次迴圈 (m<n)

n 走n-1, n, n+1次迴圈 (n為設計時迴圈能跑的次數的最大值)

2. Nested Loops

n 從內部迴圈開始測試, 先將外部迴圈跑的次數設為最小值

n 內部迴圈跑min+1, 正常值, max-1, max次, 這時外部迴圈跑的次數還是最小值

n 將內部迴全設正常值, 外部迴圈坐上述步驟

3. Concatenated Loops

n 如果迴圈各是獨立的, 則照simple loop的方法測試

n 若否, 則照Nested Loops的方法測試

Black-Box Testing

是指只知道受測元件的介面與功能等規格(例如它應當執行哪些功能，以及「什麼樣的 input 該得到什麼樣的 output」等等)，但完全不瞭解或是完全不在乎受測元件是如何完成這些功能

Comparison Testing

n 只測試有可能的狀況

Orthogonal Array Testing

n 用於當所有input的數量少且每一個input的值不多的時後

Model-Based Testing

n 分析一個軟體中的模組.

n 放入特定的input讓模組可以跑不同的state.

n 記錄他所產出的output.

n 執行測試案件.

n 比較output以及測試案件的結果, 再按需求去做修改.

Software Testing Patterns

請見ch12 design patterns

這個就是干鬼原本好像要我們交作業後來又沒有收的那個, 寫出保全系統執行時的各種狀況, 不知大家是否還記得

ch19 Testing OO Application

OO Testing

3件事要做，來充分檢驗OO系統:

1)testing的定義要擴大包含錯誤發現技術應用在OO分析和設計模組

2)單位和成積的檢測策略要顯著的改變

3)test case的設計要說明OO軟體的獨特性

OO Testing Strategies

Unit testing

Integration Testing

Thread-based testing:綜合class集合要求去回應一個系統的輸入或事件

Use-based testing:開始系統得架構藉由檢測server class少用的class。Independent class檢測後，下一層的class稱dependent class

Cluster testing:定義一叢合作的class被試圖發現在這個合作中錯誤的designing test cases實行

Validation Testing

OOT Methods

1.每個test case應被唯一識別及跟配測class有明顯關聯

2.測試的目的要說明

3.表列testing steps為每個test發展及要包含

a.表列每個受測物的詳細說明

b.表列將被當成測試結果來實行的訊息和過程

c.表列受測時會發生的例外

d.表列外部狀況

e.補充的訊息將有助於了解和執行測試

Testing Methods

Fault-based testing

Class Testing and the Class Hierarchy

Scenario-Based Test Design