私 立 東 海 大 學
資 訊 工 程 與 科 學 所

碩 士 論 文

指導教授：朱正忠 教授

JavaBeans 程式碼再使用度量指標之研究

A Metrics Suite for Measuring Reusable JavaBeans Source Code

研 究 生：王士嘉

中 華 民 國 九 十 三 年 六 月

中文摘要

基於元件式的開發流程是多數開發團隊所採用的開發方式之一，而且也是目前最被接受的方法，此類型的開發必須基於前人所設計出來的元件之上而加上個人的需求設計，這些元件中存有前人的設計智慧與 Domain Know-How 於其中，以這些元件之上做個人需求設計可以快速的達到目前所要的需求，並且，無形中沿用了前人所設計系統架構，由此可知元件式的開發之所以會被廣泛接受並不無道理所在。

1997 年由美國 Carnegie Mellon University 的軟體工程學院(Software Engineering Institute, SEI)所提出的 CMMI 開始，軟體工程品質的議題逐漸延燒，在品質的領域中，雖然很早之前就已有學者提出，但長久以來一直缺乏標準的規格被提出與公認，CMMI 的出現為軟體品質的領域注入一道新血，開始有許多學者以 CMMI 為基礎研究軟體品質相關議題，舉凡軟體流程改善(Software Process Improvement, SPI)、Software Measurement/Metrics…等等都是其中的研究議題之一。

在本研究中所要討論的議題就是針對軟體品質中軟體度量/指標(Software Measurement/Metrics)為主，如同上述所講，元件式的開發雖然帶來了許多開發上的優點，舉凡避免重複開發元件而增快專案進度、Domain Know-How 的採用、系統架構的延用…等，但是以品質的角度去分析，人們開始懷疑對於已存在元件的品質，會想要知道這些已存在元件的品質狀況如何，因此，在這方面的研究即是利用 Software Measurement/Metrics 的理論來達到這個目的；研究中就是從這方面的理論著手，針對目前應用相當廣泛的 JavaBeans 元件程式碼做度量，度量程式碼再使用(Reuse)的可行性，達到除了元件的再使用之外，也能夠做到程式碼的再使用，所以在本研究中基於軟體度量的理論並實作一軟體度量輔助工具，使用者可以透過此度量工具進行 JavaBeans 程式碼的度量，另外還可以藉由度量工具所提供的修改建議而進行適當的改善，以期望 JavaBeans 程式碼能夠符合再

使用的設計方式。

關鍵字：軟體度量、軟體品質、軟體再使用度量指標、度量指標

Abstract

Component-base software development process is the most accepted process. This kind of process is base on exist component and adds specify requirement into design. These components keep design technology and domain know-how inside. We can develop quickly base on this kind of component. In the other side, insidiously we extend original architecture. Therefore, because of these advantages causes component-base software development process widely used.

When CMMI was proposed in 1997 by Carnegie Mellon University Software Engineering Institute in American, software quality issue has become more enthusiastic. Although software quality has already been addressed before, but it always lacks of standard specification for a long time. Till CMMI was proposed, many researchers start to research in this issue, such as “Software Process Improvement”, “Software Measurement”, “Software Metrics”, etc.

In this paper, the subject we discuss is focus on Software Measurement/Metrics in software quality. As describe above, although component-base software development process has so many advantages such as reuse component, increase project schedule, adopt domain know-how, extend architecture, etc. But from the viewpoint of quality, people start to doubt about the existed component quality. They want to know the quality status of the component. Thus, we can use Software Measurement/Metrics theory to achieve this. We use this as our core theory in this paper. Because JavaBeans component are used widely, we aim at JavaBeans source code for our research target. Measuring the reusability of JavaBeans source code is good or not. We hope beside component reuse, we can even reuse source code. Further, we can modify source code by suggestion that provided from measurement CASE tool in order to map the reusability design of JavaBeans.

Keyword：Software Measurement、Software Quality Measurement、Software Metrics、Reusable Metrics.

目錄

表格目錄

圖目錄

第 1 章　導論

1.1 前言

硬體設計的結果是很實體化的物品，想要知道此硬體品質的好壞可以透過相關的流程來加以驗證，ISO 9001 就是為了這一方面的議題而制定出的國際標準；相對的，軟體的設計應用發展至今，品質的要求也漸漸受到相當的重視，面對這抽象化的軟體設計，要如何去做軟體品質的控管是一個很重要的問題所在。

軟體品質的議題從 1997 年由美國國防部委託 Carnegie Mellon University 軟體工程學院(Software Engineering Institute, SEI [13])提出的整合能力成熟度模式(Capability Maturity Model-Integrated, CMMI)為止，國際間才有一個公認的品質標準依據，CMMI 是 SEI 聚集了多國學者，歷經數年收集這些學者寶貴的經驗，加以整合整理而來，也因此，CMMI 所制定的規範成為了軟體生產流程標準的一個準則。在 CMMI 裡面制定了相關控管軟體品質的議題，其中包括專案的管理、開發流程的控管、人員的流動、組織間的溝通方式、等，都是會影響到軟體品質的優劣，因此，想要確保軟體品質的優良勢必需要從這幾個問題點進行改善。

在這許多影響軟體品質的因素中，大致上分類可以分為管理層面與設計層面，在本文中以設計層面去探討軟體品質的議題；以現在的程式設計而言，多傾向導入設計樣版(Design Pattern)的設計方式，或者是架框(Framework)的開發，會這樣的設計，主要是為了未來的再使用(Reuse)而做，再使用的設計對開發而言，是一個很重要的因素，因為再使用的設計節省了重複開發所花費的時間與人力，另一方面也確保系統開發的一致性，有助於日後系統的維護，這些優點都是開發人員所強調，而現在想要知道系統再使用的設計程度如何，即必須透過軟體度量(Software Measurement [1])的相關理論的方式去得知，透過度量的方式，用量化

的數據來客觀呈現系統品質的程度。

1.2 研究動機

軟體 Reuse 的觀念存在於軟體開發中已有數年之久，其所帶來的好處不勝枚舉，以現在的軟體開發不管有意或是無意，都已經落實在設計中，對於 Domain Know-How 的解決方式已有前人設計出程式來解決之，後人面對這已存在的元件程式做 Reuse 是輕而易舉，只要將系統環境建置好、部署元件即可，但是，如果現在的需求需要做到些微的修改而元件無法達成，勢必要重新設計這些程式，這對專案來說是很大問題，不管成本或是時間上都是不符合成本的，而現在若是能夠做到 Reuse 程式碼，對於開發而言或許無法像 Reuse 元件一般快速，但也能夠從程式碼中去擷取到 Know-How 解決問題的方法，另一方面亦可從原本的程式碼中學習到快速的設計技巧，這對新系統的設計而言是有好無壞的。

因此，在本文中考慮對象針對 JavaBeans 類型的程式碼來做 Reuse 的研究，從手邊擁有這些 JavaBeans 程式碼去做到如何知道其中有哪些程式碼對於 Reuse 是適合的，有哪些是不適合的，透過這種方式可以快速知道這些 JavaBeans 程式碼是否適合未來的 Reuse。

1.3 研究目的

本文的研究目的有：

(1) 提出 JavaBeans 程式碼 Reuse 的度量指標。
(2) 透過組合式的度量指標，來度量 JavaBeans 程式碼 Reuse 的狀況。
(3) 度量出程式碼的 Reuse 狀況之後，提供修改建議以便使用者能夠進行適當的修改。

1.4 章節安排

本篇論文架構如下：

第一章 導論：介紹本文研究動機及研究目的。

第二章 背景知識及相關研究：介紹與本文有關度量理論與相關技術。

第三章 JavaBeans 程式碼再使用度量指標研究方法：提出本文對於 JavaBeans 程式碼的研究方法，其中包涵分析 JavaBeans 程式碼特性的度量指標，與最後利用組合式的度量指標最為度量 Reuse 程式碼的依據。

第四章 系統架構：介紹本文所要實作的系統架構與運作流程。

第五章 系統實作：最後實作結果的系統畫面。

第六章 案例研究：本文 JavaBeans 程式碼 Reuse 的實驗過程與結果

第七章 結論與未來方向：結論與未來尚可繼續的研究方向

第 2 章 背景知識及相關研究

2.1 因素/準則/指標 模式(Factor/Criteria/Metrics Model, FCM Model)

軟體品質評估模式，最早是由 Cavano 與 McCall 在 1978 年所提出，當時所提出的評估模式就是此 FCM Model [1]，在 FCM Model 中所要做的事情就是從客戶端的角度出來，根據客戶的需求而推演分析出會影響軟體品質的指標，客戶的需求可以反映出軟體產品的特性，在 FCM 中，即將軟體產品分為三類，如圖 1 所示：

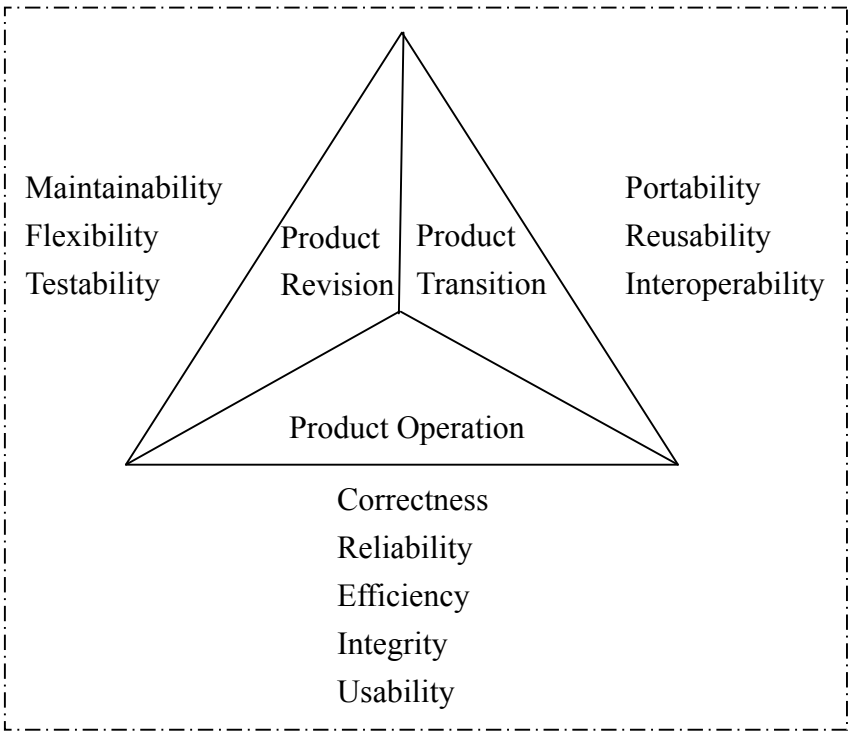


圖 1. McCall's Software Quality Factors

從圖 1 三角核心部份為軟體產品的特性，分為產品修正、變遷、與操作三種方向，根據此三方向對應出各自的因素(Factor)，總共含有 11 種 Factor，在 Factor 出現之後，會有對應的準則(Criteria)出現，每條準則在深層細分出度量指標(Metrics)，因此，此種做法結果猶如樹狀結構從根源點開始展開，如圖 2. McCall Factor/Criteria/Metrics Model：

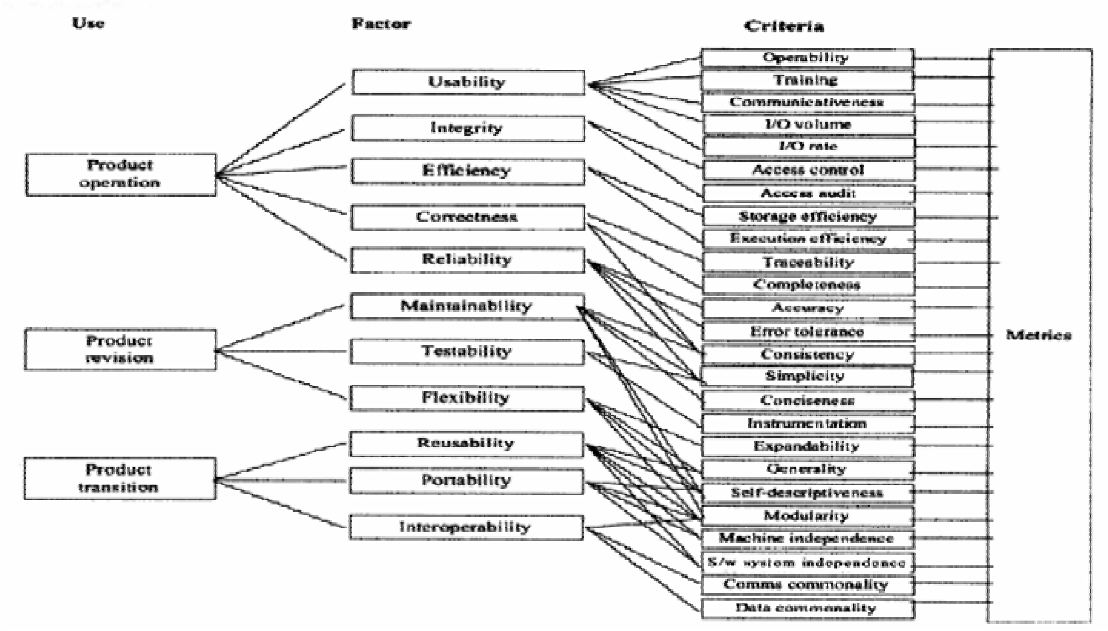


圖 2. McCall Factor/Criteria/Metrics Model

透過 FCM Model 的分析模式，使用者依據軟體產品的特性去選擇 Factor，在根據對應出的 Criteria 找出要執行的 Metric，因此軟體產品的品質控管，即可以經由執行這些 Metric 的度量結果去加以掌握。

2.2 量化專案管理(Quantitative Project Management, QPM)

CMMI 能力成熟度第四級提出組織流程績效改善與量化專案管理 [13]，組織流程績效的目的是為了建立並維護對組織標準流程績效的數量化了解，提供流程績效資料、基準及模式，以量化的方式管理組織的專案，所以，在量化專案管理的部份，首先要了解流程的執行程度，利用統計化管理得到的資料，於適當的時機採取矯正措施，降低專案錯誤率的發生以提升專案品質。

2.3 JavaBeans 設計規格書

JavaBeans 是描述 Java 的元件模型，是符合公定 JavaBeans API [10]的 Java 類別(Class)，包含屬性(attribute)及方法(method)並能夠獨立運作及重複使用，有

點類似 Microsoft 的 COM 元件，在規格書 [10] 中制定了 JavaBeans 的命名與設計規範，遵守這些規範的 Class，都可以稱為 Bean 元件；大部份的 JavaBeans 類別名稱都包含“Bean”的字眼，例如：StudentBean、DBAccessBean 等，此規範為命名慣例，為的是讓開發者立刻了解此類別是 JavaBeans，並非強制規定，但是，JavaBeans 有很重要的功能就是：內省機制(Introspection [11])，Introspection 是透過 Java 的反射(Reflection [11])架構，讓其他元件在執行階段(Run Time)取得 Bean 所具有的 public method，進一步分析這些 method 有哪些符合 JavaBeans API，如此便可以得知 Bean 所具備的屬性，因此，在 method 的設計上須具備下列條件：

- setXXX()
- getXXX()

JavaBean 的屬性，在設計上會存在一對 setter 與 getter，此種設計讓外部能夠藉 setter 設定 JavaBean 的資料，而由 getter 取得資料，其內部的運算不受外部控制，這種方式我們統稱為屬性存取方式。

2.4 ISO 9126

ISO 9126 [8]是國際公認軟體品質的標準，對軟體品質定義六個主要特徵和其使用的指引，圖 3 是 ISO 9126 所定義的 Quality Model：

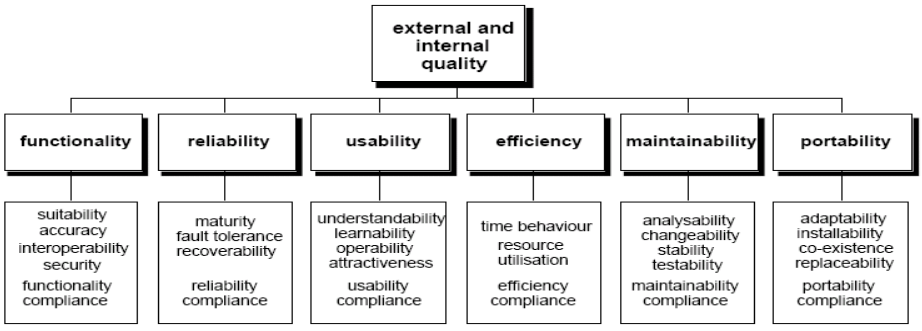


圖 3. ISO 9126 Quality Model

Quality Model [9]中分 External 和 Internal 度量指標兩類，External metrics 的定位主要以 System 為主(如圖 4 所示)，為的是度量系統的確認性(Validation)，而 Internal metrics 則是度量軟體於設計實作階段的驗證(Verification)，大體而言，

External 與 Internal metrics 的分類如圖 3 分為六大類，每大類又可以細分 metrics，雖然 metrics 的名稱相同，但在執行上 External 與 Internal 所要執行的過程卻是不同的。

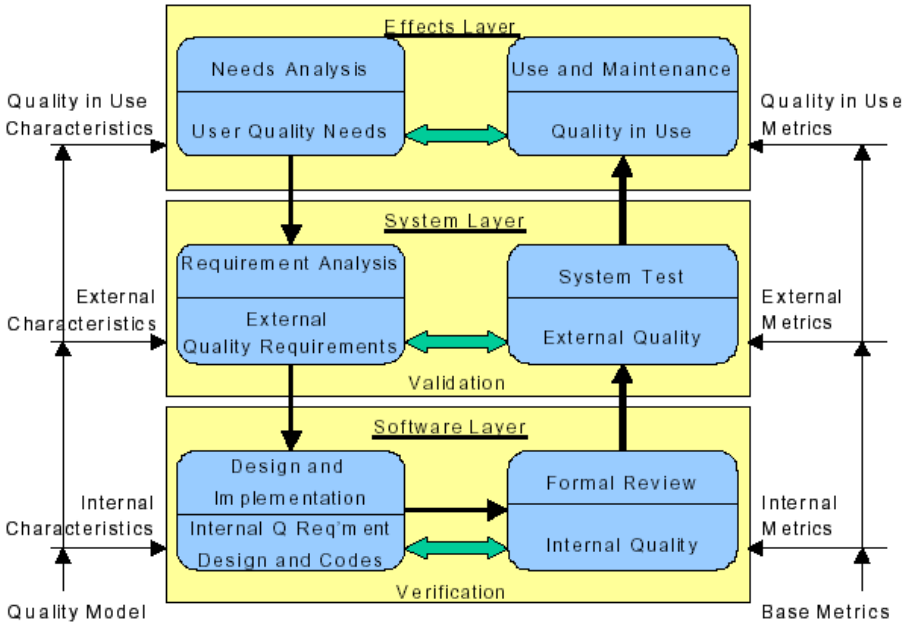


圖 4. ISO 9126 U-model

2.5 再使用元件度量指標(Metrics for Reusability of Software Component)

Hironori Washizaki 2003 年九月於 IEEE ISMS 刊登“A Metrics Suite for Measuring Reusability of Software Components” [4]，提出了對 JavaBeans 元件的度量方法，在此篇文獻中將 JavaBeans 視為一黑箱(Black Box)，在沒有原始程式的情況下，如何對此黑箱做度量，即透過 JavaBeans API 的規格書所制定的規範，提出了五個度量指標(Metrics)，分述如下：

- Existence of Meta-Information (EMI)：判斷 BeanInfo 類別的存在與否，此 BeanInfo 的存在有助於增進對 JavaBeans 元件的了解。

  $$EMI(c)=\begin{cases}1(\text{BeaniInfo類別存在})\\0(\text{不存在})\end{cases}$$

  優良區間：[0.5, 1.0]

- Rate of Component Observability (RCO)：可讀取 method 與 attribute 的

比例，計算 getter 的分部比例，位於優良區間表示此 JavaBeans 元件對外界通透性高，易於瞭解。

$$RCO = \begin{cases} \frac{Pr(c)}{A(c)} (A(c) > 0) \\ 0(其它) \end{cases}$$

Pr(c)：在 component c 中可讀取的 method 數目
A(c)：在 component c 中，所有 attribute 的數目
優良區間：[0.17, 0.42]

- Rate of Component Customizability (RCC)：可寫入 method 與 attribute 的分部比例，計算 setter 的分部比例，位於優良區間表示此 JavaBeans 元件適應性高，易於做客製化的修改。

$$RCC(c) = \begin{cases} \frac{Pw(c)}{A(c)} (A(c) > 0) \\ 0(其他) \end{cases}$$

Pw(c)：在 component c 中可寫入資料的 method 數目
A(c)：在 component c 中，所有 attribute 的數目
優良區間：[0.17, 0.34]

- Self-Completeness of Component's Return Value (SCCr)：無回傳企業邏輯(business method)數目與全部 business method 數的比例，於優良區間表示此 JavaBeans 元件對外部的依賴性低，易於跨平台。

$$SCCr(c) = \begin{cases} \frac{Bv(c)}{B(c)} (B(c) > 0) \\ 1(其他) \end{cases}$$

優良區間：[0.61, 0.96]
Bv(c)：在 component c 中，不需回傳值的 business method 數目
B(c)：在 component c 中，business method 的數目

- Self-Completeness of Component's Parameter (SCCp)：不需參數傳遞的 business method 與全部 business method 數的比例，於優良區間表示此 JavaBean 元件對外部的依賴性低，易於跨平台。

$$SCCp(c) = \begin{cases} \frac{Bp(c)}{B(c)} (B(c) > 0) \\ 1(其他) \end{cases}$$

優良區間：[0.42, 0.77]

Bp(c)：在 component c 中，不需傳參數的 business method 數目

B(c)：在 component c 中，business method 的數目

基於此五個度量指標，作者做一實驗統計與分析 [5]，發現 RCO 與 RCC 的評估結果有極大的相似趨向，而 SCCp 的評估結果並未如作者預期想像中的優良，因此在最後複合式的度量方式中作者取 EMI、RCC、SCCr 三個度量指標為基準，作為再使用元件的度量指標 Component Overall Reusability(COR)，計算如下所示：

$$COR = 1.76 \frac{V_{EMI}(c) + V_{RCC}(c) + V_{SCCr}(c)}{3} - 1.13$$

$$\text{where } V_M(c) = \begin{cases} 1(M(c) \geq LL_M \wedge M(c) \leq UL_M \\ 0(otherswise) \end{cases}$$ 優良區間：[0.5, 1.0]

M(c)區間值如表格1所示

表格 1. 區間總表

Metric Name	Lower Bound	Upper Bound
EMI	0.5	1
RCC	0.17	0.34
SCCr	0.61	0.96

透過 EMI、RCC、SCCr 三個度量指標做 COR 複合式的度量指標，以此評估 JavaBeans 元件再使用的度量依據。

第 3 章 JavaBeans 程式碼再使用度量指標研究方法

在“A Metrics Suite for Measuring Reusability of Software Components” [4]文章中，對於 Software Reusability 的度量是從 Component 的角度為出發點，對於度量的 FCM Model 根據 JavaBeans 的特性而制定，在 Sun 所制定的 JavaBeans Specification 1.1 [10]中，對於 JavaBeans 的設計會存在一個對應的 BeanInfo 類別，此設計的用意乃為了能夠提供 JavaBeans 更清楚的資訊，以便讓外界知道此 Bean 的設計用意和其用途，另外 JavaBeans 中 attribute 的存取設計上，亦有 setter 與 getter 設計方式的規定，這些都是其度量的依據，以上這些因素對應出 Understandability 與 Adaptability 兩 Factor，然而對於 Component 再使用，有關“元件移植性”問題的度量則是透過企業邏輯的回傳值作為其考量，對於此度量的方式似乎不太恰當，企業邏輯的設計，會因設計者的需要而有所不同，雖然 JavaBeans Specification 中規定了 attribute 的屬性是由 getter 所存取，然而並未限制企業邏輯的設計方式，因此作者用此度量方式作為元件移植性的考量有待商榷。

本文所要研究的對象即從 JavaBeans 程式碼為出發點，從 JavaBeans 程式碼為出發點去做度量會和 Component 有些許不同，這是因為 Component 的度量方式，沒辦法深入到內部，而只能透過 Component 對外的介面(Interface)去取得資訊而度量，而現在如果能夠透過程式碼去做度量，即可以得到詳細的資訊，在度量上即可根據再使用(Reuse)的需求而做更適當的度量。

3.1 程式碼再使用 FCM 模式 (Reusable Source Model)

從[12]書中對於程式再使用的定義發現，有關 Reuse 的過程都必須先對再使用對象有相當程度的瞭解，從瞭解的過程中知道此對象是否符合需求，清楚知道對象的功能之後才會有再使用的考量，因此，在 Reuse 的過程中，對於再使用對象瞭解程度是一個很重要的依據之一；在瞭解過再使用對象之後，緊接著就是要考慮到此對象跨平台的特性與其資料的修改可行性，再使用的過程，考慮到平台的變換是一個關鍵，能夠解決平台變換的需求，提升了再使用的可能性；資料的修改則是為了能夠輸入不同的資料而做不同的處理所設計；綜觀上述幾點對於再使用的設計觀點，與 ISO 9126 Quality Model 並引用“A Metrics Suite for Measuring Reusability of Software Components”[1][2][3][6]所提而做一整理比較，發現 Understandability、Portability、Adaptability 三個特性符合上述所說的再使用設計觀點，因此特別選用此三個特性作為本文中 FCM Model 的 Factor，除此之外，對於 Understandability 的特性，補上 Complexity 以作為呼應，以此四個 Factor 作為程式碼再使用相關的特性，此四點的特性分述如下：

- Understandability 特性：此特性決定了 JavaBeans 程式碼的可再使用與否，程式設計師對於程式碼的再使用必先對程式碼有初步的瞭解，清楚程式的架構、邏輯，方可進一步加以利用，並且在需要的時刻做客製化的修改，此特性對於程式碼的再使用與否有其影響所在。
- Complexity 特性：意義類似 Understandability 的用意，此特性乃針對程式碼的可瞭解程度而設計，複雜度越高造成程式碼越不易瞭解，相反則越易瞭解，對於再使用的程式碼，應該力求簡單明瞭。
- Portability 特性：此特性關係到 JavaBeans 程式碼的設計時，有否考慮到移植的問題，也就是和系統環境之間耦合性的高低，對於 JavaBeans 程式碼再使用的情況，耦合性越低，提升了再使用的方便性，降低移植的問題與系統之間的關聯。
- Adaptability 特性：此特性牽涉到 JavaBeans 程式碼的可客製化程度，對於再使用程式碼而言，會因使用者資料的不同，而處理出不同結果的資料，同樣的程式邏輯但不同的資料，如此一來，此 JavaBeans 的再使用才有其意義。

FCM 理論的 Factor 在定義出來之後，緊接著就是準則(Criteria)與指標

(Metrics)的推演，整個 FCM 樹狀結構的模式圖如圖 5 所示：

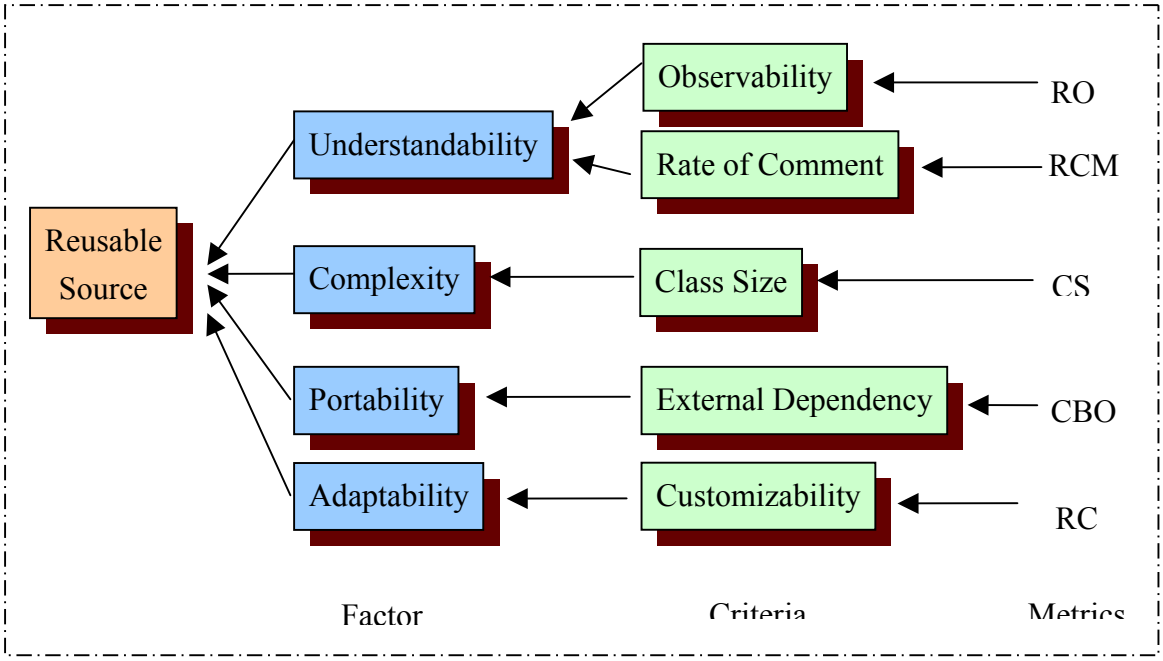


圖 5. Reusable Source Metrics FCM Model

Criteria 的用意分述如下：

(1) 可視性(Observability)：可視性的度量指標，就是在評估程式碼對外界的溝通性，也就是程式碼提供對外界介面(Interface) 的多寡，透過這公佈(Public)的 Interface，外部才可以做資料的輸入、資料處理、與資料的存取，如此一來，程式碼的再使用才有其可行之處，否則，程式碼不提供對外界的 Interface，就猶如黑箱(Black-Box)一般，無法提供清楚的資訊給外部，難以知道如何再使用此程式碼。

(2) 註解比例(Rate of Comment)：註解比例的度量指標，評估在程式碼中穿插註解的多寡；撰寫 Java 程式的同時，通常為了程式在未來的維護(Maintain)與傳承，以及利用 javadoc 的功能自動產生說明文件，都會撰寫註解、說明於程式碼中，而其他人員透過這些註解即可以迅速知道這程式碼的意義所在，因此，透過註解比例的度量結果，可以知道此程式碼說明文件的多寡，相對的顯示出此程式碼被瞭解程度的優劣。

(3) 程式大小(Class Size)：程式碼多寡的度量指標，評估程式碼行數的多寡，程式碼的行數越多，顯示其中的處理邏輯越複雜，相對的減低了被瞭解程度，相反則增加被瞭解程度，因此，程式碼的瞭解程度亦可用此為參考依據。

(4) 外部相依性(External dependency)：與外界環境的相依性的度量指標，相依性越高，說明了程式碼越不易移植；當有外部的相依性存在，增加了程式碼再使用的困難度，需要做客製化的修改必須增加許多，也有可能

因此而破壞了原本程式設計的用意與架構，因此，此度量結果點出再使用的難度性。

(5) 客製化(Customizability)：可客製化的度量指標，意義同 Observability，只不過 Observability 的度量對象為讀取資料，而 Customizability 則是做寫入資料的動作，根據不同的使用需求，將不同資料輸入給程式，程式碼的再使用勢必需要提供此功能，使用者方可根據不同的資料而達到再使用的目的。

3.2 程式碼再使用度量指標 (Definition of Reusable Source Metrics)

根據上述的 Criteria，在這章節定義出對應的度量指標(Metric)，依序為 RO、RCM、CS、CBO、RC 等五個度量指標，五個 Metric 有各自的度量方法，與其優良的度量區間，利用數學式的表示法，*M* 代表度量指標，其優良區間上限值用 *UB*(Upper Bound)，下限值用 *LB*(Lower Bound)分別表示之。

圖 6 是利用 Rational 公司開發的 UML 朔模工具-Rose，所設計的簡單 JavaBeans Class Diagram，圖中有三個程式，PeopleBean 是父類別，StudentBean 與 TeacherBean 是子類別，繼承自父類別 PeopleBean 而來，接下來本文所定義的五個度量指標就以此圖來解釋。

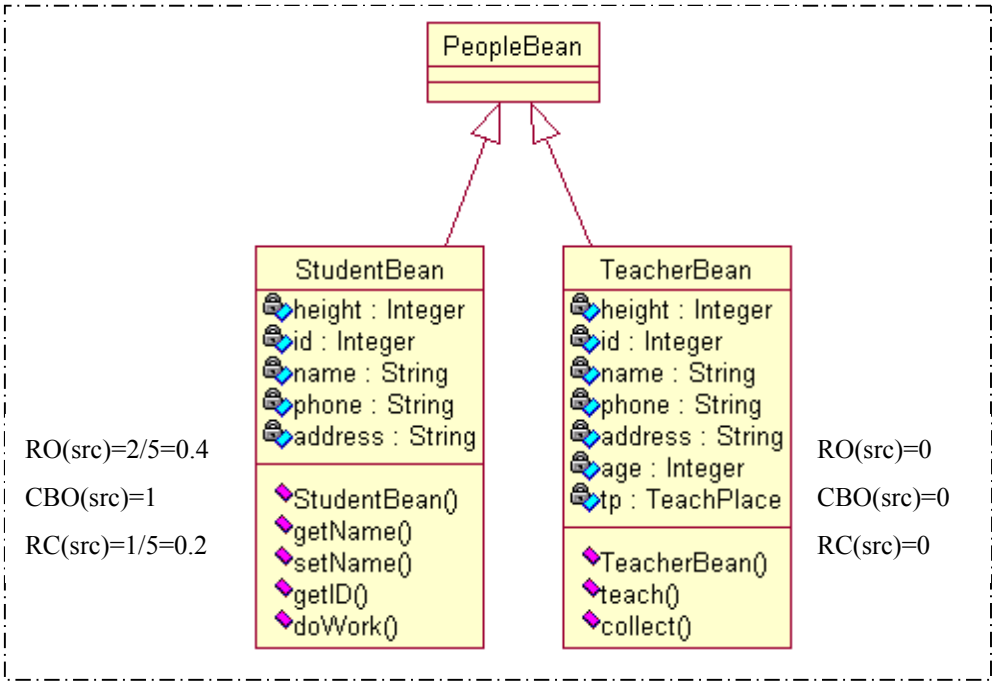


圖 6. Bean Class Diagram 範例

3.2.1 可視性比例(Rate of Observability, RO)

此 Metric 引用 “A Metrics Suite for Measuring Reusability of Software Components” [4]所提 RCO(Rate of Component Observability)的 Metric，其定義如下：

$$RO = \begin{cases} \frac{Pr(src)}{A(src)} (A(src) > 0) \\ 0(otherwise) \end{cases}$$

where

Pr(src) : source中可讀取method數目

A(src) : source中所有attribute數目

優良區間：[0.17, 0.42]

RO 計算程式碼中 attribute 有被外界讀取數目的比例，區間值位於 0~1 之間，透過$\frac{Pr(src)}{A(src)}$的計算方式得知其比率，此優良區間乃根據[4]所提而定，RO 的值位於此優良區間表示此可視性佳，被瞭解程度也提升，相反的，RO 的值低於優良值 0.17，表示此程式碼中的資料不被外部讀取，可用黑箱來形容之，而若 RO 的值高於優良值 0.42，則表示過多的 attribute 被外部讀取，反而無法知道哪個 attribute 重要，造成難以理解程式的設計用意，因此，0.17 與 0.42 之間算是一個較為均衡的區段。

參考圖 6 StudentBean 與 TeacherBean 作為比較，StudentBean 所計算出來的 RO 值為 0.4，而 TeacherBean 為 0，顯然 StudentBean 的 CO 值優於 TeacherBean。

3.2.2 註解行數百分比(Rate of Comment, RCM)

定義：計算註解行數與 method 行數的比例

$$RCM(src) = \frac{\sum comment\_line}{\sum method\_line}$$

優良值：[30%, 75%]

RCM 計算每個 method 的註解行數比例，在撰寫 Java 程式會將註解嵌入其中，如圖 7 所示：

A
```
/**
 * Returns true if the menu is currently selected (highlighted).
 *
 * @return true if the menu is selected, else false
 */
```
B
```
public boolean isSelected() {
    return getModel().isSelected();
}
```

圖 7. RCM 範例程式

在圖 7 中 B 區塊就是所撰寫程式的部分，而與此段程式有關的註解則會根據 Java 程式語法的規定寫在“/*…*/” [22]之間，如 A 區塊所示，所以 RCM 即計算註解與程式兩者的行數比例，開發軟體測試的 Testwell Oy(Ltd) [16]公司更建議註解行數與程式行數的比例，最好維持在 30%以上而保持在 75%以下，以提高程式的可讀性。

3.2.3 程式規模(Class Size, CS)

定義：計算程式碼的行數

$$CS(src) = \sum source\_line$$

優良值：[50, 400]

CS 計算 JavaBean 程式中程式碼的行數，以初步瞭解程式碼複雜度的高低，Testwell Oy(Ltd) [15][16]公司對於程式碼的大小，建議行數在 400 行內是開發人員最能接受的範圍之內而複雜度屬於可以接受[23]，超過 400 行增加了複雜度而使得程式碼變的不易瞭解。

3.2.4 物件間的耦合度(Coupling Between Objects, CBO)

定義：JavaBean 程式中有引用外部的物件

$$CBO = \begin{cases} 0(\text{引用外部物件}) \\ 1(\text{otherwise}) \end{cases}$$

優良區間：[0.5, 1]

計算 JavaBean 程式中有否引用外部物件，外部物件的引用增加了物件之間

的耦合度，雖然提升了彼此之間的關聯性，但是卻也增加了物件移植的困難度，再使用的程式碼即要避免耦合度的發生，因此，透過 CBO 的度量指標來檢查耦合度的發生。

以圖 6 為例，StudentBean 中未引用外部物件，所以其 CBO 值為 1，而 TeacherBean 中引用了 TeachPlace 的外部物件，所以其值為 0，因此 StudentBean 程式碼的再使用優於 TeacherBean。

3.2.5 客製化比例(Rate of Customizability, RC)

此 Metric 亦引用[4]所提 RCC(Rate of Component Customizability)的 Metric，其定義如下：

$$RC(src) = \begin{cases} \dfrac{Pw(src)}{A(src)} (A(src) > 0) \\ 0(otherwise) \end{cases}$$

where

Pw(src) : source中可寫入資料的method數目

A(src) : source中所有attribute的數目

優良區間：[0.17, 0.34]

RC 計算程式碼中 attribute 可被外界設定資料的數目比例，區間值位於 0~1 之間，優良區間根據[4]所提，優良區間表示可客製化修改佳，所以低於優良區間表示不容易做客製化的修改，但高於優良區間，則違反資料封裝(Encapsulation)的特性，而且可能導致程式誤用的情況發生。

以圖 6 來說，StudentBean 有一個 setter，計算出來的 RC 值為 0.2，而 TeacherBean 的 RC 值計算出為 0，表示 StudentBean 的可客製化修改優於 TeacherBean。

3.3 組合式度量指標 (Composition Metrics)

程式碼的再使用度量指標，從上述來看，有五個度量指標會影響程式碼再使用的特性，因此，在這邊提出複合式的度量指標 Reusable Source Metric(RSM)來

作為度量 JavaBean 程式碼可再用的依據：

$$RSM(src)=V_{RO}(src)+V_{RCM}(src)+V_{CS}(src)+V_{CBO}(src)+V_{RC}(src)$$

$$\text{where } V_M(src)=\begin{cases}1(M(src)\geq LB_M \wedge M(src)\leq UB_M)\\0\end{cases}$$

優良區間：[3, 5]

綜合整理此五個 Metric 的優良區間如表格 2 所示：

表格 2. 度量指標區間總表

Metric Name	Lower Bound	Upper Bound
RO	0.17	0.42
RCM	30%	75%
CS	50	400
CBO	0.5	1
RC	0.17	0.34

在計算 RSM 此度量指標的過程中，必須先將前一節所提的五個度量指標，RO、RCM、CS、CBO、RC 都先計算出來，而其各自的 LB 與 UB 如表格 2 所整理，當這些細部的度量指標位於優良區間，則其 V 的值為 1，否則為 0，RSM 的結果即是透過這五個 Metric 所計算出，預估優良的可再用 JavaBeans 程式碼為 3 到 5 之間，因此，JavaBeans 的可再用程式的度量即是透過 RSM 來做評估。

第 4 章　系統架構

本文所提的方法，就是提供度量程式碼再使用的理論，進一步可以輔助使用者增進程式碼的再使用，圖 8 就是所要實作的系統架構示意圖：

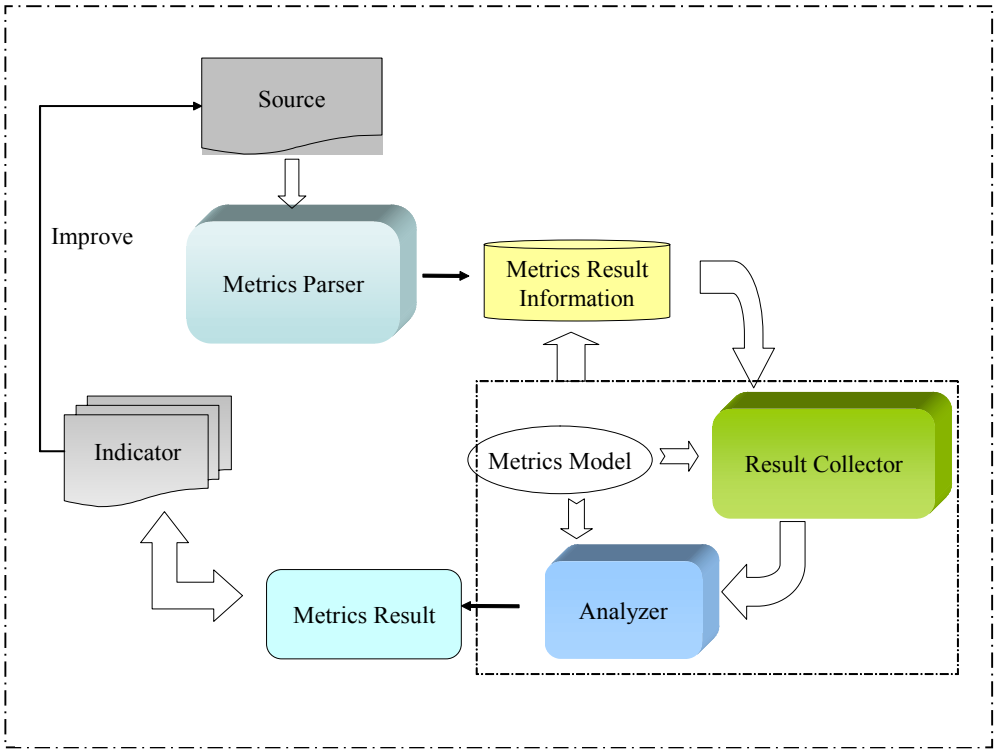


圖 8. 系統架構示意圖

本系統所要處理的資料，即以 JavaBeans 的程式碼(Source)為主，針對使用者輸入進來的 Source，首先會經過 Metrics Parser 做第一階段的解析，在 Metrics Parser 這一步驟中，其解析的目的是為了將 Source 中的程式區塊做切割，帶程式區塊切割後，將區塊的位置(Index)與重要字串(Primary String)存到 Metrics Result Information，透過 Metrics Result Information 將這些資訊存放之後，由扮演 Controller 腳色的 Result Collector 做資料的存取，在存取的過程上，乃根據 Metrics Model 的規則而做，Metrics Model 中即是上述所提 RSM 的 FCM Model，此中就是由 RO、RCM、CS、CBO、RC 所組合而成，資料收集完即透過 Analyzer 做數值的計算，最後送交 Metrics Result 做資料的呈現。

整個系統的 Layer 分布如圖 9 所示

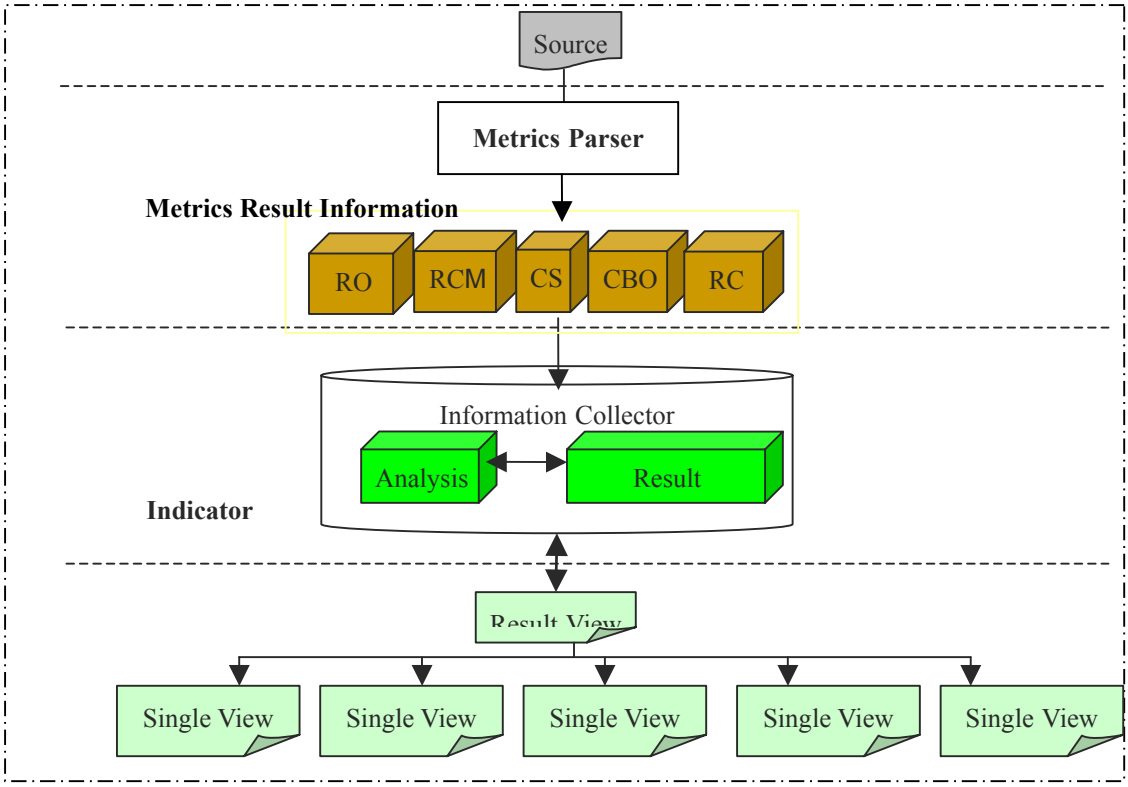
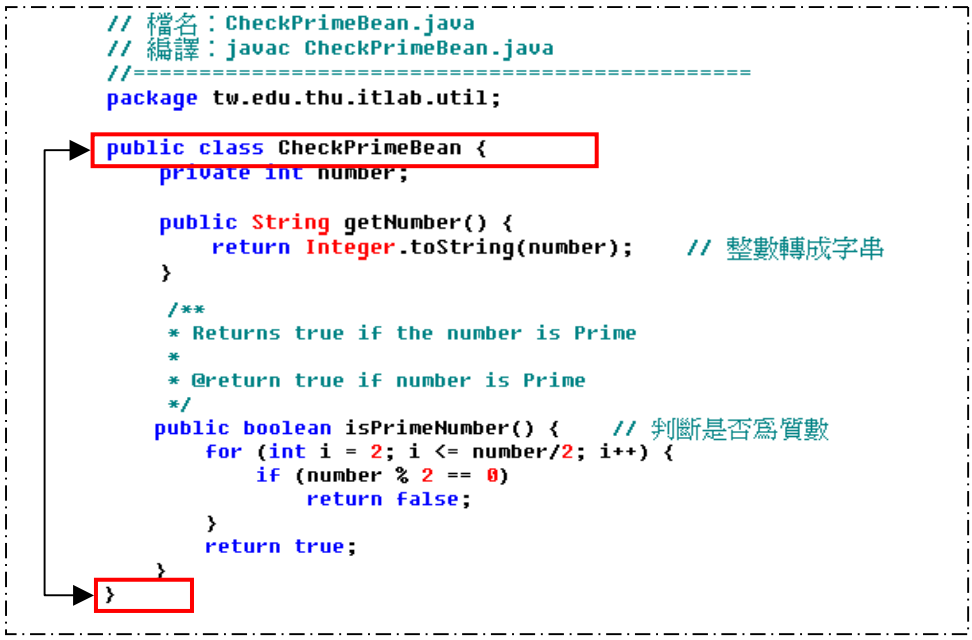


圖 9. 系統 Layer 架構

接下來將依序介紹 Metrics Parser、Information Collector 與 Indicator 的運作目的。

4.1 度量指標解析器

在度量指標解析器這一區塊中，即是針對 Source 做 Metrics 的解析，在做 Metrics 的解析之前，先對 JavaBean Source 做初步的處理，如圖 10 所示：



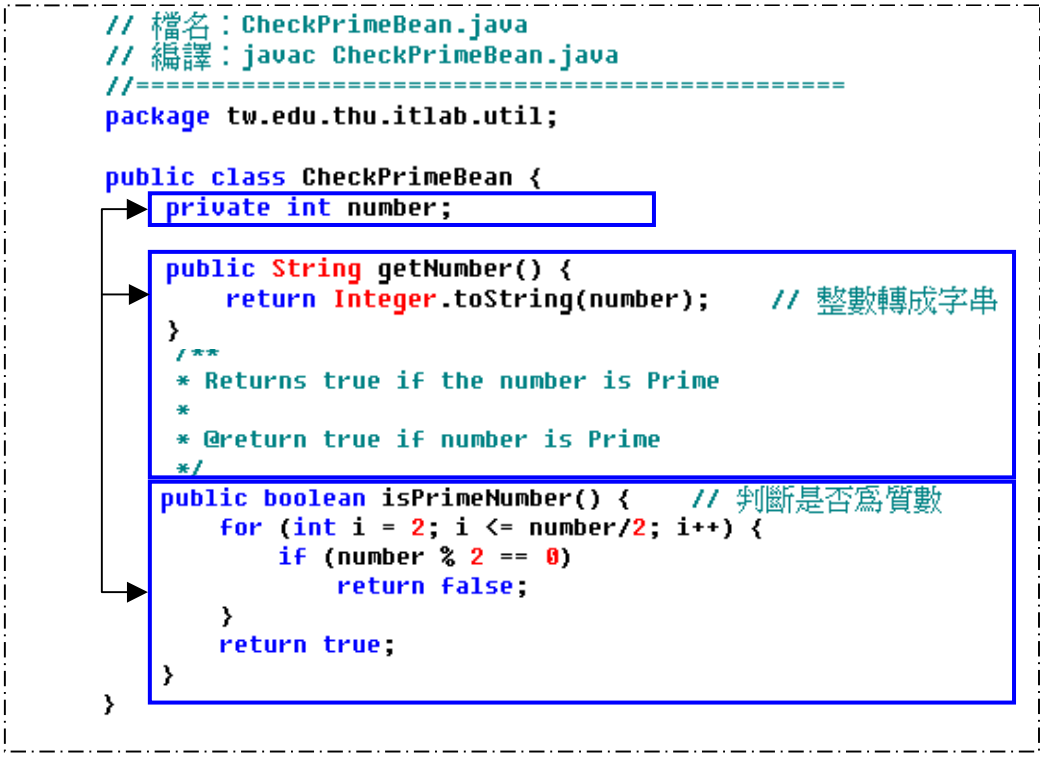
```
// 檔名：CheckPrimeBean.java
// 編譯：javac CheckPrimeBean.java
//================================================
package tw.edu.thu.itlab.util;

public class CheckPrimeBean {
    private int number;

    public String getNumber() {
        return Integer.toString(number);    // 整數轉成字串
    }
    /**
     * Returns true if the number is Prime
     *
     * @return true if number is Prime
     */
    public boolean isPrimeNumber() {    // 判斷是否爲質數
        for (int i = 2; i <= number/2; i++) {
            if (number % 2 == 0)
                return false;
        }
        return true;
    }
}
```

圖 10. JavaBeans Source 範例：內部碼切割

在處理的過程上，先將 Java 程式的撰寫規則：程式命名宣告切割出來，根據 Java 程式撰寫規則，先對最外層的區塊(bracket)“{…}”做切割，如圖 10 所選擇的區塊，透過此方法的切割，內部的程式邏輯與變數宣告獨立出來，稱之為內部碼，接著再繼續對這內部碼做 public/private 的字串切割：



```
// 檔名：CheckPrimeBean.java
// 編譯：javac CheckPrimeBean.java
//================================================
package tw.edu.thu.itlab.util;

public class CheckPrimeBean {
    private int number;

    public String getNumber() {
        return Integer.toString(number);    // 整數轉成字串
    }
    /**
     * Returns true if the number is Prime
     *
     * @return true if number is Prime
     */
    public boolean isPrimeNumber() {    // 判斷是否爲質數
        for (int i = 2; i <= number/2; i++) {
            if (number % 2 == 0)
                return false;
        }
        return true;
    }
}
```

圖 11. JavaBeans Source 範例：public/private 切割

內部碼的切割，針對 public/private 的字串做處理，以 public/private 為基準點，切割每塊區塊，稱之為內部碼區塊，如圖 11 所示，圖中即切出三個內部碼區塊，在處理上將這些內部碼區塊稱之為“區塊集合”，到此時，程式碼中 public/private 的 method 或 attribute 都以區塊的型式處理，因此，下一步驟就是要區分這些區塊中 method 與 attribute 的分布：

```
// 檔名：CheckPrimeBean.java
// 編譯：javac CheckPrimeBean.java
//================================================
package tw.edu.thu.itlab.util;

public class CheckPrimeBean {
    private int number;

    public String getNumber() {
        return Integer.toString(number);    // 整數轉成字串
    }
    /**
     * Returns true if the number is Prime
     *
     * @return true if number is Prime
     */
    public boolean isPrimeNumber() {    // 判斷是否爲質數
        for (int i = 2; i <= number/2; i++) {
            if (number % 2 == 0)
                return false;
        }
        return true;
    }
}
```

圖 12. JavaBeans Source 範例：method 與 attribute 分布

圖 12 中所選擇的區塊，則是 method 與 attribute 的分布情況，但是，礙於每個程式設計師的設計風格不依，其撰寫風格多樣，如圖 13：

```
    private Date date = new Date();
    private String str, str2 = new String("");
    private int number, input,
                output;

    public String getNumber()
    {
        return Integer.toString(number);    // 整數轉成字串
    }
    public String setNumber(String number) {
        return Integer.toString(number);    // 整數轉成字串
    }
```

圖 13. 程式撰寫風格

圖 13 attribute 的宣告會出現斷行與不斷行的情況發生，而 method 的撰寫，也同樣會出現 bracket 斷行與否的情況，因此內碼區塊 attribute 與 method 的辨別，

就是根據“;”與“{”符號來做其辨視，如圖 13 選取的區塊，直到目前為止，已經可以確認 attribute 與 method 的分布數目如何，此時，參考 ValueObject Pattern []的概念，將這些 attribute 內部碼區塊與 method 內部碼區塊的分布位置(Index)存在 ValueObject 的物件中以方便接著度量指標的處理。

4.1.1 RO 與 RC 度量指標資訊處理

RO 與 RC 兩個度量指標即是在計算 setter 與 getter 的 method 數與 attribute 總數的比例，因此，從 ValueObject 中取出 method 內部碼區塊來做處理，計算區塊集合中 setter 與 getter 的數目之後，即可以與 attribute 的數目做計算而取得 RO 與 RC 的值。

4.1.2 CS 度量指標資訊處理

CS 即計算程式碼行數的多寡，計算中從計算 Java Source 名稱的宣告開始計算，而 package 與 import 的宣告欲以忽略不計算在內，收集 attribute 內部碼區塊與 method 內部碼區塊，計算這些內部碼區塊集合的行數即 CS 的值。

4.1.3 RCM 度量指標資訊處理

RCM 行數的計算，即根據 method 內部碼區塊來做處理：

```
public void setNumber(String s) {
    try {
        number = Integer.parseInt(s);   // 字串轉成整數
    }
    catch (NumberFormatException e) {
        number = -1;
    }
}
 /**
 * Returns true if the number is Prime
 *
 * @return true if number is Prime
 */
public boolean isPrimeNumber() {    // 判斷是否爲質數
    for (int i = 2; i <= number/2; i++) {
        if (number % 2 == 0)
            return false;
    }
    return true;
}
```

圖 14. 內部碼區塊註解的分布

圖 14 中可以發現，method 內部碼區塊中會包含程式碼註解於其中，因此，註解行數的計算即透過“/* …. */”的區塊來求得，RCM 的值即可以求出。

4.1.4 CBO 度量指標資訊處理

度量外部物件的引用，乃根據 attribute 內部區塊來計算，

```
private Date date = new Date();
private String str, str2 = new String("");
private int number, input,
            output;

public String getNumber()
{
    return Integer.toString(number);    // 整數轉成字串
}
public String setNumber(String number) {
    return Integer.toString(number);    // 整數轉成字串
}
```

圖 15. attribute 型態範例

如圖 15 中選擇區塊，可以解析出物件型態，並利用 ClassLoader 的機制對物件型態呼叫，當 ClassLoader 發生錯誤例外事件(Exception Event)時，即表示 JVM(Java Virtual Machine)找不到此物件的存在，也就是說此錯誤的發生自來外部物件所造成的，依此判斷外部物件的引用與否。

4.2 資料處理運算

Information Collector 的腳色就如同 MVC Pattern 中“Controller”的腳色，圖 16 為 MVC Pattern 的內部元件間溝通的示意圖：

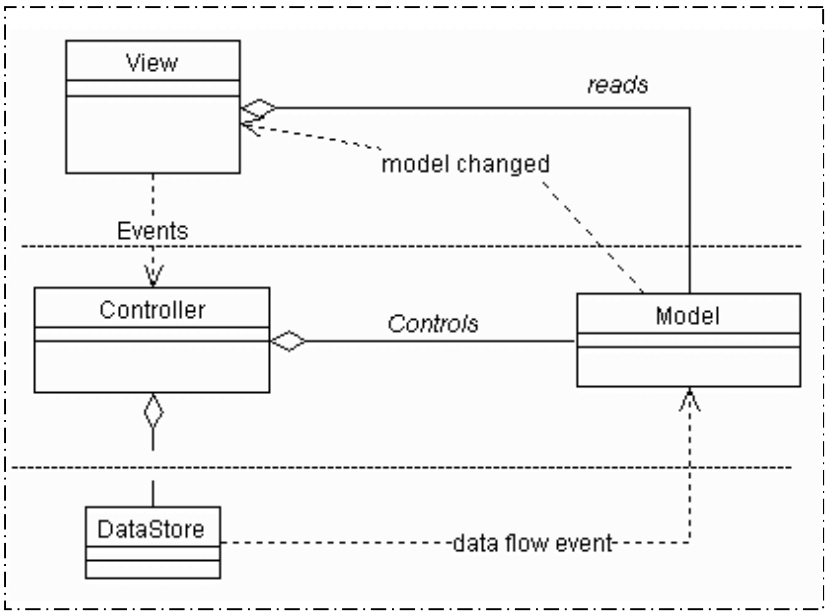


圖 16. MVC 架構圖

Metrics Result Information 中的資料如同 Model 的腳色，而 Result Collector 就如 Controller 負責把 Metrics Result Information 中的資料讀出來加以整理分析；Metrics Model 中即是本文所提的五個 Metrics 的度量規則，Result Collector 即根據 Metrics Model 做適當資料的收集與處理，並且交由 Analyzer 做結果分析，分析的過程中，透過 Metrics Model 的定義做改善這些度量指標，度量指標結果值不良者，經由分析而提供改善建議，如此一來使用者方可知道如何做改善的動作，因此 Metrics Result Collector 中分析的依據即是根據每個度量指標的度量依據，以綜合整理改善意見，並將這些資訊回存到 Metrics Parser 之中。

4.3 度量指標指示器

資料的呈現，Indicator 的資料內容乃根據 Metrics Parser 的資料而來，也就是 MVC 中 Model 和 View 的關係，Indicator 除了做資料的呈現之外，更進一步還要提供度量的改進建議，在 Indicator 這一部分中，並不會處理任何的程式邏輯，也就是說，所有的資料呈現，都會由 Metrics Parser 中取出，此乃 MVC Pattern 中的設計用意。

4.3.1 RO 指示器

此指示器顯示所有 getter method 與所有 attribute 位置清單，RO 值位於優良區間顯示 RO 值正常，否則，RO 值太高或太低，則會顯示 getter method 清單太多或太低，並提供應該增加對應 attribute 的 getter method 數目以改進 RO 值。

圖 17 圖中所顯示的是某一隻程式中所有 method 與 attribute 的分部情況，根據 RO 的定義所計算出來，此程式中 RO 的值為大於 1，超過優良區間 0.17 與 0.42 之間，因此 RO 指示器則必須計算出要減少多少的 getter 數目，使得 RO 的值能夠位於優良區間，經過指示器的計算，指示器提出 getter 數目需減少 3 個，剩一個以使得計算 RO 值能夠位於優良區間。

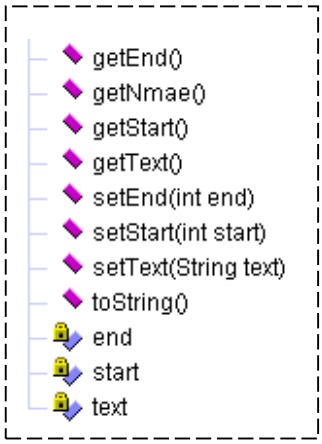


圖 17. 指示器範例

4.3.2 CS 指示器

CS 乃度量 JavaBean 程式碼行數的多寡，此指示器顯示 CS 值得優劣之外，並提出該增進程式碼行數或減少以改進 CS 值。

如圖 18 所示，CS 指示器乃提供使用者知道目前此程式的大小，當有超過優良值的情況發生，才會告知以超過優良值。

```
12      public void setNumber(String s) {
13          try {
14              number = Integer.parseInt(s);   // 字串轉成整數
15          }
16          catch (NumberFormatException e) {
17              number = -1;
18          }
19      }
20      public boolean isPrimeNumber() {    // 判斷是否爲質數
21          for (int i = 2; i <= number/2; i++) {
22              if (number % 2 == 0)
23                  return false;
24          }
25          return true;
26      }
27  }
28
```

圖 18. 程式碼大小範例

4.3.3 RCM 指示器

此指示器除了算出註解行數的比例之外，亦呈現每個註解區塊的分部，當算出 RCM 的值不在優良區間時，則告知使用者此時程式碼總大小與註解行數的多寡，並須增加註解行數而使得 RCM 的值能夠趨近優良區間。

圖 19 中所顯現的是完整的一隻程式，指示器將註解區塊與 method 區塊(如圖中所標示處)將以樹狀方式呈現出來，待使用者選擇某一區塊號碼，則會有對應程式碼的視窗呈現給使用者，供使用者做適當的修改。

```java
// 檔名：CheckPrimeBean.java
// 編譯：javac CheckPrimeBean.java
//=============================================
package tw.edu.thu.itlab.util;

public class CheckPrimeBean {
    private int number;

    /**
    * 取得字串型態的數字
    */
    public String getNumber() {
        // 整數轉成字串
        return Integer.toString(number);
    }

    /**
    * 設定字串型態的數字
    */
    public void setNumber(String s) {
        try {
            // 字串轉成整數
            number = Integer.parseInt(s);
        }
        catch (NumberFormatException e) {
            number = -1;
        }
    }

    /**
    * 檢驗所設定的數字是否爲質數
    */
    public boolean isPrimeNumber() {
        // 判斷是否爲質數
        for (int i = 2; i <= number/2; i++) {
            if (number % 2 == 0)
                return false;
        }
```

圖 19. 程式碼註解比例範例

4.3.4 CBO 指示器

CBO 乃偵測 JavaBeans 程式中有否引用到外部物件，因此，指示器則顯示出在程式碼中所有 attribute 的清單，並點出清單中哪些 attribute 是外部物件，需減少此外部物件的引用以改進 CBO 的值。

圖 20 則是一個 CBO 指示器的範例，指示器將所有外部物件以樹狀方式呈現出來，待使用者選擇某一物件名稱，則會有對應程式碼的視窗呈現給使用者，供使用者做適當的修改。

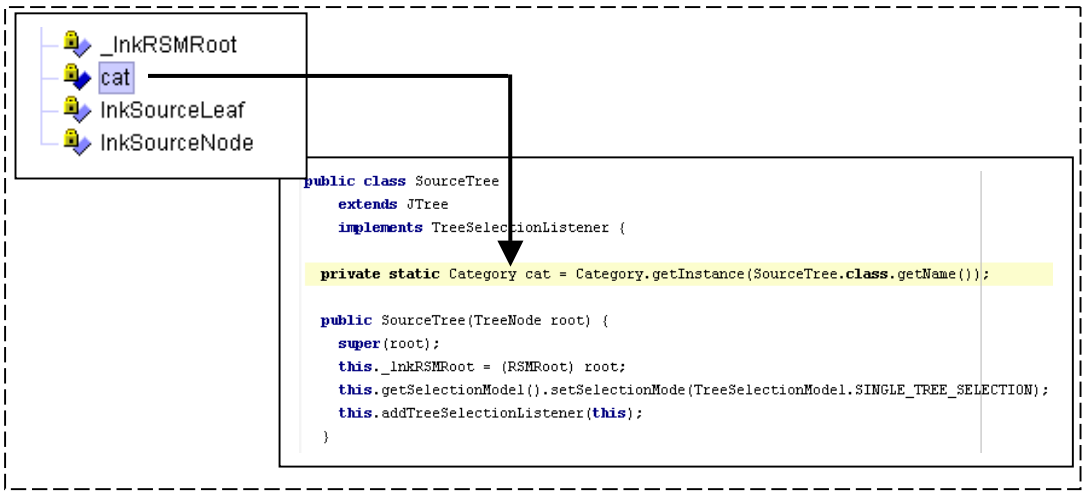


圖 20. 外部物件範例

4.3.5 RC 指示器

此指示器顯示所有 setter method 與所有 attribute 位置清單 RC 值位於優良區間顯示 RC 值正常，否則，RC 值太高或太低，則會顯示 setter method 清單太多或太低，並提供應該增加對應 attribute 的 setter method 數目以改進 RC 值。

參考圖 17，圖中所顯示的是某一隻程式中所有 method 與 attribute 的分部情況，根據 RC 的定義所計算出來，此程式中 RC 的值為等於 1，超過優良區間 0.17 與 0.34 之間，因此 RC 指示器則必須計算出要減少多少的 setter 數目，使得 RC 的值能夠位於優良區間，經過指示器的計算，指示器提出 setter 數目需減少 2 個，剩一個以使得計算 RC 值能夠位於優良區間。

第 5 章　系統實作

本章節將介紹系統在實作上的設計規劃與系統畫面。

在介紹之前，依照軟體開發流程的順序而言，先介紹系統規劃時所設計的 Use Case Diagram，如所示：

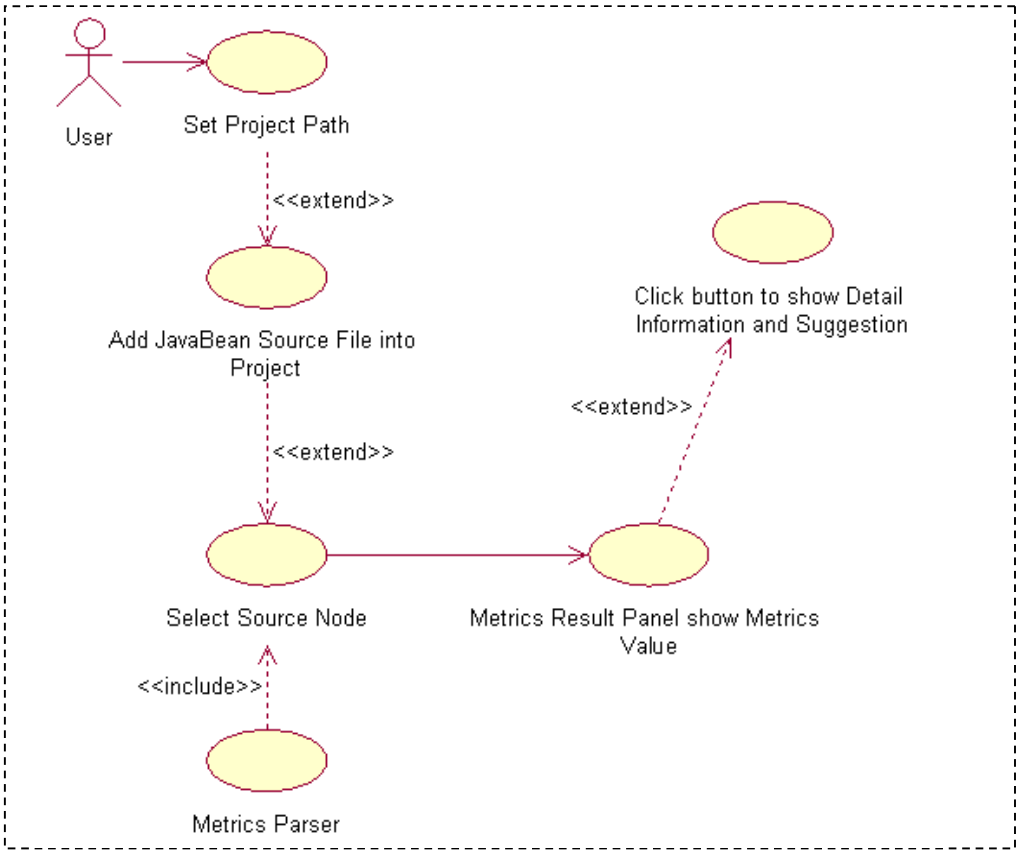


圖 21. 系統 Use Case Diagram

從圖 21 可以看出，系統在運作上，使用者的操作直接與 Use Case：Set Project Path 直接運作，從此開始一連串的系統運作，接下來分別介紹每個 Use Case 中所代表的意義。

表格 3. Use Case 定義

Use Case 名稱	定義
Set Project Path	系統初始階段，由使用者設定其所要度量的程式碼目錄；因系統運作中會以專案目錄為基準點，所以系統運作

	上，必須由使用者先設定其專案目錄起始路徑方可正常的運作，若使用者未設定專案目錄，系統預設起始目錄設定為 C:\temp。
Add JavaBeans Source File into Project	由使用者觸發新增檔案的系統事件；由設定專案目錄之後所延伸出來的 Use Case，使用者點選所要新增的 JavaBeans 檔案後，進入偵測檔案階段，偵測檔案所存在的目錄位置是否與專案目錄相同，如果不在專案目錄下結束此流程，如果相同專案目錄，則建立此檔案在專案目錄中所應該存在的樹狀位置。
Select Source Node	由使用者透過畫面選擇所要度量的 Source Node；系統從專案目錄中取得所有 JavaBeans 程式碼後，以樹狀結構呈現之，透過使用者去選擇所要度量的樹狀節點，即開始進入自動化的解析程式碼。
Metrics Parser	此 Use Case 是系統內部的運作；從其字面來看，就是系統中解析程式碼的過程，因為藉由使用者去選擇其所要度量的程式碼後，系統自動觸發此事件，開始進行解析程式碼的流程，也就是本文所提的五個度量指標的規則，透過規則去解析並計算結果，因此 Metrics Parser 與 Select Source Node 之間的關係，從圖 21 可發現中間存在有 include 的 stereotype，也就是說 Metrics Parser 要被包含在 Select Source Node 的 Use Case 之中。
Show Metrics Value	將度量結果呈現於畫面上；此 Use Case 即是做資料畫面的呈現，不含複雜的程式邏輯，和 Select Source Node 之間的關係有其順序先後，也就是說此 Use Case 所要呈現的畫面，需要先經過 Select Source Node 的處理，待資料解析過後，才會送到此階段做畫面呈現，因此從圖 21 中兩者間的關係存在一有方向性的 Association。
Show detail Information and Suggestion	顯示近一步的修改建議；在系統度量出程式碼的五個度量指標之後，使用者想要近一步知道這些度量指標的情況如何，又或是使用者想要近一步的修改以增進這些程式碼的再使用情況，因此，透過系統而得到更近一步的建議。

接下來將依圖 21 與表格 3 中所敘述的 Use Case 分別介紹系統畫面：

5.1 Use Case: Set Project Path

先設定所要度量的專案目錄位置，畫面如所示：

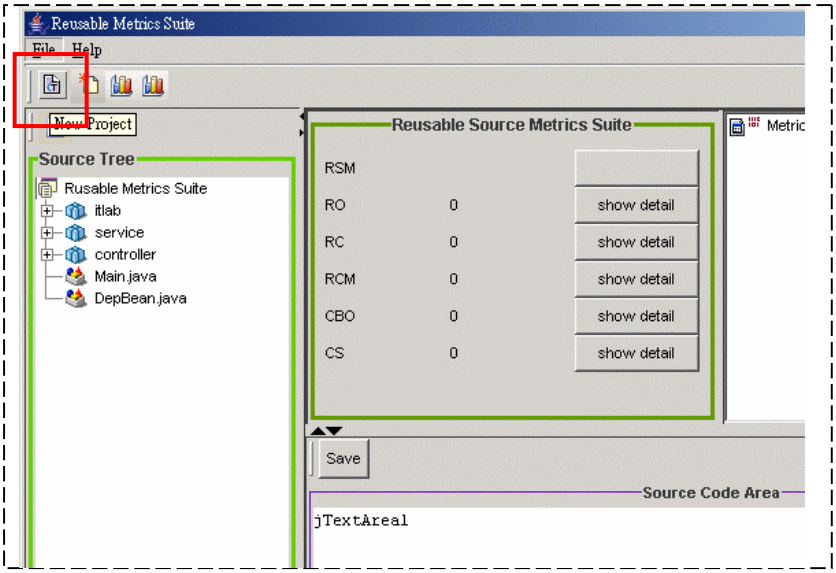


圖 22. 設定專案目錄按鈕

在系統畫面中，使用者可以從工具列去選擇“New Project”的按鈕，如圖 22 所示，觸發新增專案目錄的事件，畫面如圖 23 所選取區塊所示：

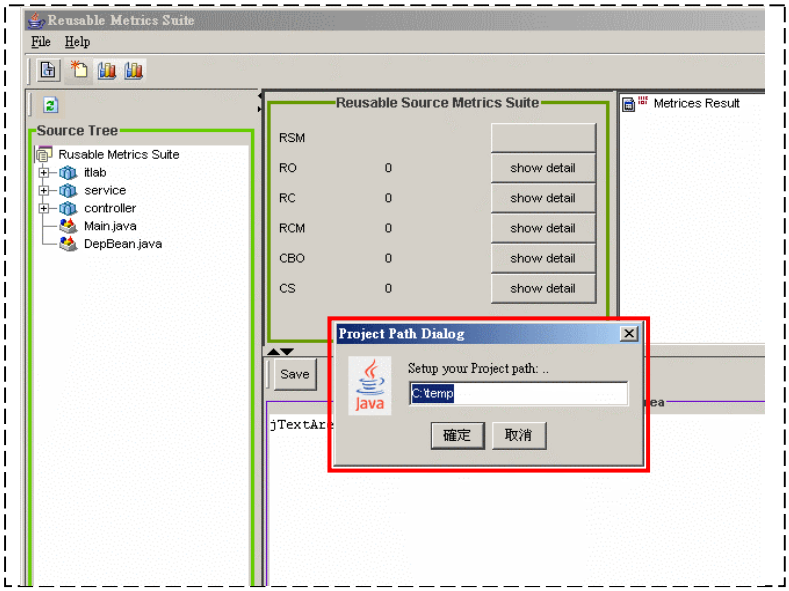


圖 23. 設定專案畫面

待使用者輸入其專案目錄位置後，即進入系統設定流程，設定流程用 State Diagram 來顯示之：

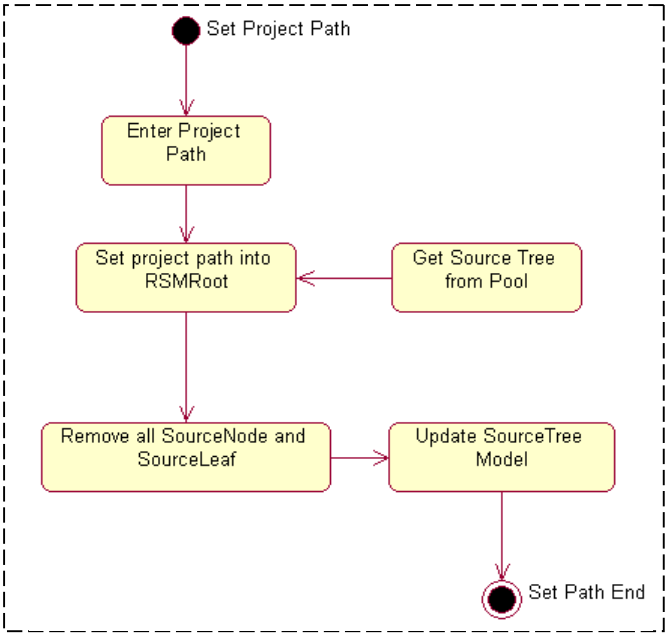


圖 24. Set Project Path State Diagram

此流程中說明了系統的運作過程，根據使用者所輸入的專案目錄，會設定到程式中樹狀結構的根節點中，而此根節點是從程式中所設計的 Instance Pool 取出樹狀的 instance，緊接著畫面上的處理重新更新過一次，清除舊有的資料並更新而結束此段流程。

5.2 Use Case: Add JavaBeans Source File into Project

使用者新增 JavaBeans 檔案進入系統，系統會以專案目錄為起始點作為新增的規則，因此，沒有位於專案目錄下的檔案無法新增進入專案：

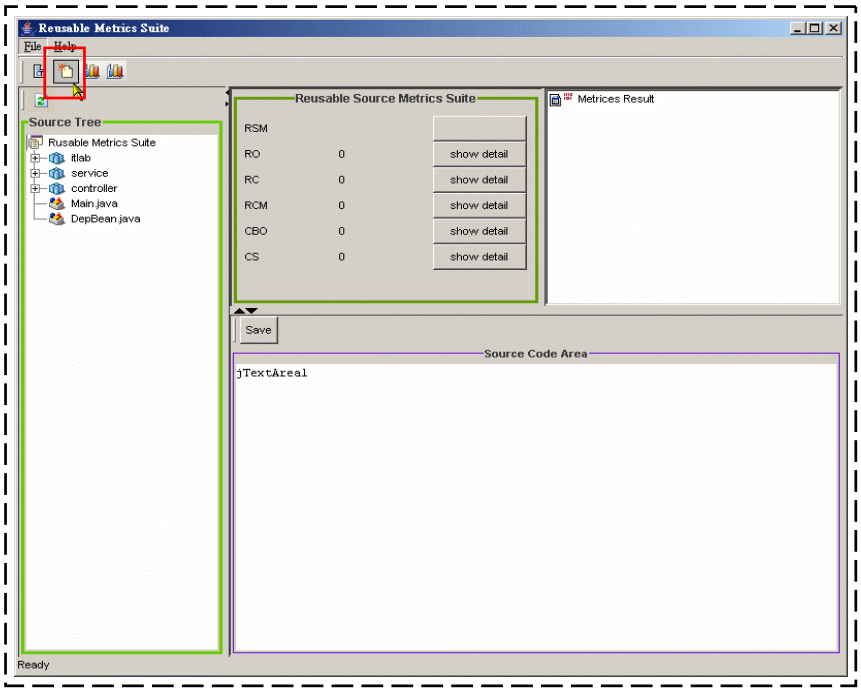


圖 25. 新增 JavaBeans 檔案進專案

使用者從圖 25 中點選“Add File..”出現選取畫面：

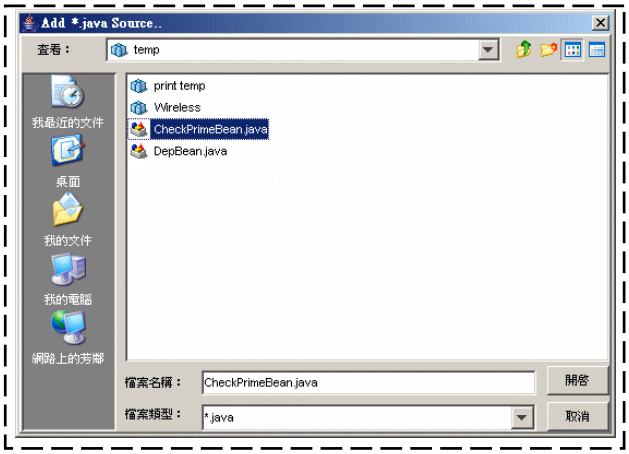


圖 26. 選取檔案畫面

待使用者選取所要新增的檔案後，會根據 JavaBeans 檔案於專案目錄中所在位置新增至樹狀結構中，如圖 27 中所選取區塊所示：

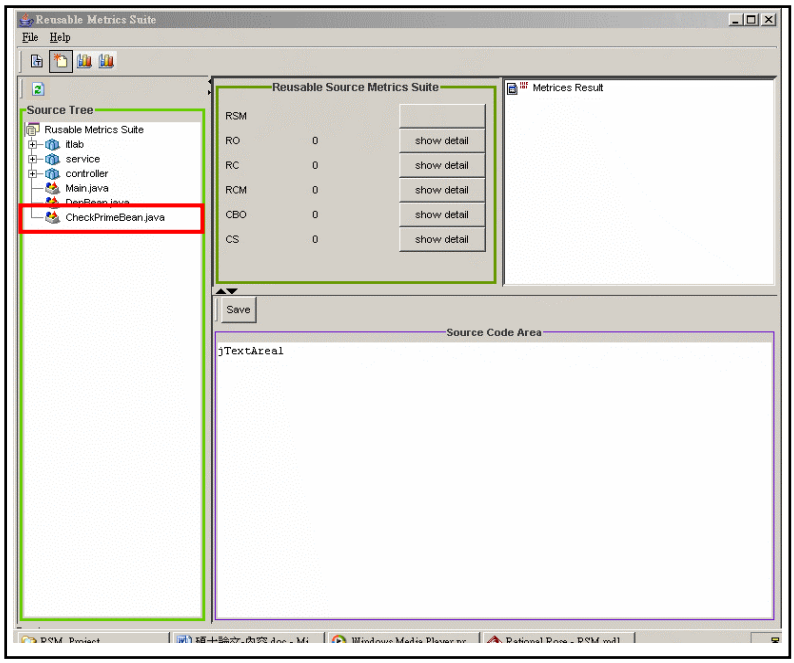


圖 27. 新增檔案完成

到此為止，此階段的 Use Case 完成，接下來將介紹到在系統運作流程過程中，State Diagram 的表示法：

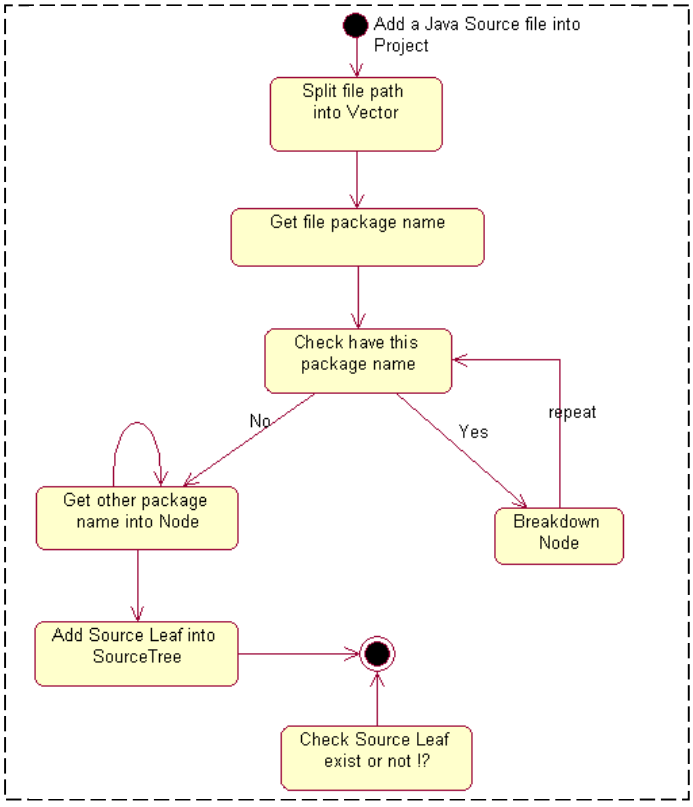


圖 28. Add File State Diagram

新增檔案的過程中，透過使用者所選取的檔案之後，必須去分析此檔案其所

在的 package 位置，因此，依照檔案的目錄去做分析，以取得其所在的 package 名稱，取得所在 package 名稱之後，開始要跟樹狀結構中的節點作檢驗，若 package 名稱存在，則往細層尋找恰當的位置，不存在則直接依照檔案的 package 建立樹狀節點，最後將節點加到樹狀結構中並做畫面更新。

5.3 Use Case: Select Source Node

使用者藉由選取樹狀結構中的節點，開始進行半自動化的度量，如圖 29 所示：

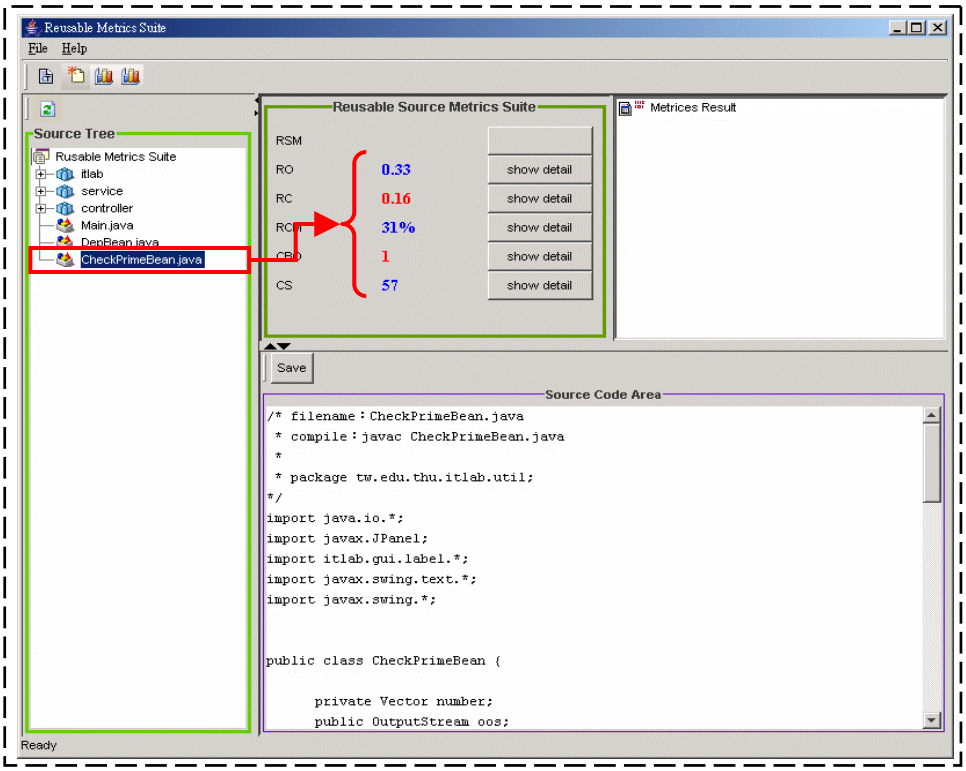


圖 29. Select Source Node

在這個階段的 Use Case 是重點所在，因此階段要進行程式碼的解析，需要去 parse 程式碼中 code 的狀態，以圖 30 做說明：

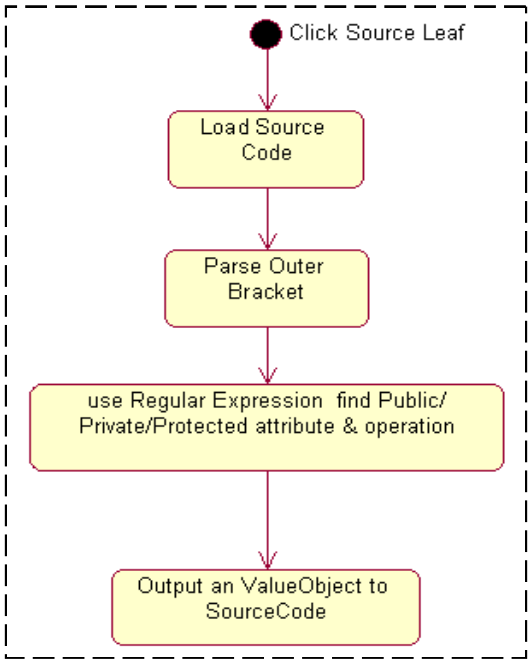


圖 30. Select Source Node State Diagram

由使用者選擇樹狀節點後，開始進入解析的流程，首先將程式碼從硬碟中載入到記憶體，緊接著對程式碼的設計規則去拆解，針對最外部的 Bracket 符號、利用正規化表示法偵測 public/private/protected 的字串去解析是 attribute 或是 operation，待這些 index 都找到之後，以 ValueObject 的型態存起來，如圖 31 所示，就是整個流程過程中所參與的物件與其之間溝通關係。

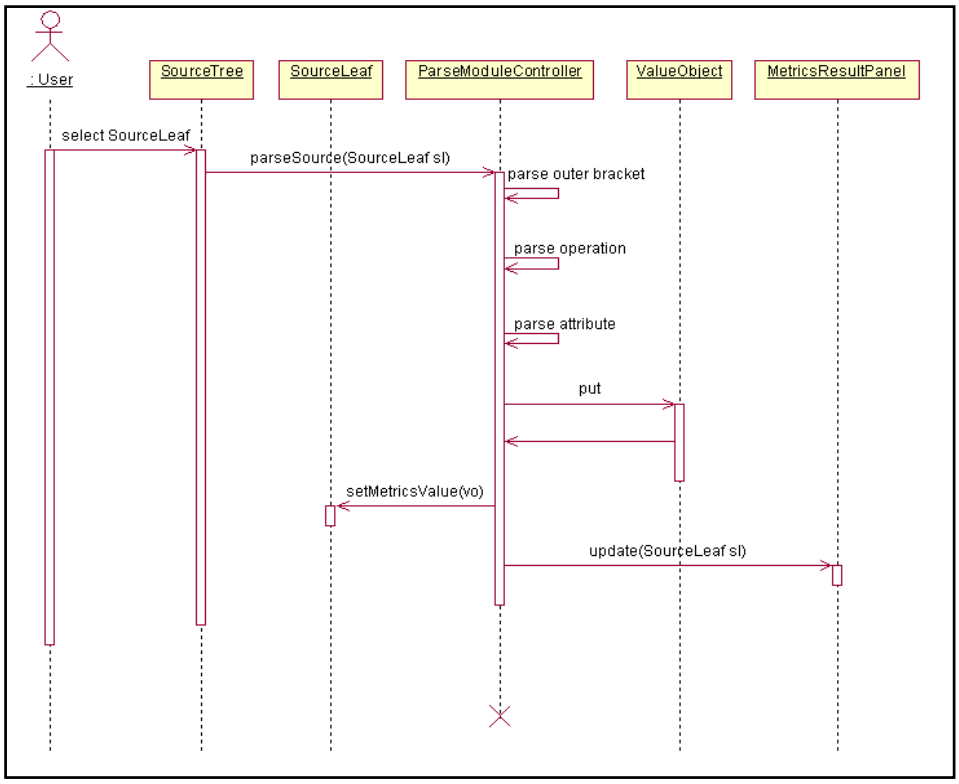


圖 31. Select Source Node Sequence Diagram

詳細的內部解析過程步驟請參照 4.1。

5.4 Use Case: Show Metrics Value

此階段 Use Case 承繼 5.3 而來，乃系統於解析程式碼之後，在取得相關資訊並進一步計算其度量指標的值，用圖 32 來說明：

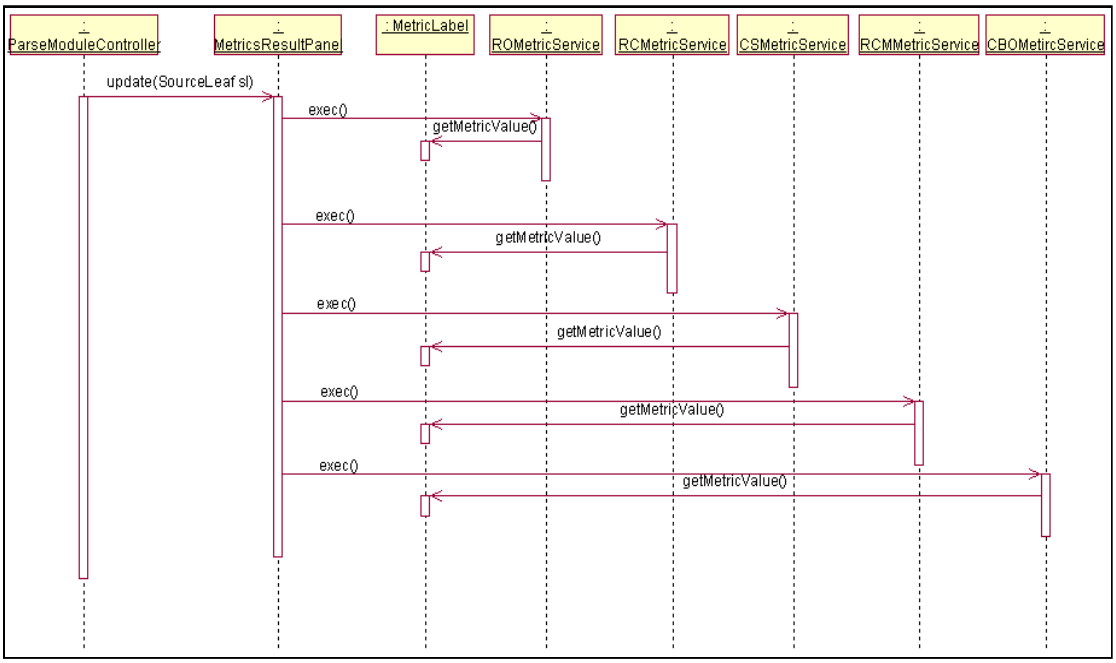


圖 32. Show Metrics Value Sequence Diagram

ParseModuleController 於系統中是負責程式碼 parse 的重要控制元件，parse 的結果最後要送交 MetricsResultPanel 畫面做呈現，設計上由五個對應的 MetricService 處理對應的工作，最後送交五個對應的 MetricLabel，如圖 33 選取區塊所示：

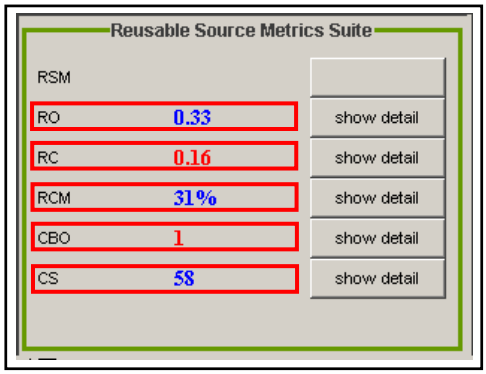


圖 33. Metrics Value 畫面配置

本文所提的五個度量指標分別有其坐落的最佳區間，設計上，上述所提的 MetricLabel 則依照最佳區間的規則設計，從 MetricService 所送達的資料，會因為不同的 MetricLabel 而有不同的回應，如圖 33 中所選取的區塊就是所提的 MetricLabel，這些所計算出來的值會顯示其中，為了讓使用者能夠更明瞭，位於區間的值會以“藍色”來顯示，否則會以“紅色”來顯示，如此一來哪些度量指

標是好是壞都可以迅速知道。

5.5 Use Case: Show detail Information and Suggestion

此階段進入到系統建議階段，在得知度量指標的優劣之後，系統提供改進的建議事項，以告知使用者要如何去改善其所度量的 JavaBeans 程式碼方可利再使用。

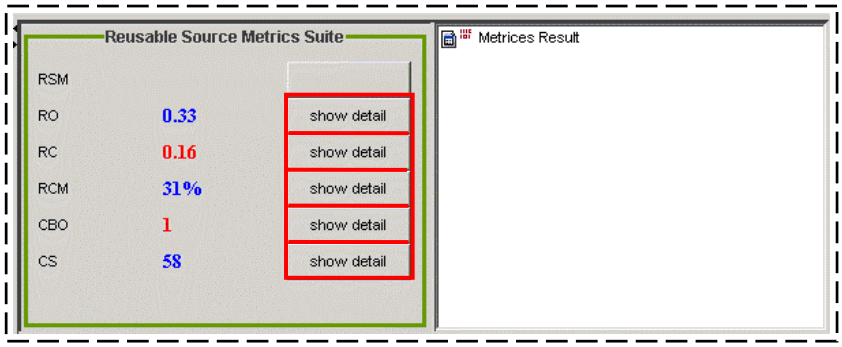


圖 34. Show detail Information screen

圖 34 中所選取的區塊，就是系統提供建議的功能按鈕，五個度量指標會有其各自的度量資訊與建議，接下來介紹所提供的建議畫面過程：

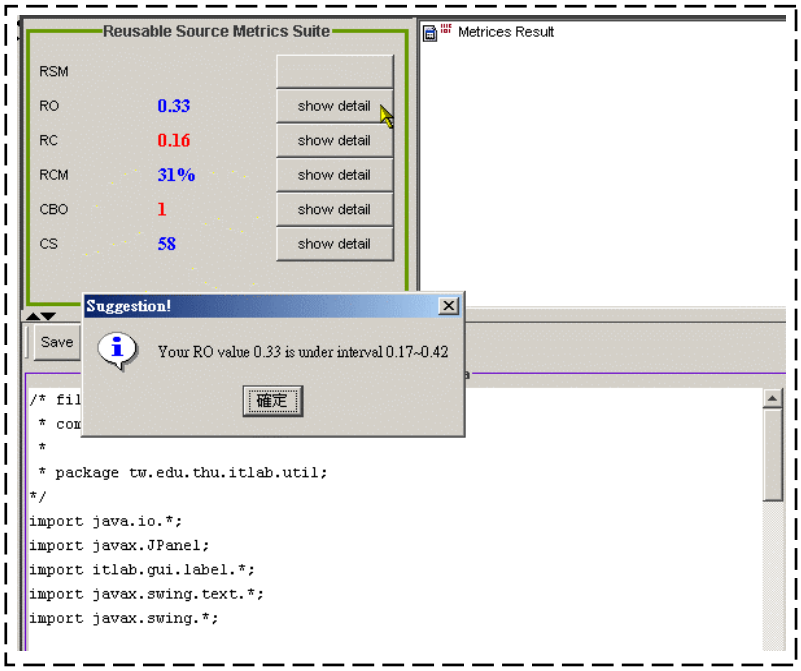


圖 35. RO Suggestion

圖 35 中，滑鼠敲擊按鈕之後，因 RO 的值位於優良區間，所以系統給的提

示乃表示 RO 的值優良，並不需要特地的修改，緊接著，在右側的空白處會出現細部的度量因子：

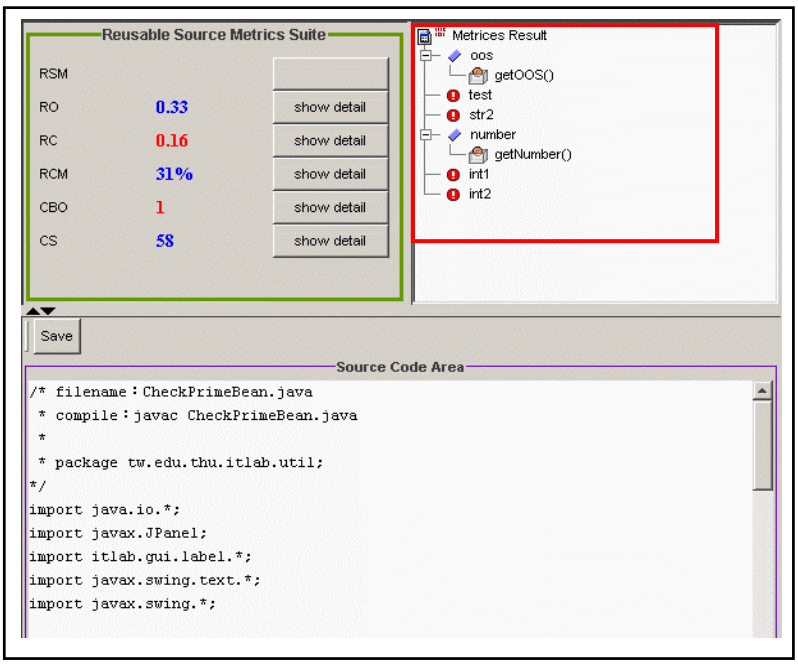


圖 36. RO Detail

如圖 36 中所選起區塊所示，會以一樹狀結構呈現相關訊息，在此 RO 的計算過程是計算在 JavaBeans 中 getter method 與 attribute 的數目比例，因此，將程式碼中所有 attribute 取出，而若存在對應的 getter method，則將 method name 與 attribute 連結，為了讓使用者清楚的知道目前情況如何，特別利用不同的圖示來表示之：

：attribute 圖示，但存有對應的 getter method。

：attribute 圖示，但不存在對應的 getter method。

：getter method 圖示。

透過使用者去選取 getter method，會觸發 Source Code Area 將其所在位置顯示於畫面中：

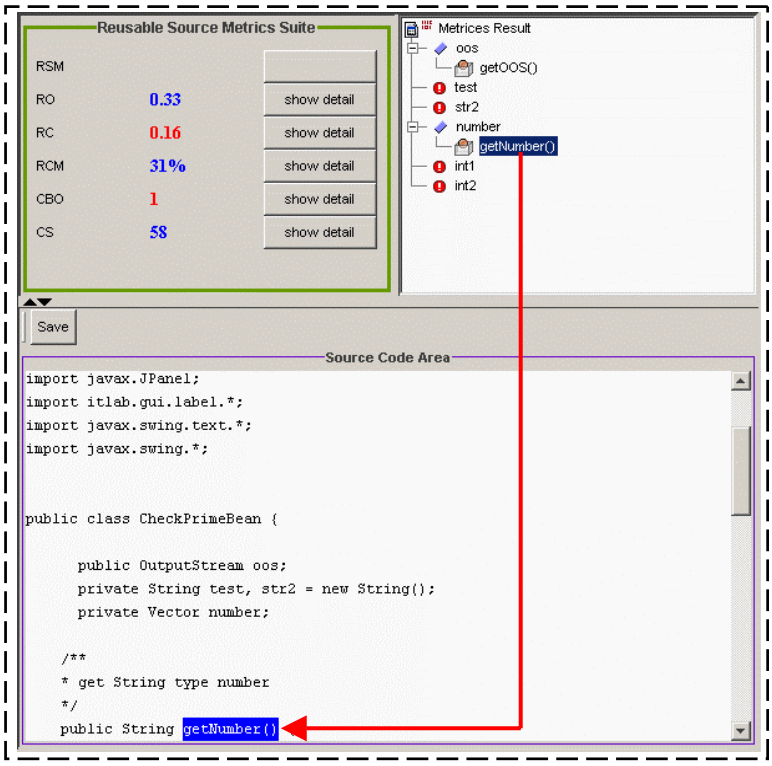


圖 37. RO Detail Modify

以上是 RO Metric 的建議修改部分，接下來介紹 RC，在例子中，RC 的值以紅色顯示，所以可以知道 RC 的值並未落於優良區間，所以系統提示會如圖 38 所示：

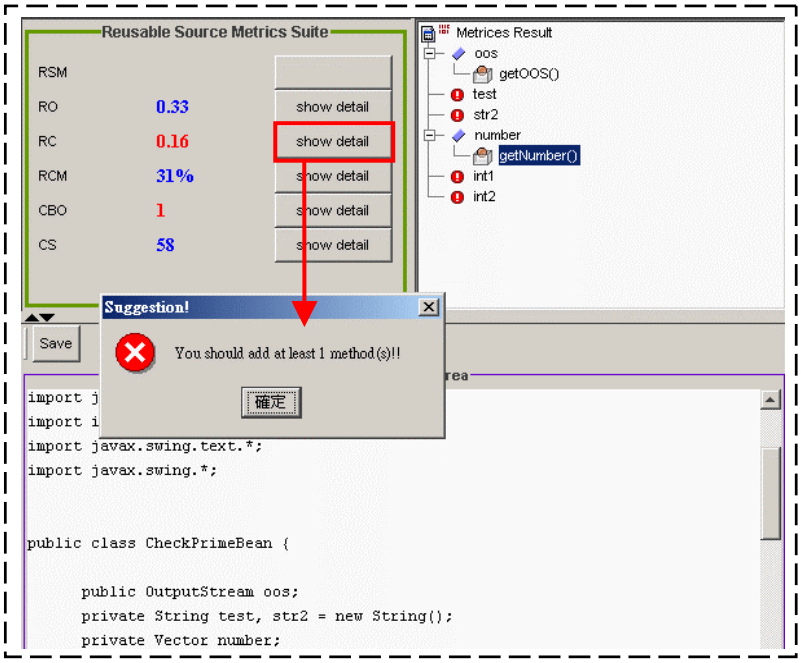


圖 38. RC Suggestion

經系統計算後，發現 RC 的值需要增進 method 數，如此 RC 才會落於優良

區間，因此系統會先警告使用者，如圖 38 所示，接著才會顯示更詳細的資料：

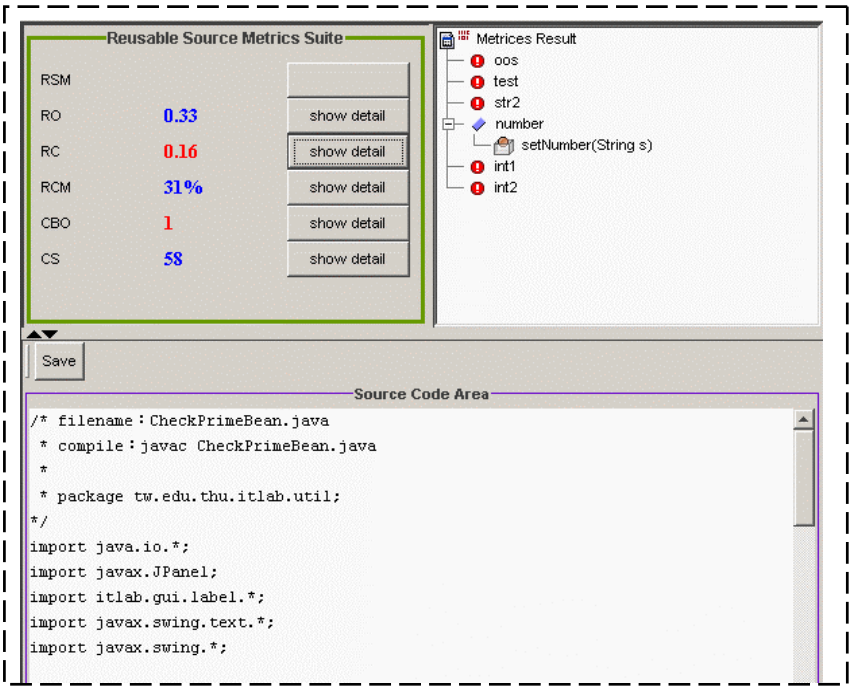


圖 39. RC Detail

RC 所顯示的資訊和 RO 有絕大多數雷同，系統所使用的圖案是一樣的，不同的地方在於 RC 所取得的 method 為以 setter name 為主。

而 RCM 的度量指標一樣會透過系統所計算並提出適當的建議，

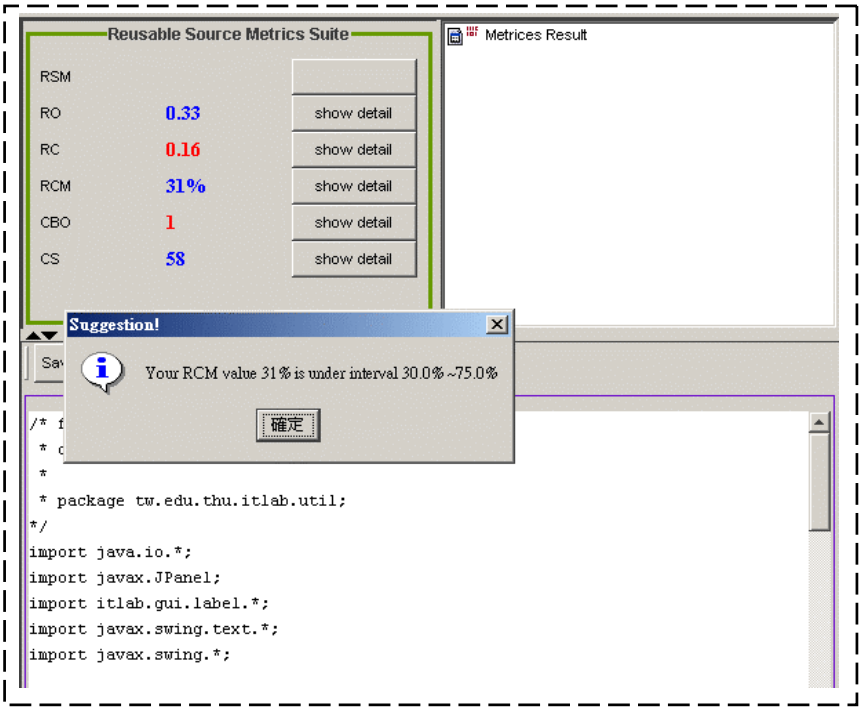


圖 40. RCM Suggestion

設計上是依照註解區塊來作依據，因此在樹狀結構的顯示上，則按照其取得的順序呈現並為其編號：

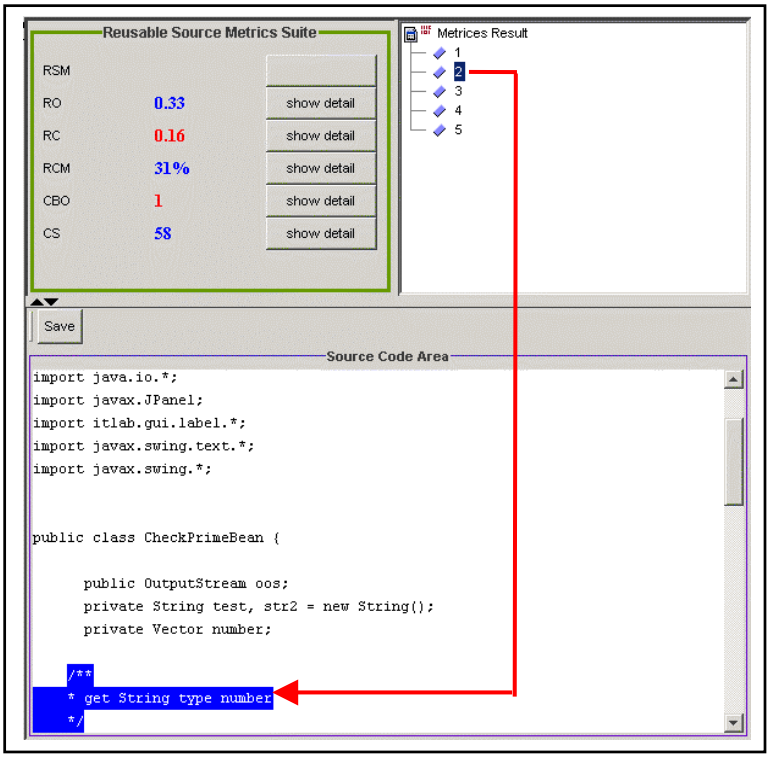


圖 41. RCM Detail

使用者可以透過選取樹狀結構而知道對應的註解區塊位置，透過這種方式可以讓使用者馬上知道註解內容，且知道註解區塊的多寡。

CBO 則是顯示目前 JavaBeans 程式碼中有用到外部物件的數目，當其顏色為紅色表有外部物件存在，點選按鈕系統則會告知應該減少這些外部物件的使用：

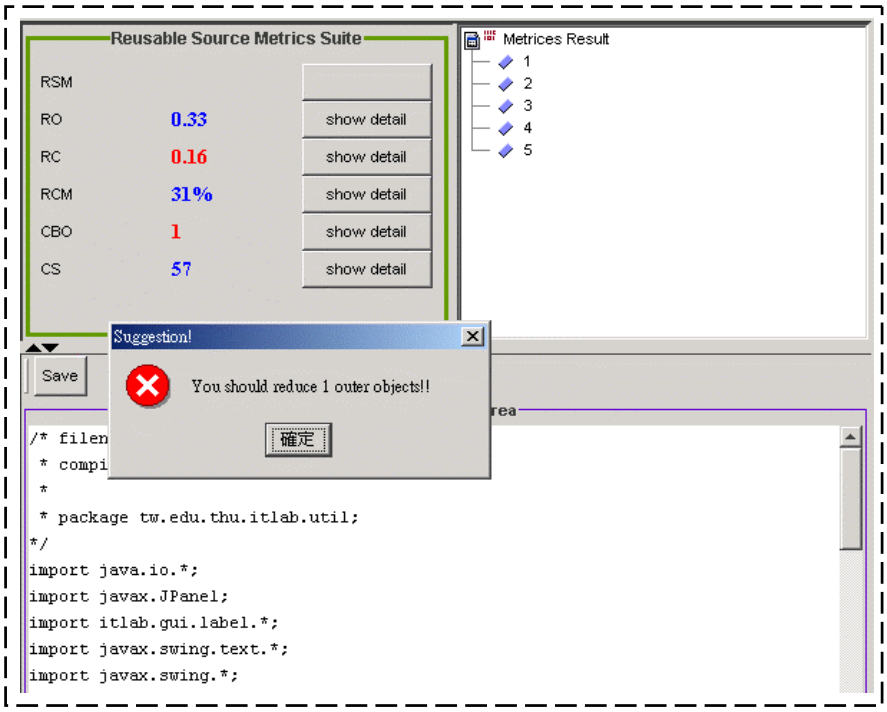


圖 42. CS Suggestion

緊接著就是在樹狀結構中顯示所有 attribute 的數目：

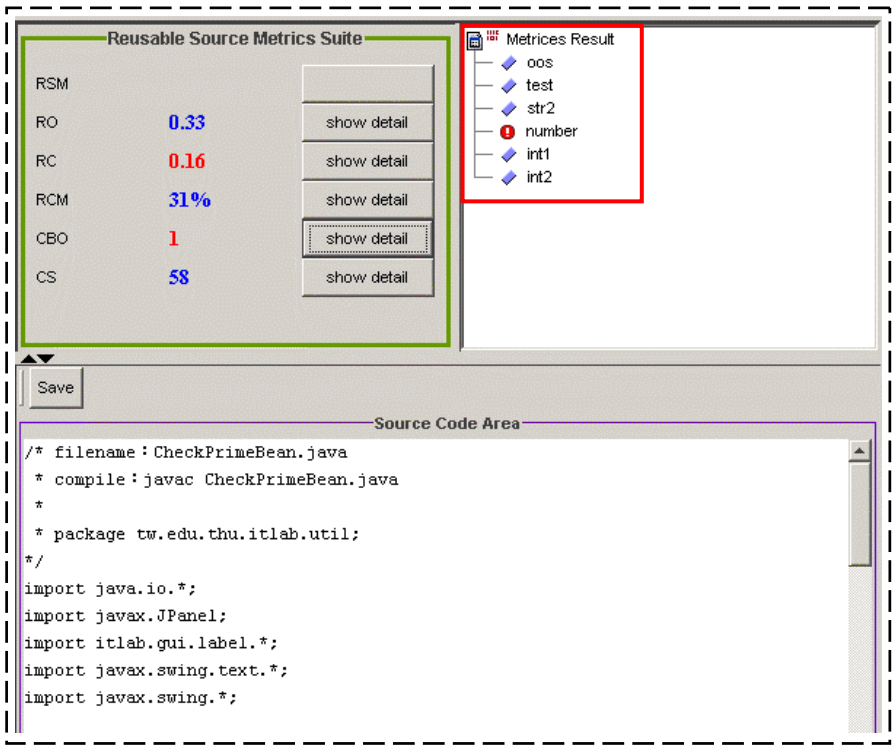


圖 43. CS Detail

如圖 43 中所選取區塊所示，樹狀結構中所顯示的資料即是此 JavaBeans 中所有用到的 attribute 清單，設計上用圖示來區分是否為外部物件，以便使用者快

速知道外部物件的引用是誰造成的。

：一般物件 attribute 的圖示。

：外部物件 attribute 的圖示。

待使用者選取此外部物件，Code Area 則會顯示其位置所在，如圖 44 所示：

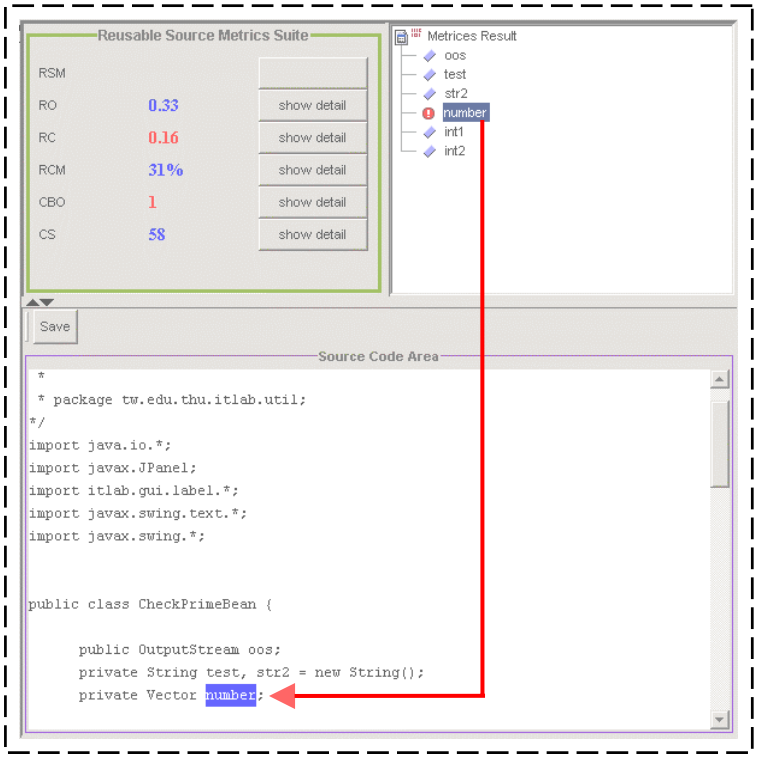


圖 44. CBO Detail Modify

如此一來，使用者可以方便修改外部物件的引用。

最後 CS 的度量指標則只是簡單告知目前此 JavaBeans 程式碼行數多寡

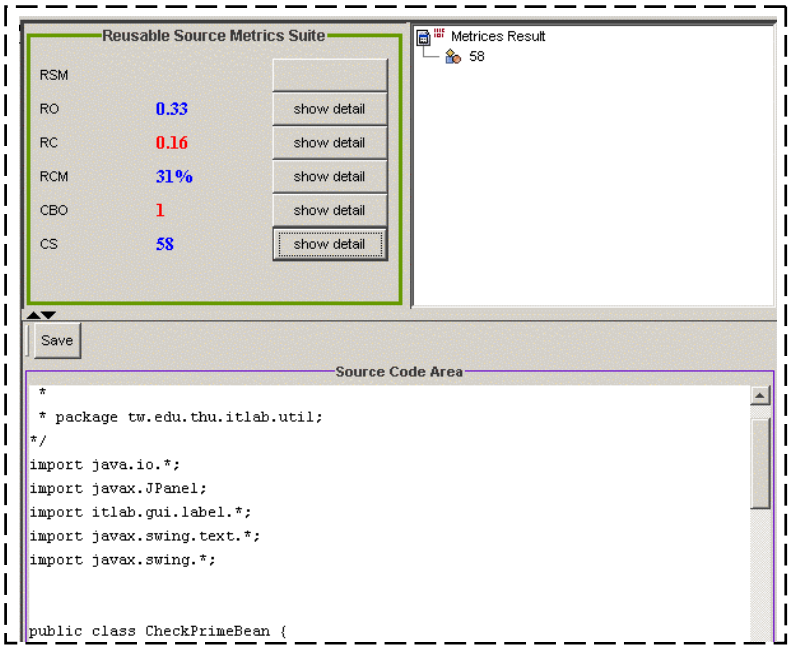


圖 45. CS Detail

第 6 章　案例研究

在本章中將以一書目管理系統來作為本文研究案例的對象。

在案例中，此書目管理系統主要目的乃設計一 Web 介面的管理系統，透過登入的機制，分別會有不同的功能機制，透過管理者的建立書目檔案，系統能夠提供搜尋功能、借還書的機制而達到管理書目的需求。

初步分析此書目管理系統中所會應用到的技術，分別包含網頁設計上的技術，譬如 Client-Server 的網路技術，因透過 Web 介面以方便使用者操作，所以基本的架構會以 2-tier 以上的機制去建置；在來，面對書目資料的建立，牽涉到資料庫設計方面的技術，舉凡資料庫的連結、資料庫與 Server 之間的溝通，都將是此管理系統中所會面臨到的技術問題，簡言之可以用圖 46 表示之：

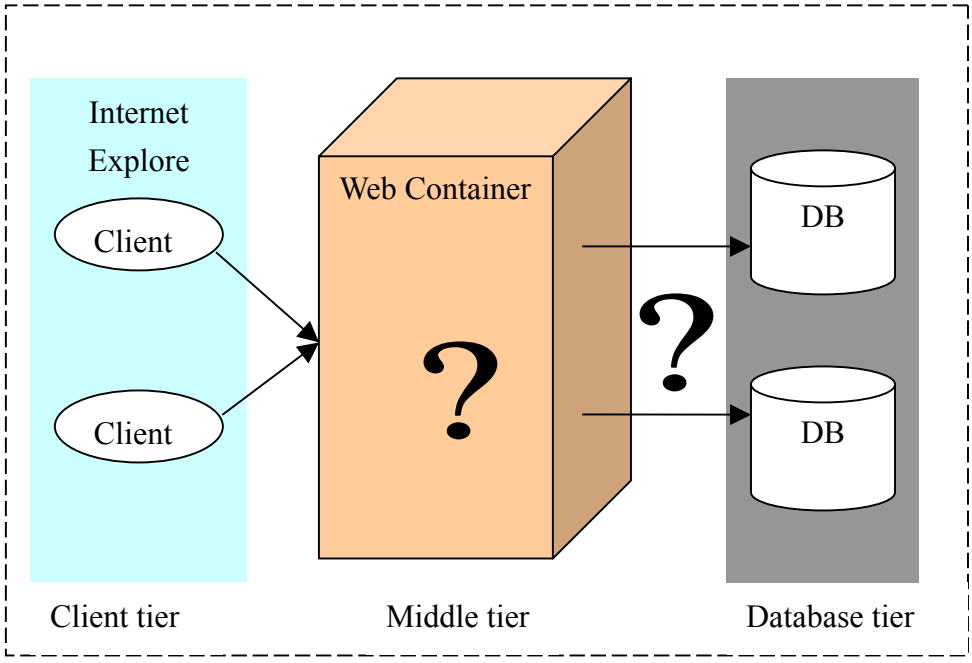


圖 46. 案例系統層次示意圖

從案例系統層次示意圖可以知道，其中設計上的主要問題存在於 Web Container 與 Database 兩部分之間的設計，此案例開發上採用 Java 為主要的開發程式語言，直至目前為止，雖然可以將整個 tier 的層次切割成如此，其中仍然隱

含著某些技術問題，譬如在 Middle tier 之中的設計方式，是否有其設計上的盲點存在[18][19]，此亦為問題所在，因為網路程式語言的設計所要考慮的因素並不像單機上的操作單純，分散式的程式設計是一關鍵所在，另外，Web Container [25]與 Database 之間的連結亦是所要克服的重點之一。

簡單分析此案例系統的問題點之後，我們試圖想要從學長之前所設計的 Office Organizer [17]中去習得這領域的相關知識，Office Organizer 是學長姐們所設計的一套 Web Application，針對中小企業內部的資訊管理，所涵蓋的功能舉凡人事管理、財產管理、文件檔案管理…等，都是在 Office Organizer 中所提供的功能之一，基於 Office Organizer 亦是以 Java 為基礎語言，在本案例中即以此系統為所實驗的對象。

在取得系統的原始程式碼之後，試圖想要從檔案命名規則去知道當初的設計風格，簡單做一整理如所示：

表格 4. 分配比例表

類型(角色)	分配比例
JavaBeans	11%
Database Data Bean	23%
Special Function	47%
Visitor Class	19%

在表格 4 中，將近有 11%的 JavaBeans Class 就是在此案例中所要參考的對象之一，其中包含了於 Web Container 中的設計方式與資料庫之間的連結，而約有 23%的 Database Data Bean，這些在 Office Organizer 中所扮演的角色則是為了對應資料庫中 table 的資料所設計，也就是說，在資料庫中所取出來的 tuple 資料都會分別包裝成 Database Data Bean 的物件型態，如此一來方便系統對資料的處理亦降低資料存取的難度，對設計上而言有其共同的設計規則。

其中佔有將近 47%的 Class 是系統中所需要的特殊功能，在此與所要實驗的

對象不甚符合，因此這些 Class 不列入實驗過程中，最後 19%的 Class 設計是屬於 Visitor 類型，主要是因為當初設計上採用了 Design Pattern 中的 Visitor Pattern，而此類型的 Class 扮演不同需求的 Visitor 角色，雖然非本案例研究中的實驗對象，但 Pattern 的設計概念卻是可作為參考。

因此，以這些符合我們需求的 JavaBeans 元件開始進行程式碼再使用的實驗，實驗結果的統計圖分述如下：

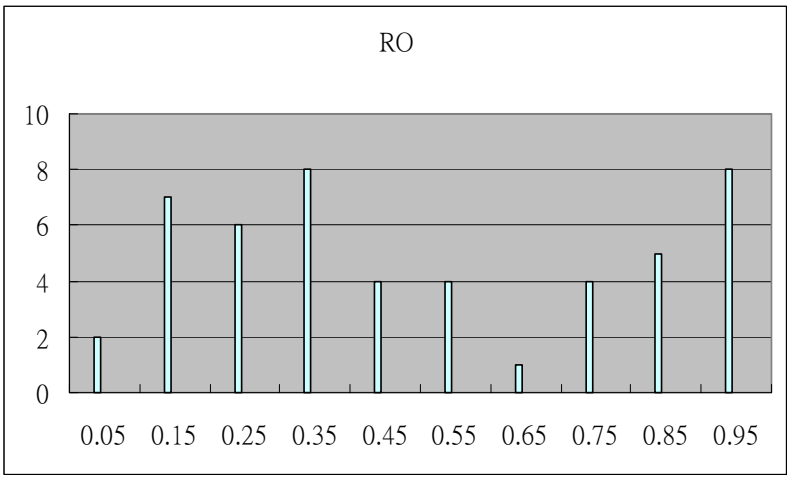


圖 47. RO Histogram

實驗的統計結果，RO 的分布圖如圖 47 所示，而 RC 的分布圖如圖 48 所示，從圖中可以發現有多數的比例分別位於 RO 與 RC 的優良區間，顯示出所驗證的資料對於設計上仍有多數的設計有依循 JavaBeans 的設計規格，但從 RO 的分布圖中可以發現有少數的資料顯示其 RO 的值是偏高，位於 0.75 之上，對這些數據作深入的探討發現，當初參與設計的人員不同，而導致在設計的風格不依，且檔案命名上的不一致性而造成如此的結果。

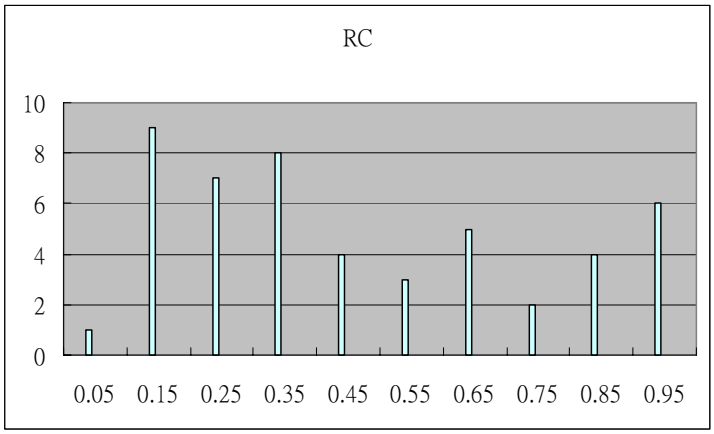


圖 48. RC Histogram

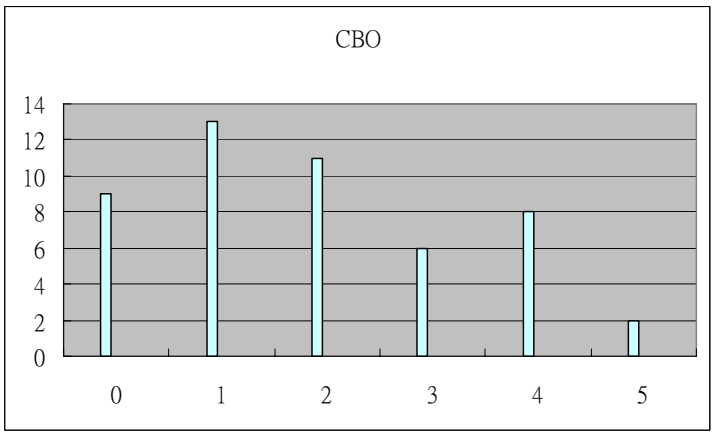


圖 49. CBO Histogram

從圖 49 可以發現 CBO 的統計結果有絕大多數的程式都會引用外部物件，經過深入的分析之後發現在系統的設計上都會依循如圖 50 的架構在設計，也就是說這邊的 JavaBeans 在設計上，首先會和 Connection Pool 有關聯，需從其中取出 Connection 的 instance，取出之後方可與資料庫連結，在來，為了要封裝從資料庫取出來的資料，也會設計一對應的 Data Bean 於其中以便存取處理，基於這兩種原因與設計師的設計風格不依，造就了驗證的結果是如此，不過，這種情況也只是少數特例，並不會嚴重影響程式碼再使用的過程。

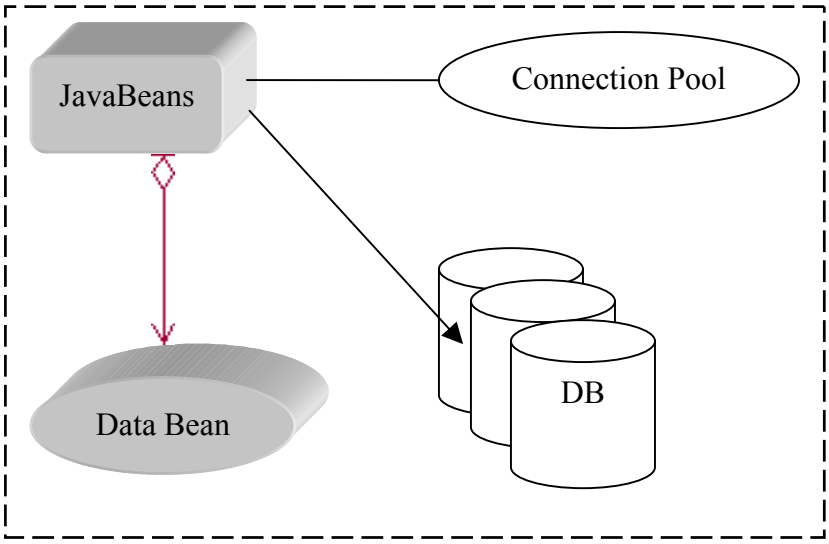


圖 50. 連結資料庫示意圖

而程式註解的部分猶如圖 51 所示，雖然位於優良區間 35%以上的比率並沒有很高，但是由於設計上的有依循圖 50 的架構在設計，因此，即使註解的比率不高，依然可以從簡易的註解中得知程式碼的設計用意，不過，透過註解可以更容易知道程式設計的理念與目的。

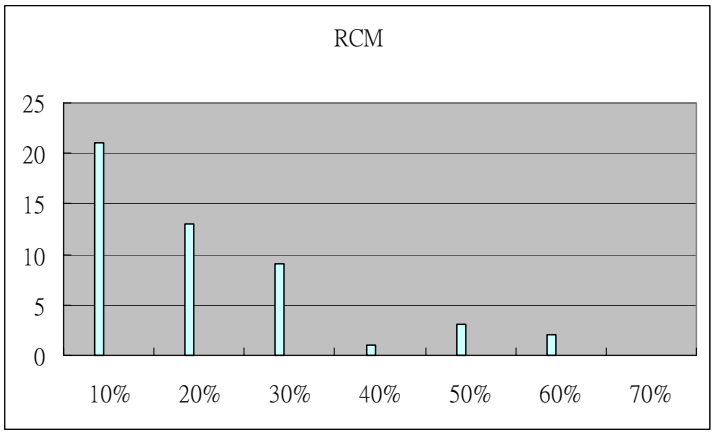


圖 51. RCM Histogram

程式碼大小的部分，綜合整理發現設計上多數都能控制在優良區間內而沒有過多的情況發生，由此看來，程式設計師的習性上並不會製造太多的 code 而加深程式的複雜度，反而都是盡量將功能切細，讓每個元件的單位越小越好，程式越單純越好。

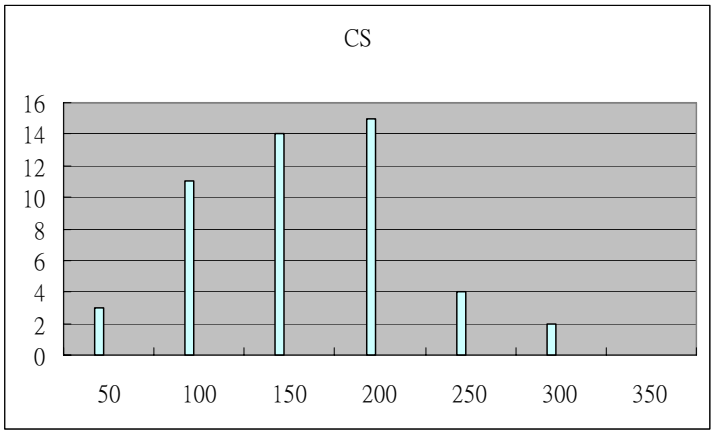


圖 52. CS Histogram

實驗最後在追蹤程式碼發現，整個系統的運作機制有如圖 53 所示，在 Server 部分的運作，包含了三部份，三部份有各自不同的用意：

- Web Container：在這部份中包含 Servlets、JSP、HTML、與 XML，理念上 JSP 與 HTML 都只是做單純的畫面呈現，也就是 MVC Pattern 中所謂的“View”，其中盡量不包含複雜的程式運作邏輯，而 Servlets 的定位則以“Controller”為主，負責控制整個網頁程式運作流程的導向，而 XML 的部分則作為存放資料的代表。
- EJB Container：負責存放分散式 EJB 物件的容器，主要所要處理的事件來源除了 Client 端之外，亦包含從 Web Container 過來的事件，基於 EJB 設計的架構，減輕網路間的訊息傳遞次數，期望能有效減低網路等待時間而增進效能的設計方式。
- DAO(Data Access Object) / JavaBeans：此部份元件的定位在存取資料庫資料的元件，設計這些元件上，針對資料庫中存在的 Table 或者 Schema 設計，每個 Table 會有一對應的 DAO 或者 JavaBeans 的元件來負責存取，因此，從 Web Container 或者是 EJB Container 中的元件需要存取資料庫中的資料時，即透過此部份的元件從後端資料庫中取出資料而交由前端，如此一來又將功能切分更細。

直到目前為止，約略可以清楚 eOffice 整個系統的運作機制就如同圖 53 的

運作機制，所以案例系統從圖 46 演變為如圖 54 所示。

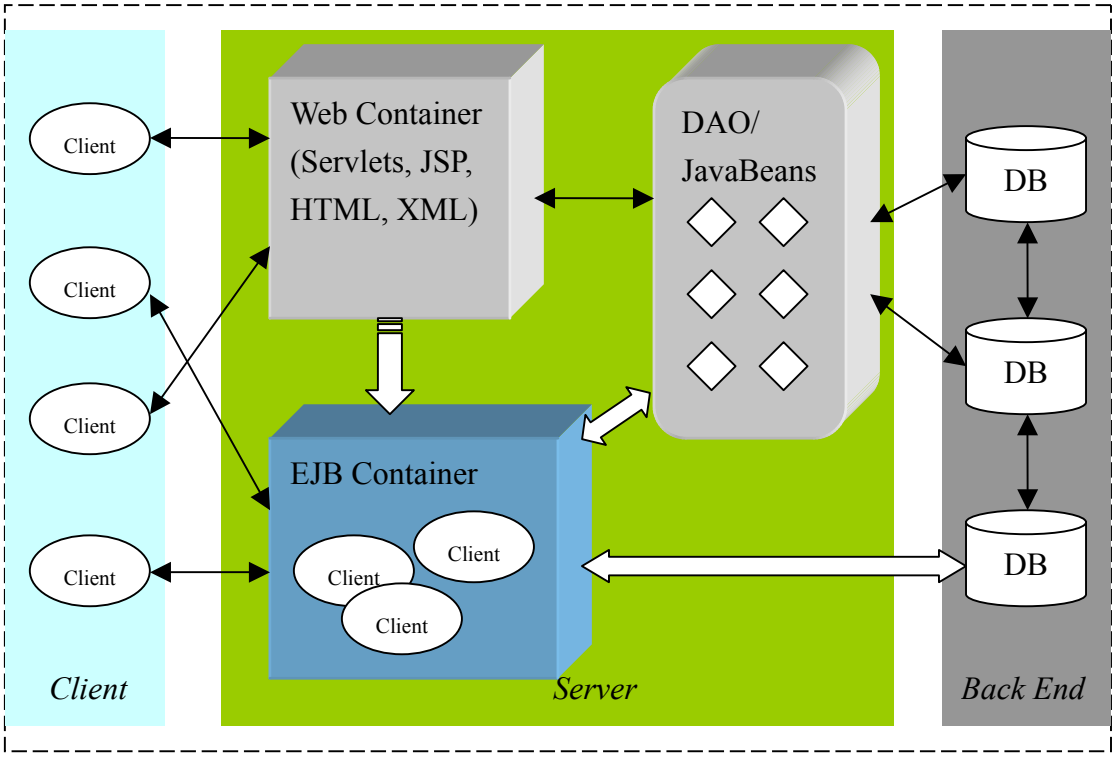


圖 53. eOffice 運作機制

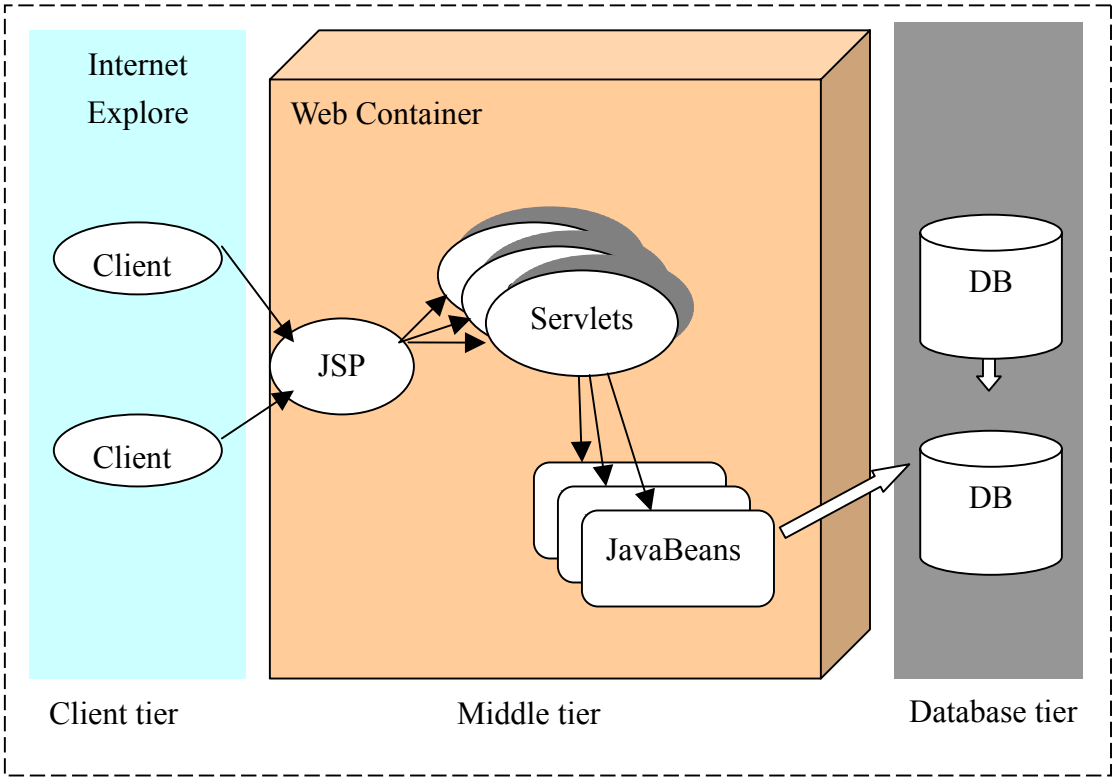


圖 54. 案例系統運作架構

如同圖 54 所示，整過架構分為三個 tier，分別為 Client tier、Middle tier、

Database tier，Client tier 所要說明的就是來自網際網路上要求，在這個 tier 上並無所謂的技術問題，只有單純的 Brower 瀏覽介面而已，和系統之間的運作不無太密切的關聯；而在 Middle tier 內就是核心技術所在部分，在這部分主要化分三個主要的角色：

- JSP：依照 MVC Model 2 [20][21]的設計方式，JSP 的定位就是設計做資料的呈現，也就是 View 的功能，最多在 JSP 中嵌入 Tag 標籤的功能，不過不管如何，JSP 檔案中的內容不會有太多的程式邏輯於其中。
- Servlets：定位在 Controller 的角色，負責整個 Web 運作流程的控制角色，負責參數的傳遞與流程的控制，這種設計方式定位 Servlets 的意義不至於與 JSP 有腳色衝突[24]。
- JavaBeans：如同 DAO(Data Access Object)的設計，負責存取後端資料庫的資料，資料庫中每一 Table 有一專門的 JavaBeans 負責存取，在 Web 流程中只要對這些 JavaBeans 做處理，而不用去考慮到如何對資料庫下語法處理，使程式設計師專注於目前的流程運作即可。

最後就是 Database tier 這一層，有關資料庫的 Schema 的規劃都落於這一部份，在案例系統中不強調這一部份，視為一單純的資料庫。

這邊我們列舉“新增書目”與“搜尋書目”兩功能為例說明，如圖 55 所示為兩功能的 Collaboration Diagram：

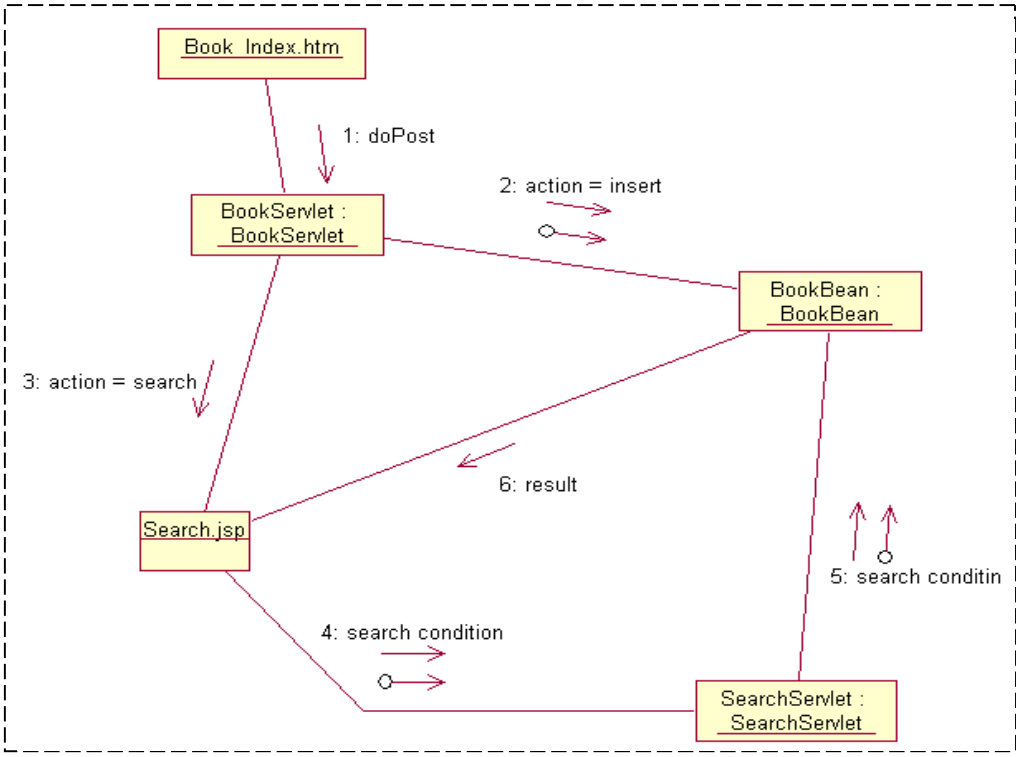


圖 55. Insert and Search Book, “Collaboration Diagram”

由圖 55 中可以發現，在案例系統的設計上是從使用者端的 HTML/JSP 接受要求，任何的 request 會送達 Servlets 做轉送，而對資料庫的存取，則交由 JavaBeans 負責存取，如此一來，每個腳色的定位即劃分清楚。

因此所需要再使用程式碼的部分就是 JavaBeans 的部分，以圖 55 中的 BookBean 為例，要設計 BookBean 的內部結構，和 eOffice 所用到的 JavaBeans 有雷同之處，因此即再使用 eOffice 中的 JavaBeans 程式碼作為實驗的案例。

任意選取某一 JavaBeans 作為實驗例子，以 eoffice.company.department 下的 DepTitleObject 為例，其度量值如表格 5 所示：

表格 5. DepTitleObject 度量值

Metric Name	Value
RO	0.33
RC	0.16
RCM	33%
CBO	1

CS	168

如同表格 5 中所示，RO、RCM 與 CS 的值都落於優良區間，顯示 RSM 的值良好，可以作為實驗的對象，細部觀察程式碼發現其中所設計的架構如圖 56：

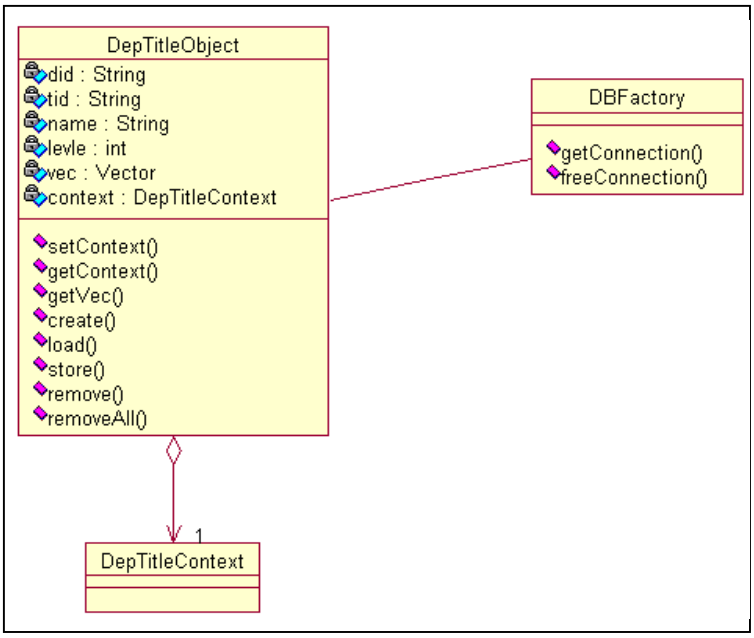


圖 56. DepTitleObject Structure

其設計方式中，需藉由 DBFactory 一物件取得資料庫的連結，在取得連接之後，即可從資料庫中取出資料，並將資料庫中的資料封裝成 DepTitleContext 的型態以便處理；亦因為這種設計的方式，使得度量出來的結果 CBO 的值為 1，就是 DepTitleContext 所造成的，其他 RO 與 RC 的值細部查看，發現 Context 的取得存有一對應的 getter 與 setter，另外尚有一 getVec() method，由此更加深了 DepTitleContext 其存在的意義就是為了封裝資料庫中的資料而設計，而 getVec() 則是為了存放多筆 Context 資料而設計；所以在案例系統中，BookBean 所要再使用程式碼的部分即有參考對象，如圖 57 所示就是所設計的 BookBean Class Diagram：

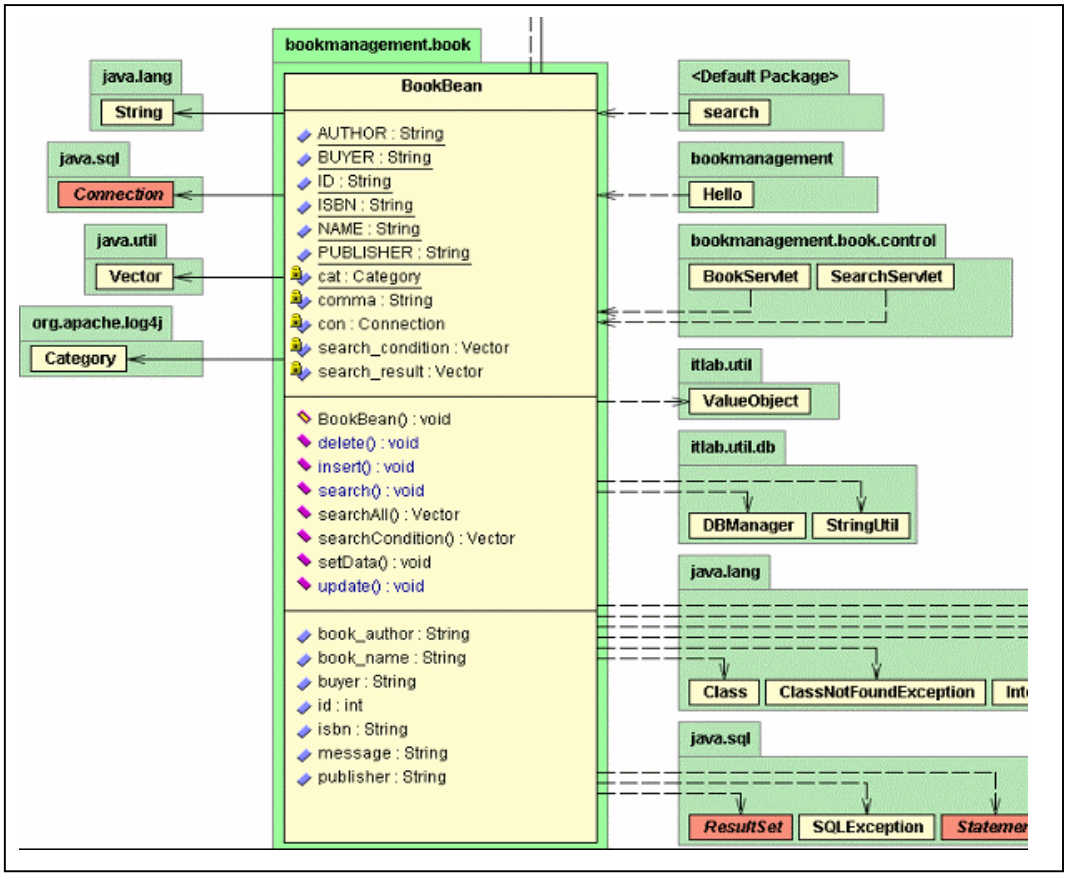


圖 57. BookBean Class Diagram

在 BookBean 中的設計，包含基本的資料處理 method，如 insert、delete、search、update 等四大主要功能，這四功能分別對應 DepTitleObject 中的 create、remove、load、store，因此在程式碼的再使用上，可說是完全參考其設計方式，目前只是做 interface 的介紹，接下來就是細部程式碼的撰寫部分，在整個實驗的過程中，所接觸到的幾個問題分述如下：

- 資料庫連線架構：為了要再使用就有的程式碼，面臨到原先所設計的連線架構，原先對資料庫的連線架構早有其設計的規則，在案例系統中勢必也要將這連線架構也納入，否則連線的機制無法建立即無法進一步的案例系統設計。
- 字串編碼：案例系統中所採用的資料庫最先為 Open Source 中的 MySQL，後來亦曾經換過 SQL2000，其中最大的問題就是字串的編碼問題，從 Client 端的 Browser 送至 Web Container 的資料，在送至 JavaBeans 最後與資料庫做存取，其中 Web Container 部分字串就必須因

為編碼的問題先處理過一次，接著就是與資料庫做存取時的編碼需要再做一次，而 MySQL 與 SQL2000 對編碼的接受度又些許不同，所以在案例系統的實做上對於編碼這一部份的知識學到不少。

- Web 應用程式的設計：網頁應用程式在設計上有其天生的限制，譬如說薄弱的(stateless)連線機制就是其基本限制，因此網頁程式在設計上也因此而必須考量這個問題，舉凡購物車、線上交易機制都將面臨到這個問題，另一問題就是網路連線的問題，過多的網路連線，除了造成網路的擁擠之外，伺服器亦可能因此而無法正常運作，如何減低頻繁的連線也是網頁程式在設計上要考量的；譬如上述所提的購物車，為了要減低頻繁的網路連線，在後端以 JavaBeans、EJB 來設計處理商業邏輯的流程，即可有效降低過多的網路連線，這也是分散式物件應用於網頁程式設計上的一種。
- Design Pattern 的設計技巧：在追蹤程式碼的期間，發現在設計上有遵循許多 Design Pattern 的設計理念，其中以應用 Visitor Pattern 最為頻繁，因此案例系統中的帳號權限部分參考 Role-base Access Control 的機制設計，並引用 Visitor Pattern 來應用在權限追蹤。

因此，在整個 JavaBeans 程式碼再使用的過程中，除了一些程式的 Domain Know How 的設計之外，其所帶來的優點並非只於此，甚至是如上所提的，連線機制的架構、設計樣版的使用時機與技巧，這些都是當初所沒有想到的。

對於所實驗的案例系統做一花費時間/優缺點的對照表，針對有再使用程式碼與重新開發做一整理比較如表格 6 所示：

表格 6. 花費時間/優缺點比照表

	再使用程式碼	重新開發
字串編碼	0.5/hr	2.5/hr (包含從網路上與書本上取得資料的時間)

Visitor Pattern 設計	1/hr	3/hr (包含從 Design Pattern, Gof 與 Pattern in Java 書中取得資料，並透過範例程式的演練所花費的時間)
連線池設計架構	有此設計、 2.5/hr	無此設計、 單存連線機制：1/hr
MVC 架構的設計	依照 MVC Model 2 的設計理念進行。	無此規則，程式邏輯的設計存在於 JSP、Servlets 與 JavaBeans 中。

從表格 6 可以發現，透過所提的再使用度量方法去對程式碼度量，並進一步對這些程式碼做再使用所帶來的好處，不僅縮短了程式開發的時間，另一方面優良的舊有系統設計架構甚至是可作為一延伸改良；而從時間上的考量，對於程式碼再使用的度量確實可作為參考依據。

第 7 章　結論

軟體的度量一直是軟體品質的討論話題，如何去做度量的標準，常常是因人而異，現在在研究中所針對的即是 JavaBeans 程式碼再使用的度量，其度量的基本關鍵仍需 JavaBeans 程式的撰寫風格必須遵守 Sun 所制定的 JavaBeans Specification 1.1 規格，否則其度量出來的結果仍無其意義所在，也因為如此，本文才會選擇標準的 Specification 作為度量的規則(Rule)，本文依照 FCM 的推演理論分析程式碼再使用的過程，以確保符合再使用的基本理念，如此一來，度量指標才有符合再使用的觀點，度量出來的結果也才有其意義價值，透過本文所提的方法，才能有效的度量 JavaBeans 程式碼的再使用並得知如何改善的方法與建議。

以現今整個軟體開發的流程中，要貫徹度量的目的仍有許多領域可以研究，目前，UML [14]文件的製作相當廣泛，UML Diagram 文件複雜度的度量也是未來可研究的議題之一，譬如從需求、分析階段所設計出來的 Use Case Diagram、Sequence Diagram，度量系統的複雜度是否與這些 Artifact 有關，又或者是設計階段出來的 Class Diagram，能否從這些 Artifact 中去度量整個系統的相關資訊，都有其研究所在，另外，以 JavaBeans 而言，可以繼續針對 Bean 元件的架構著手，度量 JavaBeans 元件間架構的資訊，舉凡度量 Design Pattern 的應用或者是 MVC 架構的設計，這些都可以列為未來的研究議題之一。

參考文獻

[1] Cavano J., McCall J, “A Framework for the Measurement of Software Quality,” Proceeding of ACM Software Quality Assurance Workshop, 1978, pp. 133-139.

[2] Childamber S.R., Kemerer C.F., “Towards a Metrics Suite for Object-Oriented Design,” Proceeding of Object Oriented Programming Systems Languages and Applications (OOPSLA ’91), Vol. 26, 1991, pp. 197-211.

[3] Childamber, S.R., Kemerer, D.F., “A Metrics suite for Object-Oriented Design,” IEEE Transaction on Software Engineering, Vol.20, Issue 6, 1994, pp. 476-493.

[4] Hironori W., Hirokazu Y., Yoshiaki F., “A Metrics Suite for Measuring Reusability of Software Components,” International Symposium of Software Metrics, 2003, pp. 211-225

[5] Hironori W., Hirokazu Y., Yoshiaki F., “Software Component Metrics and It’s Experimental Evaluation,” Proceeding of International Symposium on Empirical Software Engineering, vol. 11, 2002.

[6] McCabe T., “A Software Complexity Measure,” IEEE Transaction on Software Engineering, Vol.2, No.6, 1976, pp. 308-320.

[7] Sandeep P., Vijay V., “Product metrics for object-oriented systems,” ACM Computing Surveys (CSUR), Vol. 35, No. 2, 2003, pp. 191-221.

[8] ISO/IEC Standard ISO-9126, Software Product Evaluation-Quality Characteristics and Guidelines for Their Use, 1991.

[9] ISO 9126-1-2-3-4, http://www.kenji.com.br/ita/ce230/apresentacoes/

[10] JavaBean Specification, http://java.sun.com/products/javabeans/docs/spec.html.

[11] Java programming dynamics, Part 2: Introducing reflection, http://www-106.ibm.com/developerworks/java/library/j-dyn0603/

[12] Ian Sommerville, Software Engineering, August 2000, 6/e, Addison Wesley, pp.

309.

[13] Carnegie Mellon, Software Engineering Institute, Capability Maturity Model® Integration, http://www.sei.cmu.edu/cmmi/

[14] Object Management Group, Unified Modeling Language, http://www.omg.org/uml/

[15] Parasoft Technology, JTest, http://www.parasoft.com/jsp/home.jsp

[16] Testwell Oy(Ltd), CMT++, http://lambertz.chez.tiscali.fr/cmtppen.htm.

[17] ZeN Technologies, OfficeOrganizer , http://www.zen.com.tw

[18] JARS.COM, http://www.jars.com/

[19] AAA Java Source, http://members.tripod.com/headline/jcode/index.html

[20] Apache Software Foundation, http://www.apache.org/

[21] Struts, http://jakarta.apache.org/struts/

[22] AmbySoft Inc. Coding Standards for Java v17.01d, by Scott W. Amber, http://www.ambysoft.com/javaCodingStandards.html

[23] Draft Java Coding Standard, by Doug Lea, http://g.oswego.edu/dl/html/javaCodingStd.html

[24] 華酷教程, http://tutorial.huacool.com/article.php?articleid=549

[25] Duane K. Fields, Mark A. Kolb, Shann Bayern 著, 賴怡名、楊惠國 譯, "JSP 網站開發實務", ISBN 857-717-971-1, 92 年 4 月初版, 旗標出版社

致謝

碩士的兩年學習生涯，說長不長，說短不短，這兩年對我來講充滿了回憶；首先，我要特別感謝我的指導授受 朱正忠老師，這兩年來身受朱老師的教導，讓我從老師身上學到許多處事技巧與學術研究的方法，這些對我來講是無法從書上或文獻中學得的，在此由衷的感謝朱老師讓我有機會能夠從旁學習；另外也要感謝郭譽申老師、吳毅成老師、楊朝棟老師與徐讚昇老師，感謝口試老師們在颱風天中來參與學生的口試。

兩年生活和我息息相關就數實驗室的事務，碩一那年和學長在一起的生活讓我最印象深刻，每個學長各有各的特色、專長，而現在也各有各的路在走，非常謝謝碩一那年學長們的教導與指教，另外，也非常感謝曾經一同執行案子的林志偉同學，那半年的歷程我想我永遠都會記得；在來，新進來的小學弟，彼此特色與眾不同，替實驗室灌注了新的氣象，未來實驗室就由他們來主導，希望實驗室能夠越來越好；對我而言，你們都是我生活的小老師，從你們身上我學到了許多，感謝這兩年的點點滴滴。

這邊還要特別感謝家父、家母，現在我能夠一路讀上來都是他們的支持，沒有他們就沒有今天的我，除了感謝還是感謝；要感謝的人太多，在這無法一一列出，總之感謝關心我的人與曾經照顧我的人，謝謝你們，沒有你們，我就沒有今天的成就，在此致上我最誠摯的謝意。