國 立 交 通 大 學

管理學院（資訊管理學程）碩士班

碩 士 論 文

使用開放原始碼工具實作軟體產品線方法

Implementing the Software Product Lines with Open Source Solutions

研究生　繆維武

指導教授　楊千 博士

中　華　民　國　一百　年　六　月

使用開放原始碼工具實作軟體產品線方法

Implementing the Software Product Lines with Open Source Solutions

研 究 生： 繆維武　　Student: Wei-Wu, Miao

指導教授： 楊 千 博士　　Advisor: Dr. Chyan Yang

國 立 交 通 大 學

管理學院（資訊管理學程）碩士班

碩 士 論 文

A Thesis

Submitted to Institute of Information Management

College of Management

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Information Management

June 2011

Hsinchu, Taiwan, the Republic of China

中 華 民 國 一 百 年 六 月

使用開放原始碼工具實作軟體產品線方法

研究生：繆維武　　指導教授：楊 千 博士

國立交通大學管理學院（資訊管理學程）碩士班

摘　要

軟體產品線方法是由軟體工程機構(Software Engineering Institute)提出之軟體開發概念性架構，經 SEI 研究證實，使用此方法可以幫助組織以更快的效率、更高的生產力與品質、以及更少的成本建置資訊系統；本論文的目標有二，其一，嘗試提出軟體產品線方法之技術架構；其二，整合 JavaEE 平台和開放原始碼工具來實作此技術架構，以期能提供標準化、開放式架構、低成本並且穩定度高之解決方案；最後會量測本論文實作其之重用程度、節省成本和投資報酬作為驗證，並將其驗證成果就實務上提出導入原則與建議。

關鍵字：軟體產品線、開放原始碼、領域模型、軟體開發、統一塑模語言、軟體專案管理

Implementing the Software Product Lines with Open Source Solutions

Student：Wei-Wu, Miao　　Advisor：Dr. Chyan Yang

Institue of Information Management

National Chiao Tung University

Hsinchu, Taiwan, Republic of China

Abstract

The Software Productline method is a conceptual architecture provided by SEI (Software Engineering Institute). As confirmed by SEI researches, this method can help organizations increase efficiency, productivity and quality, and also decrease cost of implementing enterprise information system. There are two objectives for this thesis. Firstly, the thesis provides a technical architecture for Software Productline method. Secondly, it provides a standardized, open architecture, low cost and stable solution by implementing the provided technical architecture which is integrated with JavaEE platform and Open Source tools. Lastly, the implementation will be verified by measuring it's software reuse percentage, Reuse Cost Avoidance, and Return On Investment; then the verified results will be used to provide practices for organizations to build their own Software Productline implementations.

Keyword: Software Productline, Open Source, Domain Model, Software Development, UML, Software Project Management

致　謝

在交大修習研究所的這兩年，一邊兼顧事業、一邊兼顧課業，學生不得不說真的是很辛苦，但同時也很慶幸自己可以成為這大家族中的一份子，在修習課業的過程中，教授們的悉心指導，開拓了學生在資訊領域的廣度和深度，所辦助理們快速、主動地服務也有如及時雨一般，經常幫學生處理學校行政程序的各項疑難雜症。學生也結識了許多可愛、真誠的同窗好友們，他們有些是如睿智的長者、有些是如靈巧的頑皮小孩，他們的見聞也經常刺激學生去思考新的事物，幸好有這些同窗好友的相互支持，學生才能順利地繼續走下去。

在準備論文的過程中，楊老師經常切中要點的指導和言簡意深的舉例，再加上意鈞學長經常施展的溫柔壓力，帶領著研究室的同袍們，有如指導牙牙學語的小 baby 們學走路一般，一點一點地，從一開始的跌跌撞撞，到現在總算是有個可以拿出來的成果！學生在這個過程中學到很多，也因此跟研究室的同袍們有著堅深的革命情感，令學生非常感動與珍惜。

最後，也要謝謝一路支持學生的父母親、老婆和公司的長官與同事們，沒有你們的諒解與支持，相信我也無法撐過這一段難熬又彌足珍貴的時期，真的非常感謝大家！總而言之，能夠來交大作學問真的太好了！

繆維武 謹誌於

2011 年 6 月國立交通大學管理學院（資訊管理學程）碩士班

目錄

表目錄

圖目錄

第1章　緒論

1.1 研究動機與目的

近年來，世界全球化的影響日趨明顯，各國之間政治、文化、經濟、貿易、人與人之間的關係等，互動愈來越頻繁，相互依存性越來越高，各國之間不斷簽訂 FTA(Free Trade Agreement) 、跨國界的電子商務通行無阻，也因此商業組織之間的競爭程度也越趨激烈；商業組織要提高其競爭力，被動的來說，必須更快速地、更精準地對內、外部事件的做出適當反應，維持市場地位，主動的來說，可以提出創新的產品、服務，甚至是新的商業模型，為企業創造新的利潤和市場！而其中資訊科技占有相當重要的地位，作為一個現代化的企業，除了在生產、銷售、人資、研發、財務等各個功能性部門導入 ERP(Enterprise Resource Planing)，加以整合企業的整體資源，更有甚者，還會建置供應鏈管理(Supply Chain Management)相關之資訊系統，加強從全球供應商開始一直到全球終端客戶手上為止的整體流程，讓企業的競爭力更上層樓！

資訊科技對企業各式管理活動的重要性與影響甚廣，同時要開發一個彈性高、穩定性高、擴充彈性佳和符合顧客期待的資訊系統是相當困難的；舉個例子，當組織的業務隨著時間成長，關鍵集成系統越來越多、實作技術差異越來越大、修補程式上再上修補程式，系統擴張再擴張，最後系統成為了無法移動的大象，以至於無法再繼續提供價值，導致組織不得不考慮使用新技術來替換掉既有的老舊資訊系統，但老舊系統的商業邏輯和相關文件早已散布各地且技術異質性大，取得難度高，所以組織仍然無法徹底遷移至新系統，因為風險實在太高最後組織的彈性反過來受到老舊系統的影響而越來越僵化，進而影響到組織對內、外部環境的反應能力，導致錯過新的商機，抑或是走入衰退(ThoughtWorks 公司，2009)！

在軟體工程領域裡，對於上述的問題，早已有許多優秀的工程師花了幾十年的研究、實作和經驗總結，提出了合理的解決方案—**Software Product Lines**(SEI，2010)；由 SEI(Software Engineering Institute)主導，一種從大量客製化生產製造方法所衍生出來的軟體開發方法，例如筆記型電腦生產製造，由於產品設計的先期投入，使得工廠生產線可針對同一產品線的產品，大量生產統一的機體，等到接到客戶訂單時，依據客戶的實際需求，在機體的統一介面上搭配不同的模組，如顯示卡模組、記憶體模組等，因而產生不同的類型但是相似的產品；此概念如今被拿來在軟體開發上使用，將欲解決相似問題之系統歸納在同一產品線下，當它們在被開發時就可共用統一的核心資產(core assets) ，如系統架構、可重用軟

體元件(Reusable Software Components)、領域模型(Domain Model)、需求文件、測試程式碼...等，每一個系統(產品)就可採用組裝式軟體開發(Development by assembly)方式(吳信輝，2006)，以期達到以更快的效率、更高的生產力與品質和更少的成本來建造資訊系統，讓組織更為敏捷(agility)。

圖 1.1 為軟體產品線之概念簡圖，假設一組織有生產、行銷、人事、研發和財務部門等，給予每一部門一條軟體產品線，而旗下的每一個產品即是欲解決相同領域問題之不同面向的資訊系統，因問題領域同質性高，故每一產品使用核心資產的機會則大大提升，採用組裝式軟體開發的程度提高，最終可以用較少的資源、更快的速度、更好的品質來建造資訊系統。

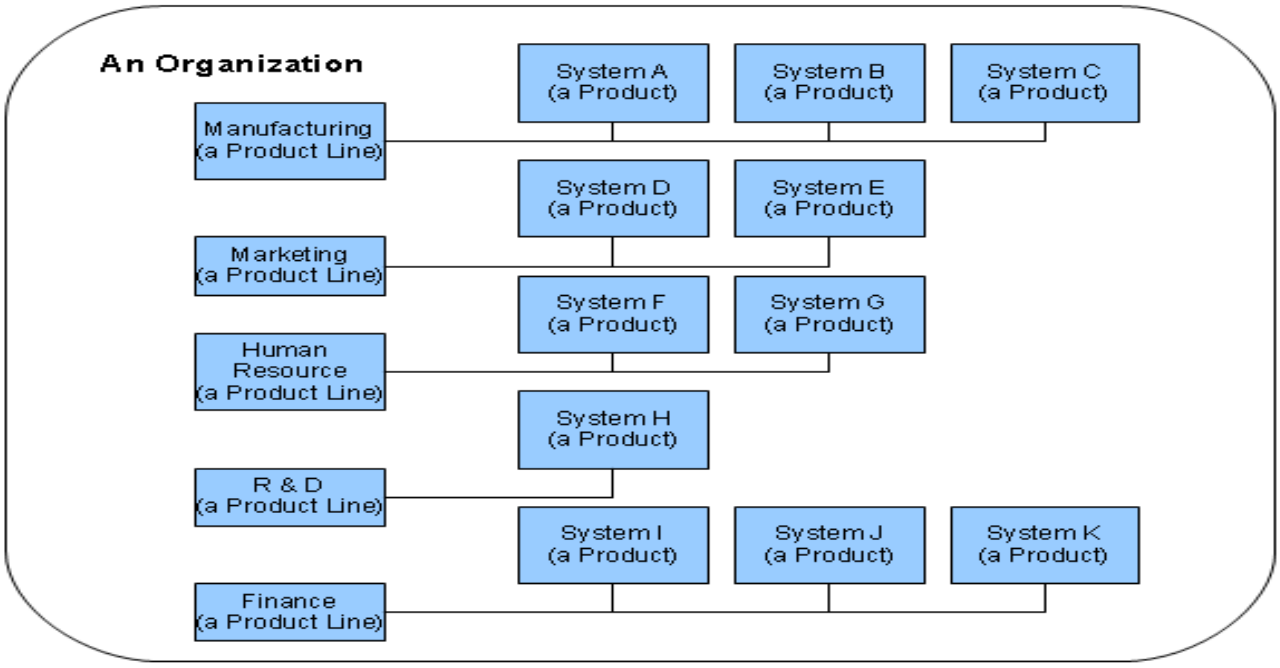


圖 1: Software Product Line 之概念簡圖

資料來源：[本研究整理]

依照 SEI 的網站的公布，採用 Software Product Line 的組織所獲得的績效(SEI，2010)，簡列如下

- 生產力改善將近 10 倍
- 品質增加將近 10 倍
- 成本減少將近 60%
- 人力資源減少將近 87%
- Time to market(組織內部部門、外部使用)時間減少將近 98%

- 進入新市場的時間，從原本幾年減少到幾個月

以類似概念執行的軟體專案在台灣還不普遍，故在此論文中欲驗證觀點有二，(1)以實作觀點切入，提出軟體產品線方法之技術架構，(2)基於論文提出之技術架構，嘗試使用開放原始碼工具，欲提供標準化、開放式架構、低成本並且穩定度高的解決方案。此論文最後將實作一個包含線上書店前台和後台管理的網站系統模組並對其結果作分析與探討，希望能藉由此論文的產出，能對於實務上提出欲導入之原則與建議。

1.2 論文架構與研究流程

本論文架構分為五個章節，敘述如下：

第一章：詳細介紹研究動機及目的、論文架構、研究流程。

第二章：研究方法的相關背景與概念介紹。

第三章：軟體產品線方法之技術架構與開放原始碼工具的整合與對映關係。

第四章：使用軟體產品線與開放原始碼工具設計與實作。

第五章：對本研究產出結果說明，並提出對於實務需求之導入原則與建議。

論文研究流程如圖 1.2：

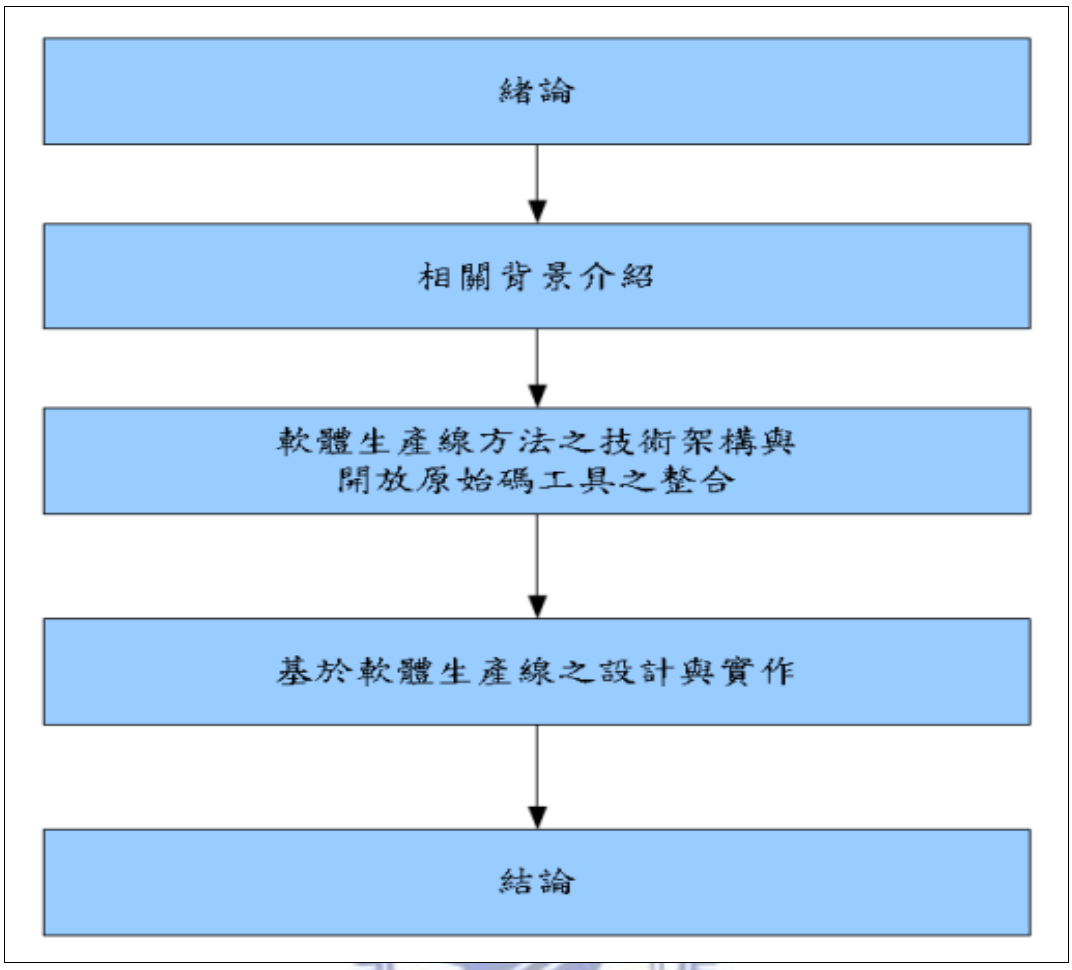


圖 2: 論文研究流程

第2章　相關背景介紹

2.1 軟體抽象化層次之演進

從程式語言開始發展的1950年早期，到目前為止，軟體系統的開發方向是往越來越高階的抽象化層次(higher level of abstraction)發展(Balasubramanian、Gokhale、Karsai、Sztipanovits、Neema，2006；吳信輝，2005)，依抽象化層次條列如下。

- 機器語言(Machine Language)

  電腦問世後的第一個程式語言，直接使用二進位表示，可由機器CPU直接讀取的語言，執行速度非常快，因為是由機器直接讀取，所以可讀性(human readability)低，故學習曲線相當高，且在不同的機器上，程式就得改寫，(不同的CPU吃的指令集不同)，完全無可攜性(portability)可言。

- 組合語言(Assembly Language)

  為提高機器語言可讀性，一種便於記憶和理解的符號表示式，由組譯器(Assembler)轉譯成特定CPU的機器語言，然後再給其CPU執行，可攜性部分，是針對一種家族的CPU，如果是其它廠牌的CPU，轉譯出來的機器語言可能無法執行，目前在底層硬體操作或是驅動程式仍然可見到它的蹤跡。

- 高階程式語言(High Level Programming Language)

  為了提高程式設計師的生產力，故開發了可讀性更高的程式語言種類，由人類容易閱讀的英文單字組成，不同的機器有提供不同版本的編譯器(compiler)，將撰寫好的高階語言吃進其編譯器，編譯成相對可執行之機器語言。

- 結構式語言(Structured Programming)

  高階程式語言的一種，具有三種特性，為依序執行(Sequence)、條件判斷(Selection)和反覆執行(Repetition)，結構式語言通常是以演算法的資料結構和執行的流程與順序來解決問題領域的問題，故系統開發人員是從電腦運算平台的角度來看待問題領域。

- 物件導向語言(Object-Oriented Programming) (Eckel，2002)

  高階程式語言的一種，除了包含結構式語言的特性之外，還另外具有三種特性，封裝

(Encapsulation)、繼承(Inheritance)和多型(Polymorphism)，物件導向語言可將問題領域之重要概念的相關資料與處理行為加以封裝，形成類別(Class)，將概念相關之類別作一般化/特殊化(Generalization/Specialization)繼承處理，抽取出共用與非共用的程式碼區塊，最後也可藉由多型去改變類別行為的變異點，達到程式演算法共用的效果。故物件導向語言可將問題領域加以抽象化，將程式演算法從問題領域概念中抽離出來，系統開發人員便有機會從問題領域本身的角度來看待問題，接著才去思考運算平台的實作。

- 應用虛擬機器(Application Virtual Machine)

  一種在作業系統運行的軟體，當特定程序(Process)被執行起來時，AVM 便先會被執行，並且包含此特定程序，提供所需資源，當此特定程序執行結束時，AVM 也會執行結束並從記憶體中移除；主要功能是提供此特定程序一個完全獨立於底層作業系統與硬體的執行環境，藉以讓此類型的程序在不同的運算平台上都可以使用相同方式運行，而不需考慮運算平台的異質性。近代的物件導向語言如 Java、.NET 等，都包含此一技術，方便系統開發人員 WORA(Write once run any where)。

- 中介軟體(Middleware) (Sun Microsystems，2008)

  也稱為應用伺服器(Application Server)，介於系統程式與應用程式之間的軟體，便於應用系統之軟體元件間的連線與資源控管，中介軟體基於應用虛擬機器的實作，可使應用系統具有更高的可攜性，而且中介軟體通常會提供應用系統常使用的服務，如資料庫連線、交易控管機制、外部系統溝通、電子郵件設定和安全性設定等，減輕系統開發人員考慮這些服務的負擔，而把心力花在問題領域本身。

雖然軟體產品線的類似概念早在 1985 年就有成功的案例(Clements、Northrop，2002)，CelsiusTech 為當時瑞典的大型軟體公司，可投入的資源和技術環境就比大部分小型軟體公司或是非軟體產業的公司要來的優渥，但在上述提及的相關技術環境成熟的前提之下，如今要實作軟體產品線實為管理階層和相關執行人員的思維轉換(paradigm shift)，相較之下，技術上的限制已大幅降低。

2.2 物件導向系統的設計風格

近十幾年軟體系統開發的主流是使用 OOAD(Object-Oriented Analysis and Design)，現今經常聽到的產品與平台有 Sun Microsystems Java 和 Microsoft .NET 等，除了要使用相關平台與

技術之外，系統設計的風格也是影響到軟體系統後續的彈性、穩定性和可維護性，目前物件導向系統設計風格的主流分為兩大類(1)Transaction Script 和(2)Domain Model，分別描述如下。

2.2.1 Transaction Script 設計風格

何謂 Transaction Script 的設計風格，依 Martin Fowler 的定義如下

> *Organize business logic by procedures where each procedure handles a single request from presentation.*(Fowler，2002)

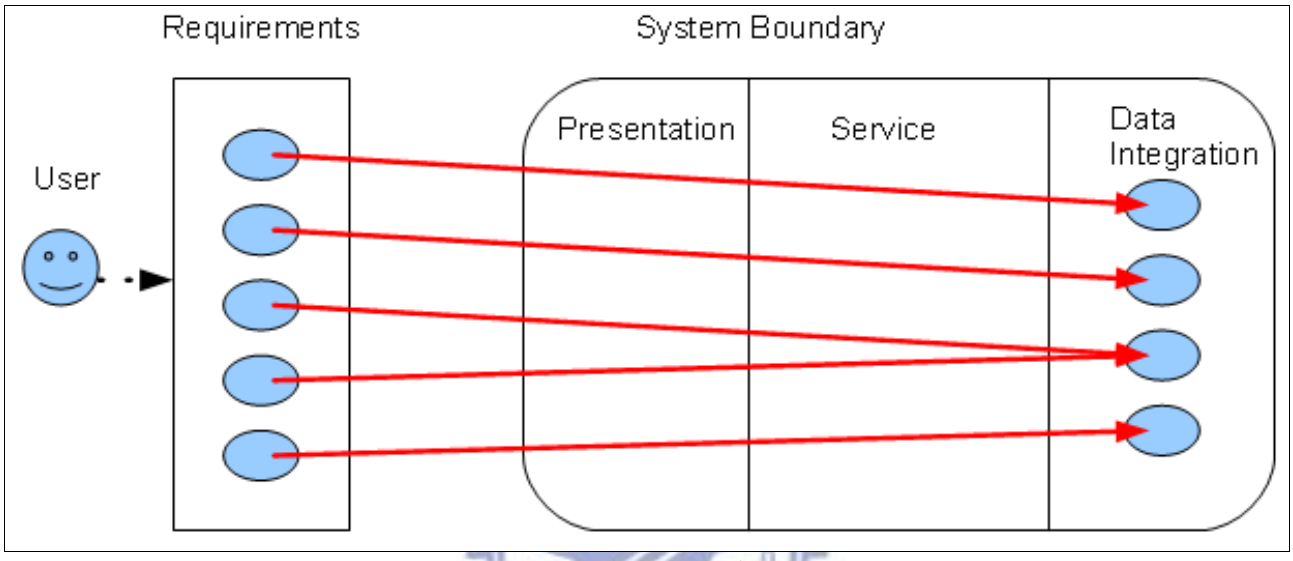


圖 3: Transaction Script 系統設計風格
資料來源：[本研究整理]

簡言之，就是系統設計師依照 SA 開出的需求規格，重頭到尾地把相關設計與開發實作出來，然而這樣的設計風格，會不自覺地使得系統的架構與商業邏輯之程式碼綁在一個或多個需求上，如圖 2.1 所示，系統的每一個 tier 都綁在由需求延伸出來的紅線上，然而 end user 的需求是一天到晚都在變動的(I know it when I see it)，所以 Transaction Script 設計風格的系統架構很容易就會因需求變動發生大範圍的影響，使得接受需求變動的彈性較低。

2.2.2 Domain Model 設計風格

何謂 Domain Model 的設計風格，依 Martin Fowler 的定義如下

> *An object model of the domain that incorporates both behavior and data.*
> (Fowler，2002)

此設計風格主要是強調在開發系統時，除了 user 的需求之外，其實我們還要更關注 user 所在的商業環境！因為 user 會提出需求，其背後是因為商業環境的驅動使然，所以系統除了

要提供滿足 user 需求的服務之外，其核心應該建立在 user 背後的商業環境上，因為其和需求本身相較，是本質上穩定不變的部分。如此，即使 user 的需求天天在變，系統受到影響的部分將會有效減少，因為 user 的商業環境並不是天天在變動的。然而此種設計風格是需要 SA 大力支持的，因為需要有人作商業塑模，所以 SA 對於軟體工程的認知和系統抽象化思考是此設計風格事觀重要的一環。

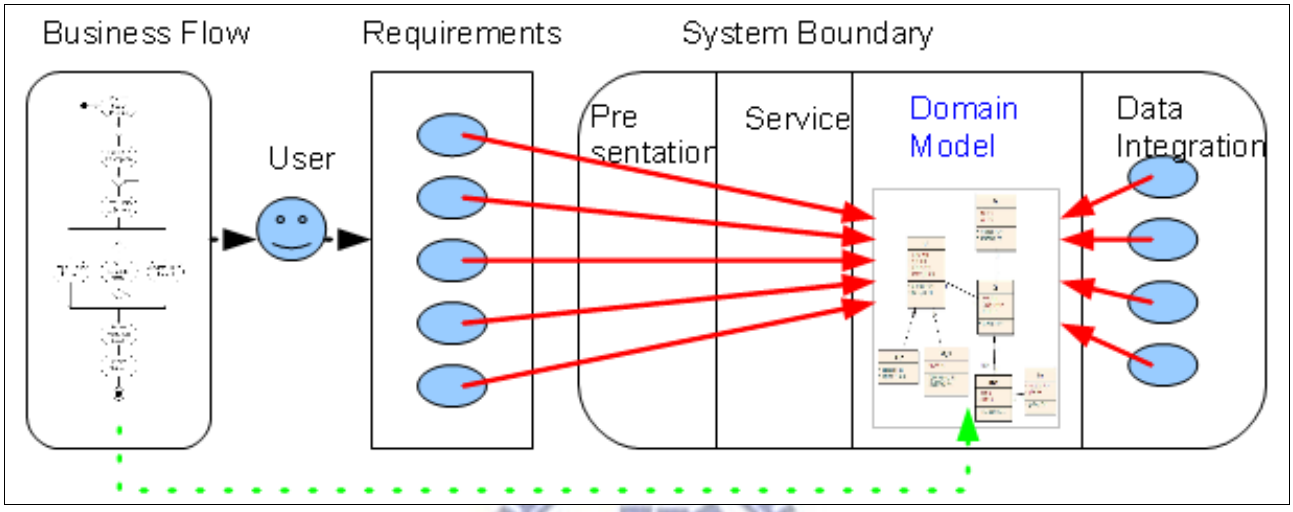


圖 4: Domain Model 系統設計風格
資料來源：[本研究整理]

Martin Fowler 是美國的軟體架構研究者，他在『Patterns of enterprise application architecture』一書中，提到他以自己三十幾年來軟體開發的經驗，總結以下圖表 (質化，非量化方法)

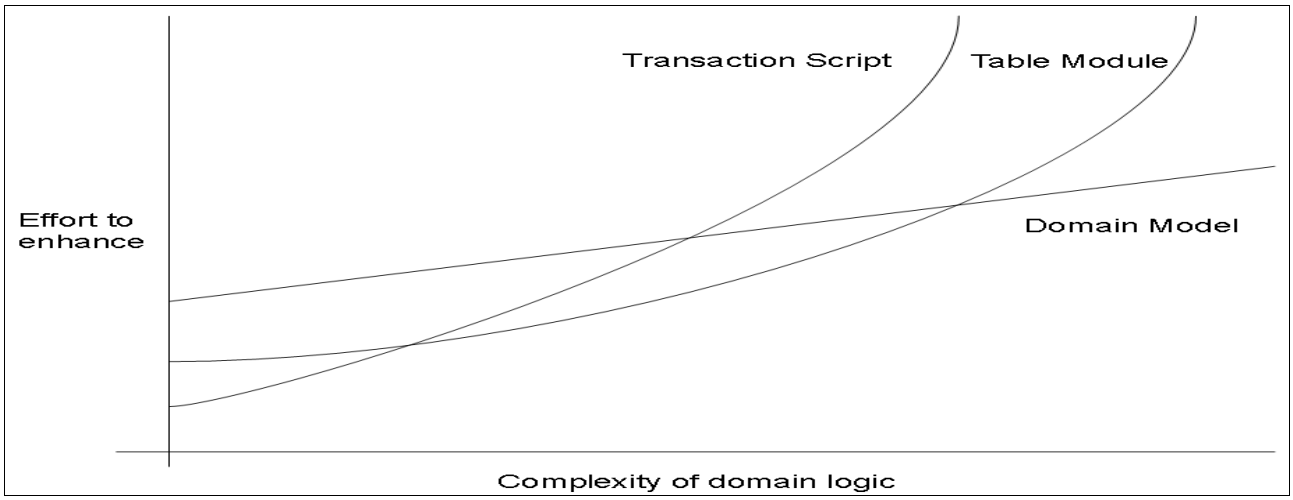


圖 5: A sense of the relationships between complexity and effort for different domain logic styles
資料來源：[Fowler，2002]

在圖 2.3 中的三條線各自代表不同的設計風格，其中 Table Model 為.NET 相關技術，故不在此討論範圍內，X 軸代表系統的複雜程度，Y 軸代表系統的維護心力，由此可看出 Transaction Script 的設計風格是適合較小型、邏輯簡單的系統，而 Domain Model 的是適合較大型、邏輯複雜的系統。

2.3 Domain-Driven Design

由 Eric Evans 提出的 Domain-Driven Design，可說是 Domain Model 設計風格的進階版，其主要中心思想有兩點 (1)在大部分的軟體專案中，設計主要是聚焦在問題領域和商業邏輯，(2)要解決複雜問題領域，須基於模型做設計。它系統化地提供了專有術語(Terminology)和最佳實務(Best Practices)，來幫助開發團隊如何做出更好的設計抉擇，可以更快速地開發欲解決複雜問題領域的軟體系統(D.-D. D. Community，2009)。

2.3.1 Domain Model 的精確定義

D.D.D 對於 Domain Model 有著更精確的定義，描述如下(Richardson，2006)：

1. Entity：一個物件不是被它所擁有的屬性(property)定義，而是被一個識別子(Identity)所定義
2. Value Object：一個物件是被它所擁有的屬性所定義，沒有識別子，且一旦實例化(instantiate)後就無法變更(immutable)。
3. Aggregate：一個根物件指向一個或多個物件集合，由根物件負責維護物件集合的一致性，故物件集合不會被其他外部物件直接呼叫，而是透過根物件間接呼叫(delegate)；根物件也被稱為 Aggregate Root。
4. Service：當系統的某個程序在邏輯上並不屬於特定領域物件(Domain Object)所擁有，或是該程序需要多個 Domain Model 互動來完成，就需要 Service 物件來擔負這些責任。
5. Repository：負責跟永久層(Persistence Tier)請求領域物件的 CRUD(Create, Retrieve, Update and Delete)操作，基本上 Domain Model 必須不能知道跟永久層互動的細節，因為永久層包含了跟特定資料庫連結或是其他種類的檔案格式的細節，所以需要 Repository 以介面(interface)的方式存在於 Domain Model 中，而相對應之實作(implementation)則由永久層提供。
6. Factory：當要實例化一個領域物件時，建議使用 Factory 物件，所有的領域物件都由

相對的 Factory 物件來實例化，如此一來，當更換領域物件的實作時，只需更改 Factory 物件裡的程式碼，而不須更動客戶端(Client-side)的程式碼。

2.3.2 Ubiquitous Language

DDD 也明確提出當軟體專案溝通協調的語言不一致時，會帶來相當大的風險；例如在系統開發階段的討論時，領域專家(Domain Experts)使用他們的專業術語(jargon)，然而技術團隊成員使用他們自己的術語，再加上這些討論結果無法直接對應於系統的實作上(中間還會歷經系統設計師和不同開發人員的設計)，以至於同一件重要概念會有不同的名詞描述，被使用在不同的角色中(領域專家對系統分析師、系統分析師對系統設計師、系統設計師對系統開發人員，同一角色的不同人員等)；最後不同名詞的翻譯妨礙了專案溝通，而導致無法找到問題領域的特性(insights)，並且沒有一個成員使用的名詞可以有效地在不同專案參與人達成充分理解；到最後專案團隊整體溝通和錯誤理解(misunderstanding)的成本是相當高的(Evans，2003)。

針對上述問題 DDD 提出了 Ubiquitous Language 作為解決方案，Ubiquitous Language 是一種基於模型(Model-based)的語言，為整個系統開發生命週期(SDLC, System Development Life Cycle)之溝通協調的基本語言；專案團隊必須承諾 Ubiquitous Language 被嚴格使用在所有的溝通上，包含口頭、文件、UML 圖，甚至是實作程式碼等；專案參與人之間必須不斷地精煉(refine)此語言，領域專家必須要提出任何對於問題領域造成誤解或是不佳的設計與架構，系統開發人員必須找出不一致或是不精確的部分，直到所有人對於此語言沒有任何誤解與疑義為止；在這期間，所有相關的口頭、文件、UML 圖和實作程式碼也必須要不斷地跟 Ubiquitous Language 的最新版本保持一致。最後 Ubiquitous Language 不只是系統開發生命週期的一部分產出，而是所有專案參與人溝通的基本語言，從需求收集到後續系統維護階段，使得溝通協調和錯誤理解的成本降至最低。

2.4 Unified Modeling Language(UML)

UML 是圖形化的塑模語言(Graphical Modeling Language)，在 1980 年代末期到 1990 年代初期由 OMG 組織發展出來，是一種圖形表示方法的集合，被用來視覺化、詳述、建構和紀錄軟體密集系統(Software-Intensive System)的各種面向 。之所以要使用圖形化塑模語言的主要原因是，光從程式語言本身來看，無法提供更高的抽象化層次來幫助我們建立更複雜、更龐大的軟體系統；雖然圖形化塑模語言在軟體業發展已久，但是百家爭鳴，每家的語言表達

的含意都有差距，以至於軟體從業人員無法真正達到有效溝通，而 UML 的出現，有如救世主一般，結束了圖形化塑模語言的戰國時代(Fowler，2005)。目前通行的 UML 版本為 2.x 版，以下是 2.3 版的 UML 圖分類簡圖。

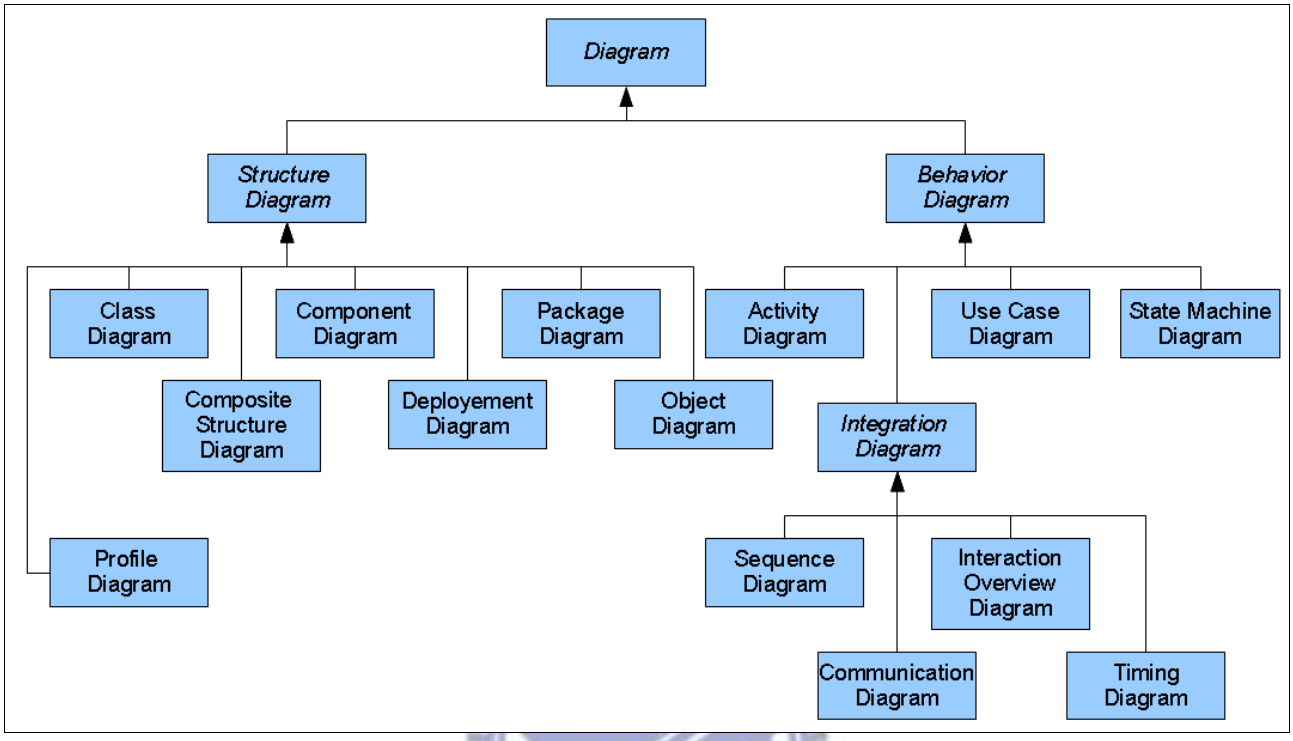


圖 6: UML2.3 版分類簡圖
資料來源：[Group，2010]

2.5 Software Product Line Engineering

在第一章時稍微描述過其概念和效益，此軟體開發方法為此論文欲驗證之重點，其正式定義為「一個軟體集成系統的集合，集合中的系統彼此分享一套共通和管理的功能集合，此功能集合是為了滿足某一特定市場區隔或是任務之需求，而基於事先規劃好的共享核心資產所開發出來的」(A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.) (Clements、Northrop，2002)。

2.5.1 軟體產品線之必要管理活動

實作軟體產品線的主要活動為(1)在有效的技術和組織管理(Management)支援之下(2)開發核心資產(Core Asset Development)，接著(3)基於已開發之核心資產來開發產品(Product Development)，核心資產開發通常被稱為領域工程(Domain Engineering)，而產品開發通常被

稱為應用工程(Application Engineering)。

如下圖 2.7 所示，有三個順向旋轉的圈圈，每一個圈圈代表著一個必要的活動，這三個圈圈彼此連結且永不間斷地運行著，可以從任何圈圈起始並且高速運轉(Highly iterative)。這三個圈圈是相互平等的，依使用者環境的需求，可以從管理活動、從核心資產開發活動亦或是從現有的產品(個別系統)抽取出核心資產開始推動整個軟體產品線的運行。從個別圈圈的箭頭方向，也可看出它們之間互有回饋機制，核心資產會因為新產品開發的需求而被要求更新核心資產的更新需要被追蹤，並藉由產品開發時的使用結果來當作核心資產的回饋，如此核心資產才會產生實際價值；而產品開發也要盡量使用核心資產的元件，盡量反映需求給核心資產開發活動，避免自己閉門造車。然而這一切都需要強力的、有遠見的管理活動之領導才能發揮效益。

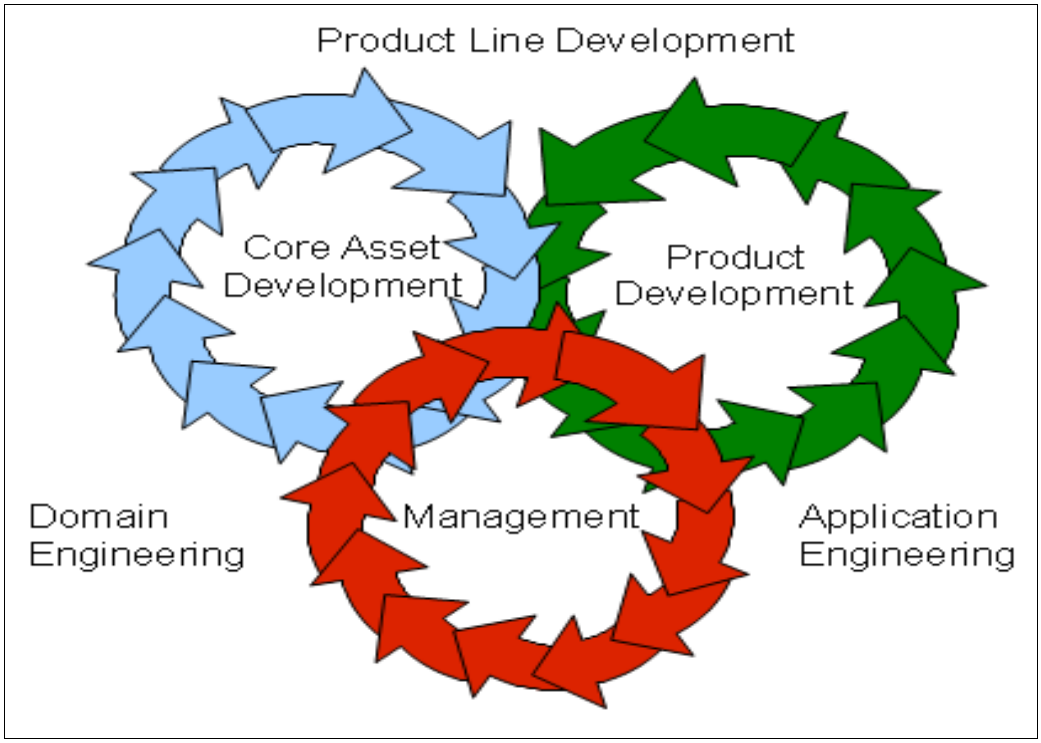


圖 7: 軟體產品線的必要活動
資料來源：[Clements、Northrop，2002]

以下再就此三項必要之管理活動加以描述。

2.5.2 核心資產開發活動(Core Asset Development)

核心資產開發活動的目標就是建立產品需要的生產要素。圖 2.8 表明了核心資產開發活動需要的輸入和輸出，順時針運轉的箭頭也表明了輸入和輸出之間並沒有明確的因果關係，輸出和輸入是會相互影響的；例如增加了生產線範疇的廣度(其中一項輸出)，就必須考慮是否

要增加新的元件輸入至核心資產開發的活動之中。

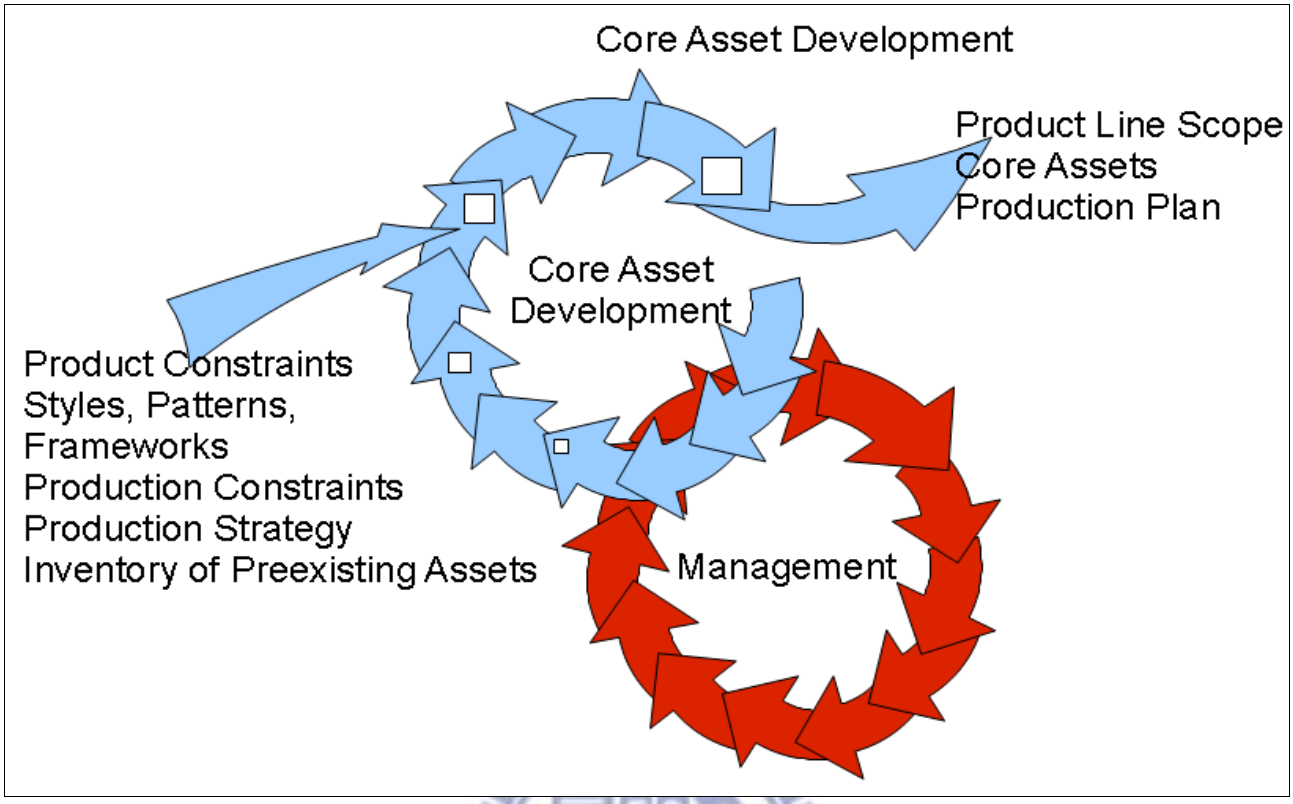


圖 8: 核心資產開發

資料來源：[Clements、Northrop，2002]

核心資產開發活動會使用各種資源當作輸入，分類如下。

1. 產品限制(Product Constraints)

   生產線旗下產品的共通與相異點、生產線的功能性需求、系統效能需求、使用何種技術標準、與哪些外部系統溝通等。

2. 風格、樣式和框架(Styles, Patterns and Frameworks)

   現今軟體開發大部分都是採用既有的程式，例如使用 Domain-Driven Design、J2EE 的架構建議、四人幫的設計樣式(GOF Design Patterns)或者是採用開放原始碼社群的框架等，這些解決方案也提供使用者以客製化方法(介面、繼承、多型等)實作系統相異點，作為軟體產品線的輸入是很自然的。

3. 生產限制(Production Constraint)

   生產線是否要遵守任何商業、軍事或是組織的特定標準？旗下產品一定要使用的設施？進入市場的時間限制？套裝軟體的整合方式？是否要既有元件和如何使用？

4. 生產策略(Production Strategy)

   使用 Top-Down(從核心資產到旗下各個產品)或是 Bottom-Up 生產策略(從既有產品萃取出核心資產)？自己內部製作或是從外部購買核心資產？如何管理核心資產生產流程？

5. 既有核心資產的管理(Inventory of Preexisting Assets)

   既有系統(Legacy System)包含了組織的領域知識甚或是決定市場形象，軟體產品線通常都會借用這些已驗證的系統和元件作為核心資產的一部分。詳加分析和使用這些既有系統是重要的。

核心資產開發活動會產生三個輸出，之後作為產品開發活動的輸入使用，以下描述其輸出。

1. 生產線範疇(Product Line Scope)

   描述此軟體產品線是由哪些產品所組成。最簡單的方式是列出此生產線所包含的產品清單；更正統的做法是列出包含產品之間的共同點和相異點，例如產品之間功能性需求之相同與相異處、非功能性需求之相同與相異處、產品運行之平台與使用技術等。

2. 核心資產(Core Assets)

   軟體產品線製造旗下產品的重複使用之基本要素。包含系統架構、軟體元件和領域物件、系統效能分析、商業邏輯、軟體開發工具和流程、測試計畫和程式、熟悉此生產線的人力資源(熟悉一個產品 100%等同於熟悉其他產品的 60~70%)。

3. 生產計劃(Production Plan)

   描述如何使用核心資產生產產品。每一個核心資產都應該要有相對應的生產計劃，來指導使用者如何在產品開發活動使用此核心資產。核心資產通常會定義相異點(Variation Points)讓不同的產品使用，使用生產計畫描述如何使用這些相異點(剪裁、重新組合等)是尤其重要的。

2.5.3 產品開發活動(Product Development)

成熟的軟體產品線組織重視軟體產品線的健康程度勝於個別產品，但軟體產品線的最終目標還是要產生出個別產品。產品開發活動除了基於核心資產開發活動的三項輸出，生產線範疇、核心資產和生產計畫之外，再加上個別產品的需求文件。圖 2.9 所示，逆時針方向運

行的箭頭代表著一個產品的輸出會成為回饋和影響到下一個產品的生產；例如當產品 A 在製作時，發現有個功能可以整合到核心資產共用，而下一個產品 B 需要同樣功能時，就直接使用核心資產的元件，而不需再重新發明輪胎。

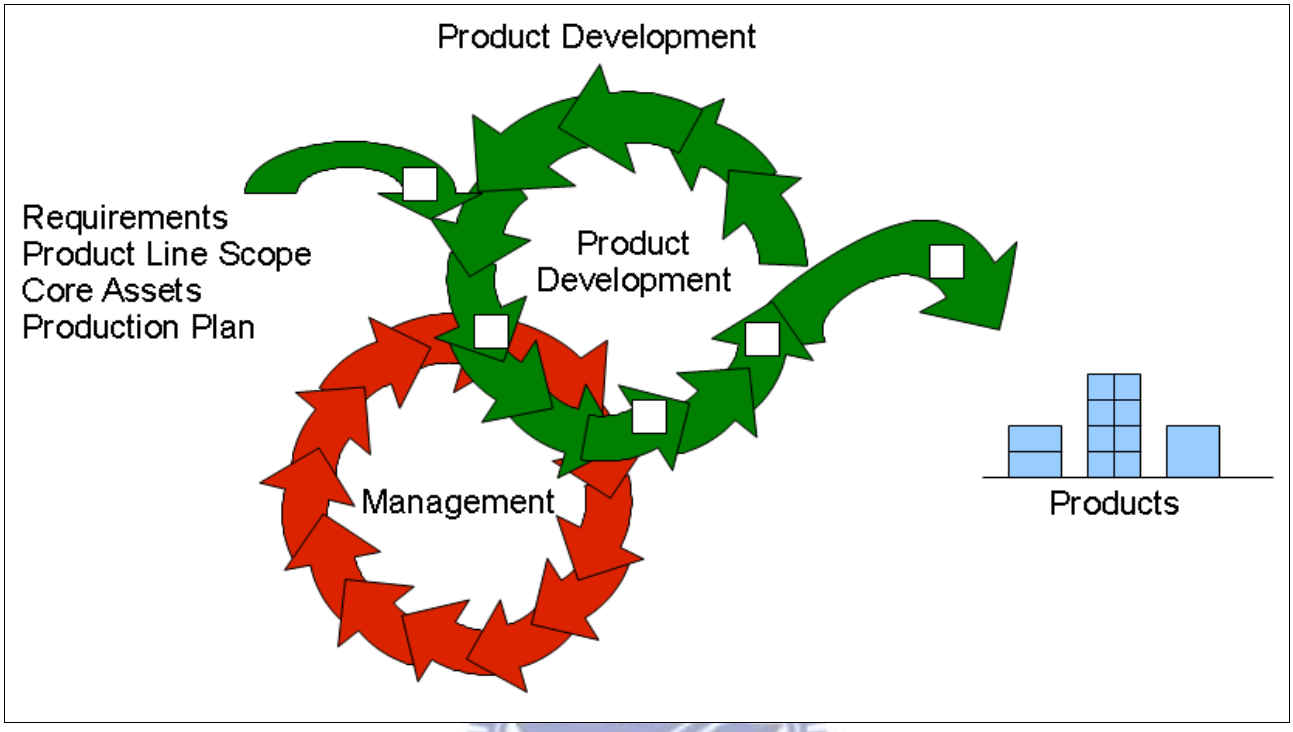


圖 9: 產品開發

資料來源：[Clements、Northrop，2002]

產品開發活動的四項輸入，描述如下。

1. 需求文件(Requirements)

   特定產品的需求文件就如同是生產線範疇文件的變異點描述。

2. 生產線範疇(Product Line Scope)

   定義特定產品的變異點是否包含在軟體產品線中。

3. 核心資產(Core Assets)

   產品的大部分功能將由核心資產組成。

4. 生產計劃(Production Plan)

   指導產品製作者如何使用核心資產來實作產品特定功能。

2.5.4 管理活動(Management)

管理活動是軟體產品線成功的因素，技術和組織的管理階層必須對軟體產品線建置許下承諾，給予適當的資源、協調和管理。技術管理階層必須確保核心資產和產品開發活動必須是在軟體產品線的定義下進行、所有必要的活動都有被執行、並且收集資料和追蹤兩者的健康狀態。組織管理階層則須採取一切必要的管理活動，來確保組織的架構和資源都適當地分配給軟體產品線運作。

2.6 軟體工廠與軟體產品線

在中研院的期刊『資訊話題』中也有軟體產品線的相關討論，吳信輝先生在「細說軟體工廠概念(十一)」文中提及：「有關於軟體工廠的內容，可以區分為兩個部份：第一部份是軟體產品線的發展，第二部份是產品的發展。在第一部份，軟體產品線的發展又可以區分為三個階段，分別為軟體產品線分析，進行的工作內容包括：產品線的定義、問題領域確定以及解決方案領域確定；第二為產品線設計，進行的工作內容包括：軟體架構的發展、客戶需求的映對以及產品發展流程；第三為產品線的實作，進行的工作內容包括：軟體資產的供應以及軟體資產的包裝。經過產品線的發展之後，產生出軟體工廠綱要與可變動的軟體資產。在第二部份產品的發展中，首先會利用產品的規格與產品線發展的軟體工廠概要兩個結合利用工具產生產品的基本架構；接下來再利用客製化工具將變動軟體資產依其所需取出會利用到的地方，再進行產品的實作。如果我們結合第一部份的經濟學的觀點，我們會發現軟體工廠的架構就是要同時處理軟體的規模經濟問題與範疇經濟問題，產品線發展就是要發展出可重複利用的軟體元件，以達到規模經濟的目標；而產品發展則是以客製化為目標，以達到範疇經濟的目標。」(吳信輝，2006)

2.7 測試驅動開發

1999 年由 eXtreme Programing 提出 test -first programing 概念，和其後於 2002 年由 Kent Beck 所撰寫的 Test Driven Development 一書和其相關開源專案 jUnit 而加以發揚光大的開發模式，Kent Beck 強調開發軟體專案時先撰寫自動化的測試程式，隨後再撰寫實作，會帶來簡單乾淨的設計，並且有效提高系統的品質和增加產值。以下條列出此開發模式之步驟(Beck，2003；Wells，2009)。

1. 思考你要測什麼？

   幫助開發人員深度思考是否真的了解需求，如不了解可快速跟 SA 反映。

2. 思考你要怎麼測？

   思考程式的介面該如何設計，以確保每個程式的最小單位都具有高度可測性。

3. 寫出測試程式碼並執行，確認他們全都是有問題的(failed)

   先確認它們都是有問題，免得測試程試結果有誤。

4. 開始實作一段、測試一段，直到全部測試程式都通過測試(pass)

   實作程式碼的每一個區塊目標都很明確，就是要通過測試程式，幫助開發人員思緒集中。

5. 移除實作中重複的程式碼，或是加上設計樣式(也就是作重構)，接著再確認測試程式是否全部通過

   在不影響實作程式功能性需求的情況下重構程式，讓程式的品質提昇。

6. 持續使用以上步驟開發系統新功能

   漸漸擴大測試程式涵蓋率，直到整個系統。

要特別注意的是，以上的步驟都是以自動化可回歸測試為前提，如此一來，每當一段新的功能被加在系統時，就可立刻執行整個系統的測試程式，以期達到每次系統的新發佈都是在品質良好的狀態；未來系統正式上線後，欲增加新需求時，如果測試程式的涵蓋率夠高的話，就能提昇開發人員對於系統作重構時的勇氣(如果重構造成問題，可藉由測試程式立即發現)，最後即使在後續系統進入維護的生命週期，仍然可以保持良好的可維護性。

本論文在系統實作結束後，會執行 jUnit 製作測試報告，包含測試執行結果報告和測試涵蓋率報告，作為論文結論中的一部分；測試執行結果一定要 100%通過測試，系統才能進入建置階段；目標測試涵蓋率為 80%以上，測試的系統層級包含持久層、商業邏輯層和服務層，但不包含使用者介面層級(Controller、jsp、js 和 html 等)。

2.8 量測軟體重用程度

既然軟體產品線的目標是製作可重複利用的核心資產，所以個別產品生產之後的量化量測方式也是相當重要的；在管理方面，量測軟體重用程度可讓管理階層更了解軟體產品線的健康程度和為接下來的管理活動提供方向；在效益方面，量測軟體重用程度可幫助管理階層量化軟體產品線的投資報酬，了解資源投資的程度和實際的效益，更有甚者，可以拿來作為

在組織中推廣軟體產品線的利器！

本論文在系統實作結束之後，將會採用 Jeffrey S. Poulin 所撰寫『Measuring Software Reuse』一書所提供的量測模型(Poulin，1997)，來量化本論文軟體產品線實作之效益，量測模型簡述如下。

2.8.1 Reuse Software Instruction (RSI)

由於軟體本身就具有高度複雜性的天性，在計算軟體重用性時，各別組織亦或是個別開發團隊對於軟體重用性的定義都不同，以至於各家的重用性量測報告都沒有基於同一基準上作量測而無法比較，所以 Poulin 提出了一系列定義，符合定義才被歸納為重用或是不重用軟體元件，最後被歸納為重用軟體元件之集合，被稱之為 RSI。

表 1: 軟體可重用和不可重用元件之定義

Type of reuse	Measured?
Maintenance	No
Operating System	No
High-level language	No
Tools	No
Multiple uses	One time
Commercial off the shelf(COTS) software	No
Porting	Separately
Application generators	Separately
Utility libraries	No
Local utility libraries	Maybe
Project/domain-specific libraries	Yes
Subclassing, with polymorphism	No
Subclassing, without polymorphism	Yes
Object composition	Yes
Parameterized types	Yes
Black-box	Yes
White-box	No

資料來源：[Poulin，1997]

除了表 2.1 所列出之基本定義之外，本論文基於軟體產品線實作之特性，另外再加了三條定義如下。

1. 組態設定檔

本論文實作在佈署、持久層控制和安全性控制等，有一些組態設定檔，它們雖然不是程式碼，但仍然或多或少都有被拿來重用，但百分比占整個實作來說是非常些微，故為降低計算複雜度，本論文不把它們列入 LOC(Line of codes)計算。

2. 測試程式

本論文實作採用測試驅動開發，故在商業邏輯方面的程式，幾乎都有相對應的測試程式碼，所以當產品重用核心資產的軟體元件時，其實同時也省去了開發測試程式碼的資源，故本論文把測試程式碼列為重用軟體元件。

3. 系統架構範本

本論文在核心資產中，有一項是系統架構範本，此範本將會被旗下產品直接複製並修改，是採取白箱重用(White-box reuse)，在表 2.1 中，白箱重用被歸納為不重用軟體元件。

2.8.2 Reuse Metrics Starter Set

Poulin 提供了一個易於使用、了解的量測模型，來讓有興趣的組織來快速使用，稱為 Reuse Metrics Starter Set，包含以下三個部分。

1. 計算重用率(Reuse %)

$$Reuse\,\% = \frac{RSI}{Total\ Statements} \times 100\,\%$$

2. 計算重用節省之成本(Reuse Cost Avoidance)

$$RCA = DCA + SCA$$

Where DCA (*Development Cost Avoidance*):

$$DCA = RSI \times (1 - RCR) \times (New\ code\ cost)$$

And SCA (*Service Cost Avoidance*)

$$SCA = RSI \times (Your\ error\ rate) \times (Your\ error\ cost)$$

以上公式包含了一些預設變數，是 Poulin 分析各家相關論文和報告歸納出來的數值，組織可以依自身的實際狀況加以修改，本論文則是採取變數預設值作計算，變數和預設值簡述如下。

- *RCR(Relative Cost of Reuse)*

使用重用軟體元件所花掉的資源是重新開發軟體元件的 20%，所以預設值是 0.2。

- *RCWR(Relative Cost of Writing for Reuse)*

開發可重用軟體元件所花掉的資源是開發一般軟體元件的 1.5 倍。

- *New code cost* = $100/LOC
- *Error rate* = 1 error/KLOC
- *Error cost* = $10K/error

3. 計算投資報酬(Return On Investment)

先計算開發重用軟體元件所花費的額外成本(Additional Development Cost)。

$$ADC = (RCWR - 1) \times (Code\ written\ for\ reuse\ by\ others) \times (New\ code\ cost)$$

接下來再計算投資報酬如下。

$$Project\ ROI = \sum_{i=1}^{n} RCA_i - ADC$$

2.8.3 重用率效益分析案例

Poulin 也收集不同公司的重用率效益分析案例，條列如下。

- Nippon Electric Company(NEC)

達到 17%之重用率，產生 6.7 倍的生產力和 2.8 倍的品質提昇。

- GTE Corporation

達到 14%之重用率，在軟體開發方面，節省一千四百萬元美金。

- Toshiba

達到 60%之重用率，每一行程式碼減少 20~30%缺陷發生率。

- Hewlett-Packard(HP)

針對兩個旗下專案作品質改善，使其達到 70%之重用率，分別減少 76%和 24%缺陷發生，增加 50%和 40%的生產力，和同樣減少 43%產品進入市場時間。

- Raytheon

使用 COBOL 語言達到 60%之重用率，增加 50%的生產力。

第3章　軟體產品線設計與開放原始碼工具之整合

本論文在此章節提出相對應之軟體產品線架構設計與使用之開放原始碼工具，在之後章節將基於此架構與工具著手系統實作的部分。

3.1 軟體產品線架構設計

基於 2.5 節對於軟體產品線的概念描述，提出圖 3.1 相對應之軟體產品線設計架構，就圖中編號一一描述概念、設計與相關角色之間的關係和將會使用之開放原始碼工具，而開放原始碼工具之介紹將會在下個小節。

I. Management

標明出管理活動的範圍，涵蓋整個軟體產品線之活動，各項活動運行是否良好端看管理活動的支援是否充足。

II. Core Asset Development

標明出核心資產開發活動的範圍，涵蓋軟體產品線之(1) ~ (9)活動，通常由核心資產開發團隊負責維護。

III. Product Development

標明出產品開發活動的範圍，涵蓋軟體產品線之(10) ~ (11)活動，通常由各別產品開發團隊負責維護。

1. User Requirements

相當於核心資產開發活動的輸入端(2.5.2 節)，輸入類型相當多樣，輸入角色也相當多元，如客戶、使用者、中高階主管等，負責角色通常是核心資產開發團隊的 SA(System Analyst)，PM(Project Manager)從旁協助，本論文以使用案例(Use Case)作為需求文件的主軸(Cockburn，2000)；使用 Open Office 為辦公室自動化工具。

2. Domain Model (UML)

基於使用案例使用統一塑模語言(2.4.2 節)為問題領域建造模型，使其產生 Domain-Driven Design(2.3 節)之基礎架構，負責角色仍為核心資產開發團隊的 SA；使用 Jude UML 作為塑模工具。

3. Domain-Driven Design (UML)

基於使用案例和 Domain Model 做出 Domain-Driven Design 設計，負責角色為核心資產開發團隊的 SD(System Designer)，如何設計出穩定性高、具有彈性和易於維護的模型是一項不小的挑戰，在此階段 SD 的經驗和功力相當重要；使用 Jude UML 作為塑模工具。

4. General Architecture Design

基於使用案例和 Domain-Driven Design 設計出相關之系統架構和應用服務，負責角色為核心資產開發團隊的 SD(System Designer)；系統架構方面，通常會將系統切分為 Presentation、Controller、Service 和 Data Integration Tier 等 ；而在應用服務方面，是指 Domain Model 沒有涉及到的應用邏輯，例如「有一個客戶想要為放在購物車的商品建立訂單，但系統要求客戶必須先作帳號和密碼登入，才允許客戶建立訂單。」以上描述，建立訂單是商業邏輯，而系統要求客戶登入則是為應用邏輯(Fowler，2002)；使用 Jude UML 作為塑模工具。

5. Reusable Library Design

為系統經常使用的功能製作或收集公用程式(Utilities)和函式庫(Libraries)，例如系統記錄(logging)、字串處理、讀取檔案等功能，負責角色方面，可由 SD 挑選候選工具，再由開發人員(Developer)繼續做驗證工作；使用網路上有名聲的開放原始碼社群的產品，如 Jakarta Commons API 等。另外要提醒的是，(3)、(4)和(5)的活動會因為設計的考量而產生相依性(Dependency)，(4)會相依於(3)，而(4)和(3)會相依於(5)，也因此在更改其中一項的設計時，也要同盤考量其它的兩個設計架構才是；使用 Jude UML 作為塑模工具。

6. General Architecture Source Code Template

由活動(4)產生出的實作程式碼，程式介面由工具自動產出，由核心資產團隊 Developer 實作應用邏輯，過程中會因為實作考量而回頭修改設計的情形發生；當 Code Template 開發定版 check-in 至 VCS(Version Control System)後，其後再由產品開發團隊 Developer 於 VCS 將 Code Template 原始碼 check-out 出來重複使用；使用 Jude UML 作為塑模工具，並使用開放原始碼 Subversion 工具作為 VCS 使用。

7. Domain Model Source Code

由活動(3)產生出的實作程式碼，程式介面由工具自動產出，由 Developer 實作應用邏輯，過程中會因為實作考量而回頭修改設計的情形發生；當 DDD 開發定版並包裝成 jar 檔，check-in 至 VCS 後，其後再由產品開發團隊 Developer 於 VCS 將 jar 檔 check-out 出來重複

使用；使用 Jude UML 作為塑模工具，並使用開放原始碼 Subversion 工具作為 VCS 使用。使用 Jude UML 作為塑模工具。

8. Reusable Library Source Code

由活動(5)產生出的實作程式碼或是從網路上的開放原始碼社群搜尋可用的工具，自行實作的部分，程式介面由工具自動產出，由 Developer 實作應用邏輯，過程中會因為實作考量而回頭修改設計的情形發生；使用開放原始碼的部分，必須將可執行程式(jar 檔)、原始碼和說明文件一起完整下載並妥善保存；當開發定版並包裝成 jar 檔，check-in 至 VCS 後，其後再由產品開發團隊 Developer 於 VCS 將 jar 檔 check-out 出來重複使用；使用 Jude UML 作為塑模工具，並使用開放原始碼 Subversion 工具作為 VCS 使用。使用 Jude UML 作為塑模工具。

9. Test Code、Production Plan、...etc

在現代化的軟體開發架構中，系統自動化可測試性的要求越來越高，主要是為了在每次 Iteration 中，能夠給予 SD 和 Developer 能夠重構系統的勇氣並有效確保系統的整體品質，因此在本論文的核心資產和產品開發團隊之實作階段中，同時也會開發相對應之測試程式碼，主要是做 Unit Test 和 SIT(System Integrated Test)的部分，UAT(User Acceptance Test)的部分需要有更多的資源投入(Test Team)才有辦法使其自動化，故不在本論文討論範圍中；測試部分使用的工具是開放遠始碼社群的 jUnit 測試框架。

當(6)、(7)和(8)的元件定版後，接著核心資產開發團隊的 SD 或是 Developer 必須就其元件撰寫使用手冊，也就是 Production Plan，便於之後產品開發團隊的 SD 和 Developer 使用；使用的工具有 Open Office 和 Sun Microsystem 的 Java Docs。

10. User Requirements For Specific Product

產品開發團隊的 SA 跟產品 End User 蒐集需求後撰寫需求文件，其中最主要的文件為使用案例，並會同 SD 參考現行定版之核心資產是否能提供 End User 的需求；如可滿足，便走到(11)，如不可滿足，便走到(12)。

11. Product

使用滿足產品需求的核心資產使用組裝式開發方法(Development by Assembly)，順利地把產品組裝完成。

12. Feedback

當在執行(3)、(4)、(5)或是(10)步驟時，經常會發現需求超出原本核心資產的需求文件或是生產線範疇定義，可能是初步規劃時的遺漏亦或是 End User 沒想到等各種原因，如此就會起始回饋流程，此時核心資產開發團隊就必須協同產品開發團隊，去確認此需求是否應該加入核心資產以擴充軟體產品線的範疇。如需加入，就會驅動核心資產進版，之後再由產品開發團隊使用新版的核心資產作組裝式開發；如不加入，則由產品開發團隊自行開發此需求；在此步驟判斷的準則就是要維持軟體產品線規劃的一致性和共用性。使用的工具主要有，Apache Maven 作為專案管理工具(Project Management Tool)和 Subversion 作為版本控制系統。

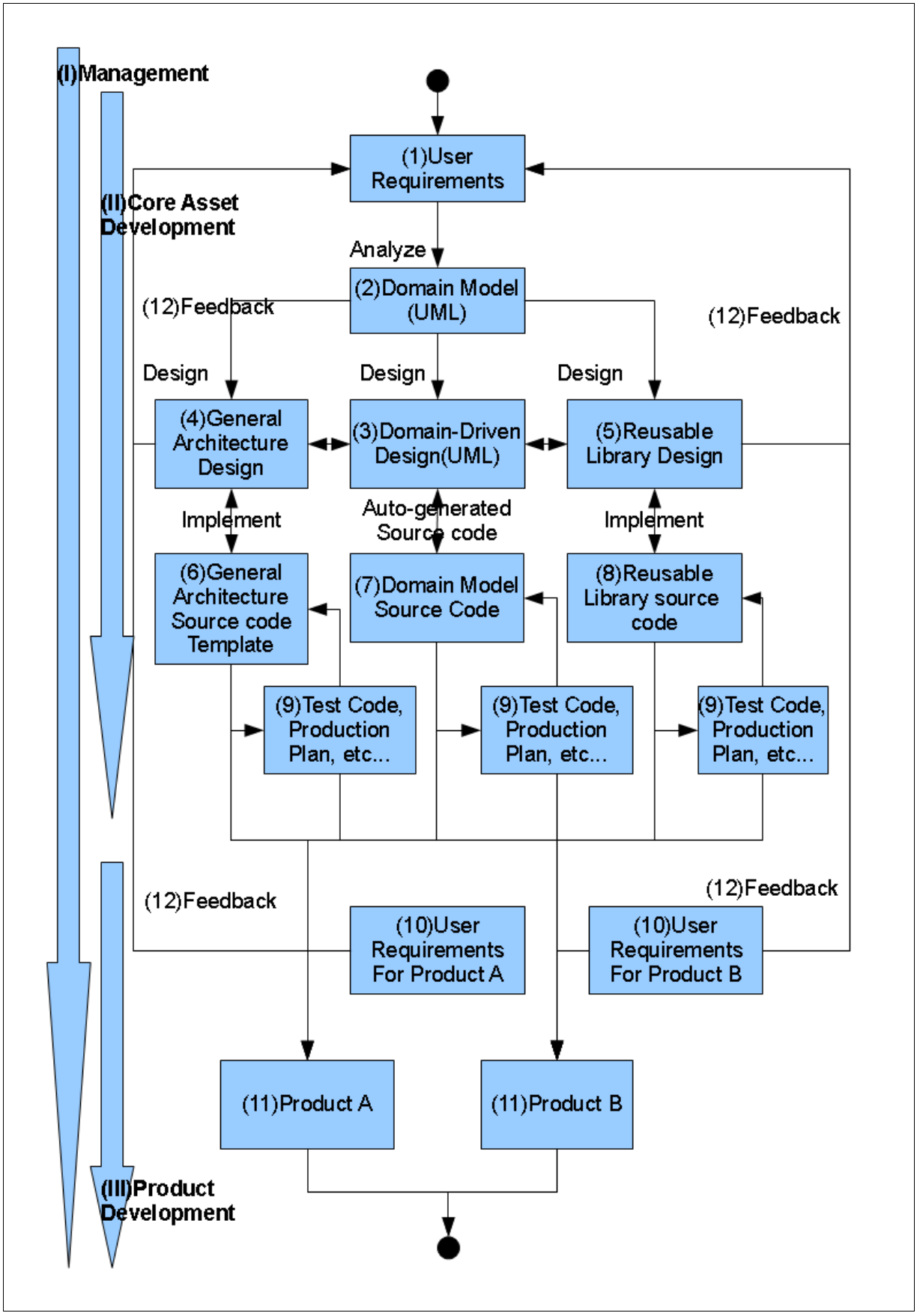


圖 10: 軟體產品線架構設計

資料來源：[本研究整理]

3.2 開放原始碼工具介紹

以下介紹本論文使用到的開放原始碼工具，在第四章設計和實作部分將會使用它們和 3.1 節的架構整合在一起。

3.2.1 OpenOffice.org

OpenOffice.org 同時是產品也是開放原始碼專案，從 2000 年 10 月 13 日開始，OpenOffice.org 1.0 在 2002 年 4 月 30 日釋出 OpenOffice.org 的服務宗旨是要建立國際化辦公室套裝軟體領導品牌的社群，並且可以在任何主要平台運行，可以透過基於開放式元件可程式介面(Open-Component based APIs)存取該軟體所有的功能和資料，和儲存成標準的 XML 檔案格式。OpenOffice.org 專案主要由 Oracle 支持，該公司主要提供專案的程式碼。其他的支援廠商包含 Novell、RedHat、RedFlag CH2000 和 IBM。此外還有超過 450,000 來自世界各地的使用者來加入這個專案，讓 OpenOffice.org 成為任何使用者都可輕易使用的辦公室套裝軟體。這就是開放遠始碼社群的精髓所在(OpenOffice.org，2010)。

目前 OpenOffice.org 的產品有

 Writer ─ 文字處理軟體(Word Processor)，就像 Microsoft Office Word。

 Calc ─ 電子表格軟體(SpreadSheet)，就像 Microsoft Office SpreadSheet。

 Impress ─ 簡報軟體(Presentation)，就像 Microsoft Office PowerPoint。

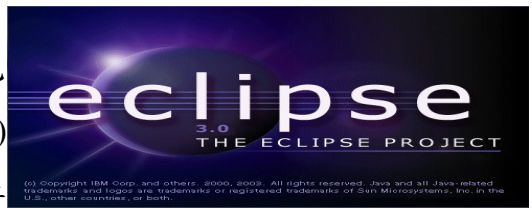
 Draw ─ 就像 Microsoft Office Visio。

 Base ─ 就像 Microsoft Office Access。

 Math ─ 編輯數學運算式的軟體。

在本論文的實作中，將會使用此 OpenOffice.org 來撰寫相關文件，如使用案例、設計文件和 Production Plan 等。

3.2.2 Eclipse IDE



Eclipse 是開放原始碼的多語言軟體開發環境，包含整合開發環境(Integrated Development Environment)和可延伸外掛系統(T. E. Foundation，2011b)。它主要

是支援基於 Java 平台的軟體系統開發，同時也可以加上其它外掛來支援不同的平台和語言的軟體系統開發，包括 Ada、C、C++、COBOL、Perl、PHP、Python、Ruby 等(E. Foundation，2010a)。

在本論文的實作中，將會使用此工具來作為系統開發時的主要平台，如撰寫系統程式碼，並使用它跟其它工具溝通，包含 VCS、Maven、Middle Ware 和 Database 等。

3.2.3 Jude UML

Jude UML 是在本論文中唯一不是開放原始碼專案之工具，它是由日本公司 ChangeVision 開發，本論文使用的是 Jude Community 5.5.2 版本，為免費版本，雖然跟 Professional 版本比起來功能較少，但已滿足本論文設計和實作之需求，功能條列如下(Vision，2011)。

1. Support of UML 2.x
2. Class diagram (Object, Package, Subsystem and Robustness Diagrams are included.)
3. Use case diagram
4. Sequence diagram
5. Collaboration Diagram
6. State diagram
7. Activity diagram
8. Deployment diagram
9. Component diagram
10. Generate Java 1.4 sourcecode from model.
11. Import Java 1.4 source files to create model.

在本論文的實作中，將會使用此工具來描述 Domain Model、Domain-Driven Design、General Architecture Design 和 Reusable Library Design(圖 3.1 (2) ~ (5))，它的功能雖然不像商業 CASE 工具那樣強大(會花很多錢)，同時可做到順向工程和逆向工程，嚴格來說，它只能提供單一順向工程或是單一逆向工程的能力，也就是無法將程式碼和既有 UML 圖作合併；在此論文中的主要使用方式是使用順向工程功能來從 UML 圖產生程式碼骨架，當程式碼實作發生不預期之變動時，再回頭手動更改 UML 圖。

3.2.4 Apache Subversion

Subversion 是開放原始碼的版本控制系統，在 2000 年由 CollabNet 公司建立。在過去的十年它獲得了相當大的成功，現在 Subversion 仍然在開放原始碼和企業應用領域持續耕耘(Subversion，2011)。

3.2.5 Open Foundry

由自由軟體鑄造場(Open Source Software Foundry)開發之網路專案管理平台，OSSF 為政府投資，旨在推廣自由軟體在台灣發展之腳步而創立之組織；Open Foundry 主要的服務如下(自由軟體鑄造場，2011)。

1. 專案開發的面向

   OpenFoundry 在專案開發方面，其本身即為一個供自由軟體愛好者討論、編寫、管理專案內容所設置的共同開發平台，自由軟體的開發人員可以透過此 OpenFoundry 所提供的工具模組建置其自由軟體專案；一般的使用者則可以透過 OpenFoundry 下載所需要的自由軟體，並提供下載程式的改良意見。

2. 資訊匯集的面向

   OpenFoundry 在資訊匯集方面，登錄有本專案的最新動態、自由軟體社群活動、法律授權參考資訊以及自由軟體相關研究報告與教材資源等，可供使用者自行下載參照學習。

3. 人才媒介的面向

   OpenFoundry 在人才媒介方面，收集了國內軟體社群、學界與業界自由軟體相關人才的資訊，延伸介紹台灣自由軟體社群發展的現狀，以整合跨領域的人際網絡與人才資源，並讓使用者可以分享自由軟體活動訊息、開發經驗及個人近況，使用者亦可以透過此平台與國內需要自由軟體人才的廠商聯繫接洽。

3.2.6 Apache Maven

Apache Maven 是開放原始碼的軟體專案管理(Software Project Management)工具，以 POM(Project Object Model)為基礎，Maven 就可以支援軟體專案的建置、報告和文件管理等功能(T. A. S. Foundation，2010c)。

Maven 的主要目標是讓剛加入專案的 Developer 可以在最短的時間內了解軟體專案的整體

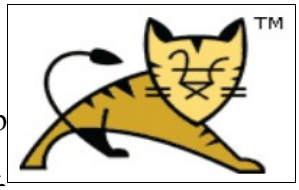
狀態；為了達到這主要目標，Maven 改善以下軟體專案開發的面向

- 讓專案建置更簡單
- 提供統一的軟體專案建置系統
- 提供專案品質資訊
- 提供基於 best practices 的指導原則
- 簡便的昇級和外掛系統

基於以上對 Maven 的功能簡介，它為本論文的實作架構帶來的好處如下

1. 標準的專案開發架構
2. 將不同目的環境的設定資源加以區分(例如 Production 環境和 Test 環境連線的資料庫是不同的)，並協助自動化佈署。
3. 在每次建置專案可執行程式(Build Code)時，能自動化地執行所有測試程式，並且會自動寄發測試報告給專案相關人員，進而保證每次產生的專案可執行程式的品質是在預期範圍。
4. 強制的版本控制，基於 Maven 的專案都被強制給予一版本編號，並且在最後產出的專案可執行程式中，都會將此版本編號附加在其中，也因此專案相關人員都會相當清楚自己使用的可執行程式版本為何，降低混淆不清的情況發生。
5. 解決程式相依性問題，Maven 具有中央控管的程式碼容器，在本論文的實作中，將會有許多的程式相依性存在，例如產品的 2.x 版本相依於核心資產的 1.x 版，而核心資產的 1.x 版又相依於某個框架的 3.x 版，類似的相依性會相當多，使用人為判斷很容易出錯，而 Maven 基於 POM 檔案，可以自動化地幫忙控管此相依性關係。

3.2.7 Apache Tomcat



由 Apache 開放原始碼社群所開發與維護之中介軟體(Middle Ware)，主要提供 JavaEE 規格中 Servlet 和 JSP 所定義的功能，主要包含 Web Container(T. A. S. Foundation，2011a)。本論文將使用 Tomcat 作為主要中介軟體。

3.2.8 H2 DB

100% 基於 Java 平台所開發的開放原始碼關聯式資料庫(Relational Database)，為 MPL 1.1 (Mozilla Public License)和 EPL 1.0 (Eclipse Public License)授權(Engine，2011)；本論文將 H2 DB 用在如下所述的兩個環境。

1. Production 環境，由 H2 DB 起始的一個獨立運作之網路伺服器對系統提供服務，可支援多個連線(Session)。
2. Test 環境，由測試程式碼起始並內嵌在記憶體的一個 H2 DB 程序(Process)，專門對測試程式碼提供服務，這樣作法的好處是(1)可讓測試程式不需依靠外部系統(一個單獨運作的資料庫)，增加測試程式的可靠性和可攜性，(2)測試程式可自行維護測試資料，不需擔心被任何人員修改，在每次測試程式起始時，將測試資料塞入測試資料庫中，在測試程式結束時，就連同測試資料和資料庫程序從記憶體中一起銷毀。

3.2.9 JUunit

目前在開放原始碼社群中基於 Java 平台上最出名的單元測試框架(Unit Test Framework)(JUnit.org，2011)，且市面上，不論是商業授權或是開放原始碼版本的 IDE 都支援其框架，在現今軟體開發方法中，都相當強調自動化測試的重要性(Beck，2003；Wells，2009)，而 JUnit 這樣的測試框架正是自動化測試基石。

在本論文中，除了前端 UI(User Interface)之外，所有自行開發的程式碼都要納入 JUnit 的測試範圍，接著使用 Apache Maven 跟 JUnit 整合，如此每次專案在建置(Build)時，都會執行測試程式並且將結果作成報表，方便開發團隊即時發覺每次專案程式修改是否合乎預期，如有問題也能馬上修改，並且使用自動化測試即時驗證修改結果；如此便能提升專案團隊對於專案程式碼品質提升的意願，勇於在必要時刻進行程式碼重構(code refactor)。

3.2.10 Spring Framework

由 SpringSource 開發之開放原始碼專案，它作為一個輕量級的框架(SpringSource，2011)，目前在 javaEE 領域中可說是相當的熱門，在本論文中主要使用到之功能如下。

1. Inversion of control(Dependency Injection)
2. Aspect-oriented programing

3. Data accessing framework
4. Transaction management framework
5. Model-view-controller framework

3.2.11 Hibernate Framework

Hibernate 是在 javaEE 領域相當有名的 Object-Relation Mapping 開放原始碼框架(J. Community，2011)；在本論文中對於資料庫的存取，是透過 Java Persistence API 作為介面，底層是依靠此框架作管理。

3.2.12 使用工具與軟體產品線開發活動之關係

以下駝峰圖描繪出關於各工具如何被使用在本論文軟體開發活動中的程度，圖中上半部之 X 軸是系統開發生命週期，下半部之 X 軸則是將本論文軟體產品線之開發活動，依照系統開發生命週期作分類及歸納；Y 軸是本論文使用之主要工具；兩者交會處便代表某特定工具在系統開發生命特定週期和本論文特定開發活動之使用狀態。

以 Eclipse IDE 為例，在系統開發初期，處於需求蒐集及確認和需求分析等階段，尚不須撰寫程式碼或建置開發環境，所以使用狀態為無；之後進入系統設計階段，SD 在設計系統時，需考慮到系統架構和設計可行性之確認，通常會使用到 IDE 去確認細部需求，所以使用狀態漸漸提高；之後歷經實作、測試和維護階段，當然 IDE 都脫不了干係。

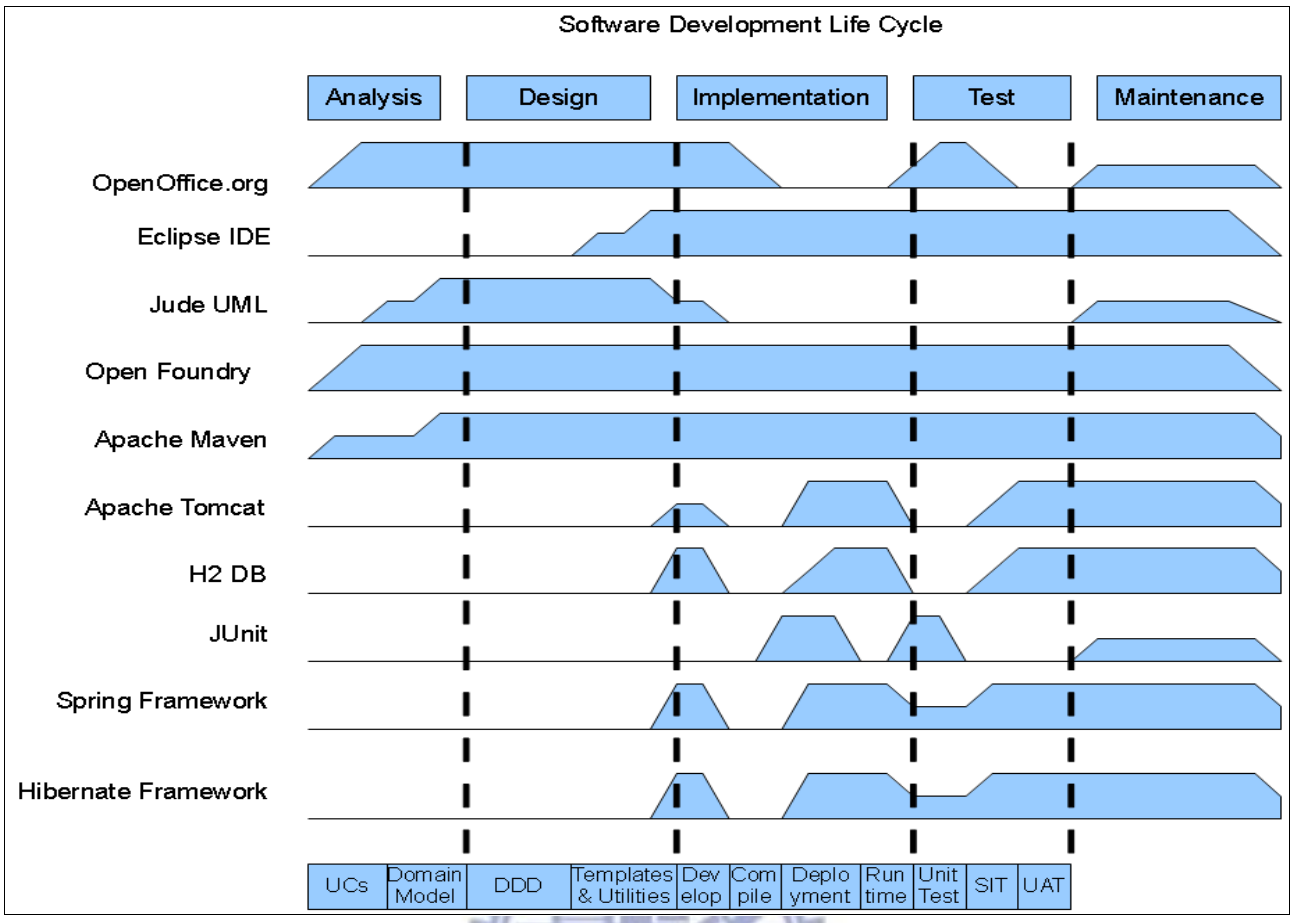


圖 11: 使用工具與本論文軟體開發活動之關係

資料來源：[本研究整理]

第4章　軟體產品線之設計與實作

4.1 背景描述

目標是實作一線上書店網站前後台模組(兩個產品)，其為網路應用程式；前台模組主要是讓線上書店的使用者和顧客能夠方便地做查詢、瀏覽、購買書籍和維護個人資料等作業；後台模組主要是讓線上書店的管理者和店員能夠方便地作書籍上架、促銷、接收顧客訂單和處理顧客抱怨等；其系統實作主要是驗證軟體產品線的可用性，故背景單純設定為一家線上書店、多家出版商和多位線上客戶的交易平台，且此線上書店有自己的書籍庫存。

而核心資產的組成是採取 Top-Down 的方式進行，故將會收集兩個產品的需求並將其精練，萃取出共用的部分(Project Template、Business Logic 和 API Library 等)，以期之後能夠使用組裝式開發來達成的兩個產品之實作。

4.2 軟體產品線之需求

以下描述此軟體生產線欲實作之整體需求。

4.2.1 角色清單

下表列出會與系統直接有互動之角色。

表 2: 核心資產互動角色清單

角色名稱	描述
AnonymousUser	不具名使用者，通常是隨機連線至網站潛在使用者；可瀏覽、查詢網站上架的書籍和註冊成為網站顧客等。
Customer	在網站已有註冊資料且已登入網站表明身分的使用者；可購買書籍、維護個人資料和聯絡客戶服務等。
Clerk	線上書店的店員；線上書店的維護帳號、訂單、書籍資料、庫存等。
Administrator	線上書店的管理員，擁有包含 Clerk 的權限；維護 Administrator 和 Clerk 的權限等。
Carrier	把已訂購書籍宅配到顧客家中或是便利超商的貨運商；貨運商會連線到系統更改訂單資訊，例如顧客已簽收就進系統把訂單 close 掉等。

資料來源：[本研究整理]

下表列出不會與系統有直接互動之利害關係人

表 3: 核心資產利害關係人清單

利害關係人名稱	描述
Publisher	書籍出版商；不會與系統有直接接觸，但系統必須記錄有關他們的資訊。
IT Staff	系統的維護人員；需要系統有良好的 logging 機制，幫助排解 PROD 環境的系統問題。

資料來源：[本研究整理]

4.2.2 使用案例

下表是系統使用案例清單，其中先後順序為分數越大的優先性越高，另外要注意的是分數 100 並不代表比分數 20 還要重要五倍，僅僅只是表示分數 100 比分數 20 要重要(優先)。

表 4: 核心資產使用案例清單

UC 名稱	UC 描述	編號	先後順序
browseBookStore	AnonymousUser 可以使用下列方法瀏覽書店 1.書籍分類 • 中、英文 • 最新書籍 • 書籍銷售排行 2.搜尋	UC-001	100
logInOutBookStore	AnonymousUser 登入/註冊/登出 網路書店 前/後台系統	UC-002	110
maintainCustomerInfo	Customer 維護個人資料	UC-003	85
queryACustomerOrdersInfo	Customer 查詢自己擁有的訂單，顧客無法自己更改訂單狀態，需要通知 Clerk 作修改(UC-006)	UC-004	80
maintainShoppingCart	AnonymousUser 選取有興趣的書籍並加入到購物車中	UC-011	97
placeAnOrder	Customer 選取欲購買之書籍，系統自動加上運費計算總價，並讓使用者選擇付款和配送方式，最後產生訂單。	UC-005	95
payAnOrderByCreditCard	Customer 選擇一訂單且付款狀態為"刷卡未完成"，繼續付款作業。	UC-010	90
updateOrdersStatusByDelivery	Carrier 視送貨單之交貨狀態去更改訂單狀態	UC-006	75
maintainAccounts	Administrator 維護所有角色之帳號	UC-007	40
maintainCustomersOrdersInfo	Clerk 維護所有顧客的訂單	UC-008	65
queryCustomersInfo	Clerk 觀看客戶資訊	UC-009	70

資料來源：[本研究整理]

以下是基於系統互動角色和使用案例清單所勾勒出的使用案例圖，依照 Alistair Cockburn 的使用案例寫法，把每個使用案例區分為不同的 User-Goal Levels(顆粒程度)，藍色為 User-Goal Level(大小適中且應該是比例占最多的使用案例)，靛藍色為 Function(粒度較小)(Cockburn，2000)。

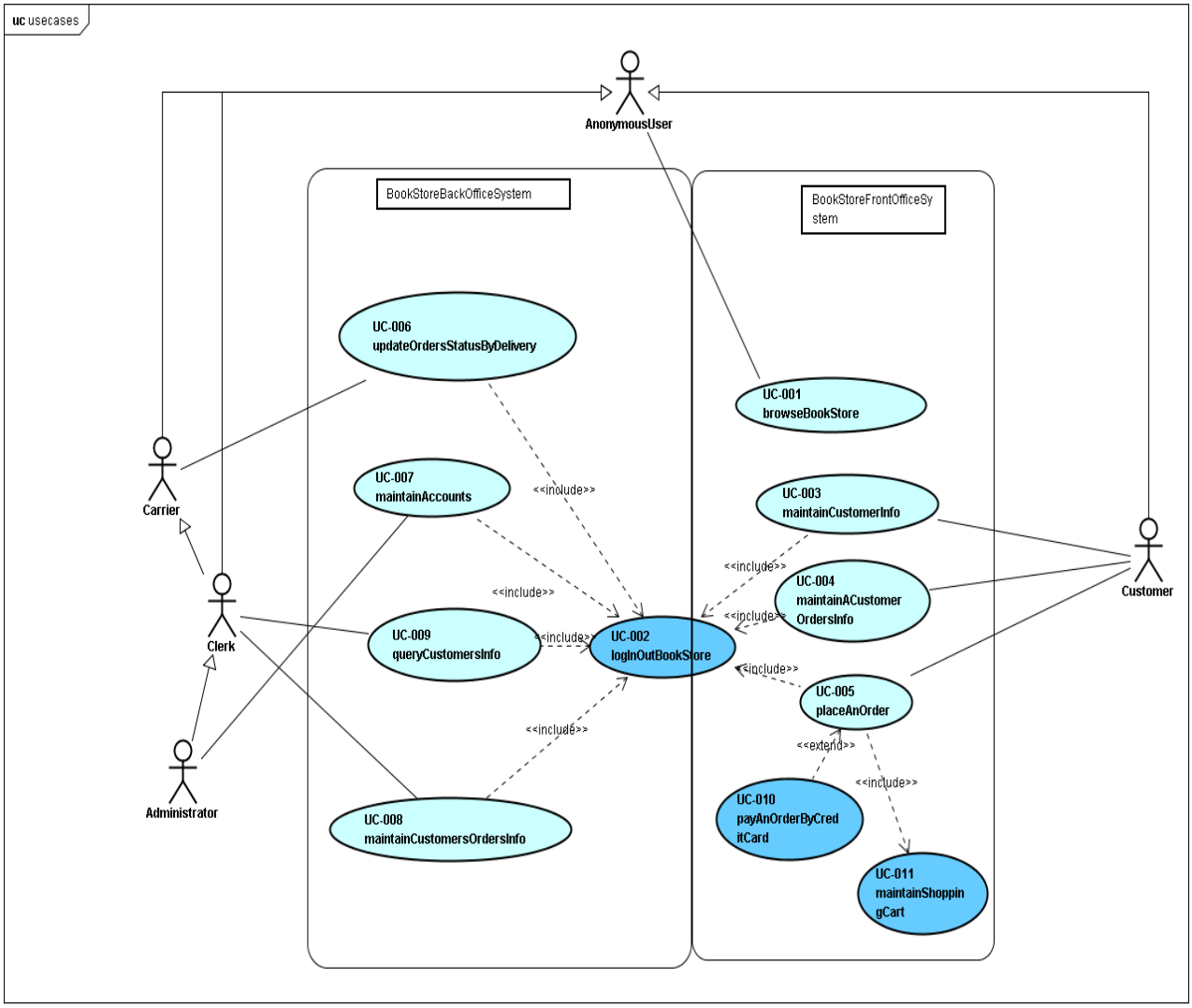


圖 12: 核心資產使用案例圖

資料來源：[本研究整理]

4.2.3 非功能性需求

下表條列出主要非功能性需求

表 5: 核心資產非功能性需求清單

非功能性需求名稱	描述
系統回應時間	要求所有系統作業回應時間要少於 5 秒；如果遇到需要較長的運算時間之案例，需使用非同步的技術，減少使用者等待的時間。

資料來源：[本研究整理]

4.3 核心資產之設計

依照圖 3.1 提出的軟體產品線高階架構設計為藍本，拆分出核心資產相關之小專案，每一

個小專案都有自己在核心資產中需扮演的角色，配置如圖 4.2，在圖 4.2 中可以跟圖 3.1 相互比對，(4)General Architecture Design 的部分包含了一個專案 Bookstore-project-template-spring，(3)Domain-Driven Design 則包含了三個專案 Bookstore-application-service、Bookstore-domain-model 和 Bookstore-domain-model-persistence-jpa，(5)Reusable Library Design 則包含了一個專案 Bookstore-utilities；每個專案都包含相對應的測試程式和文件等，也就是圖 4.2 (9)Test Code, Production Plan 的部分，專案之間的箭頭表示專案之間的相依關係；以下小節針對每個專案作出簡述。

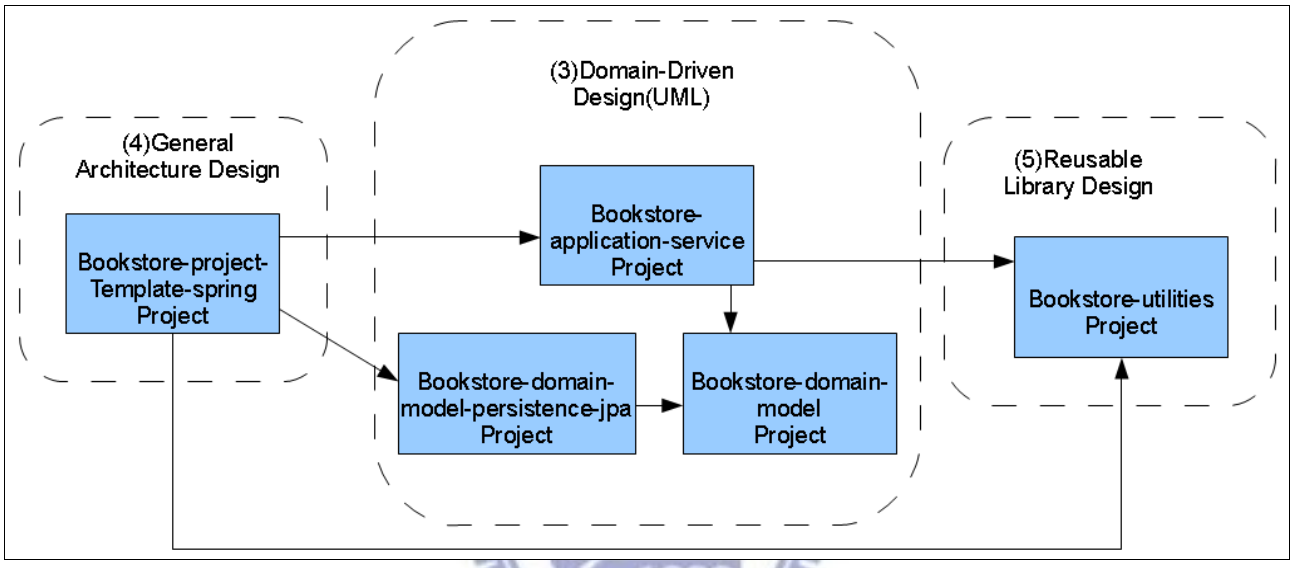


圖 13: 核心資產之專案配置
資料來源：[本研究整理]

4.3.1 Bookstore-project-template-spring 專案

為軟體產品線之下各個產品的專案初始範本，此專案已實作了基本系統需求與技術整合，在此論文欲實作之線上網路書店前台與後台系統共兩個產品，這兩個產品都會使用此專案作為初始範本，再依每個產品之不同需求再作修改；圖 4.3 表示出此專案的技術整合細節。

	Client	Presenation	Business	Domain Model	Integration	Resource
Application	Client UI	Bookstore-project-Template-spring	Bookstore-Application-Service	Bookstore-Domain-Model	Bookstore-domain-Model-Persistence-jpa	DB Schema
Virtual Platform	HTML v4.0	Servlet v2.5 JSP v2.1	Springframework-* v3.0.5	Java-persistence-api v1.0	Java-persistence-api v1.0	SQL/DDL
Upper Platform	Any Browser	Tomcat v6.0.20 JavaSE v1.6.0_17	JavaSE v1.6.0_17		hibernate v3.3.2.GA JavaSE v1.6.0_17	H2 DB JavaSE v1.6.0_17
Lower Platform	Any OS	Any OS				
Hardware Platform	Any PC	Any PC				

圖 14: 專案範本整合之技術
資料來源：[本研究整理]

圖 4.3 使用的是 Sun Microsystem 的 Tiers and Layers Diagram ，Tiers 指的是 Client、Presentation、Business、Domain Model 和 Resource 的欄位，Layers 指的是 Application、Virtual Platform、Upper Platform、Lower Platform 和 Hardware Platform 的列位，Tiers 和 Layers 交集的區域標明出使用之主要技術為何；此專案相依於 Bookstore-application-service 和 Bookstore-domain-model-persistence-jpa 專案。

4.3.2 Bookstore-domain-model 專案

此專案主要用來實現 Domain-Driven-Design 的概念，對於整個軟體產品線之問題領域加以塑模並提供商業邏輯來服務客戶端程式，是整個核心資產中相當中心的部分，所以它不會相依於其他專案，相反地，是其他專案相依於它；基於 4.2 節軟體產品線之需求，使用 UML 設計出如圖 4.4 之模型。

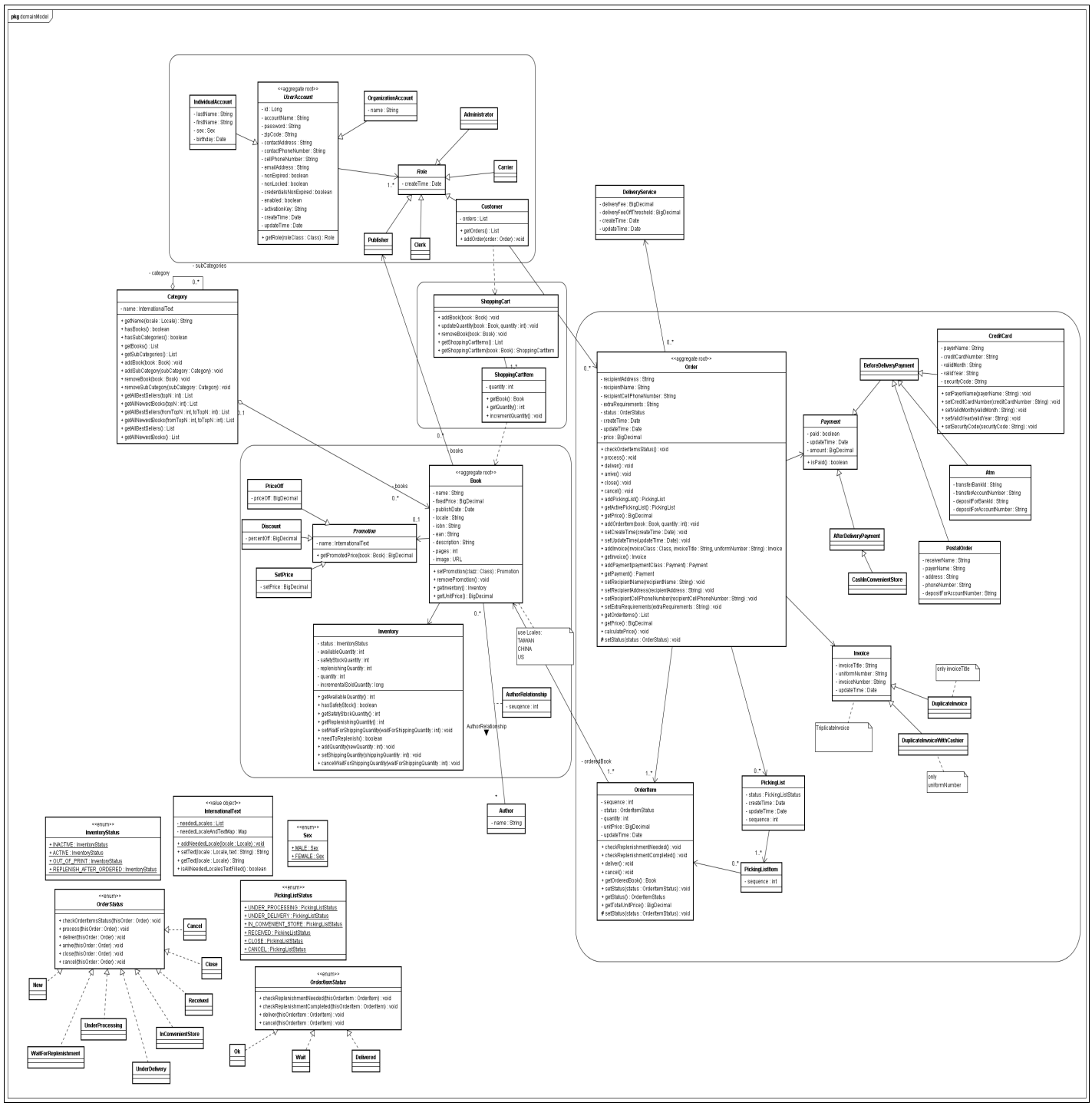


圖 15: 核心資產之領域塑模

資料來源：[本研究整理]

就圖 4.4 的 Class Diagram 可看出，圖中包含了 Entity、Value Object、Aggregate 和 Aggregate Root 等(為了不要讓圖更複雜，Factory 和 Repository 類別沒有顯示出來)，正如 2.3 節之 Domain-Driven Design 所提出之設計規範。從圖中左上角開始再進一步仔細觀察，可看到系統帳號和其登入角色，Customer 角色可擁有多個訂單物件，而訂單物件則是可包含相關之訂單項目(OrderItem)、指定之物流服務(DeliveryService)、付款方式(Payment)、欲購書籍

(Book)、書籍存貨(Inventory)、書籍分類(Category)和提貨單(PickingList)等。

每個類別都有相對應的操作(方法)對客戶端程式提供商業邏輯，當客戶端程式呼叫一個領域物件的操作時，領域物件則透過自己持有的商業邏輯和資料或是委派給相關連之其他領域物件加以處理，也就是一個或是多個領域物件之互動，最後再把處理完成的結果返回給客戶端程式碼；例如一客戶端程式欲計算一個訂單的總價，只要取得特定訂單物件並呼叫它的getPrice()方法，接著訂單物件會去呼叫它所持有的每一個訂單項目物件的 getUnitPrice()方法，取得每一個訂單項目的價錢並加總，最後訂單物件再把加總價格返回給客戶端程式。如此一來，同一軟體產品線底下的相關產品(系統)要實作不同邏輯的服務，只要把領域物件的不同組合給組裝起來，或只是呼叫領域物件不同的方法，即可達到提供不同服務邏輯的目的。

此設計也和 2.3.2 節 Ubiquitous Language 相呼應，圖 4.4 的 Class Diagram 已經表現出此軟體產品線的問題領域的主要概念，只要在加上些許文件或口頭描述，便可以很快地教導軟體產品線的使用者如何使用此核心資產。

為了要增加領域模型的彈性，在設計時也加入了四人幫的 Design Pattern(Gamma、Helm、Johnson、Vlissides，1995)，在此列舉出比較有代表性的設計，圖 4.5 是使用 Composite Pattern 來設計書籍分類的領域物件，如此一來就可以保證書籍分類可設定的廣度(子分類的數量)和深度(子分類的層級)，而且增加或減少任何子分類時，也不需要更改資料庫綱要；圖 4.6 是使用 State Pattern 來處理訂單領域物件的狀態變更邏輯，訂單的生命週期在系統中是很長的，圖 4.7 描繪出從客戶欲購買書籍產生訂單開始，其中歷經書籍存貨檢查、付款檢查、打包出貨、運送和最後到客戶手上為止，其中訂單的狀態會一直變換來反應訂單的真實情況，這些訂單狀態變換時涉及的複雜邏輯和條件就相當適合使用 State Pattern 來處理。

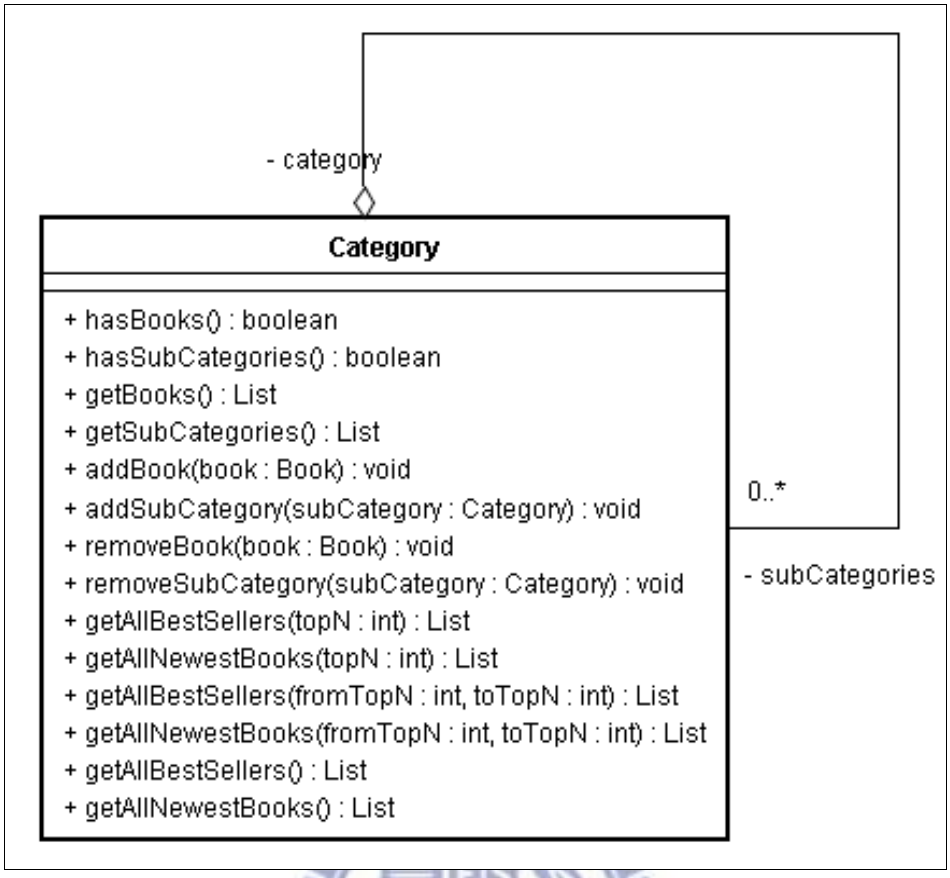


圖 16: 使用 Composite Pattern 設計書籍分類
資料來源：[本研究整理]

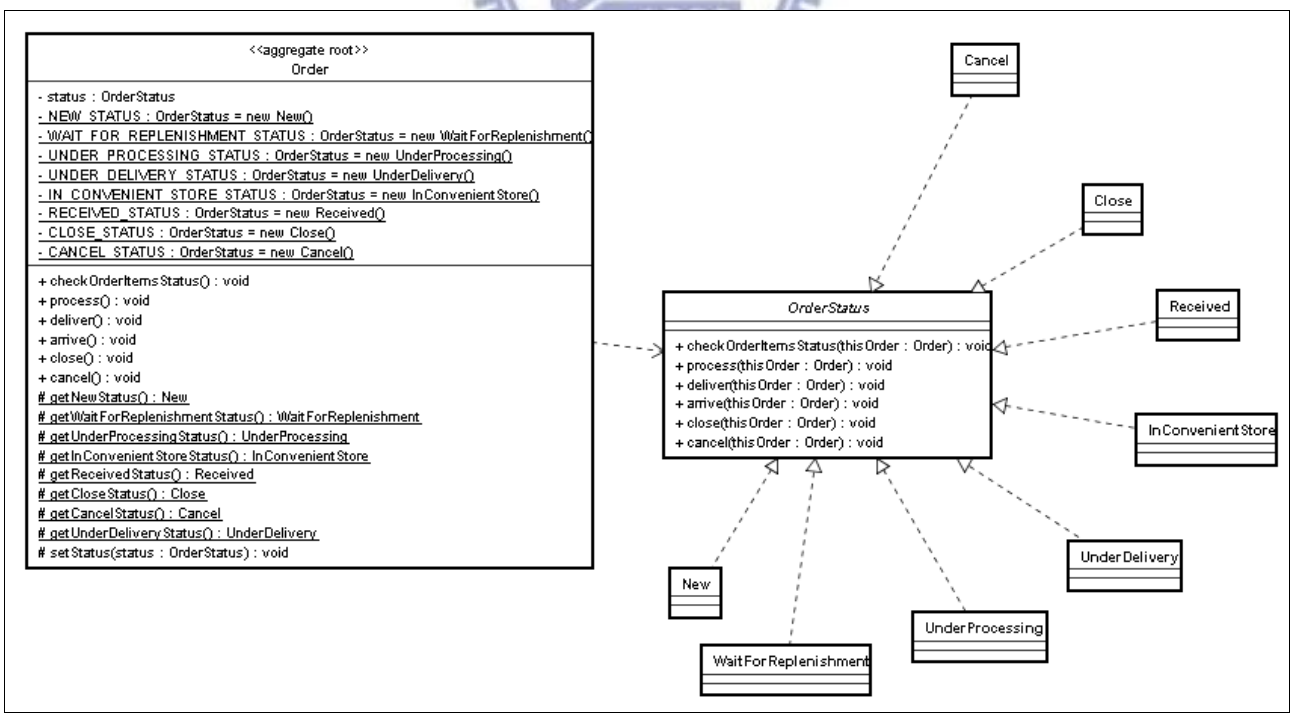


圖 17: 使用 State Pattern 來設計訂單狀態變更邏輯
資料來源：[本研究整理]

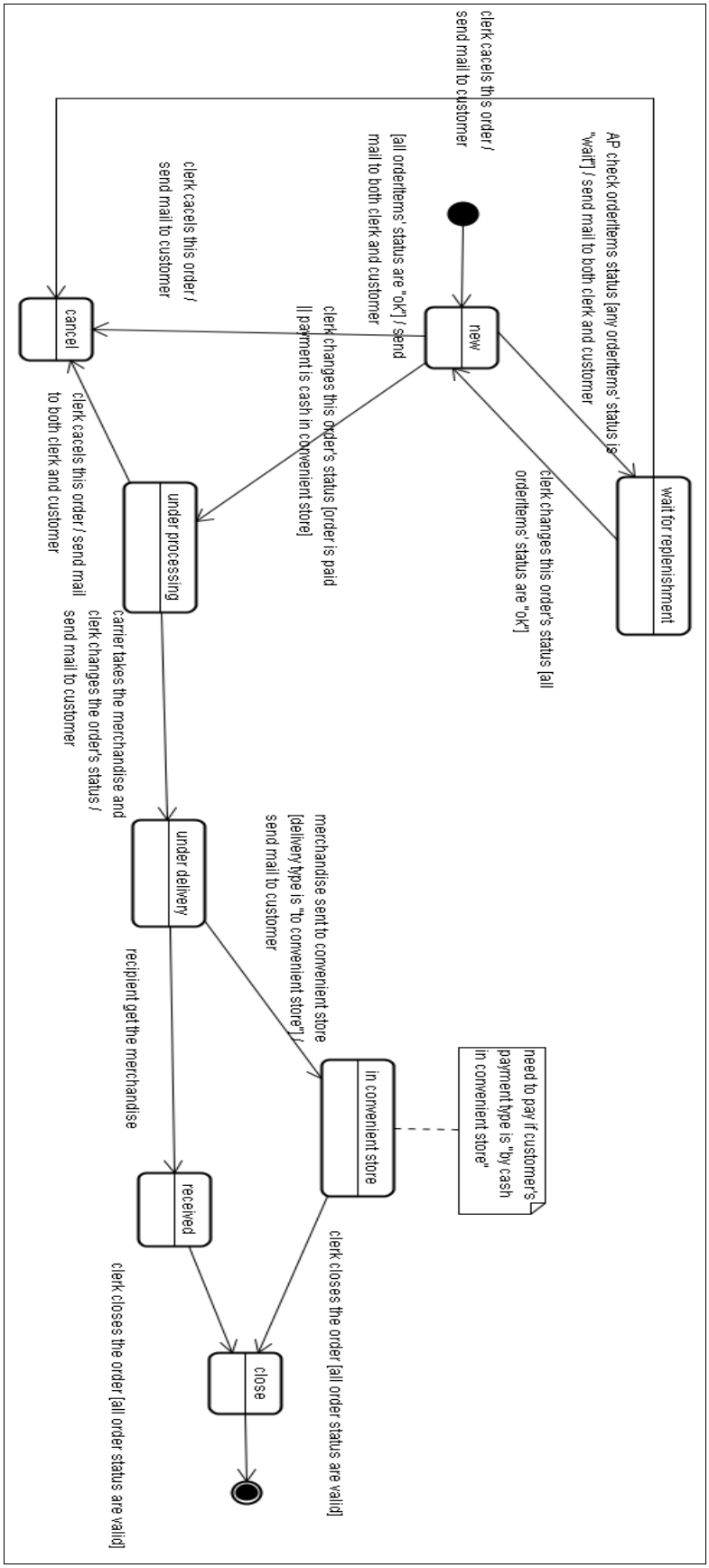


圖　18: 訂單領域物件的狀態圖
資料來源：[本研究整理]

4.3.3 Bookstore-domain-model-persistence-jpa 專案

上一節設計出領域模型，但是當系統實際在運行時，是需要將這些領域物件(資料)給永久保存(persistence)起來，以便系統後續使用，Domain Driven Design 的 Repository 領域物件就是設計一個介面，來將永久保存邏輯和相關實作給切分開來，便於以後可以方便地切換永久保存的實作(現在使用關聯式資料庫，未來會切換成物件導向式資料庫)，而此專案就是永久保存的實作；永久保存所使用的技術有很多種選擇，例如關聯式資料庫、物件導向資料庫、文字檔案或是其它外部系統等，所以需要獨立出一個或多個專案來實作領域模型的永久儲存機制，此專案是使用 JavaEE 的 JPA API (Network，2011) 和 Hibernate 框架 (J. Community，2011)來實作與關聯式資料庫的儲存機制，下圖是使用 Hibernate 框架基於領域物件自動產生出來的關連式模型。

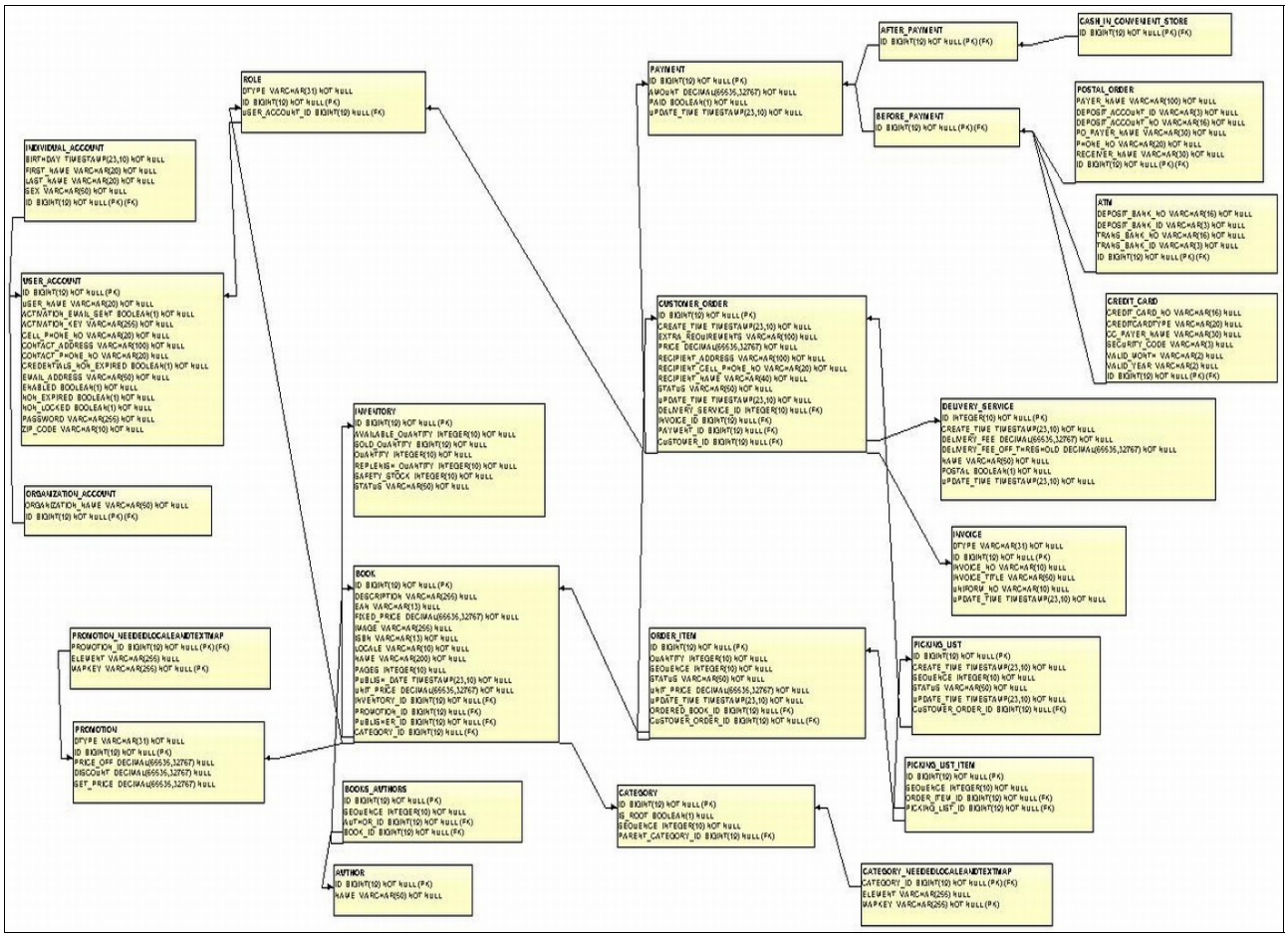


圖 19: 基於領域物件自動化產生之關連式模型
資料來源：[本研究整理]

4.3.4 Bookstore-application-service 專案

系統運作的邏輯有兩種，一個是專門針對問題領域的商業邏輯，另外一種是系統的應用邏輯，在 3.1 節有提到一個例子，在這邊再重申一次『「有一客戶要求當產品訂單改變狀態時，要寄發 e-mail 給相關人員並作系統記錄(log)」，以上描述，產品訂單改變狀態和相關人員為商業邏輯，而寄發 e-mail 和系統記錄則為應用邏輯』，所以前述的領域模型專門是處理問題領域的商業邏輯，而此專案則是建立產品共用的應用邏輯服務，在此專案中目前包含兩個服務，一個是安全性相關的服務，另一個是郵件寄發服務，如圖 4.8。

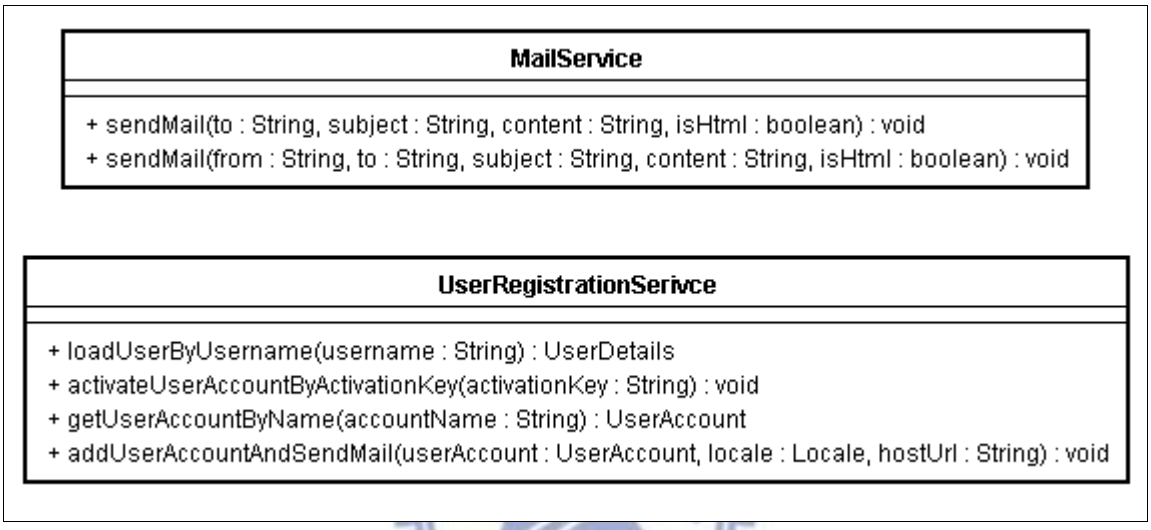


圖 20: 應用邏輯服務
資料來源：[本研究整理]

4.3.5 Bookstore-utilities 專案

此專案是為了建立為此軟體產品線而實作的工具程式，整個軟體產品線所使用的工具程式大多都是直接使用第三方開放遠始碼社群的 API 程式，但仍會有另外再撰寫或是客製化工具程式的需求，這些工具程式是相當中性的，所以另外再開一個專案來專門放置這些自製的工具程式。

4.3.6 系統自動化測試之管理

每一個專案是基於 Maven 來做專案管理的，Maven 預設的執行指令稱之為預設生命週期(Default lifecycle)，預設建構生命週期又包含了執行一連串的階段(Phase)，而每個階段才包含了多個真正執行的動作，稱之為目標(Goal)；表 4.5 整理出預設生建構命週期、階段和目標的關係(T. A. S. Foundation，2010b)。

表 6: Maven 預設生命週期、階段和目標的關係整理

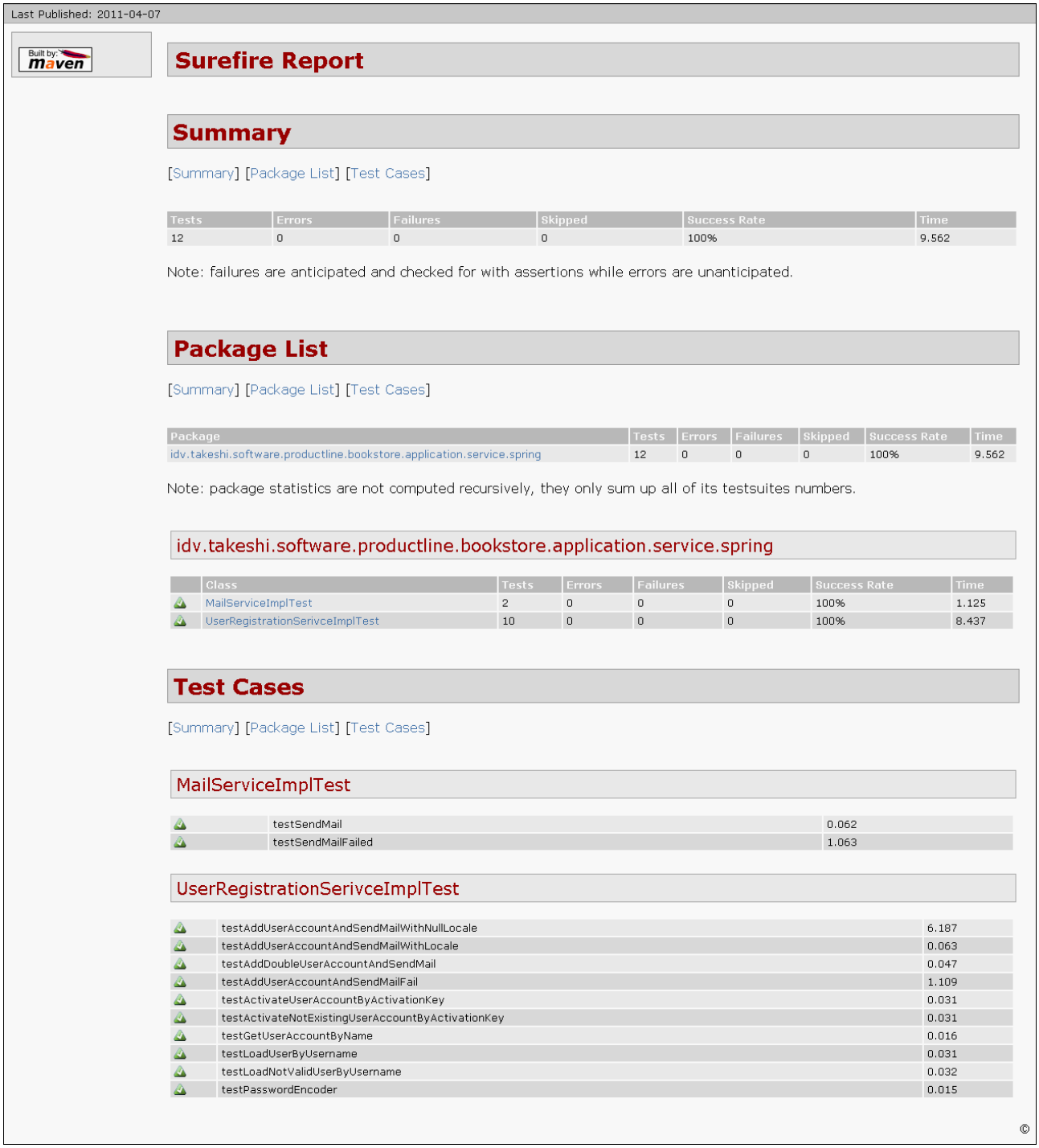
生命週期	階段	目標	描述
Default	process-resources	resources:resources	準備程式資源檔/設定檔
	compile	compiler:compile	編譯原始碼
	process-test-resources	resources:testResources	準備測試程式資源檔/設定檔
	test-compile	compiler:testCompile	編譯測試原始碼
	test	surefire:test	執行測試程式，並確保測試程式100%通過，才會前往下一個階段
	package	jar:jar	將編譯後的可執行檔包裝起來
	install	install:install	將包裝好的可執行檔登錄在本機Maven 容器裡
	deploy	deploy:deploy	自動佈署可執行檔至本機或是遠端伺服器上

資料來源：[T. A. S. Foundation，2010b]

從表 4.5 可看出 Maven 建置可執行檔的預設行為就已經把測試相關行為涵蓋範圍中，換句話說，Maven 已經預設提供了一個自動化測試的時間點和平台，所以專案團隊在執行專案時，就要養成撰寫測試程式的習慣，本論文使用 Junit 測試框架和測試驅動開發(Test Driven Development)方法來支持自動化測試以期提昇系統的品質；此外 Maven 也提供了測試報表的外掛元件(Plugins)，本論文使用的測試報表有兩種，簡述如下。

1. Surefire-Report 外掛測試報表

當 Maven 執行完預設生命週期指令後，再追加 Maven surefire-report:report-only 指令，Maven 便會基於之前的測試結果產生如圖 4.9 的報表，此報表主要是呈現測試程式的測試結果為何，從圖 4.9 Summary 的欄位可看出結果有分為測試的總量、錯誤數量、失敗數量、跳過數量、測試成功率和花費時間等。



Last Published: 2011-04-07

Built by: maven

Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
12	0	0	0	100%	9.562

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
idv.takeshi.software.productline.bookstore.application.service.spring	12	0	0	0	100%	9.562

Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.

idv.takeshi.software.productline.bookstore.application.service.spring

Class	Tests	Errors	Failures	Skipped	Success Rate	Time
MailServiceImplTest	2	0	0	0	100%	1.125
UserRegistrationSerivceImplTest	10	0	0	0	100%	8.437

Test Cases

[Summary] [Package List] [Test Cases]

MailServiceImplTest

testSendMail	0.062
testSendMailFailed	1.063

UserRegistrationSerivceImplTest

testAddUserAccountAndSendMailWithNullLocale	6.187
testAddUserAccountAndSendMailWithLocale	0.063
testAddDoubleUserAccountAndSendMail	0.047
testAddUserAccountAndSendMailFail	1.109
testActivateUserAccountByActivationKey	0.031
testActivateNotExistingUserAccountByActivationKey	0.031
testGetUserAccountByName	0.016
testLoadUserByUsername	0.031
testLoadNotValidUserByUsername	0.032
testPasswordEncoder	0.015

©

圖 21: Surefire-Report 測試報表

資料來源：[本研究整理]

2. Cobertura 測試報表

由 Cobertura 外掛產生的測試報表，使用指令 mvn corbertura:cobertura 指令，maven 會重新執行測試程式並且記錄執行緒所執行到的原始碼段落，最後計算並產生出測試碼涵蓋率報表，從圖 4.10～4.12 可看出此報表威力強大之處，從全部專案到單一類別之原始碼的測試涵蓋率和測試程式執行緒沒有測到的程式區塊等，全都一覽無遺，是提昇自動化

測試品質的好幫手。

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	7	81% 83/102	75% 9/12	1.414
idv.takeshi.software.productline.bookstore.application.service	4	50% 2/4	N/A N/A	1
idv.takeshi.software.productline.bookstore.application.service.spring	3	82% 81/98	75% 9/12	1.545

Report generated by Cobertura 1.9.4.1 on 2011/4/7 下午 3:11.

圖 22: Cobertura 測試報表—全部套件
資料來源：[本研究整理]

Coverage Report - idv.takeshi.software.productline.bookstore.application.service.spring

Package	# Classes	Line Coverage	Branch Coverage	Complexity
idv.takeshi.software.productline.bookstore.application.service.spring	3	82% 81/98	75% 9/12	1.545

Classes in this Package	Line Coverage	Branch Coverage	Complexity
AbstractUserRegistrationSerivce	74% 32/43	80% 8/10	1.615
IndividualAccountRegistrationServiceImpl	100% 21/21	N/A N/A	1.333
MailServiceImpl	82% 28/34	50% 1/2	1.5

Report generated by Cobertura 1.9.4.1 on 2011/4/7 下午 3:11.

圖 23: Cobertura 測試報表—單一套件
資料來源：[本研究整理]

Coverage Report - idv.takeshi.software.productline.bookstore.application.service.spring.MailServiceImpl

Classes in this File	Line Coverage	Branch Coverage	Complexity
MailServiceImpl	82% 28/34	50% 1/2	1.5

```
 1   /**
 2    *
 3    */
 4   package idv.takeshi.software.productline.bookstore.application.service.spring;
 5
 6   import javax.mail.MessagingException;
 7   import javax.mail.internet.MimeMessage;
 8
 9   import org.apache.commons.lang.StringUtils;
10   import org.apache.commons.lang.Validate;
11   import org.apache.commons.logging.Log;
12   import org.apache.commons.logging.LogFactory;
13   import org.springframework.beans.factory.annotation.Autowired;
14   import org.springframework.mail.javamail.JavaMailSender;
15   import org.springframework.mail.javamail.MimeMessageHelper;
16   import org.springframework.stereotype.Service;
17   import org.springframework.transaction.annotation.Propagation;
18   import org.springframework.transaction.annotation.Transactional;
19
20   import idv.takeshi.software.productline.bookstore.application.service.MailService;
21
22   /**
23    * MailService implementation.
24    * @author takeshi
25    *
26    */
27   @Service("mailService")
28   @Transactional(propagation = Propagation.REQUIRES_NEW, rollbackFor = Exception.class)
29 1 public class MailServiceImpl implements MailService {
30
31 1         private static final Log log = LogFactory.getLog(MailServiceImpl.class);
32
33          @Autowired
34          private JavaMailSender mailSender;
35
36          private String fromAddress;
37
38
39          /* (non-Javadoc)
40           * @see idv.takeshi.software.productline.bookstore.application.service.MailService#send(java.lang.String, java.lang.String, java.lang.String, java.lang.String, boolean)
41           */
42          public void send(String from, String to, String subject, String mailContent, boolean html) throws MessagingException {
43 6                 Validate.notEmpty(to, "to address shall always not be null/empty");
44 6                 Validate.notEmpty(subject, "subject address shall always not be null/empty");
45 6                 Validate.notEmpty(mailContent, "mailContent address shall always not be null/empty");
46
47 6                 MimeMessage message = mailSender.createMimeMessage();
48 6                 MimeMessageHelper helper = new MimeMessageHelper(message);
49                  try{
50 6                         if(StringUtils.isNotEmpty(from)){
51 6                                 helper.setFrom(from);
52 6                                 log.debug("from = " + from);
53                          } else{
54 0                                 helper.setFrom(this.fromAddress);
55 0                                 log.debug("from = " + this.fromAddress);
56                          }
57 6                       helper.setTo(to);
58 6                       helper.setSubject(subject);
59 6                       helper.setText(mailContent, html);
60
61 6                       log.debug("to = " + to);
62 6                       log.debug("subject = " + subject);
63 6                       log.debug("html = " + html);
64 6                       log.debug("mailContent = " + mailContent);
65
66
67 6                       this.mailSender.send(message);
68 2                 } catch(Exception e){
69 2                       String msg = "send mail failed";
70 2                       log.error(msg, e);
71 2                       throw new MessagingException(msg, e);
72 4                 }
73 4         }
74
75          /* (non-Javadoc)
76           * @see idv.takeshi.software.productline.bookstore.application.service.MailService#send(java.lang.String, java.lang.String, java.lang.String, boolean)
77           */
78          public void send(String to, String subject, String mailContent, boolean html)
79                          throws MessagingException {
80 6                 this.send(this.fromAddress, to, subject, mailContent, html);
81 4         }
82
83          /**
84           * @return the mailSender
85           */
86          public JavaMailSender getMailSender() {
87 0                 return mailSender;
88          }
89
90          /**
91           * @param mailSender the mailSender to set
92           */
93          public void setMailSender(JavaMailSender mailSender) {
94 0                 this.mailSender = mailSender;
95 0         }
96
97          /**
98           * @return the fromAddress
99           */
100         public String getFromAddress() {
101 0                return fromAddress;
102         }
103
104         /**
105          * @param fromAddress the fromAddress to set
106          */
107         public void setFromAddress(String fromAddress) {
108 1                 this.fromAddress = fromAddress;
109 1         }
110
111 }
```

Report generated by Cobertura 1.9.4.1 on 2011/4/7 下午 3:11.

圖 24: Cobertura 測試報表—單一類別

資料來源：[本研究整理]

4.3.7 系統元件相依性之管理

隨著系統規模越來越大，使用並相依的系統元件也會越來越多，例如框架、函式庫或是外部系統介面等，系統元件相依性的控管也會越發複雜，本論文同樣也是借助 Maven 優秀的相依性控管機制來解決這些問題；在傳統的系統開發中，對於相依性控管的方式是把所有使用到的系統元件全部放在版本控管伺服器中，然後再把使用的系統元件清單列在系統設計文件裡；但傳統的方式通常會遇到以下問題。

- 系統文件與真實系統狀態不一致

這是最常發生的狀況，隨著系統的演進，假如管理者沒有針對文件與系統的一致性作管理，通常到最後文件與系統真實狀態很容易脫節；或是要撰寫的文件太多，也很可能寫錯或是缺漏。

- 系統元件沒有一致的命名規則

通常大部分系統元件不是自己團隊開發的，而是直接使用第三方團隊開發的元件，例如公司的其他團隊或是不同的開放原始碼社群等，由於出處不同，這些元件都有它們自己的命名規則，有些元件只用功能作命名，有些元件只用團隊或專案名稱作命名，而且有時還沒有元件的版本號，這些小細節都會帶來管理的困擾。

Maven 的系統元件相依性控管機制解決了以上問題，在 Maven 的世界裡，每一個系統元件都會有一個唯一的座標，組織名稱 : 元件名稱(通常會帶功能) : 版本序號，這個名稱用來宣告自己專案的座標，並且也用來宣告相依的系統元件座標，這些資訊都寫在一個 pom.xml 檔案裡，pom.xml 裡記錄了有關特定專案的所有資訊，Maven 在執行任何指令都是基於 pom.xml 來運作的；所有的系統元件都有了唯一的名稱，不需要另外撰寫文件來記錄相依元件的清單(以 pom.xml 為準)；在版本控管伺服器裡也不需要特地儲存所有系統相依性元件，因為 Maven 在運行的過程中會透過網路到指定的容器尋找相依性元件(基於唯一的座標)。

圖 4.14 呈現出 Maven 在處理系統相依元件的順序。

1. 使用者下預設生命週期指令給 Maven
2. Maven 讀取專案的 pom.xml
3. Maven 到本機容器尋找相依元件
4. 如找不到，Maven 則到遠端容器尋找
5. 將找到的相依元件載到本機容器裡
6. 將找到的相依元件載到 classpath 裡
7. Maven 建置專案的可執行檔
8. 專案可執行檔建置成功
9. Maven 註冊專案的可執行檔到容器裡

10. 專案可執行檔成功註冊到本機和遠端容器中

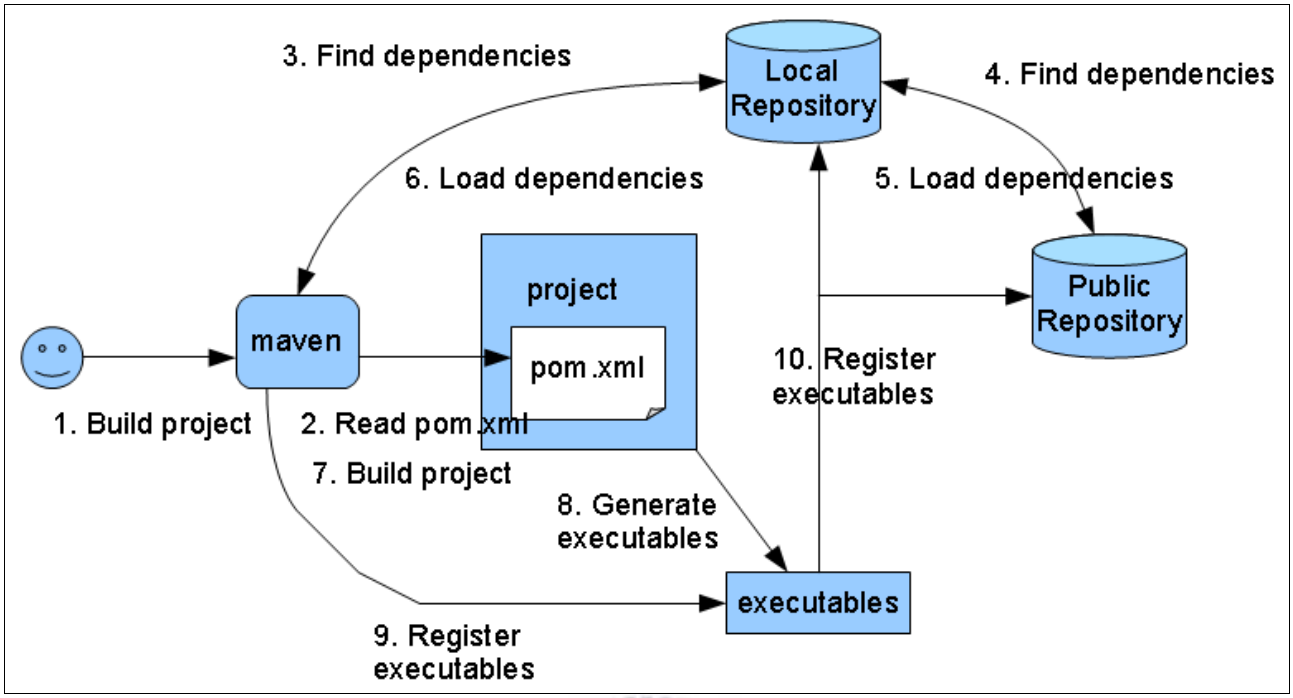


圖 25: Maven 建置專案時對於相依元件的處理順序

資料來源：[本研究整理]

圖 4.15 是使用 IDE 工具透過 Maven 來圖示化系統元件相依性的關係。

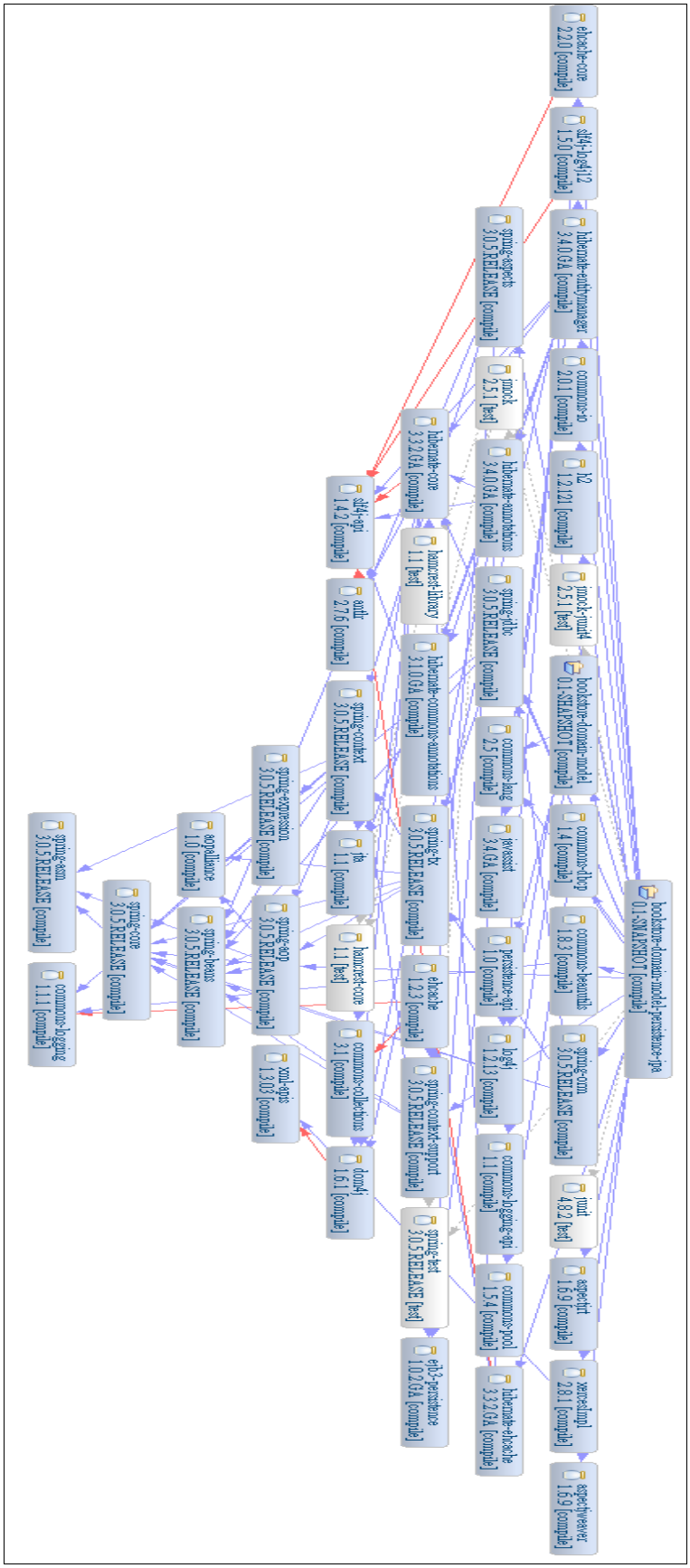


圖 26: 由 Maven 整理系統元件的相依性關係

資料來源：[本研究整理]

4.3.8 系統元件版本之管理

在 2.5 節有提到軟體產品線之中核心資產、產品開發和管理活動是三個不斷反饋持續順向運轉的圈圈，而資訊系統被開發出來並上線之後，後續的生命週期通常也會長達好多年，其間系統需求的變更或增加實屬正常，所以除了系統元件相依性管理之外，當程式碼被更改之後的版本控管也是相當重要的議題，在此節描述如何使用版本控制伺服器和上一節討論的相依性管理，來協同運作達到系統元件版本之管理。

在此先提出一個不同產品在開發時會使用不同版本核心資產的情境，產品 A 在開發時，使用的核心資產為版本 1.0，隨著時間的推移，組織決定要開發產品 B，經分析需求後發現，核心資產版本 1.0 無法完全支持產品 B 所需要的功能，所以組織決定將產品 B 的需求反饋給核心資產，所以核心資產將會進版到 1.1；之後產品 C 也開發出來，但因為核心資產版本 1.1 符合產品 C 的需求，故產品 C 就直接使用了版本 1.1；接著產品 A 因為需求變更而使核心資產版本 1.0 和 1.1 皆不適用，故核心資產又進版至 1.2 來符合產品 A 的需求，歷程如圖 4.15 所示；為了簡化說明，在此情境只針對核心資產版本控管的需求，產品版本控管的概念和核心資產一致，故在此不列入討論。

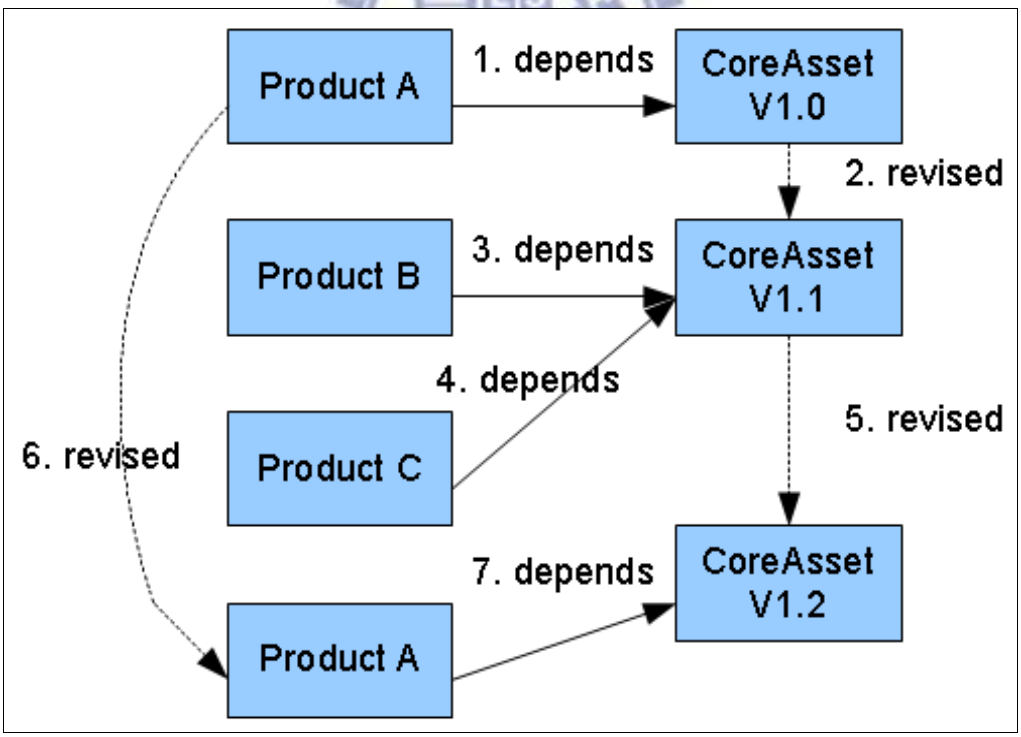


圖 27: 核心資產版本控管情境

資料來源：[本研究整理]

依照上一段提出的情境，使用版本控制伺服器和系統元件相依性管理機制協同運作來達

到版本控管的需求，步驟就如圖 4.17 所示，步驟描述如下。

1. 將開發好的核心資產原始碼版本 1.0 作標籤(tagged)
2. 建置核心資產版本 1.0 的可執行檔並註冊到 Maven 容器裡
3. 產品 A 從 Maven 容器取得核心資產版本 1.0 的可執行檔
4. 因需求回饋，核心資產原始碼版本 1.0 修改成版本 1.1
5. 將開發好的核心資產原始碼版本 1.1 作標籤(tagged)
6. 建置核心資產版本 1.1 的可執行檔並註冊到 Maven 容器裡
7. 產品 B 從 Maven 容器取得核心資產版本 1.1 的可執行檔
8. 產品 C 從 Maven 容器取得核心資產版本 1.1 的可執行檔
9. 因需求回饋，核心資產原始碼版本 1.1 修改成版本 1.2
10. 將開發好的核心資產原始碼版本 1.2 作標籤(tagged)
11. 建置核心資產版本 1.2 的可執行檔並註冊到 Maven 容器裡
12. 因新需求修改產品 A 程式碼
13. 產品 A 從 Maven 容器取得核心資產版本 1.2 的可執行檔

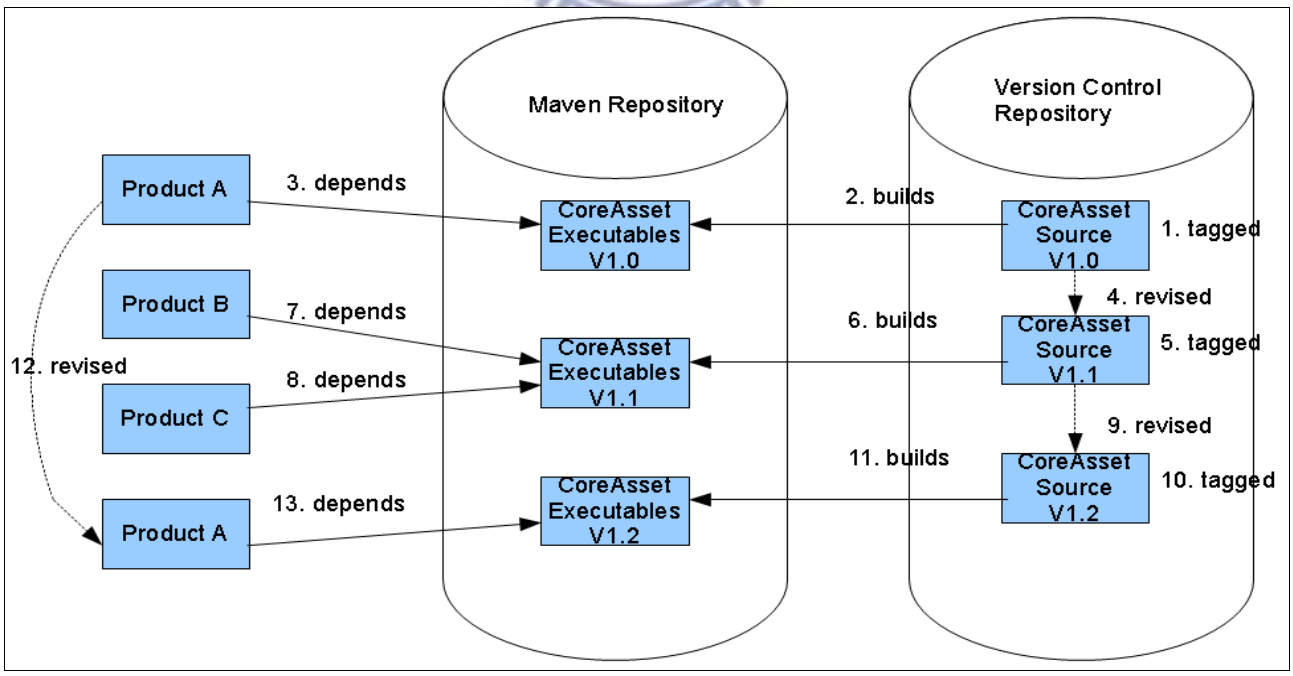


圖 28: 核心資產版本控管機制

資料來源：[本研究整理]

分別由版本控制伺服器管理程式原始碼、Maven 容器管理可執行檔，再加上系統元件相依性控管機制，大家各司其職，如此一來，在新需求回饋給核心資產而導致進版管理所花費的時間可以大大減少，讓軟體產品線越運作越順暢；當然此情境是為了幫助說明其概念而經過簡化，但面對更複雜的情境，版本控管伺服器原本的功能應足以應付。

4.4 產品開發之設計

針對線上書店前台和後台系統作穩健分析(Robustness Analysis)，為系統介面和基於以上小節整理之使用案例和領域物件設計作連結，個別設計圖列於下。

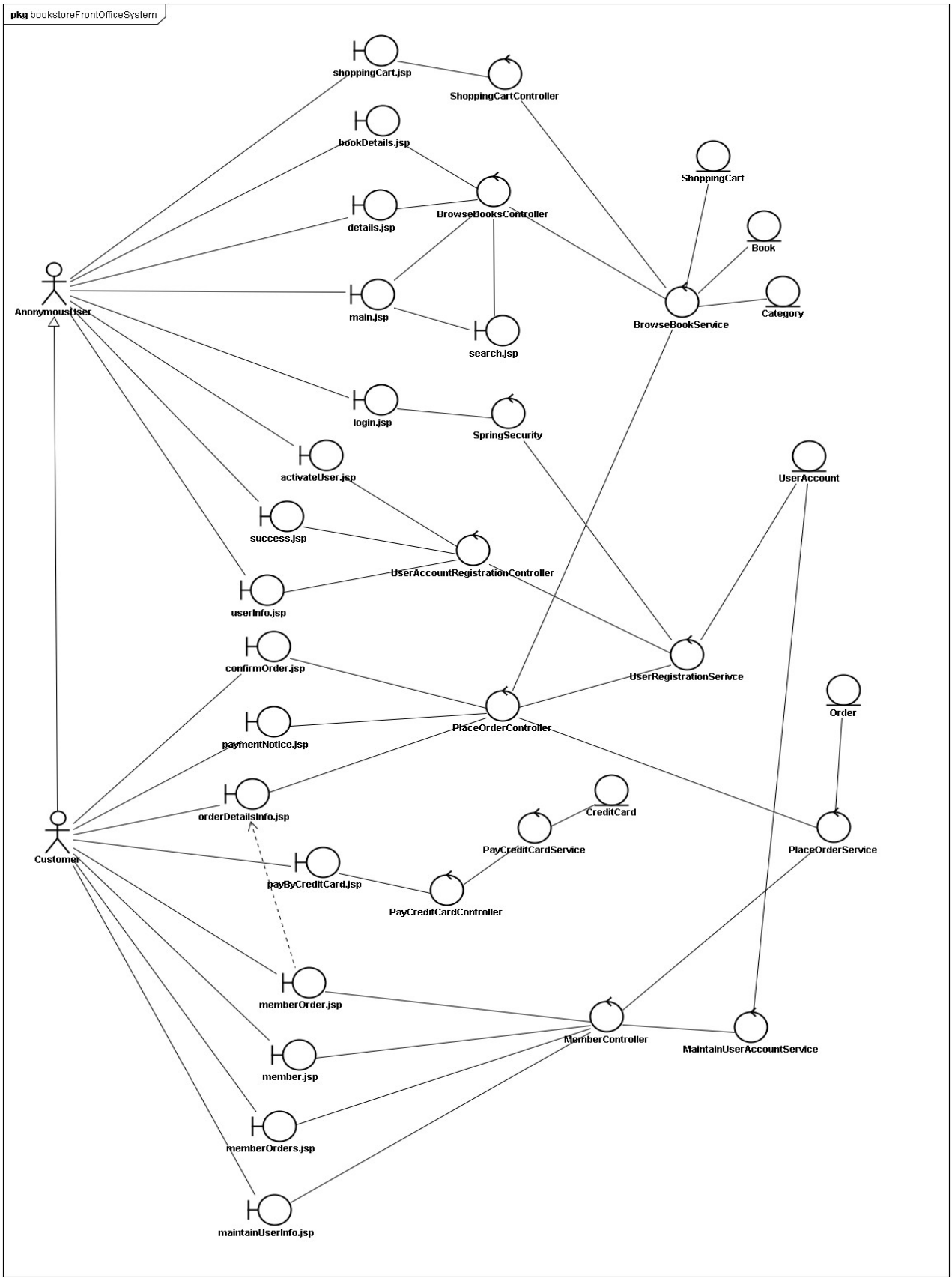


圖 29: 線上書店前台系統之穩健分析圖

資料來源：[本研究整理]

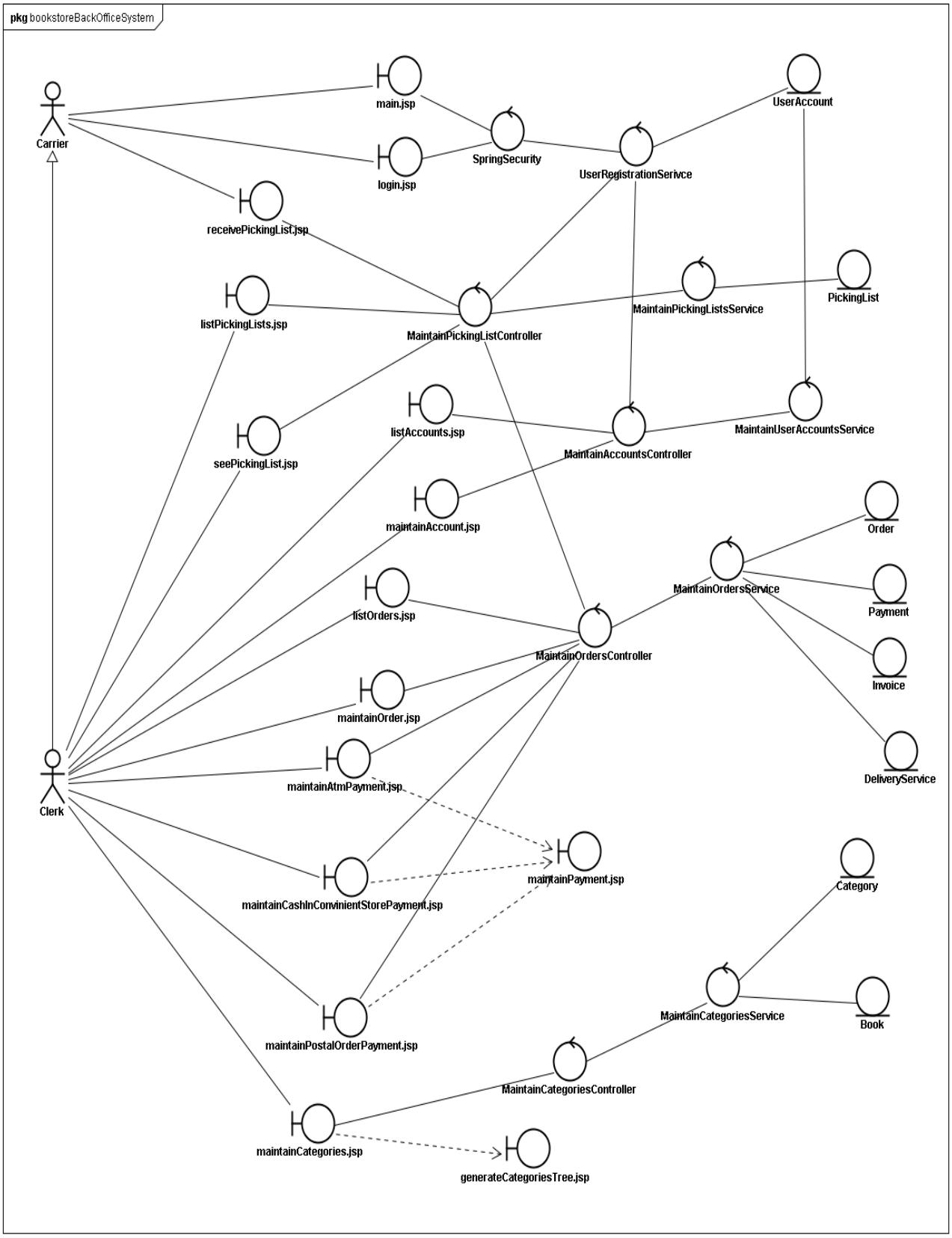


圖 30: 線上書店後台系統之穩健分析圖

資料來源：[本研究整理]

以下是線上書店前/後台系統的佈署圖，由下圖可看出系統的佈署環境相當單純，目標是可以讓內、外部使用者可以使用任何廠牌之瀏覽器，透過網路來操作系統。

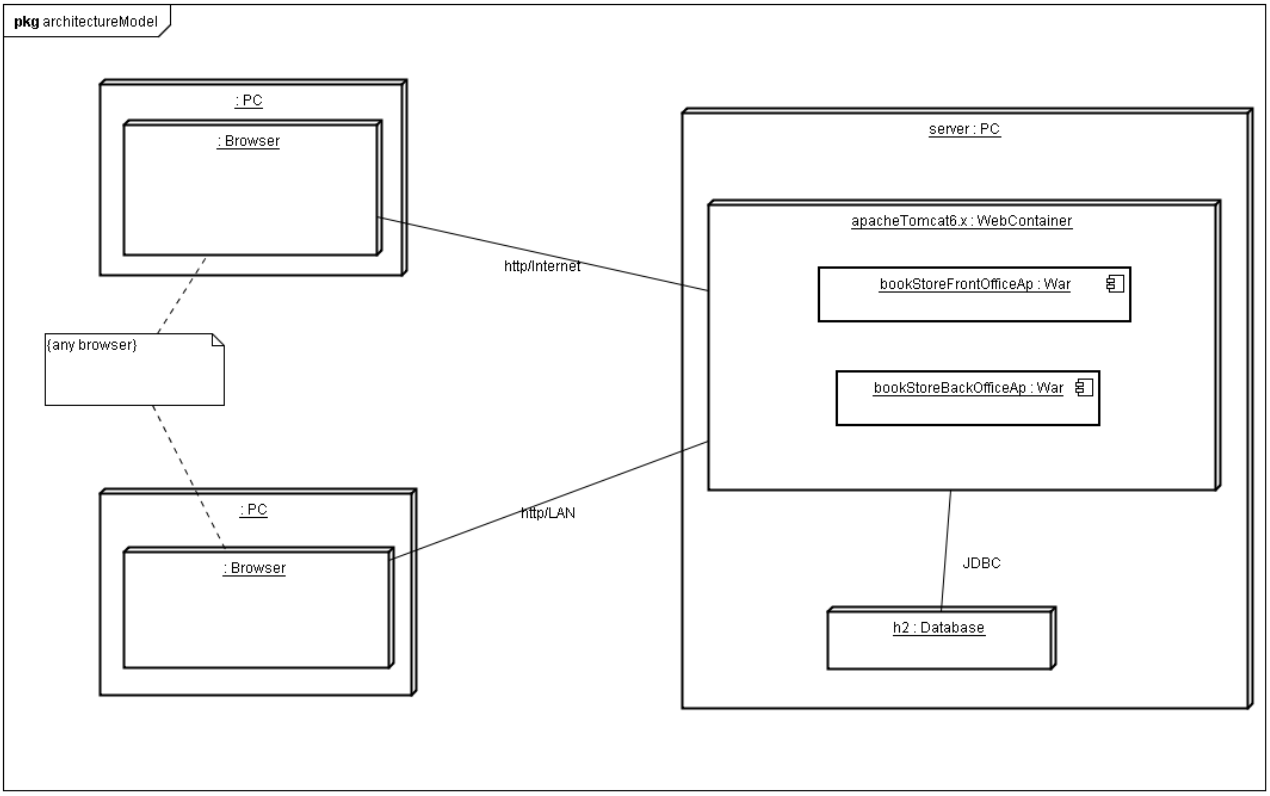


圖 31: 線上書店系統之佈署圖
資料來源：[本研究整理]

4.5 系統實作

系統實作有線上書店前台系統和線上書店後台系統，系統畫面分別如下。

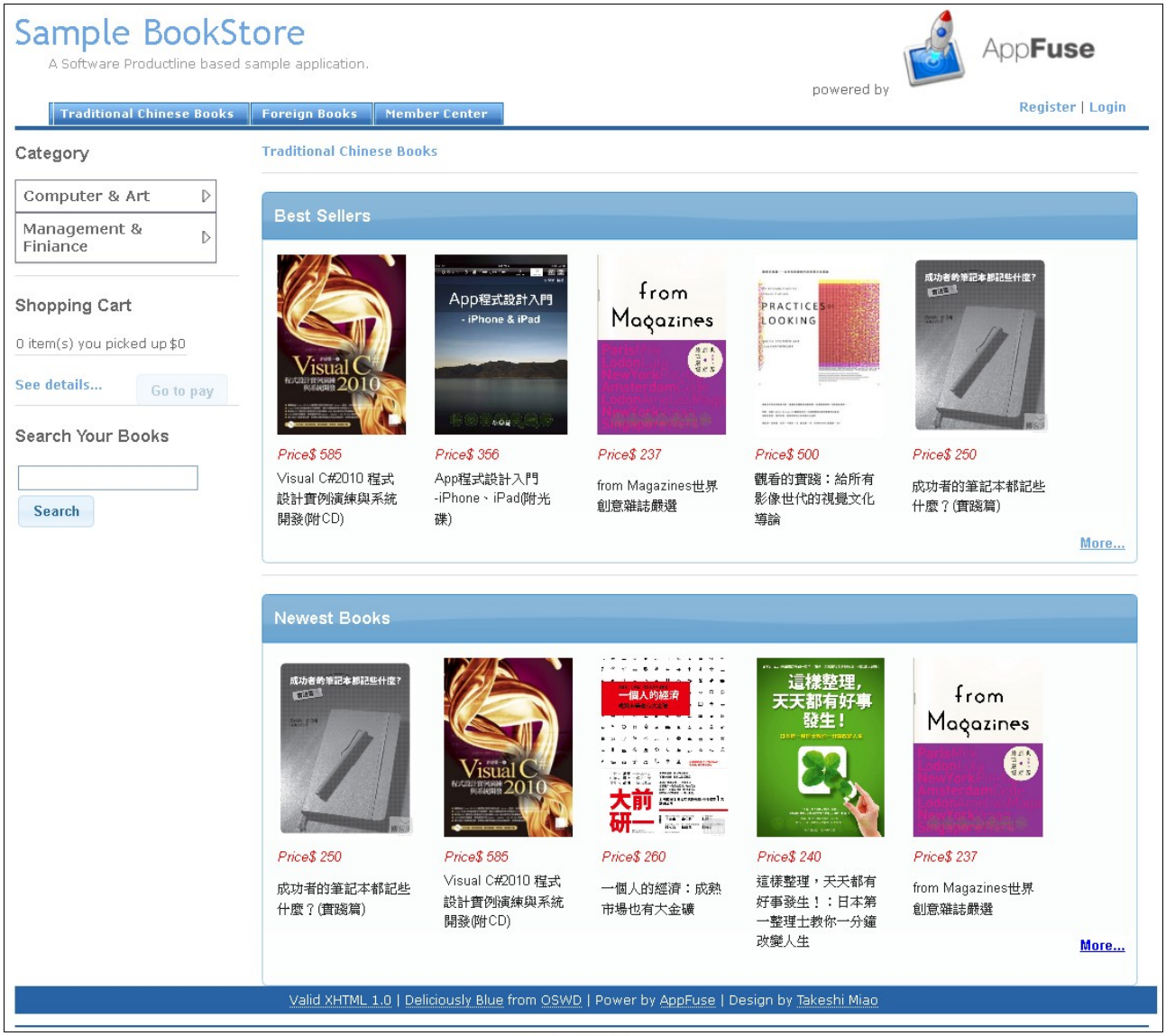


圖 32: 線上書店前台系統畫面

資料來源：[本研究整理]

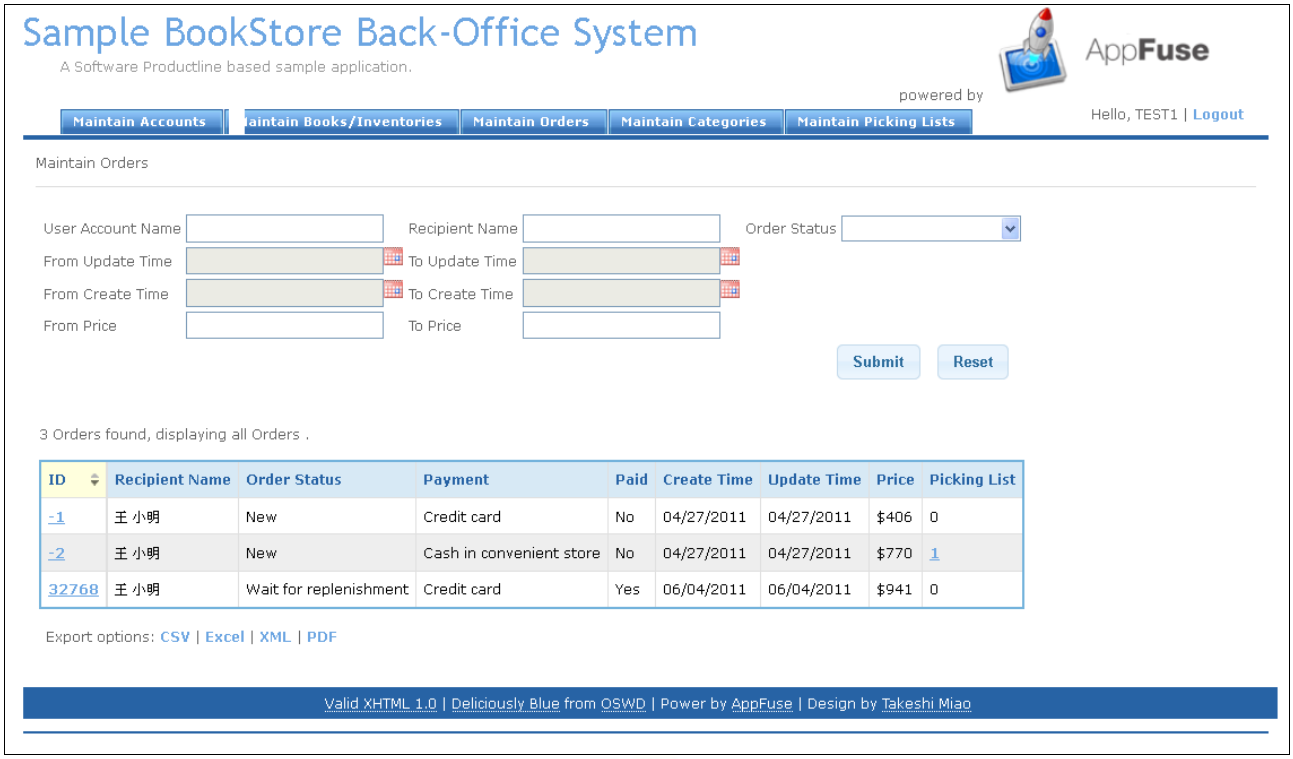


圖 33: 線上書店後台系統畫面

資料來源：[本研究整理]

以上系統實作之專案放置在自由軟體鑄造場之環境中，URL 為 http://www.openfoundry.org/of/projects/1924。

第5章　結論與建議

5.1 結論

從以下五個角度來驗證此軟體產品線的效益，有兩個屬於量化角度，其他三個為質化角度。

5.1.1 量化角度

1. 軟體元件

為核心資產的主要元件，這些軟體元件經過詳細的設計，可以在不更改軟體元件的程式碼下，經由繼承、多型和設計樣式的幫助，不斷的被旗下產品重複使用，當重覆使用的次數越多，軟體元件的單位開發成本將會越來越低，以達到範疇經濟的目標；以下是此論文實作之軟體元件重用率。

下表是各專案之程式碼統計。

表 7: 各專案之程式碼統計

Project	LOC
bookstore-domain-model	9002
bookstore-domain-model-persistence-jpa	1682
bookstore-utilities	352
bookstore-application-service	594
Bookstore-project-template-spring	863
bookstore-front-office	4364
bookstore-back-office	5606
Total	22463

資料來源：[本研究整理]

下表是各產品 RSI 統計。

表 8: 各產品RSI統計

Product	Domain-specific RSI	Application-specific RSI	Utilities RSI	New or changed LOC	Total LOC
bookstore-front-office	9693	577	253	4364	14887
bookstore-back-office	8588	358	99	5606	14651
Total	18281	935	352	9970	29538

資料來源：[本研究整理]

下表是重用率、重用節省成本和投資報酬計算結果，ADC 是$624,650。

表 9: 重用率、重用節省成本和投資報酬計算結果

Product	RSI	Reuse%	RCA	ROI
bookstore-front-office	10523	70.7%	$947,070	$322,420
bookstore-back-office	9045	61.7%	$814,050	$189,400
Total/Avg.	19568	66.2%	$1,761,120	$511,820

資料來源：[本研究整理]

由以上計算結果可得知，本論文軟體產品線實作之平均重用率達 66.2%、總重用節省成本為$1,761,120 和總投資報酬為$511,820。可以跟 2.8.3 節 Poulin 收集之案例比較，66.2%之重用率算是在適當範圍內。

2. 測試案例、測試計畫和測試資料

核心資產中的軟體元件之釋出版本，都是經過自動化可回歸測試和達到一定程度的測試涵蓋率的標準；當旗下產品使用這些核心資產時，都可確保這些軟體元件都是品質良好，可靠度高的元件，故省去不少撰寫測試程式和 debug 的大量時間。

下表是此論文實作之軟體元件測試結果和測試涵蓋率，再次提醒測試涵蓋率包含持久層、商業邏輯層和服務層，但不包含使用者介面層級(Controller、jsp 和 html 等)；而專案 Bookstore-project-template-spring 是屬於架構範本，使用之商業邏輯都是從其他核心資產而來，故無需測試。

表 10: 實作之軟體元件測試結果和測試涵蓋率

Project	Test result rate(%)	Avg. test coverage rate(%)
bookstore-domain-model	100.0%	91.0%
bookstore-domain-model-persistence-jpa	100.0%	91.0%
bookstore-utilities	100.0%	80.0%
bookstore-application-service	100.0%	82.0%
Bookstore-project-template-spring	0.0%	0.0%
bookstore-front-office	100.0%	94.0%
bookstore-back-office	100.0%	90.6%

資料來源：[本研究整理]

5.1.2 質化角度

1. 系統架構

在系統分層的開發概念之下，系統架構本身跟欲解決領域問題之軟體元件是可以切分開來做設計的 ，所以系統架構比較不會受到問題領域的影響而產生較大的變動，故可當作範本來給旗下產品稍作修改便可重複使用，本論文的 Bookstore-project-template-spring 專案就是可重複使用的系統架構和組態檔設定範本。旗下產品在開發時，使用系統架構範本可以幫助快速建置專案，進而規範旗下產品系統架構。

2. 工具和流程

建置核心資產時的相關工具和流程，這些工具和流程都已被驗證過，可以直接套用在旗下產品開發，故可幫助團隊快速進入開發狀態，進而規範旗下產品之工具使用和流程。

3. 人員技能和訓練

當一人員開發旗下產品時，實際上就是接觸整個軟體產品線的核心資產、工具和流程等，所以此人員被調換到旗下之不同產品，都可以快速上手；人員只要訓練一次，對整個產品線都可快速進入生產狀態，有效降低人員訓練成本和提昇生產效率。

5.2 建議

本論文的建議條列如下。

1. 由上而下和由下而上之軟體產品線建置

本論文實作軟體產品線之方法是採取由上而下的建置方法，但實務上在企業裡會有很多舊有系統，將舊有系統合併到軟體產品線是由下而上的建置方法，這無疑是一個相當大的挑戰！例如，如何避免影響到已上線系統之運作、如何有效萃取舊有系統之商業邏輯、如何轉移大量既存之商業資料並確認資料完整性等；這些都是和由上而下的軟體產品線之建置方法相當不同並值得研究的議題。

2. 使用 MDA(Model Driven Architecture)開發

本論文開發領域模型是採取單一順向工程的方式，也就是從 UML 模型產生程式碼，在實作時發現有設計不足的地方，是手動修改程式碼和 UML 模型的，這個缺口需要人為審核，不然有可能造成 UML 模型和程式碼不一致的情形發生。採取 MDA 或許是個不錯的選擇，目前市面上的商業 MDA 軟體相當多元，很值得研究。

3. 不同語言與平台之選擇

本論文實作是採用 javaEE 之標準平台，目前企業級資訊系統開發領域之平台和語言可謂是百家爭鳴，甚至有組織提出在單一系統中使用多語言開發的構想 ；對軟體產品線有興趣的同好，可仔細評估建置環境的需求與限制，來選擇更適合的語言和平台，有機會達到更好的效益。

4. 資源考量

本論文實作沒有把時程和人力資源列入考量，核心資產之規劃和設計是需要經驗和不斷學習的，故組織如要建置軟體產品線，需要考量軟體產品線開發團隊的能力，是否能開發出足夠抽象化、重用性高的核心資產，不然可能因此而讓建置核心資產的功夫付之一炬！

英文參考文獻

1. Balasubramanian, K., Gokhale, A., Karsai, G., Sztipanovits, J., Neema, S.(2006). Developing applications using model-driven design environments. *Computer*，*39*(2), 33-40.
2. Beck, K.(2003). *Test-driven development: By example*：Addison-Wesley Professional.
3. Clements, P., Northrop, L.(2002). *Software product lines - practices and patterns*：Addison Wesley.
4. Cockburn, A.(2000). *Writing effective use cases*：Addison-Wesley Professional.
5. Eckel, B.(2002). *Thinking in java*(3 版)：Prentice-Hall.
6. Evans, E.(2003). *Domain-driven design: Tackling complexity in the heart of software*：Addison-Wesley Professional.
7. Fowler, M.(2002). *Patterns of enterprise application architecture*：Addison Wesley.
8. Gamma, E., Helm, R., Johnson, R., Vlissides, J.(1995). *Design patterns: Elements of reusable object-oriented software*(206 冊)：Addison-wesley Reading, MA.
9. Group, O. M.(2010). Uml superstructure specification, v2.3. 在 O. M. Group 編著：Object Management Group.
10. Poulin, J. S. (1997). *Measuring Software Reuse: principles, practices, and economic models*: Addison-Wesley.
11. Richardson, C.(2006). *Pojos in action - developing enterprise applications with lightweight frameworks*：Manning.
12. Sun Microsystems, I.(2008). *Thejavaee5tutorial for sun java system application server9.1*：Sun Microsystems, Inc..

中文參考文獻

1. Fowler, M.（2005）。*Uml* 精華第三版 - 標準物件模型語言（趙光正譯）：碁峯資訊。
2. ThoughtWorks 公司（2009）。軟件開發沉思錄 *thoughtworks* 文集（ThoughtWorks 中國公司譯）：人民郵電出版社。
3. 吳信輝（2005）。細說「軟體工廠」概念（一）。資訊話題，*2125*，198。
4. 吳信輝（2006）。細說「軟體工廠」概念（十一）。資訊話題，*2209*，72。

參考網頁

1. Community, D.-D. D.(2009). What is domain-driven design?. 檢自: http://domaindrivendesign.org/resources/what_is_ddd
2. Community, J.(2011). Hibernate - relational persistence for java and .Net. 上網日期: 2011. 檢自: http://www.hibernate.org/
3. Engine, H. D.(2011). H2 database engine. 上網日期: 2011. 檢自: http://www.h2database.com/html/main.html
4. Foundation, E.(2010a). About the eclipse foundation. 上網日期: 2010. 檢自: http://www.eclipse.org/org/
5. Foundation, T. A. S.(2010b). Apache maven project. 上網日期: 2011. 檢自: http://maven.apache.org/
6. Foundation, T. A. S.(2010c). What is maven. 上網日期: 2010. 檢自: http://maven.apache.org/what-is-maven.html
7. Foundation, T. A. S.(2011a). Apache tomcat. 上網日期: 2011. 檢自: http://tomcat.apache.org/
8. Foundation, T. E.(2011b). The eclipse foundation open source community website. 檢自: http://www.eclipse.org/
9. JUnit.org(2011). Junit.Org. 上網日期: 2011. 檢自: http://www.junit.org/
10. Network, O. T.(2011). Java persistence api. 上網日期: 2011. 檢自: http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html
11. OpenOffice.org(2010). About openoffice.Org. 上網日期: 2010. 檢自: http://about.openoffice.org/index.html
12. SEI(2010). Software product lines. 檢自: http://www.sei.cmu.edu/productlines/
13. SpringSource, a. d. o. V.(2011). Springsource.Org. 上網日期: 2011. 檢自: http://www.springsource.org/
14. Subversion, A.(2011). Apache subversion. 上網日期: 2011. 檢自: http://subversion.apache.org/

15. Vision, C.(2011). Astah* community - free uml modeling tool. 檢自: http://astah.change-vision.com/en/product/astah-community.html

16. Wells, D.(2009). Extreme programming: A gentle introduction - code the unit test first. 上網日期: 2011. 檢自: http://www.extremeprogramming.org/rules/testfirst.html

17. 自由軟體鑄造場 (2011). 關於 openfoundry & ossf. 上網日期: 2011. 檢自: http://www.openfoundry.org/tw/about

附錄 1　實作系統之重用率量測報告細節

bookstore-domain-model 專案 LOC 統計報告之一

Metric	Type	Package	File	Count	Sub Total	Total	Bookstore-front-office used sources	Count for Bookstore-front-office used sources	Bookstore-back-office used sources	Count for Bookstore-back-office used sources
Total Lines of Code						9002		8160		7484
	src				4075					
		idv.takeshi.software.productline.bookstore.domain.model.order			1480					
			Order.java	335			1	335	1	335
			OrderStatus.java	193			1	193	1	193
			OrderItem.java	132			1	132	1	132
			PickingList.java	125				0	1	125
			CreditCard.java	91			1	91	1	91
			PostalOrder.java	88			1	88	1	88
			Invoice.java	85			1	85	1	85
			Payment.java	80			1	80	1	80
			PickingListItem.java	70				0	1	70
			Atm.java	68			1	68	1	68
			OrderItemStatus.java	68			1	68	1	68
			OrderFactory.java	29			1	29		0
			DuplicateInvoice.java	25			1	25	1	25
			DuplicateInvoiceWithCashier.java	25			1	25	1	25
			OrderRepository.java	19			1	19	1	19
			AfterDeliveryPayment.java	7			1	7	1	7
			BeforeDeliveryPayment.java	7			1	7	1	7
			CashInConvenientStore.java	7			1	7	1	7
			OrderDuplicateException.java	6			1	6		0
			OrderException.java	6			1	6	1	6
			OrderStatusToUnderProcessingException.java	6				0	1	6
			CreditCardType.java	4			1	4	1	4
			PickingListStatus.java	4				0	1	4
		idv.takeshi.software.productline.bookstore.domain.model.book			921					
			Book.java	256			1	256	1	256
			Inventory.java	167			1	167	1	167
			AuthorRelationship.java	70			1	70	1	70
			Promotion.java	64			1	64		0
			Author.java	61			1	61		0
			SetPrice.java	55			1	55		0
			PriceOff.java	54			1	54		0
			Discount.java	53			1	53		0
			BookFactory.java	29			1	29		0
			BookRepository.java	18			1	18		0
			AuthorRepository.java	10				0		0
			PriceOffGreaterThanFixedPriceException.java	7				0		0
			SetPriceGreaterThanFixedPriceException.java	7				0		0
			AuthorDuplicateException.java	6				0		0
			BookDuplicateException.java	6			1	6		0
			BookException.java	6			1	6		0
			DiscountGreaterThanOneException.java	6				0		0
			DiscountSettingException.java	6				0		0
			DiscountSmallThanZeroException.java	6				0		0
			PriceOffSettingException.java	6				0		0
			PriceOffSmallThanZeroException.java	6				0		0
			PromotionException.java	6				0		0
			SetPriceSettingException.java	6				0		0
			SetPriceSmallThanZeroException.java	6				0		0
			InventoryStatus.java	4			1	4	1	4

bookstore-domain-model 專案 LOC 統計報告之二

Metric	Type	Package	File	Count	Sub Total	Total	Bookstore-front-office used sources	Count for Bookstore-front-office used sources	Bookstore-back-office used sources	Count for Bookstore-back-office used sources
		idv.takeshi.software.productline.bookstore.domain.model.useraccount			720					
			UserAccount.java	302			1	302	1	302
			IndividualAccount.java	78			1	78	1	78
			Role.java	57			1	57	1	57
			Customer.java	54			1	54	1	54
			OrganizationAccount.java	42			1	42	1	42
			UserAccountFactory.java	40			1	40	1	40
			Administrator.java	23			1	23	1	23
			Carrier.java	23				0	1	23
			Clerk.java	23				0	1	23
			Publisher.java	23			1	23	1	23
			UserAccountRepository.java	18			1	18	1	18
			CreateRoleFailedException.java	9			1	9	1	9
			CreateUserAccountFailedException.java	6			1	6	1	6
			RoleDuplicateException.java	6			1	6	1	6
			UserAccountDuplicateException.java	6			1	6	1	6
			UserAccountException.java	6			1	6	1	6
			Sex.java	4			1	4	1	4
		idv.takeshi.software.productline.bookstore.domain.model.category			482					
			Category.java	352			1	352	1	352
			BooksPagingService.java	79			1	79		0
			CategoryFactory.java	13				0	1	13
			CategoryRepository.java	8			1	8	1	8
			BookAlreadyAttachedException.java	6				0	1	6
			CategoryDuplicateException.java	6				0	1	6
			CategoryException.java	6				0	1	6
			CategoryNameNotSetException.java	6				0	1	6
			SubCategoryAlreadyAttachedException.java	6				0	1	6
		idv.takeshi.software.productline.bookstore.domain.model.deliveryservice			162					
			DeliveryService.java	125			1	125	1	125
			DeliveryServiceFactory.java	19				0		0
			DeliveryServiceRepository.java	6			1	6	1	6
			DeliveryServiceDuplicateException.java	6				0		0
			DeliveryServiceException.java	6				0		0
		idv.takeshi.software.productline.bookstore.domain.model.shoppingcart			163					
			ShoppingCart.java	101			1	101		0
			ShoppingCartItem.java	56			1	56		0
			UpdateQuantityOfShoppingCartItemException.java	6			1	6		0
		idv.takeshi.software.productline.bookstore.domain.model.valueobject			147					
			InternationalText.java	135			1	135	1	135
			InternationalTextException.java	6				0	1	6
			LocaleNotNeededException.java	6				0	1	6

bookstore-domain-model 專案 LOC 統計報告之三

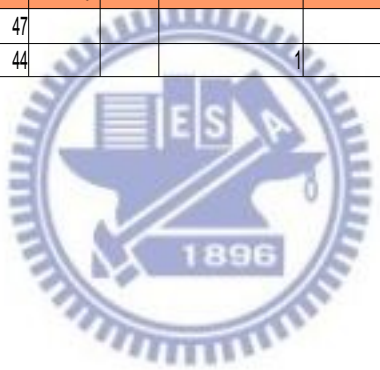
Metric	Type	Package	File	Count	Sub Total	Total	Bookstore-front-office used sources	Count for Bookstore-front-office used sources	Bookstore-back-office used sources	Count for Bookstore-back-office used sources
	test				4927					
		idv.takeshi.software.productline.bookstore.domain.model.order			2079					
			OrderTest.java	304			1	304	1	304
			OrderItemTest.java	174			1	174	1	174
			NewTest.java	156			1	156	1	156
			AtmTest.java	146			1	146	1	146
			PickingListTest.java	133				0	1	133
			PostalOrderTest.java	113			1	113	1	113
			CreditCardTest.java	112			1	112	1	112
			InvoiceTest.java	109			1	109	1	109
			PickingListItemTest.java	90				0	1	90
			WaitForReplenishmentTest.java	85			1	85	1	85
			OkTest.java	80			1	80	1	80
			WaitTest.java	80			1	80	1	80
			UnderProcessingTest.java	79			1	79	1	79
			OrderFactoryTest.java	77			1	77		0
			UnderDeliveryTest.java	73			1	73	1	73
			DuplicateInvoiceTest.java	54			1	54	1	54
			DuplicateInvoiceWithCashierTest.java	54			1	54	1	54
			InConvenientStoreTest.java	52			1	52	1	52
			ReceivedTest.java	52			1	52	1	52
			CancelTest.java	28			1	28	1	28
			CloseTest.java	28			1	28	1	28
		idv.takeshi.software.productline.bookstore.domain.model.book			886					
			BookTest.java	366			1	366	1	366
			InventoryTest.java	179			1	179	1	179
			AuthorRelationshipTest.java	89			1	89		0
			AuthorTest.java	72			1	72		0
			DiscountTest.java	63			1	63		0
			SetPriceTest.java	63			1	63		0
			PriceOffTest.java	54			1	54		0
		idv.takeshi.software.productline.bookstore.domain.model.useraccount			749					
			UserAccountTest.java	410			1	410	1	410
			CustomerTest.java	95			1	95	1	95
			OrganizationAccountTest.java	53			1	53	1	53
			AdministratorTest.java	40			1	40	1	40
			ClerkTest.java	40				0	1	40
			PublisherTest.java	40			1	40	1	40
			CarrierTest.java	39				0	1	39
			UserAccountFactoryTest.java	32			1	32	1	32
		idv.takeshi.software.productline.bookstore.domain.model.category			654					
			CategoryTest.java	569			1	569	1	569
			BooksPagingServiceTest.java	85			1	85		0
		idv.takeshi.software.productline.bookstore.domain.model.deliveryservice			164					
			DeliveryServiceTest.java	164			1	164	1	164
		idv.takeshi.software.productline.bookstore.domain.model.shoppingcart			158					
			ShoppingCartTest.java	158			1	158		0
		idv.takeshi.software.productline.bookstore.domain.test			125					
			VariousTest.java	125				0		0
		idv.takeshi.software.productline.bookstore.domain.model.valueobject			112					
			InternationalTextTest.java	112			1	112	1	112

bookstore-domain-model-persistence-jpa 專案 LOC 統計報告

Metric	Type	Package	File	Count	Sub Total	Total	Bookstore-front-office used sources	Count for Bookstore-front-office used sources	Bookstore-back-office used sources	Count for Bookstore-back-office used sources
Total Lines of Code						1682		1533		1104
	src				709					
		idv.takeshi.software.productline.bookstore.infrastructure.persistence.jpa			709					
			OrderRepositoryImpl.java	247			1	247	1	247
			UserAccountRepositoryImpl.java	218			1	218	1	218
			BookRepositoryImpl.java	108			1	108		0
			CategoryRepositoryImpl.java	55			1	55	1	55
			AuthorRepositoryImpl.java	49				0		0
			DeliveryServiceRepositoryImpl.java	32			1	32		0
	test				973					
		idv.takeshi.software.productline.bookstore.infrastructure.persistence.jpa			961					
			OrderRepositoryImplTest.java	257			1	257	1	257
			UserAccountRepositoryImplTest.java	235			1	235	1	235
			BookRepositoryImplTest.java	232			1	232		0
			CategoryRepositoryImplTest.java	92			1	92	1	92
			AuthorRepositoryImplTest.java	88				0		0
			DeliveryServiceRepositoryImplTest.java	57			1	57		0
		idv.takeshi.software.productline.bookstore.test			12					
			AbstractSpringTest.java	12				0		0

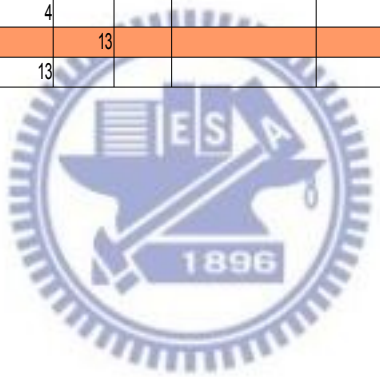
bookstore-utilities 專案 LOC 統計報告

Metric	Type	Package	File	Count	Sub Total	Total	Bookstore-front-office used sources	Count for Bookstore-front-office used sources	Bookstore-back-office used sources	Count for Bookstore-back-office used sources
Total Lines of Code						352		253		99
	src				162					
		idv.takeshi.software.productline.bookstore.utils.template			97					
			TemplateUtils.java	91			1	91		0
			TemplateGenerationFailedException.java	6			1	6		0
		idv.takeshi.software.productline.bookstore.utils.text			65					
			ServletPathStringUtils.java	52				0	1	52
			StripTagUtils.java	13			1	13		0
	test				190					
		idv.takeshi.software.productline.bookstore.utils.template			99					
			TemplateUtilsTest.java	99			1	99		0
		idv.takeshi.software.productline.bookstore.utils.text			91					
			ServletPathStringUtilsTest.java	47				0	1	47
			StripTagUtilsTest.java	44			1	44		0



bookstore-application-service 專案 LOC 統計報告

Metric	Type	Package	File	Count	Sub Total	Total	Bookstore-front-office used sources	Count for Bookstore-front-office used sources	Bookstore-back-office used sources	Count for Bookstore-back-office used sources
Total Lines of Code						594		577		358
	src				266					
		idv.takeshi.software.productline.bookstore.application.service.spring			231					
			AbstractUserRegistrationSerivce.java	112			1	112	1	112
			MailServiceImpl.java	75			1	75		0
			IndividualAccountRegistrationServiceImpl.java	44			1	44		0
		idv.takeshi.software.productline.bookstore.application.service			35					
			MailService.java	13			1	13		0
			UserRegistrationSerivce.java	10			1	10	1	10
			UserAccountAlreadyActivatedException.java	6			1	6		0
			UserAccountNotFoundException.java	6			1	6	1	6
	test				328					
		idv.takeshi.software.productline.bookstore.application.service.spring			315					
			UserRegistrationSerivceImplTest.java	230			1	230	1	230
			MailServiceImplTest.java	81			1	81		0
			Constants.java	4				0		0
		idv.takeshi.software.productline.bookstore.test			13					
			AbstractSpringTest.java	13				0		0



bookstore-project-template-spring 專案 LOC 統計報告

Metric	Type	Package	File	Count	Sub Total
Total Lines of Code					
	src				413
		idv.takeshi.software.productline.bookstore.presentation.dto			265
			UserAccountDto.java	120	
			IndividualAccountDto.java	94	
			OrganizationAccountDto.java	51	
		idv.takeshi.software.productline.bookstore.presentation.web.controller			106
			UserAccountRegistrationController.java	106	
		idv.takeshi.software.productline.bookstore.presentation.web			38
			MessageFilter.java	27	
			UserSecurityAdvice.java	11	
		idv.takeshi.software.productline.bookstore.application.service			4
			Constants.java	4	
	test				175
		idv.takeshi.software.productline.bookstore.test			102
			VariousTest.java	89	
			AbstractSpringTest.java	13	
		idv.takeshi.software.productline.bookstore.web			73
			UserWebTest.java	73	
	webapp/WEB-INF/pages				275
		/error			15
			dataAccessFailure.jsp	15	
		/register			134
			create.jsp	112	
			activateUser.jsp	14	
			success.jsp	8	
		/			126
			main.jsp	81	
			login.jsp	31	
			roleClerk.jsp	7	
			roleCustomer.jsp	7	

bookstore-front-office 專案 LOC 統計報告之一

Metric	Type	Package	File	Count	Sub Total	Total
Total Lines of Code						4364
	src				2044	
		idv.takeshi.software.productline.bookstore.presentation.dto			549	
			OrderDto.java	154		
			UserAccountDto.java	151		
			IndividualAccountDto.java	129		
			CreditCardDto.java	115		
		idv.takeshi.software.productline.bookstore.presentation.web.controller			619	
			PlaceOrderController.java	188		
			BrowseBooksController.java	120		
			UserAccountRegistrationController.java	106		
			MemberController.java	99		
			PayCreditCardController.java	53		
			ShoppingCartController.java	53		
		idv.takeshi.software.productline.bookstore.presentation.web.tag			301	
			PathCategoriesTagHandler.java	80		
			SidebarCategoriesTagHandler.java	78		
			NavCategoriesTagHandler.java	71		
			BookDescriptionTagHandler.java	52		
			AbstractSimpleTagSupport.java	20		
		idv.takeshi.software.productline.bookstore.application.service.spring			281	
			PlaceOrderServiceImpl.java	112		
			PayCreditCardServiceImpl.java	87		
			BrowseBooksServiceImpl.java	48		
			MaintainUserAccountServiceImpl.java	34		
		idv.takeshi.software.productline.bookstore.presentation.web.function			99	
			Functions.java	99		
		idv.takeshi.software.productline.bookstore.application.service			73	
			BeansInitializer.java	20		
			BrowseBooksService.java	13		
			PlaceOrderService.java	13		
			Constants.java	9		
			PayCreditCardService.java	7		
			OriginalPasswordInvalidException.java	6		
			MaintainUserAccountService.java	5		
		idv.takeshi.software.productline.bookstore.presentation.web.filter			61	
			InitialSessionAttributesFilter.java	61		
		idv.takeshi.software.productline.bookstore.presentation.web			38	
			MessageFilter.java	27		
			UserSecurityAdvice.java	11		
		idv.takeshi.software.productline.bookstore.utils.invoice			23	
			InvoiceNumberUtils.java	23		

bookstore-front-office 專案 LOC 統計報告之二

Metric	Type	Package	File	Count	Sub Total	Total
	test				1172	
		idv.takeshi.software.productline.bookstore.application.service.dto			466	
			IndividualAccountDtoTest.java	206		
			OrderDtoTest.java	147		
			CreditCardDtoTest.java	113		
		idv.takeshi.software.productline.bookstore.application.service.spring			436	
			PlaceOrderServiceImplTest.java	169		
			PayCreditCardServiceImplTest.java	125		
			MaintainUserAccountServiceImplTest.java	76		
			BrowseBooksServiceImplTest.java	66		
		idv.takeshi.software.productline.bookstore.test			155	
			VariousTest.java	142		
			AbstractSpringTest.java	13		
		idv.takeshi.software.productline.bookstore.web			73	
			UserWebTest.java	73		
		idv.takeshi.software.productline.bookstore.utils.invoice			42	
			InvoiceNumberUtilsTest.java	42		
	webapp/WEB-INF/pages				1148	
		/book			93	
			details.jsp	93		
		/error			15	
			dataAccessFailure.jsp	15		
		/member			79	
			main.jsp	5		
			maintainUserInfo.jsp	16		
			order.jsp	2		
			orders.jsp	56		
		/order			451	
			confirmOrder.jsp	178		
			orderDetailsInfo.jsp	202		
			orderInfo.jsp	2		
			paymentNotice.jsp	69		
		/payment			64	
			payByCreditCard.jsp	64		
		/register			155	
			activateUser.jsp	14		
			create.jsp	7		
			success.jsp	8		
			userInfo.jsp	126		
		/shoppingCart			80	
			shoppingCart.jsp	80		
		/			211	
			details.jsp	38		
			detailsContent.jsp	47		
			login.jsp	32		
			main.jsp	50		
			roleClerk.jsp	7		
			roleCustomer.jsp	7		
			search.jsp	30		

bookstore-back-office 專案 LOC 統計報告之一

Metric	Type	Package	File	Count	Sub Total	Total
Total Lines of Code						5606
	src				2231	
		idv.takeshi.software.productline.bookstore.application.service.dto			662	
			MultiUserAccountDto.java	238		
			UserAccountDto.java	116		
			PaymentDto.java	86		
			PostalOrderPaymentDto.java	85		
			IndividualAccountDto.java	75		
			AtmPaymentDto.java	62		
		idv.takeshi.software.productline.bookstore.presentation.web.controller			786	
			MaintainAccountsController.java	202		
			MaintainOrdersController.java	179		
			MaintainCategoriesController.java	176		
			UserAccountRegistrationController.java	124		
			MaintainPickingListController.java	105		
		idv.takeshi.software.productline.bookstore.application.service.spring			274	
			MaintainCategoriesServiceImpl.java	90		
			MaintainOrdersServiceImpl.java	86		
			MaintainUserAccountsServiceImpl.java	48		
			MaintainPickingListsServiceImpl.java	34		
			UserRegistrationSerivceImpl.java	16		
		idv.takeshi.software.productline.bookstore.presentation.web.function			220	
			Functions.java	220		
		idv.takeshi.software.productline.bookstore.presentation.web.tag			140	
			CategoriesTreeTagHandler.java	120		
			AbstractSimpleTagSupport.java	20		
		idv.takeshi.software.productline.bookstore.application.service			111	
			BeansInitializer.java	20		
			MaintainOrdersService.java	20		
			MaintainCategoriesService.java	15		
			MaintainUserAccountsService.java	12		
			MaintainCategoriesException.java	9		
			InvalidPickingListSequenceException.java	7		
			CanNotRemoveCategoryException.java	6		
			Constants.java	6		
			InvalidOrderIdException.java	6		
			MaintainPickingListException.java	6		
			MaintainPickingListsService.java	4		
		idv.takeshi.software.productline.bookstore.presentation.web			38	
			MessageFilter.java	27		
			UserSecurityAdvice.java	11		

bookstore-back-office 專案 LOC 統計報告之二

Metric	Type	Package	File	Count	Sub Total	Total
	test				1380	
		idv.takeshi.software.productline.bookstore.application.service.dto			701	
			MultiUserAccountDtoTest.java	256		
			IndividualAccountDtoTest.java	153		
			AtmPaymentDtoTest.java	149		
			PostalOrderPaymentDtoTest.java	143		
		idv.takeshi.software.productline.bookstore.application.service.spring			448	
			MaintainCategoriesServiceImplTest.java	141		
			MaintainOrdersServiceImplTest.java	142		
			MaintainPickingListsServiceImplTest.java	61		
			MaintainUserAccountsServiceImplTest.java	104		
		idv.takeshi.software.productline.bookstore.test			158	
			AbstractSpringTest.java	13		
			VariousTest.java	145		
		idv.takeshi.software.productline.bookstore.web			73	
			UserWebTest.java	73		
	webapp/WEB-INF/pages				1995	
		/accounts			429	
			listAccounts.jsp	128		
			maintainAccount.jsp	301		
		/categories			515	
			generateCategoriesTree.jsp	97		
			maintainCategories.jsp	418		
		/error			15	
			dataAccessFailure.jsp	15		
		/orders			602	
			listOrders.jsp	158		
			maintainAtmPayment.jsp	48		
			maintainCashInConvinientStorePayment.jsp	28		
			maintainOrder.jsp	287		
			maintainPayment.jsp	23		
			maintainPostalOrderPayment.jsp	58		
		/pickingLists			253	
			listPickingLists.jsp	64		
			receivePickingList.jsp	57		
			seePickingList.jsp	132		
		/register			134	
			activateUser.jsp	14		
			create.jsp	112		
			success.jsp	8		
		/			47	
			login.jsp	36		
			main.jsp	11		

1. 各專案 LOC 之統計結果彙整

各專案 LOC 之統計結果彙整

Project Source Code		LOC
bookstore-domain-model		9002
bookstore-domain-model	-persistence-jpa	1682
bookstore-utilities		352
bookstore-application-se	rvice	594
Bookstore-project-templa	te-spring(White box reuse)	863
bookstore-front-office		4364
bookstore-back-office		5606
Total		22463

2. 論文實作之 RSI 統計

論文實作之 RSI 統計

Application	Domain-specific RSI	Application-specific RSI	Utilities RSI	New or changed LOC	Total LOC
bookstore-front-office	9693	577	253	4364	14887
bookstore-back-office	8588	358	99	5606	14651
Total	18281	935	352	9970	29538

彙整以上各統計資料，接著套入 Reuse Metrics Starter Set 所給定之公式，最後得出以下結果。

ADC: $624,650

重用率、重用節省成本和投資報酬計算結果

Application	RSI	Reuse%	DCA	SCA	RCA	ROI
bookstore-front-office	10523	70.7%	$841,840	$105,230	$947,070	$322,420
bookstore-back-office	9045	61.7%	$723,600	$90,450	$814,050	$189,400
Total/Avg.	19568	66.2%	$1,565,440	$195,680	$1,761,120	$511,820

1. bookstore-domain-model 專案

Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
396	0	0	0	100%	3.252

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
idv.takeshi.software.productline.bookstore.domain.model.order	167	0	0	0	100%	1.904
idv.takeshi.software.productline.bookstore.domain.model.valueobject	9	0	0	0	100%	0.125
idv.takeshi.software.productline.bookstore.domain.model.book	91	0	0	0	100%	0.33
idv.takeshi.software.productline.bookstore.domain.model.shoppingcart	12	0	0	0	100%	0.063
idv.takeshi.software.productline.bookstore.domain.model.useraccount	65	0	0	0	100%	0.33
idv.takeshi.software.productline.bookstore.domain.test	3	0	0	0	100%	0.078
idv.takeshi.software.productline.bookstore.domain.model.category	31	0	0	0	100%	0.391
idv.takeshi.software.productline.bookstore.domain.model.deliveryservice	18	0	0	0	100%	0.031

bookstore-domain-model 專案測試結果報告

Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	106	91%	1416/1552	70%	353/504	1.685
idv.takeshi.software.productline.bookstore.domain.model.book	25	90%	276/304	68%	74/108	1.603
idv.takeshi.software.productline.bookstore.domain.model.category	18	90%	214/236	78%	86/110	1.81
idv.takeshi.software.productline.bookstore.domain.model.deliveryservice	5	90%	50/55	57%	8/14	1.375
idv.takeshi.software.productline.bookstore.domain.model.order	34	94%	538/569	69%	90/130	1.575
idv.takeshi.software.productline.bookstore.domain.model.shoppingcart	4	93%	67/72	67%	23/34	2.143
idv.takeshi.software.productline.bookstore.domain.model.useraccount	17	83%	204/244	62%	45/72	1.773
idv.takeshi.software.productline.bookstore.domain.model.valueobject	3	93%	67/72	75%	27/36	2.6

Report generated by Cobertura 1.9.4.1 on 2011/6/5 下午 5:50.

bookstore-domain-model 專案測試涵蓋率報告

2. bookstore-domain-model-persistence-jpa 專案

Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
62	0	0	0	100%	11.671

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
idv.takeshi.software.productline.bookstore.infrastructure.persistence.jpa	62	0	0	0	100%	11.671

Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.

idv.takeshi.software.productline.bookstore.infrastructure.persistence.jpa

Class	Tests	Errors	Failures	Skipped	Success Rate	Time
AuthorRepositoryImplTest	5	0	0	0	100%	0.047
BookRepositoryImplTest	13	0	0	0	100%	0.781
CategoryRepositoryImplTest	4	0	0	0	100%	0.062
DeliveryServiceRepositoryImplTest	2	0	0	0	100%	0.047
OrderRepositoryImplTest	17	0	0	0	100%	2.578
UserAccountRepositoryImplTest	21	0	0	0	100%	8.156

bookstore-domain-model-persistence-jpa 專案測試結果報告

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	6	91% 301/330	85% 91/106	2.478
idv.takeshi.software.productline.bookstore.infrastructure.persistence.jpa	6	91% 301/330	85% 91/106	2.478

Report generated by Cobertura 1.9.4.1 on 2011/6/5 下午 5:51.

bookstore-domain-model-persistence-jpa 專案測試涵蓋率報告

3. bookstore-utilities 專案

Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
5	0	0	0	100%	1.015

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
idv.takeshi.software.productline.bookstore.utils.template	3	0	0	0	100%	0.5
idv.takeshi.software.productline.bookstore.utils.text	2	0	0	0	100%	0.515

bookstore-utilities 專案測試結果報告

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	4	80% 76/95	90% 18/20	2.778
idv.takeshi.software.productline.bookstore.utils.template	2	71% 40/56	83% 5/6	3.25
idv.takeshi.software.productline.bookstore.utils.text	2	92% 36/39	92% 13/14	2.4

Report generated by Cobertura 1.9.4.1 on 2011/6/5 下午 6:09.

bookstore-utilities 專案測試涵蓋率報告

4. bookstore-application-service 專案

Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
12	0	0	0	100%	11.422

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
idv.takeshi.software.productline.bookstore.application.service.spring	12	0	0	0	100%	11.422

bookstore-application-service 專案測試結果報告

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	7	82% 89/108	75% 9/12	1.353
idv.takeshi.software.productline.bookstore.application.service	4	50% 2/4	N/A N/A	1
idv.takeshi.software.productline.bookstore.application.service.spring	3	83% 87/104	75% 9/12	1.48

Report generated by Cobertura 1.9.4.1 on 2011/6/5 下午 5:52.

bookstore-application-service 專案測試涵蓋率報告

5. bookstore-front-office 專案

Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
65	0	0	0	100%	14

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
idv.takeshi.software.productline.bookstore.test	9	0	0	0	100%	0.234
idv.takeshi.software.productline.bookstore.application.service.dto	41	0	0	0	100%	1.672
idv.takeshi.software.productline.bookstore.utils.invoice	1	0	0	0	100%	0
idv.takeshi.software.productline.bookstore.application.service.spring	14	0	0	0	100%	12.094

bookstore-front-office 專案測試結果報告

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	32	37% 314/839	13% 22/158	1.623
idv.takeshi.software.productline.bookstore.application.service	7	90% 9/10	100% 2/2	1.071
idv.takeshi.software.productline.bookstore.application.service.dto	4	97% 192/197	50% 2/4	1.047
idv.takeshi.software.productline.bookstore.application.service.spring	4	95% 99/104	71% 10/14	1.647
idv.takeshi.software.productline.bookstore.presentation.web	2	0% 0/14	0% 0/2	1.2
idv.takeshi.software.productline.bookstore.presentation.web.controller	6	0% 0/250	0% 0/60	2.586
idv.takeshi.software.productline.bookstore.presentation.web.filter	1	0% 0/28	0% 0/6	1.429
idv.takeshi.software.productline.bookstore.presentation.web.function	2	0% 0/59	0% 0/28	5.6
idv.takeshi.software.productline.bookstore.presentation.web.tag	5	0% 0/162	0% 0/34	1.886
idv.takeshi.software.productline.bookstore.utils.invoice	1	93% 14/15	100% 8/8	3

Report generated by Cobertura 1.9.4.1 on 2011/6/5 下午 5:53.

bookstore-front-office 專案測試涵蓋率報告

6. bookstore-back-office 專案

Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
92	0	0	0	100%	11.015

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
idv.takeshi.software.productline.bookstore.test	6	0	0	0	100%	1.016
idv.takeshi.software.productline.bookstore.application.service.dto	61	0	0	0	100%	1.094
idv.takeshi.software.productline.bookstore.application.service.spring	25	0	0	0	100%	8.905

bookstore-back-office 專案測試結果報告

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	33	40% 421/1041	21% 51/237	1.867
idv.takeshi.software.productline.bookstore.application.service	11	85% 17/20	100% 2/2	1.042
idv.takeshi.software.productline.bookstore.application.service.dto	6	95% 293/306	78% 33/42	1.245
idv.takeshi.software.productline.bookstore.application.service.spring	5	92% 111/120	80% 16/20	2
idv.takeshi.software.productline.bookstore.presentation.web	2	0% 0/14	0% 0/2	1.2
idv.takeshi.software.productline.bookstore.presentation.web.controller	5	0% 0/368	0% 0/88	3.128
idv.takeshi.software.productline.bookstore.presentation.web.function	2	0% 0/135	0% 0/67	5.545
idv.takeshi.software.productline.bookstore.presentation.web.tag	2	0% 0/78	0% 0/16	1.556

Report generated by Cobertura 1.9.4.1 on 2011/6/5 下午 5:54.

bookstore-back-office 專案測試涵蓋率報告