國 立 交 通 大 學

資 訊 工 程 學 系

碩 士 論 文

以「問題－解法」為基礎的軟體知識分類法

A Problem-Approach-based Software Knowledge Classification System

研 究 生：謝祖望

指導教授：鍾乾癸

中 華 民 國 九 十 三 年 六 月

以「問題－解法」為基礎的軟體知識分類法

A Problem-Approach-based Software Knowledge Classification System

研 究 生：謝祖望　　Student : Tzu-Wang, Hsieh

指導教授：鍾乾癸　　Advisor : Chyan-Goei, Chung

國 立 交 通 大 學

資 訊 工 程 學 系

碩 士 論 文

A Thesis

Submitted to Department of Computer Science and Information Engineering College of Electrical Engineering and Computer Science
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Science
in
Computer Science and Information Engineering

June 2004

Hsinchu, Taiwan, Republic of China

中 華 民 國　九十三　年　六　月

以「問題－解法」為基礎的軟體知識分類法

研究生：謝祖望　　指導教授：鍾乾癸
國立交通大學資訊工程系碩士班

摘要

軟體發展是腦力密集及知識密集的工作，在開發過程中，軟體工程師常需搜尋所需技術知識來加速其發展。由於目前技術知識庫的關鍵字缺乏良好制定法則，無法有效代表技術知識的意義，進而影響知識搜尋的範圍、精度與速度，對軟體生產力造成不良影響。

現有知識管理平台常見的知識分類法則大致可以分為四類，即 Taxonomy、Faceted Classification、Case-Based Reasoning 與 Ontology。這些方法中，Taxonomy 太過簡單故缺乏效率；Faceted classification 會因知識數量增多而過於複雜，且難以對軟體知識定義有效率的面向；CBR 則缺乏良好的分類架構；Ontology 只做到知識的描述，而欠缺知識間的關聯性。故這些方法應用於軟體知識管理仍有許多不足之處。

本論文發現多數技術知識主要內容是針對特定議題提出解決方法，因此提出以「領域類別」、「議題」、「解法」、「技術」來代表一技術知識的性質，且發現此種表示法易於建構技術知識間的關聯性。進而本論文提出以「問題－解法」為基礎的軟體知識分類架構，以領域類別為基礎，在類別下有子議題，各議題下有許多論文提出不同的解法及各種技術解決此議題。本論文並依提出的知識分類方法，實作了一雛型（Prototype）知識庫，以驗證其可行性。

本文所提之新技術知識關鍵字可清楚表示論文之性質，且可依領域類別、議題、解法或技術有效且精確搜尋出所要的技術知識，較以前方法更為實用。

A Problem-Approach-based Software Knowledge Classification System

Student: Tzu-Wang, Hsieh Advisor: Chyan-Goei, Chung

Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Chiao-Tung University

Abstract

It is well convinced that software industry is highly knowledge- and labor-intensive. During software development process, software developers usually search for technical knowledge to shorten the development time and improve the software quality. However, technical knowledge doesn't have good indexing keyword definition to stand for its main point. It will cause poor accuracy and efficiency of knowledge query. Consequently, it's the obstacle toward productivity of software development.

Most popular knowledge classification techniques are Taxonomy, Faceted classification, Case-based reasoning, and Ontology. Among these techniques, Taxonomy is too simple to be effective. Faceted classification is too complicated when the amount of knowledge increases, it's also difficult to define good facet for software knowledge. Case-based reasoning lacks classification structure for knowledge domain. Ontology uses its model to describe knowledge, but there is no relationship between knowledge. Since above techniques is not designed for software knowledge, applying to software knowledge management has drawbacks.

We find that the purpose of technical knowledge is to propose an approach to an issue. Hence, technical knowledge could be represented by its category, issue, approach and technique. With this representation, it's very easy to construct the relationship between knowledge. Accordingly, this thesis proposed a problem-approach-based software knowledge classification system. This system classify knowledge domain with hierarchy structure. Leaf-node categories have issues. Technical knowledge proposes approach to each issue, and uses certain techniques. Then we can represent technical knowledge effectively, and increase the accuracy and efficiency of software knowledge query. To prove the feasibility, we also implement a prototype system.

This thesis proposed new classification of technical knowledge keywords. It allows user to query technical knowledge by its category, issue, approach or techniques. Therefore, this classification technique is more useful than others.

誌謝

在碩士班的兩年時間中，真的非常感激鍾乾癸老師的教誨，讓我深刻學習到做學問的方法與態度，讓我往後做人做事都受用不盡。這篇論文能夠完成，也是要感謝老師不厭其煩的教導，在此要對老師致上最高的謝意。

另外，感謝劉文謙學長與鄭靜紋學姐的細心指導，協助我解決許多研究上的困難。同時也感謝一起畢業的莊志良、吳偉聖、黃鼎新，許多研究的想法都是和大家一起討論才順利產生的，我們也一同度過了令人難忘的兩年碩士班時光。感謝我從大學以來的好友陳大任、劉衍谷、張子文、邱挺以及實驗室的學弟們，你們陪我度過了寫論文最困難的時光。

最後我要感謝我的母親及怡螢，這段日子為了寫論文錯過了許多陪伴你們的時光，謝謝你們的支持與鼓勵，僅以此成果與你們分享。

目錄

圖目錄

表目錄

第一章、緒論

1.1 研究背景與動機

近年來由於網際網路的普及與資訊科技的突飛猛進，帶動了知識經濟的風潮。世界經濟的發展由以往的勞力密集、資本密集、技術密集，逐漸朝向知識密集的時代，Toffler稱之為第三波經濟革命 (Alvin, 1991)。在農業時代，土地的取得是成功的要素；工業時代則取決於資本財的投資；而當前的科技時代，知識已成為個人與組織所擁有的珍貴資產之一(Thomas, et. al., 1998)。因此，知識管理將是使組織具備競爭力的重要關鍵。所謂的知識管理，是指適時適地將正確的知識提供給需要的成員，以輔助成員採取適當的行動來增進組織績效的持續性過程。此過程包括知識的創造、確認、收集、分類組織、分享、使用與改進等步驟 (American Productivity & Quality Center, 1996)。而軟體產業本身即是一項人力與知識密集的產業 (Birk, et. al., 1999)，軟體發展更是一個知識不斷創新的過程。由於資訊硬體產品的效能及功能大幅提升及價格大眾化，導致各類軟體需求愈來愈多，功能也漸趨複雜，故軟體知識快速且多樣化地成長，因而造成發展軟體時常遭遇開發過程費時、交貨延遲、產能不易提升，且品質不易確保等問題，甚至產生了**軟體危機**[1]。故知識管理對軟體產業來說更是刻不容緩的課題。

如同前面所述，我們在過去已體認到軟體知識管理的重要性，展開一系列的研究，對軟體知識種類與型態、軟體知識螺旋、軟體重用元件、軟體知識庫及軟體發展環境之研究與實作等議題均有相關的研究，並將研究成果發表於會議論文[2][3]。在過去的研究中，我們發覺軟體的知識管理必須與軟體開發流程緊密結合，在不同的階段中，發展者需要不同類型的知識，在每一個階段中，遇到不同的問題，發展者亦需要相對應的知識作為輔助。然而，隨著知識庫中的知識愈來愈多，要如何有效地利用這些知識變成一個重要的課題。

過去，許多典型的知識庫如 Microsoft 的知識庫[4]、IEEE 電子期刊資料庫[5]、ACM 電子圖書館等[6]，都擁有非常充足的知識。以 Microsoft 的知識庫作為例，原先目的是用來作為產品技術支援服務；產品知識庫中包含數十萬份由數千位產品技術支援工程師針對客戶的問題而製作的技術文件，藉由不斷的更新與增加，以確保內容的實用性及正確性。使用者可透過搜尋介面，選擇所要查詢的範圍與選項，鍵入關鍵字來尋找。然而由於其資料實在太過龐大，同一類型的產品相關問題可能高達數千筆，即使其中真的有能解決使用者問題的知識，使用者也很難利用幾個關鍵字就找得到。IEEE 及 ACM 的電子圖書館也有類似的問題，同樣的領域可能有高達數百篇的論文，使用者利用幾個關鍵字找出來的論文也可能相當多，難以判斷知識是否能夠解決問題。因此，即使這些知識庫都收集了相關領域下最豐富、最有價值的知識，卻不能適時適地將正確的知識提供給需要的成員，這使得這些知識庫沒有發揮其應有的價值。

知識庫的效率與知識分類方法有密不可分的關係，最常見的知識分類法大致可以分為四類，即 Taxonomy[7]、Faceted Classification[8]、Case-Based Reasoning[9]、Ontology[10]。

Taxonomy 是最廣為引用的樹狀分類法，將知識領域以階層架構的方式分類。然而這種方法只將知識領域分類，對軟體知識而言，同領域下的知識仍非常多，就如前述的 IEEE 及 ACM 電子圖書館，同領域下可能有高達數百篇的論文，使用者必須閱讀全部論文後，才能判斷是否為他需要的知識，這種方法非常沒有效率。

Faceted Classification 為多面向的分類法，讓知識能依不同面向進行樹狀結構的分類。例如酒可依產地、種類來分類。但軟體知識各領域特性不同，除了依知識領域分類外，難以建構其他具有通用性的面向。而如作者、期刊等面向則無法對知識的內容描述有所幫助。當知識量增多時，從各個面向找尋知識也是費時費力，因此很少有軟體知識採用 Faceted Classification 做為主要分類方法。

Case-based Reasoning（以下簡稱 CBR）採用案例及解決方案的原理描述知識，因此會設計其知識模型。但各種知識都必須經過處理修改後才能符合其知識模型，這對增長快速的軟體知識而言是沒有效率的。此外，對 CBR 方法而言，各個案例都是單獨的存在，彼此之間僅有一些共通性的關聯，而沒有架構良好的分類系統，因此使用 CBR 系統只能找案例的解答，若對知識領域不熟悉的使用者，則無法利用類別瀏覽的方式找出想了解的知識領域。

Ontology 利用定義清楚的模型描述知識，改善傳統關鍵字搜尋的效率問題。但軟體知識領域廣泛，很難定出各領域皆通用的 Ontology 模型，且以 Ontology 方法描述知識通常較為複雜，對知識工程師而言是一大負擔。此外，以 Ontology 描述的知識彼此間缺乏關聯性，使用者無法在閱讀知識之後快速取得相關知識，也沒辦法對知識進行比較。

由上可知，目前的知識管理搜尋與分類方法應用在軟體知識上都有其不足之處，當知識累積數量增多時，知識庫的效率及正確性便會降低，浪費使用者寶貴的時間。因此我們需要更有效率的知識分類方法來改善此問題。

對知識使用者而言，利用知識庫搜尋知識，必定是對此類知識有所需求，我們可以將這些需求大致分為下列四類：

1. 使用者對某特定領域不熟，希望找出該領域的介紹性知識。
2. 使用者對特定領域熟悉，希望深入了解某特定議題。
3. 使用者遭遇了某些問題，希望找出這些問題的最佳解決方案。
4. 使用者熟悉特定技術，想了解該技術被運用在哪些領域中。

在各種分類與搜尋方法中，關鍵字法仍舊是最廣為運用的。然而一般關鍵字都較零散且缺乏規則，且關鍵字之間沒有具體的關聯性，如此對知識的使用者幫助相當有限。因此有必要對關鍵字作更嚴謹的定義，使其更明確地表示論文的性質，且便於分類。

從技術論文或技術報告的本質分析，發現這兩種知識主要內容都是為了解決某項議題，而提出其解決方法。議題必定存在於特定領域之下，而解決方法則可能運用了其他的技術，因此，描述一篇論文，只要描述這篇論文所在領域、解決的議題、採用的解法及運用的技術便能說明其核心意義。故我們能將關鍵字分為「領域類別」、「議題」、「解法」、「技術」四個類型，一篇論文依此方法定義關鍵字便能表示其意義。

採用上述方法表示技術知識後，可非常容易找出兩篇論文間之相關性。例如解決相同議題的論文、採用相同解法或技術的論文、改善前人解法的論文，利用這些關係能提供知識間的關聯性，依議題、解法、技術找出同性質的論文，提昇搜尋的精度與準度。

本研究進一步依此表示方式提出知識分類架構。以領域類別為基礎，在其下可建構議題，各議題下有許多論文提出不同的解法及各種技術解決此議題。如此使用者便可依領域類別瀏覽，找出領域類別下的子議題，並可列出特定議題的解法及技術，也可找出特定技術的運用狀況，讓上述四種需求的使用者都能在此分類架構上找到合適的知識。

因此，本研究將依上述發現，詳細定義技術知識的關鍵字類型與關係，提出知識分類架構，並依此分類方法實作一雛型（Prototype）知識庫系統，以證實可行性。最後並將比較此方法與傳統知識分類法對軟體知識管理的適用性。

1.2 各章節介紹

本論文章節安排如下，首先在第二章介紹軟體知識與軟體知識管理的概念，接著探討傳統知識分類法運用在軟體知識管理上的問題。第三章從軟體知識本質進行分析，進而提出軟體知識表示法及分類方法，並採用符號表示法提昇知識處理的效率。第四章則介紹軟體知識的關鍵字分類機制，包括建構法則、領域類別

架構、知識排序法則與搜尋方法以及符號表示法的 BNF。第五章將實作一知識庫範例以證此分類方法的可行性。第六章總結本論文之研究成果並提出未來可繼續發展的研究方向。

第二章、背景知識與相關研究

軟體產業是一高度知識密集的產業，軟體的發展更是一項不斷創新知識的過程，因此，做好知識的管理，將能加速知識傳遞與學習的過程，以便縮短軟體開發的時程，提高軟體品質，進而降低整體開發成本與加快產品上市的時間。而軟體知識分類方法的好壞，將決定軟體知識管理的效率，尤其軟體知識發展迅速，大量的知識必須透過有效率的知識分類方法加以管理方能達到軟體知識管理的目標。

本章首先在 2.1 節介紹軟體知識管理的基本概念及其流程，2.2 節將重點放在軟體知識管理中的分類方法，並介紹目前最常用的知識分類方法。2.3 節則分析現有知識分類方法運用在軟體知識管理上的問題。

2.1 軟體知識管理

軟體的發展是一項知識不斷創新的過程，每天都有新的問題被解決、新的知識被創造出來，使得軟體的知識呈現多樣化且快速的成長。Ioana Rus等人曾說明知識管理對於軟體工程的重要性包括[11]：

- **記錄與分享流程（Process）以及產物（Product）知識：**軟體發展的流程與產物會隨目標與內容的不同而有所差異，單一的發展模式無法滿足所有專案的需要，因此發展者須不斷的藉由專案的過程來累積知識與經驗。然而有些發展團隊卻常忽略這些工作經驗，以至於未能有效運用之前專案的成果而造成許多時間的浪費。而知識管理強調的知識捕捉（Capture）與分享正是解決的良方。

- **了解專門領域知識（Domain Knowledge）：**軟體發展過程常需要瞭解問題領域的相關知識，搜尋與學習這些知識往往要花上許多時間，但這些知識會散

在不同的成員身上，其效用有限，知識管理技術可幫助組織將這些專門知識（Expertise）予以確認（Identify）、包裹（Package），進而供團隊共享，以提升整體能力。

- **獲得新技術、資訊**：軟體是一個不容易掌握且發展快速的領域，科技不斷地推陳出新，也許使軟體的能力更加強大了，但也是發展人員的夢魘，除了穩定性、相容性外，也讓評估變的更加困難，需要花時間去適應與學習。知識管理旨在促進組織分享的文化，透過溝通與實踐來縮短共同學習的曲線。

- **分享所在環境的知識（Local Policy）**：成功的軟體組織應該擁有許多軟體經驗，而發展團隊的程式風格或一些慣例通常存在較有經驗的發展者腦中，且常透過較非正式的溝通來傳承給新進人員。知識管理除了協助溝通，也試著去捕捉日常生活中較非正式的知識分享，讓更多需要的人能夠受益。

- **了解有誰知道什麼（Who Knows What）**：有許多的知識可以被紀錄下來，但不可否認的，也有許多隱性的知識存在員工的腦中，對每個人所知道的知識做管理可減少成員尋找專家幫忙解決問題的時間，甚至有成員離開時，也可當作找尋替補人選的參考。

- **遠距離的合作**：由於全球化的趨勢，團隊的成員可能身處在不同的地方，因此遠距的溝通、協調與合作就愈發顯出其重要性，透過知識分享，才能提升整合效率。

所謂軟體知識是指在軟體發展過程中所有需要、產生、應用以及參考到的各項顯性與隱性的知識，我們即可稱之為「軟體知識」。一般而言，參與人員需要軟體工程知識（Software Engineering Knowledge）、產品領域知識（Product Domain Knowledge）、產品知識（Product Knowledge）、相關電腦科學知識（Computer Science Knowledge）等類的軟體知識。將上述知識與軟體發展人員的關係關連在一起，則軟體知識模型便如圖 2 - 1 所示：[2]

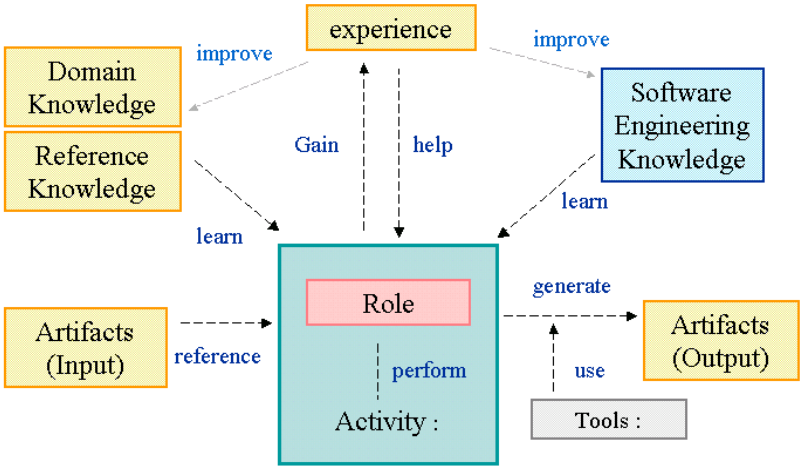

圖 2 - 1 軟體知識模型

至於知識的創造是一項連續不斷的過程，透過內隱知識與外顯知識彼此的交互作用，由個人、團體慢慢向外擴張至整個組織結構，結合新的概念和既有的知識，進而發展出新的知識，此一反覆遞增的過程，即為知識的螺旋（Spiral of Knowledge）[12]。知識的螺旋包含社會化、外化、整合、內化四種知識的轉換模式，而軟體發展的過程本身也符合知識螺旋的特性。根據這四類轉換在軟體知識管理所扮演的角色與軟體發展者在各項轉換中所必須執行的活動，可整理軟體知識管理模型如圖 2 - 2 所示[2]：

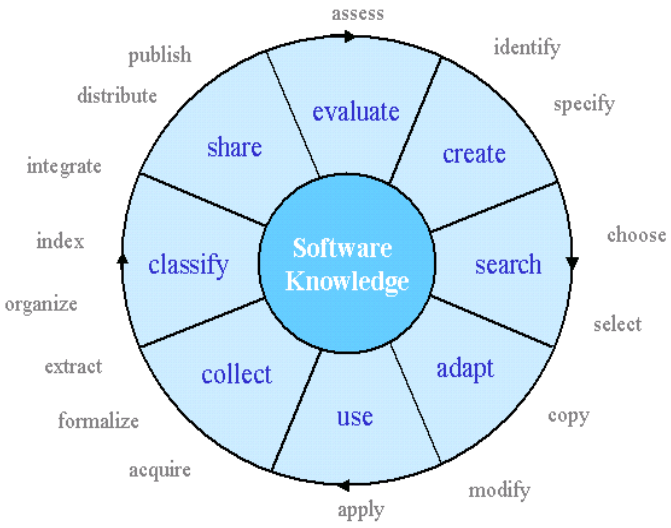

圖 2 - 2 軟體知識管理模型

由上圖可知，軟體知識管理的主要活動包括：

- 創造(Create)：創造新的、有價值的知識是知識管理主要的目的之一，但並非所有創新都是無中生有，許多知識是透過合併或更改舊有的知識，進而賦予其新的意義與價值。因此在創造的過程中首要的步驟是確認(Identify)欲得到的知識是什麼，並具體詳細地加以指明(Specify)。

- 搜尋(Search)：在目前知識爆炸的時代，要精確的從數以萬計的資料中找出所要的知識並不容易，因此搜尋的機制也是非常重要的一環。關於搜尋的研究與作法很多，但主要的想法是根據想要找的條件，選擇(Choose)適合的分類，再搜尋現有的知識庫，並從搜尋的結果中挑選出(Select)需要的部分。

- 調適(Adapt)：從前一步驟搜尋得到的知識通常與發展者實際所想要的知識，不管在內容或者形式上都可能會有些落差，因此在不破壞原來知識的情況下，先將之取出複製(Copy)一份，再經過適當的修改(Modify)或合併以符合實際的情境所需。

- 使用(Use)：知識的使用可說是知識循環中最重要的部分，各項步驟的最終目的就是希望讓知識能被有效的應用(Apply)到實際狀況中，進而發揮知識的價值。

- 收集(Collect)：擷取(Acquire)發展過程中可再應用的產物、相關文件、經驗等知識，整理成特定的適當形式(Formalize)，抽出(Extract)並儲存至知識庫中。

- 分類(Classify)：組織(Organize)知識，放入適當的分類與索引(Index)，並合併(Integrate)其中相似的部分。同時在分類過程中也可能需要動態適時的進行知識結構的調整。

- 共享(Share)：透過分派(Distribute)與出版(Publish)的機制來擴散知識，進而達成知識的共享。其中又可分為動態的散佈，主動地將重要的知識即時傳遞給

需要的人員；以及靜態的陳列，讓發展者有需求的時候再去尋找。這兩種方式的差別在於時間、對象與知識內容的取捨。

- 評估(Evaluate)：透過使用者的討論、評分、測量、回饋等方式對知識進行分析與估價(Assess)，以便瞭解知識使用和分佈的情形，淘汰其中過時以及無用的部分，並將重要或使用率高的知識擺在明顯、容易取得的位置。

在這些軟體活動中，收集（Collect）、分類（Classify）與共享（Share）是軟體知識庫的主要工作，而這些工作的效率又與軟體知識分類方法有極大的關聯，故軟體知識分類方法是決定軟體知識庫效率的重要關鍵，亦是本研究所探討的主題。

2.2 軟體知識分類方法

知識分類是將相同領域或具有相似特性的知識歸類整理，或對知識進行淬取（extract）及描述，藉以提昇使用者搜尋知識的效率的方法。最常見的知識分類法則大致可以分為四類，即 Taxonomy[7]、Faceted Classification[8]、Case-Based Reasoning[9]及 Ontology[10]。本節將分別對其進行介紹。

2.2.1 Taxonomy

Taxonomy 是最廣為引用的樹狀分類法。Taxonomy 最早是在十七世紀，Carl Linnaeus 對植物採用拉丁二項式系統（Latin binomial system）命名，用第一個字表示植物的屬（Genes），用第二個字表示種（Species），這種方法是最簡單的樹狀分類法，而其後 Carl Linnaeus 還發明了以七層架構為基礎的生物種類分類法，從上往下依序是界（Kingdom）、門（Phylum）、綱（Class）、目（Order）、科（Family）、屬（Genes）、種（Species），這個方法從十七世紀延用至今，仍然是生物分類的標準，而 Carl Linnaeus 也被稱為 Taxonomy 之父。

這種樹狀分類法被運用在各種需要分類的地方，由於其方法簡單、直觀，又能有效地對各種知識分門別類，故被廣為引用。在知識管理上，許多分類方法也都是以 Taxonomy 方法架構的。如圖書館的分類系統，最著名的杜威十進分類法（Dewey decimal classification）[13]、美國國會圖書分類法（Library of congress classification）[14]都是採用樹狀分類架構，在電腦科技期刊上，最常使用的 ACM 分類系統（ACM computing classification system）[15]也是 Taxonomy 的分類方式。此外，在網際網路上搜尋引擎最常使用的 ODP（Open Directory Project）[16]分類系統，同樣也是採用此方法。由上述可知，在各領域知識管理中，最常使用的知識分類方法即為 Taxonomy。

Taxonomy 的基本原理即樹狀分類架構，將知識先分成數個大領域，各個大領域再往下細分，每一個分類都可以繼續細分，例如軟體知識分類範例便可如圖 2 - 3 所示：

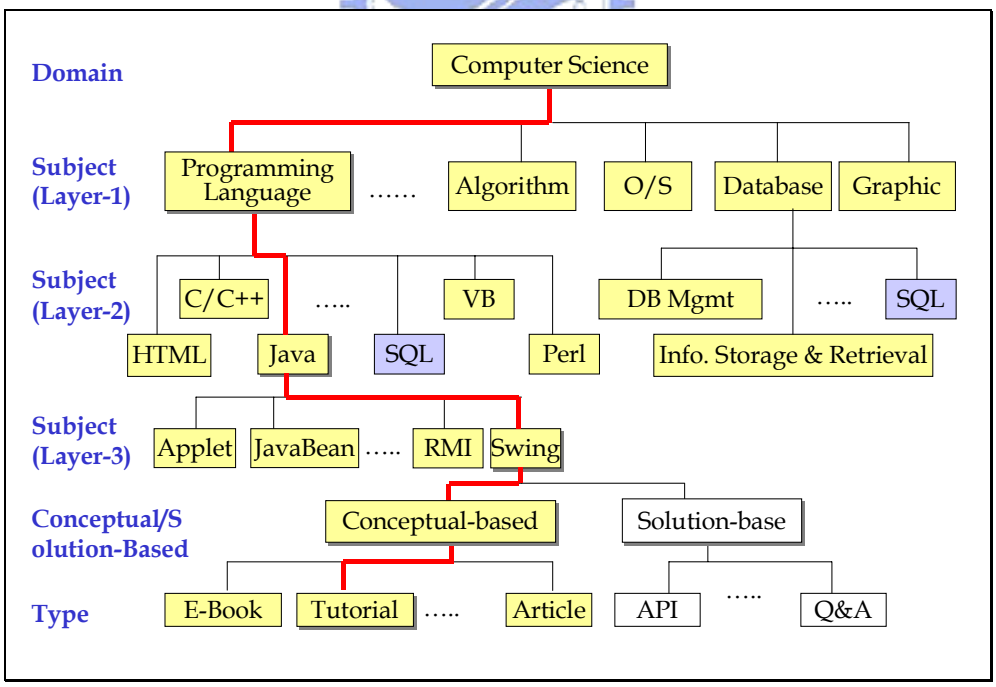

圖 2 - 3 軟體知識分類範例

2.2.2 Faceted Classification

Faceted Classification 係由印度的圖書館員 S. R. Ranganathan 於 1930 年代發明的，這是利用清楚定義（clearly defined）、完全互斥（mutually exclusive）及完整窮舉（collectively exhaustive）的知識主題、特性來進行分類[8]，其特色是讓知識分類具備更高的彈性及自由度，不必侷限於之前的分類架構，隨時都可以新增一個 Facet 分類。Facet 可以是兩種型態，第一種即前一小節所述的 Taxonomy 樹狀結構，第二種則是用表示連續數字的格式，如價格、日期等等。

Faceted Classification 自 1960 年代以來被廣泛地使用，由於其分類具有高度彈性，故被運用在各個不同領域。例如原本採用 Taxonomy 架構的杜威十進分類法，也在其第 21 版加入了 Faceted Classification 的概念，除了原有的領域分類外，還加入了如作者種族、書本語言、國家等不同面向的分類。此方法也被運用到如商業管理（London Classification of Business Studies）[17]與建築工程（Unified Classification for the Construction Industry）[18]等不同領域。運用在軟體知識管理上，也有如 KM-Connection[19]等系統採用了 Faceted Classification 做為其分類架構，此系統採用的面向包括了產品、應用領域、組織、人員、領域技術、事件、出版物等，軟體知識採用上述面向進行分類以加速知識索引與搜尋。

Faceted Classification 採用多面向的分類方法，例如酒就可以利用種類（如紅酒、白酒）、產地（歐洲、美洲）及價格加以分類，在 FacetMap[20]中就有提到如下的範例，若酒採用 Faceted Classification 分類，則搜尋介面如圖 2 - 4 所示：

圖 2 - 4 Faceted Classification 範例-1

若使用者選了紅酒之後，則系統會依符合的酒重新整理後如圖 2 - 5：

圖 2 - 5　Faceted Classification 範例-2

接著使用者可以改以產地來縮小範圍，選擇 USA 或可得如圖 2 - 6：

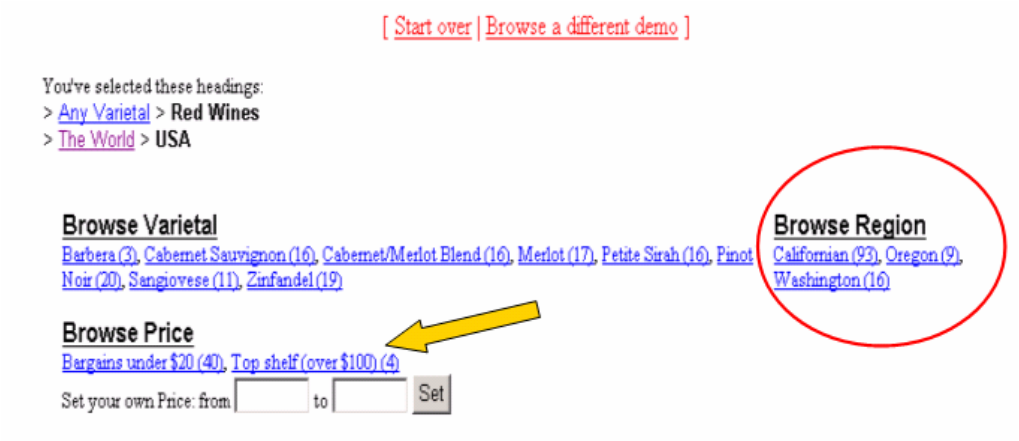

圖 2 - 6　Faceted Classification 範例-3

產地部份便會顯示在美國之下的產地，如加洲、華盛頓等，使用者可再以價格做選擇，例如選擇 Top shelf 的酒類，可得如圖 2 - 7 之結果：

圖 2 - 7　Faceted Classification 範例結果

由於這些條件下只有四種酒類，故系統可直接列出其結果。由上例可知，此方法的好處便是使用者的搜尋過程可以隨時利用不同面向縮小範圍，故可用其較熟悉的面向快速找到符合的知識。

Faceted Classification 也有標準交換格式，稱為 XFML (the eXchangable Faceted Metadata Language)[21]，此格式採用 XML 對 Facet 進行描述，讓不同的知識庫可以籍此進行 Facet 的交換。

2.2.3　Case-based Reasoning

Case-based Reasoning（以下簡稱 CBR）是由 Roger Schank 所提出[22]，一開始是為了解決 AI 的問題，他認為知識的儲存、經驗及學習是不可分隔的，進而提出了 theory of dynamic memory。而 CBR 的特性即是利用過往的經驗來解決新的問題，它試圖從過往的個案（Case）中找尋與目前問題最相似者，將其解法進行適當的修改後來提供解答，並透過解答所達成的效果來進行學習，籍以豐富其知識庫。其中還有一分支稱為 Analogy-based Reasoning[23]，試圖利用跨領域的經驗來解決問題。CBR 的模型如圖 2 - 8 所示：

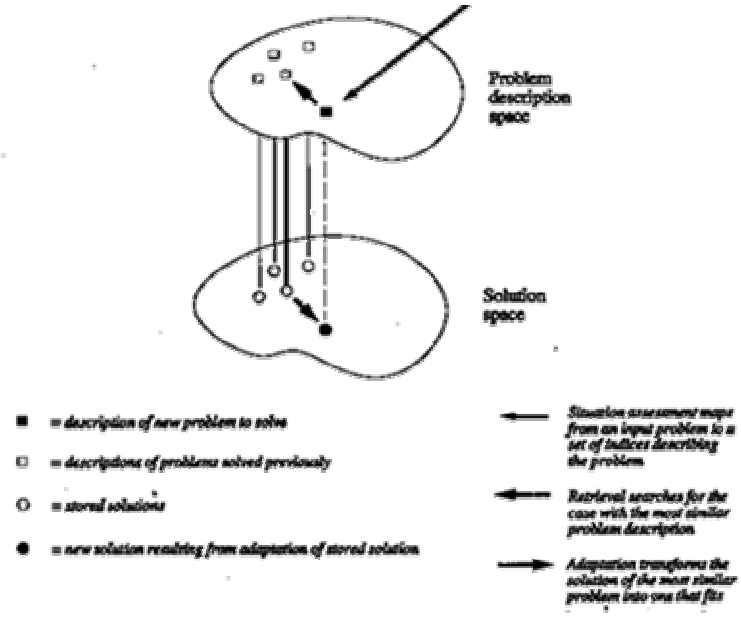

圖 2 - 8　Case-based Reasoning 模型

故 CBR 是將知識分為問題與解答來描述，知識必須先依其模型描述問題及對應解答，有新的問題出現時便依這些問題找尋最接近的答案，其結果可再儲存為新的知識。CBR 的流程圖如圖 2 - 9 所示[24]：

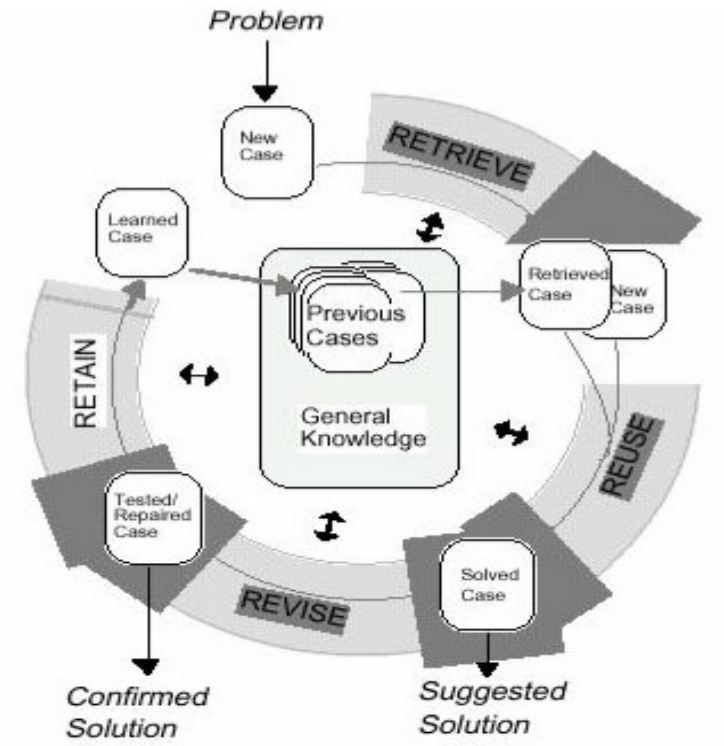

圖 2 - 9　Case-based Reasoning 流程圖

CBR 用以描述案例(Case)最常用的模型有兩種，一種稱為 Dynamic memory model，另一種稱為 Category & exemplar model。前者將案例集合成一個個片斷（Episode)，在其下有不同的索引與對應的值，這些值又可以連結到其他的片斷或直接連結到案例中，其示意圖如圖 2 - 10：

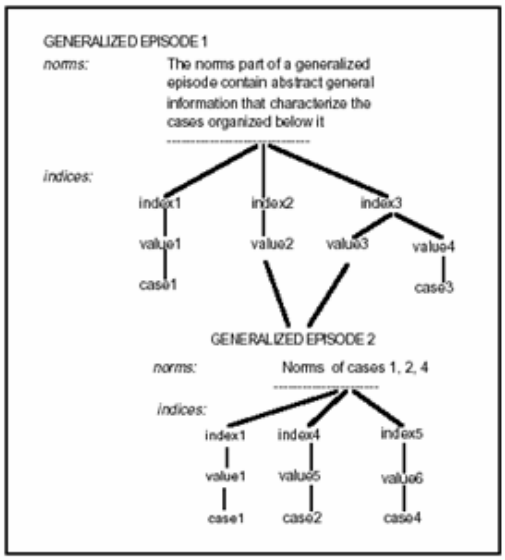

圖 2 - 10　Dynamic memory model 示意圖

Category & exemplar model 則是自案例中找出特性（Feature)，並將一些特性定義為類別（Category)，這些類別可再進一步分析而成為模範（Exemplar)，同一個類別可以產生多種模範，而案例便可依此進行歸類。當新的案例出現時，系統便對此案例分析後找出合適的模範，並將其下案例的解答分析整理後提出可能的答案。其示意圖如圖 2 - 11：

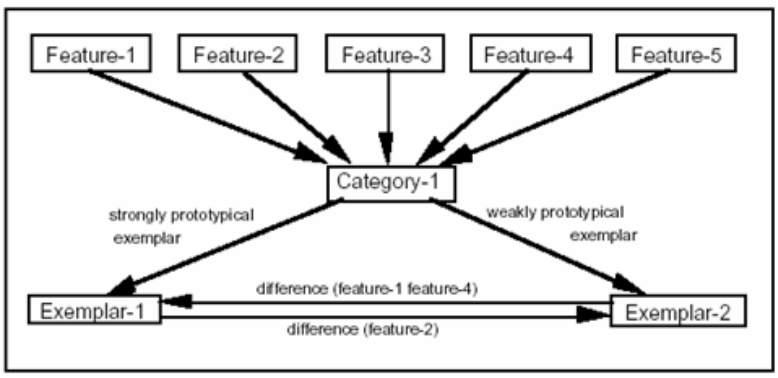

圖 2 - 11　Category & exemplar model 示圖

過去已有些許研究者，利用 CBR 進行知識管理[25]，然而由於 CBR 必須仰賴規格十分嚴謹的知識表示法，而且其複雜的模型必須完全相同才能夠讓不同的 CBR 系統溝通，因此一般都是運用在特定的領域解決特定的問題，例如處方、餐廳菜單、病情診斷等，不適合用在五花八門的一般知識庫中。

2.2.4 Ontology

Ontology 是近年來非常熱門的研究主題，在知識管理、人工智慧、自然語言解析、分散式系統都各有應用。所謂 Ontology 係指領域知識定義一個共通的分享方式，運用正式且清楚定義的模型，藉以描述知識的意義[26]。

著名的 Ontology 應用如 WordNet[27]、KIF(Knowledge Interchange Format)[28]、Ontolingua[29]、On-To-Knowledge[30]等。應用在知識管理上，其主要目的是將領域知識以完整定義的 Ontology Model 來描述，改善傳統關鍵字搜尋的效率問題。以著名的 On-To-Knowledge 為例，它首先將文件進行解析後找出文件中可能的重要概念，經由使用者編輯 Ontology 之後儲存至分享環境中。系統會通知對相關概念有興趣的使用者，使用者可進而取得此知識。搜尋的時候也是藉由其 Taxonomy 的樹狀分類搭配一些關鍵字來找尋特定概念下的知識。On-To-Knowledge 的架構圖如圖 2 - 12 所示：

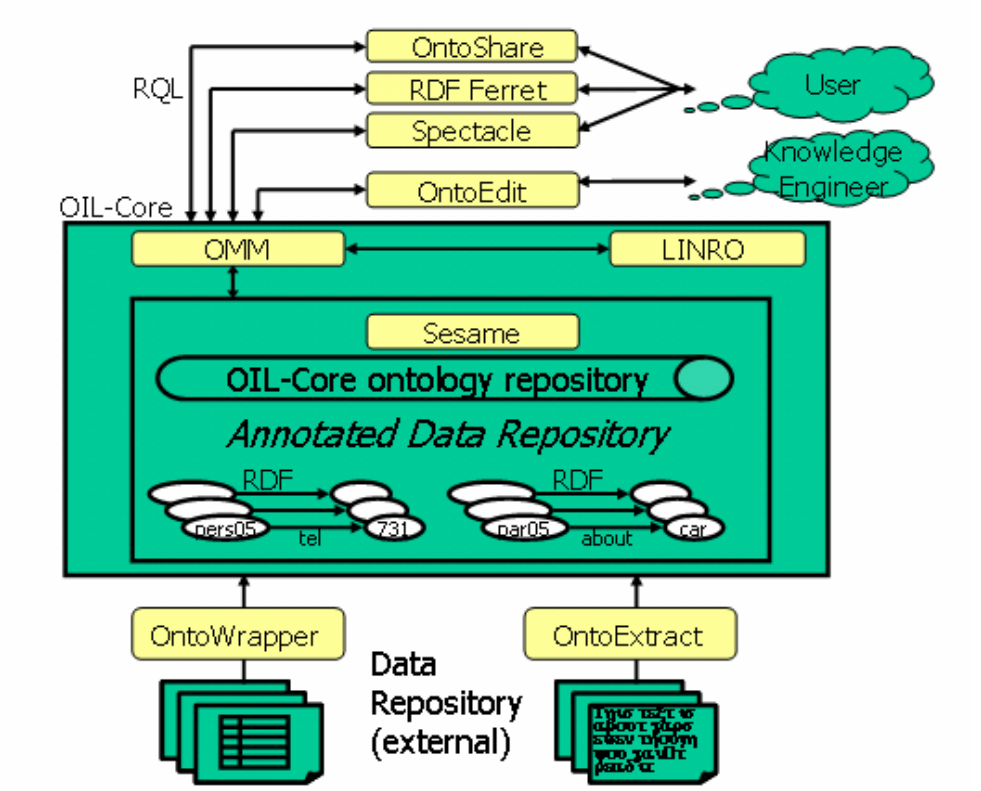

圖 2 - 12　On-To-Knowledge 架構圖

其 Ontology 採用 RDF（Resource Description Framework）進行描述，其主要子系統中，OntoEdit 是提供知識工程師手動建構 Ontology 的介面，而 OntoExtract 則是可以從自然語言的知識中半自動地建構知識的 Ontology，至於 OntoWrapper 可以從結構化的知識中淬取出其 Ontology，這些 Ontology 儲存在知識庫中，採用 OntoShare 讓使用者進行搜尋，改善傳統關鍵字搜尋效率不彰的問題。

On-To-Knowledge 對於文件建構了詳細的 Ontology 描述，如標題（title）、摘要（summary）、作者、相關概念、附件等，而作者也描述其名字、e-mail 等資訊，而相關概念也可採用分類的方式來描述，其示意圖如圖 2 - 13 所示：

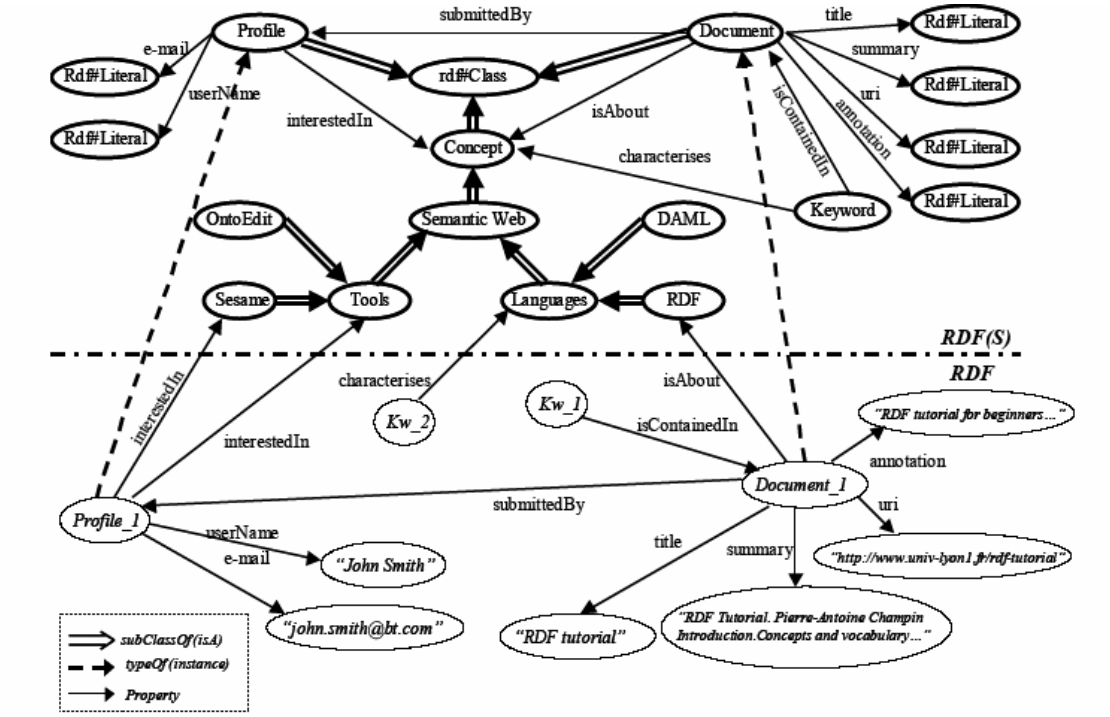

圖 2 - 13　On-To-Knowledge Ontology 架構圖

由上圖的例子，可看出 Ontology 的基本概念是定義知識的特性以及特性間的關係，甚至可利用邏輯化的方式對知識進行描述，故使用者在搜尋知識時可明確地描述知識特性及概念，提昇知識搜尋的效率。

2.3　現有知識分類方法的問題

Taxonomy

Taxonomy 是最廣為引用的樹狀分類法，但從知識搜尋的觀點來看，Taxonomy 下的知識缺乏關係與連結，也沒辦法進行知識的分析。由於樹狀分類過於簡單，因此也沒辦法依使用者需求給予適當的知識。雖然有非常多的知識庫都採用 Taxonomy 來做分類，但彼此之間由於缺乏共通的分類準則，因此沒有辦法進行跨知識庫的整合。此外，Taxonomy 的層狀架構使其分類上缺乏彈性，因為知識並非只有單一面向，而 Taxonomy 卻只能依特定方向進行分類。由於只能單純的分類，故當知識量累積愈來愈多時，同類型知識也將非常可觀，使用者沒

辦法從中快速挑選合適的知識，只能逐一閱讀，非常沒有效率。

運用在軟體知識上較有名的 Taxonomy 分類法如 ACM 分類系統（ACM computing classification system）及網路上許多搜尋引擎所使用的的 Open directory project 等，其架構都是以四層為主，但軟體知識各項領域的發展都日新月異，對於各領域的探討都愈來愈深，粗略的分類將使特定類別下的知識太過龐雜，不能真正達到知識分類的優點，例如近年來急速發展的 Internet，若要細分的話可能會出現下列的類別：Internet → WWW → Web Programming → Scripting Language → ASP → Database Connection，光是這些就需要六層的分類才真正能夠將知識恰當地分類，因此現行的知識分類架構確有其不足之處。

Faceted Classification

Faceted Classification 讓知識分類具備更高的彈性及自由度，不必侷限於固定的分類架構，但仍有其問題。在知識的關係與連結上，Faceted Classification 所建立的關係僅是「同類型的知識」，實際上使用者閱讀特定知識後有時會需要更基本或更進階的知識，甚或其他不同相關的知識，這些 Faceted Classification 都沒辦法處理；此外，Faceted Classification 的方式相當不直觀，一般人還是希望藉由幾個關鍵字就找到他所需的知識，而非在複雜的 Facet 結構中找尋；在知識庫的整合上，雖然有 XFML (the eXchangable Faceted Metadata Language)可做為交換的標準，但由於不同知識庫所採用的 Facet 不同，因此知識交換上仍然非常困難。此外，當知識的量累積愈來愈多時，Faceted Classification 也會愈來愈複雜，這樣使用者在找尋知識便更加困難，故 Faceted Classification 用在增長快速的軟體知識上也不夠有效率。

Case-based Reasoning

Case-based Reasoning（以下簡稱 CBR）是利用過往的經驗來解決新的問題，因此必須建構複雜的模型，儲存各種不同的學習結果（lesson-learned），用來做

為之後處理問題的解答。故 CBR 必須將知識進行仔細的切割，以符合其模型。例如論文、書籍中的知識都必須充份了解後再將其中知識一一分成案例（Case）及解答（Solution），因此在知識處理過程非常困難且耗時，完全沒有辦法自動化，這對於增長快速的軟體知識相當不合適；此外，由於 CBR 建構了嚴謹的知識表示方法，故只能運用在特定的領域解決特定的問題，例如處方、餐廳菜單、病情診斷等，不適合用在領域既廣且深的軟體知識庫中。由於各個 CBR 採用的基本模型都不盡相同，因此知識庫之間並沒辦法整合及合作，這對軟體知識亦不合適。

Ontology

Ontology 運用在知識管理上，主要目的是將領域知識以完整定義的 Ontology Model 來描述，改善傳統關鍵字搜尋的效率問題。然而透過 Ontology 描述的知識彼此之間缺乏關聯性，雖然透過 Ontology 描述後的知識較易搜尋，但卻沒辦法讓使用者自由取得相關的知識。此外，不同領域的知識其特性並不相同，因此用以描述的 Ontology 必然無法通用，而對每個領域都要發展其 Ontology 也非常耗時耗力，這對領域廣泛的軟體知識而言尤其不合適；其次，Ontology 雖然能提供知識意義的描述，但其實對相關領域不熟的使用者而言，他對知識的概念不足，因此在使用上反而覺得不便。誠如上述，使用者運用知識庫可概分為三類，故對不同需求的使用者提供不同的搜尋才能真正對其產生幫助。

前了上述分類法各自的問題外，軟體知識管理一直都無法做到「知識處理自動化」及「跨知識庫整合」的要求，知識處理若能自動化，將能夠提昇軟體知識管理的效率。而知識庫之間的整合，將使知識庫的知識量大幅提昇，也能大幅縮短使用者在各個知識庫之間找尋知識所耗費的時間。

由上可知，目前的知識管理搜尋與分類方法都有其不足之處，當知識累積數量增多時，知識庫的效率及正確性便會降低，浪費使用者寶貴的時間。因此我們需要更有效率的知識分類方法來改善此問題。

第三章、「問題－解法」軟體知識分類法之設計構想

前一章已對軟體知識分類方法做了一些探討及分析，可知目前的知識分類方法運用在軟體知識管理上皆有不足之處，本章將進一步提出改進的軟體知識分類方法。3.1 節首先分析軟體技術知識的本質，包括技術論文、技術報告以及書本知識的本質。3.2 節依據軟體技術知識本質提出以「問題－解法」為基礎的軟體知識分類方法。3.3 節則對我們提出的分類方法進行效益比較與分析。

3.1 軟體技術知識的本質

為了解決上述的軟體知識分類方法問題，首先要分析軟體知識的本質，方能進一步找出真正有效率的軟體知識分類法。「軟體知識」較廣義的定義係指在軟體發展過程中所有需要、產生、應用以及參考到的各項顯性與隱性的知識。這個定義中的軟體知識種類繁多，尤其像軟體產物的管理更牽涉到軟體重複利用（Software Reuse）的議題，非一般軟體知識庫的所探討的範圍。是以本論文的研究對象為軟體發展過程中所參考的結構性知識（Structured Knowledge），主要為技術論文、技術報告以及書本知識三大類型。以下將進行分析。

3.1.1 技術論文本質

技術論文是軟體知識中非常重要的部份。技術論文可能提供全新的概念、新的技術、特定議題的解決方案等，對軟體開發的效率與軟體成品都有非常大的影響。了解技術論文的特性，將有助於建構有效率的知識分類方法，進而讓使用者快速取得有幫助的技術論文。

從知識產生的目的來看，技術論文產生的目的大都是解決特定的問題，不論是傳統的演算法、資料結構乃至於發展迅速的無線網路、多媒體，技術論文的產

生都是針對技術上的問題提供解法。常見的型態包括提供較現有技術更有效率的演算法、更適當的資料結構、更佳的解法、更好的架構，或運用其他領域的技術至特定領域上處理某些狀況，整合某些解法以提供更具通用性的解決方案等。即使型態繁多，不難看出技術論文是「問題導向」的知識，換言之，技術論文與其處理的問題息息相關。

然而，縱使技術論文是「問題導向」的知識，知識庫系統卻無法自動地從技術論文的標題或內容中去找出其處理的「問題」以及論文提供的「解法」。由於技術論文是以人的自然語言寫就，故其內容無法透過知識庫處理。而技術論文的標題也沒有固定的原則可循。以下舉幾篇論文為例：

Fabrizio Sebastiani 的"**Machine learning in automated text categorization**"[31]，這個標題中，machine learning 是他提出的解法，而 automated text categorization 則是他要處理的問題，格式是名詞+in+名詞，故在這裡第一個名詞是解法，第二個名詞是問題。但另一篇 *David F. Bacon* 的"**Compiler transformations for high-performance computing**"[32]，看起來跟前一篇標題格式頗為類似，但其實這篇論文的內容是在處理 multiprocessors 以及 superscalar processors 機器上的 compiler optimization 問題，利用 compiler transformation 解決，而採用了 dependence analysis 以及 locality 等技術來解決，這時前一個名詞變成解法，第二個名詞則是問題所在領域。這兩篇標題型式如此類似的論文，其代表的意義卻相差非常多。此外論文的標題由於沒有規定的格式，因此各種型式的標題都可能出現，舉例而言，*Weiyi Meng* 的"**Building efficient and effective metasearch engines**"[33]，從標題上只能看出這篇論文要建構有效率的 metasearch engine，看不出其提出的解法，且格式是採用動詞+名詞的方式，跟前兩篇不同；又如 *Jin Jing* 的"**Client-server computing in mobile environments**"[34]，標題看起來跟第一篇類似，都是以名詞+in+名詞做為標題，但在這裡第一個名詞（Client-server computing）並非解法，而是這篇論文處理的問題，其他尚有更難

判斷的例子，如 *Steffen Heinz* 的”**Burst tries: a fast, efficient data structure for string keys**”[35]，這篇的格式是名詞+：+形容詞+名詞+for+名詞，但其實讀了內容後會發現這篇論文在講的是 text database 中的 string data structure 問題，而提出了 Burst tries 做為解法。又如 Chang H. and Iyengar S. S 的”**Efficient algorithms to globally balance a binary search tree**”[36]，格式是名詞+to+動詞+名詞，而這些名詞卻不能完全代表這論文的意義，讀完論文後會發現這篇論文處理的是 Binary search tree 中的 improve worst case 問題，採用了 sequential balancing & parallel balancing 的解法。

由上述的例子，可發現論文的標題有非常多不同的語法，由於自然語言並非電腦可處理的正規語言，同一種語法可能代表不同的意義，因此有許多地方都是含糊不清（ambiguous）的；且許多論文的標題都沒有真正提到他處理的問題以及採用的解法，因此對使用者而言，找尋論文必須閱讀內容後才能判斷是否符合所需，顯得非常沒有效率。

為改善此問題，一般技術論文都會定義「關鍵字」來增加對論文意義的描述，然而一般關鍵字都較零散且缺乏規則，且關鍵字之間沒有具體的關聯性，如此對知識的使用者幫助相當有限。最著名的 IEEE 期刊也只規定關鍵字定義的基本原則，其中列出了一個 Keyword List 供作者從中選擇，並建議作者採用五到八個關鍵字做為索引，若有新的關鍵字則提供給 IEEE 參考等等。即使如此關鍵字仍是零散無規則的，如前所述，使用者在輸入關鍵字時其目的可能是不同的，然而使用者卻沒有辦法對關鍵字加以定義，因此只能輸入他認為較重要的幾個字，這往往會找出一些完全不符合需求的知識。以下舉上面幾篇論文為例：

Fabrizio Sebastiani 的”**Machine learning in automated text categorization**”，這篇論文中定義了三個關鍵字：Machine learning、text categorization、text classification，而 text categorization 與 text classification 其實是同義字，這樣的關鍵字是很簡單的，text categorization 是問題，而 machine learning 是解法，至於

machine learning 所運用的技術就無法從關鍵字得知。另一篇 *David F. Bacon* 的”**Compiler transformations for high-performance computing**”，則定義了較多的關鍵字：compilation、dependence analysis、locality、multiprocessors、optimization、parallelism、superscalar processors 等，這些關鍵字中，有些（parallelism、superscalar processors、multiprocessors）是問題的領域，有些是問題（compilation、optimization），有些則是解法運用的技術（dependence analysis、locality），若使用者想找 superscalar 的介紹，也可能找到這一篇，想找 dependence analysis 的介紹，也可能找到這一篇，想找 compiler optimization 的方法，也可能找到這一篇，但其實這篇都不符合上述三項需求，足見雜亂關鍵字帶來的不良影響。又如 *Steffen Heinz* 的”**Burst tries: a fast, efficient data structure for string keys**”，定義了 Binary trees、splay trees、string data structures、text databases、tries、vocabulary accumulation 等關鍵字，同樣的，這些關鍵字有些是領域，有些是問題，有些是解法及運用的技術，這樣雜亂無章讓使用者難以快速掌握論文的重點，也無法真正達到索引的功能。再如 *Jin Jing* 的”**Client-server computing in mobile environments**”的，定義了非常多的關鍵字：application adaptation、cache invalidation、caching、client/server、data dissemination、disconnected operation、mobile applications、mobile client/server、mobile computing 等等，幾乎只要論文有提到的相關內容都定義為關鍵字，完全無法由關鍵字看出論文的意義。

誠如上述，技術論文是「問題導向」的，故其探討的內容是針對單一領域中的特定問題，且會提出對應的解決方案，而這個解決方案可能運用或改進了其他論文中所使用的技術。是以我們可以將關鍵字分為四類，即「領域」、「問題」、「解法」、「技術」：

(1) 領域（Domain）：技術論文的關鍵字中有部份是用來表示此知識所在的領域類別。例如前述的 parallelism、superscalar processors、multiprocessors 等。

(2) 問題（Problem）：指的是此知識所探討的主要問題，論文一般是提供某特定問題（如 Traveling Salesman Problem）的解答，將問題獨立出來有助於找尋問題的解法。

(3) 解法 (Approach)：對於特定的問題，知識可能會提供解法。各個論文若都將其提出的解法做為關鍵字，則使用者能夠快速地進行同問題間解法的比較。例如某個做法是利用 Greedy，而另一個是利用 Dynamic Programming，這就能夠讓使用者對此知識有基本概念，也能有個簡單比較的基礎。

(4) 技術（Technique）：一個特定解法可能是運用了某些技術，而這個部份也是很有價值的。使用者不明白解法時，可以籍其所運用的技術有個概括性的了解，例如前述的 dependence analysis、locality、splay trees 等。

因此，論文關鍵字可以圖 3 - 1 表示：

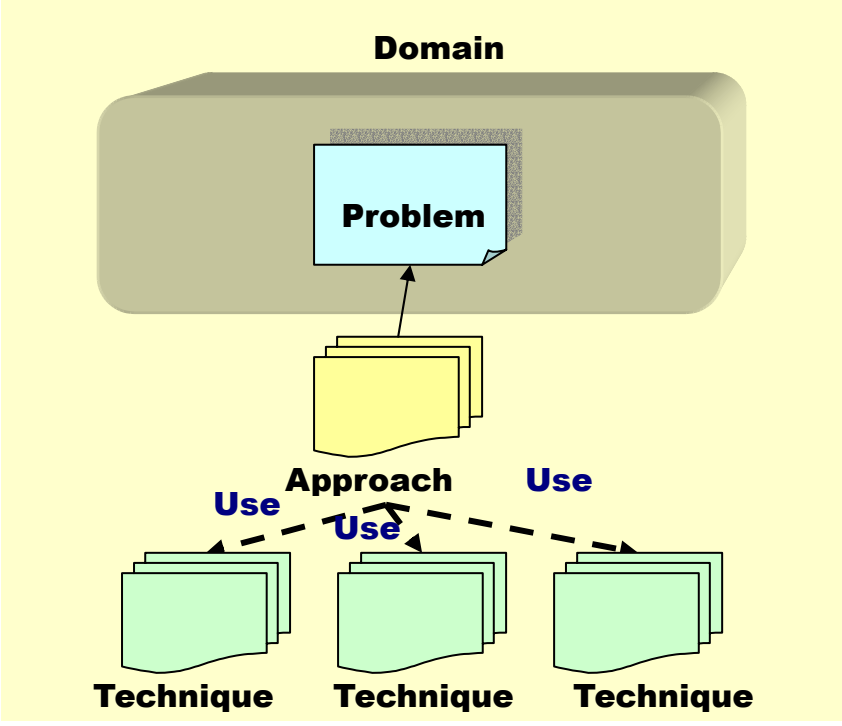

圖 3 - 1　論文關鍵字示意圖

這樣的分類是依據技術論文的「問題導向」本質而定的。由於技術論文的本

質即是根據特定的「問題」提出「解決方案」，因此只要能夠描述「問題」與「解決方案」，便能夠描述一篇技術論文的內容。而問題可分成「領域」與「問題」，解決方案則可分成「解法」與「使用技術」，現行的關鍵字由於沒有定義，因此經常發生許多論文的關鍵字與其內容關聯性不高的問題；事實上，基於技術論文的「問題導向」本質，只要定義上述四類關鍵字，便能夠充份描述一篇論文在哪個領域處理了哪個問題，提出了什麼樣的解法以及使用了什麼樣的技術，這對論文的描述、索引、查詢而言已經足夠了。

一篇論文的問題與解法，用一兩個關鍵字來描述自然有其不足之處。例如對圖學相關內容不熟悉的讀者而言，光看到 travel salesman problem 或是 Steiner tree problem 等關鍵字是無法聯想到實際的問題。或者對 binary search tree 不熟悉的讀者而言，看到某一論文使用了 Hibbard's algorithm 或是 Knuth's algorithm 也很難知道這是什麼樣的演算法。但技術論文的特性便是其深入地探討一個問題，因此會去閱讀論文的讀者應該可以假設對此議題有基本認識，否則應該先從基本的入門書籍開始了解。當使用者對特定領域中的議題有基本認識時，對於這些相關的關鍵字其實都不會陌生，且軟體技術知識有其精確性，每項技術均有其定義清楚（well-defined）的內容，例如排序法（sorting）的 bubble sort、quick sort、insertion sort 等，其實用一個關鍵字就能夠代表相當多的意義。這也是為何目前關鍵字仍是最常被用來做為論文索引的原因。而透過上述的分類，這些關鍵字除了能代表其本身軟體技術的知識外，尚能代表在論文中的意義，是屬於論文的領域、問題、解法抑或技術，更能加強對論文的描述及索引的效率。

再以之前提到的論文為例，*Steffen Heinz* 的”**Burst tries: a fast, efficient data structure for string keys**”論文定義了 Binary trees、splay trees、string data structures、text databases、tries、vocabulary accumulation 等關鍵字，其內容是處理 text database 領域中的 string data structure 問題，他提出了稱為 burst tries 的解法，此解法運用了 splay trees、tries、binary trees、hash table 等技術，因此我們

對其關鍵字稍加修改，可將之表示如圖 3 - 2：

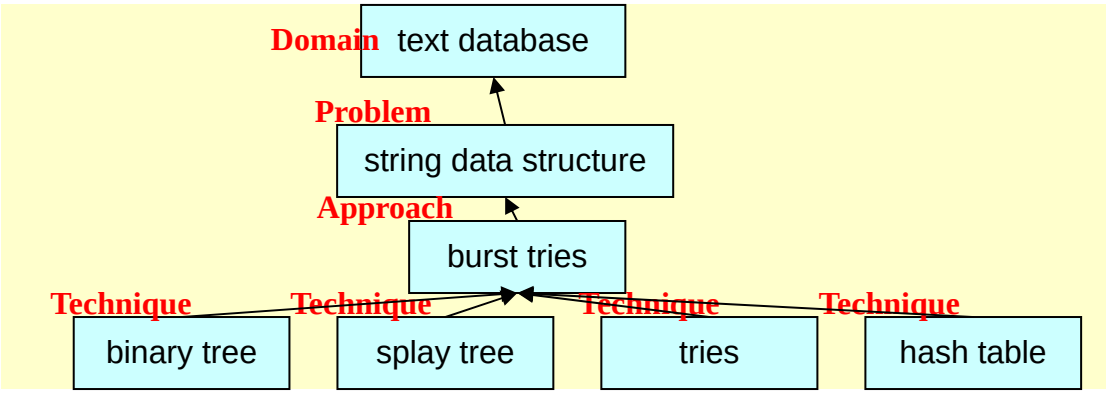

圖 3 - 2　論文關鍵字表示法範例（burst tries）

Fabrizio Sebastiani 的"**Machine learning in automated text categorization**"論文定義了 machine learning、text categorization、text classification 三個關鍵字，但這樣的定義不足以描述這篇論文的內容。這篇論文是在探討 information retrieval 的 text categorization 問題，採用了 machine learning 做為解法，並分別採用了兩種技術：train-and-test 以及 k-fold cross-validation，這篇論文可表示如下圖：

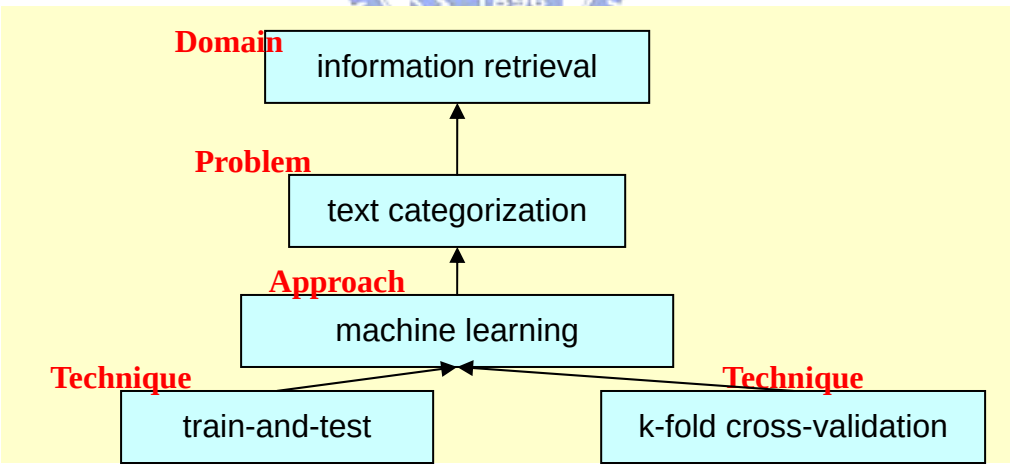

圖 3 - 3　論文關鍵字表示法範例（text categorization）

David F. Bacon 的"**Compiler transformations for high-performance computing**"論文定義了相當多的關鍵字：compilation、dependence analysis、locality、multiprocessors、optimization、parallelism、superscalar processors 等，這些關鍵字中，parallelism、multiprocessors、superscalar processors 是領域，compiler

optimization 是問題，這篇論文的解法是 compiler transformation，採用了 dependence analysis、locality 等技術，故其關鍵字可表示成圖 3 - 4：

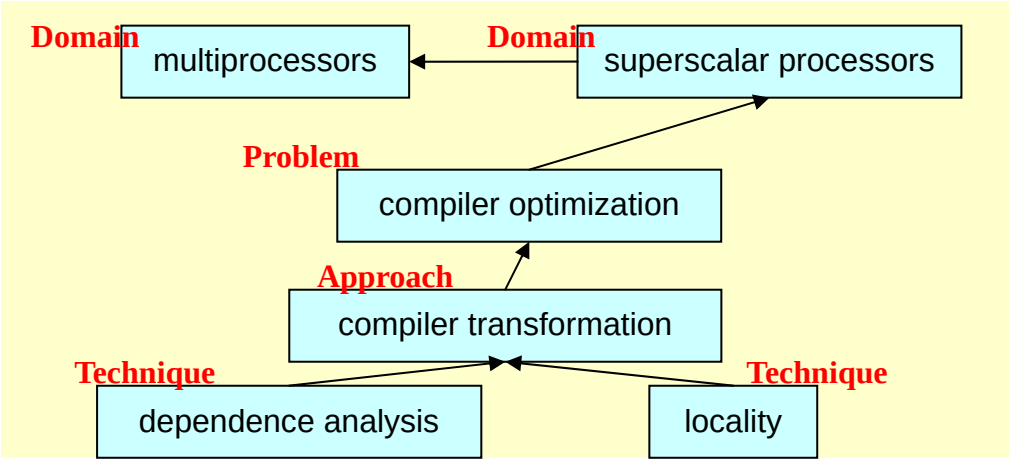

圖 3 - 4　論文關鍵字表示法範例（complier optimization）

Jin Jing 的"**Client-server computing in mobile environments**"論文中定義了非常多的關鍵字，只要論文有提到的內容都定為關鍵字，包括：application adaptation、cache invalidation、caching、client/server、data dissemination、disconnected operation、mobile applications、mobile client/server、mobile computing 等等，其實讀了這篇論文可發現主要是在說明 mobile client/server 的問題，提到了三種解法：mobile-aware adaptation、extended client/server model 以及 mobile data access 等，各自有其運用技術，可表示如圖 3 - 5：

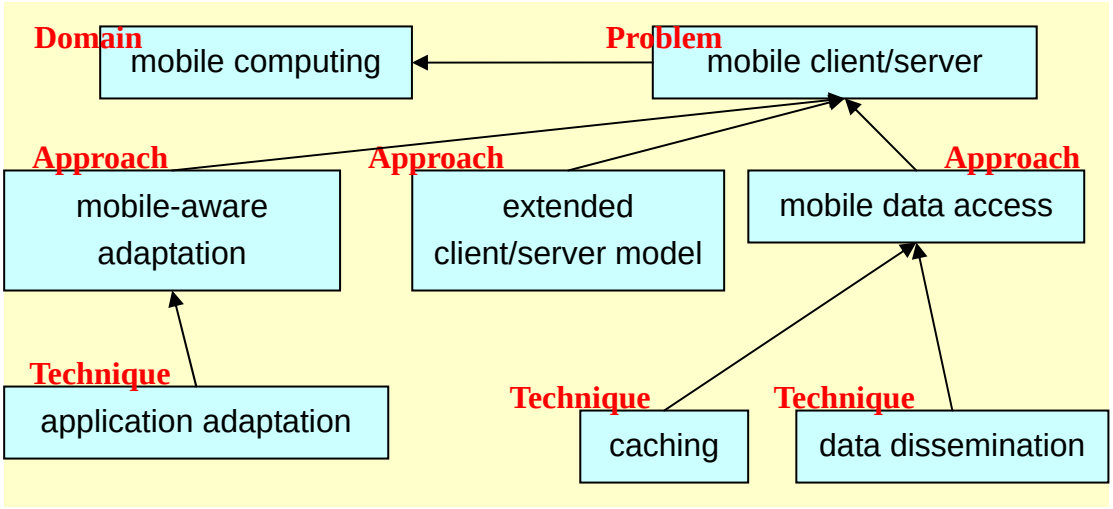

圖 3 - 5　論文關鍵字表示法範例（mobile client/server）

由上面的例子可看出技術論文的「問題導向」本質，藉由「領域」、「問題」、「解法」、「技術」四類關鍵字便能對其內容做一簡單的描述及索引，這樣的表示方法不像論文標題無法解析，也改善了傳統關鍵字缺乏定義的弱點，上述的例子有 database、information retrieval、multiprocessors、mobile computing 等各種不同領域的論文，但都能藉此共通的「問題導向」本質來描述，足見此方法的可行性。

3.1.2 技術報告本質

技術報告（Technical Report）是軟體研究者交換其研究成果常見的方式。技術報告與技術論文在格式上相當類似，但一般而言技術報告探討的主題較細，且主要目的在描述其發現，因此不會像技術論文有豐富的內容，但相對的技術報告往往都是最前瞻的知識。技術報告主要目的是在解釋研究者做了什麼、為何如此做、有哪些發現、發現的價值在哪裡等。技術報告跟技術論文一樣，也是針對一個特定主題進行探討，其探討的可能只是一個小改變、或是一些研究數據的呈現等；因此，技術報告的內容同樣可以視為「問題導向」，也就是其內容的方向由其處理的問題以及提出的解法決定。

技術報告在解決問題上要依據合理的論點（thesis），這個論點決定技術報告在探討問題上所使用的架構。舉例而言，有一篇針對 geometric Steiner tree problem 的 partitioning algorithms 論文，使用了 Steele's theory of Euclidean functionals 做為他分析 algorithms 效能的工具，則 Steele's theory of Euclidean functionals 即是這篇技術報告的論點。

技術報告的格式與技術論文類似，因此一般也會定義關鍵字做為索引，然而這也跟技術論文的問題一樣，零散且缺乏規則的關鍵字並沒有辦法有效率地做為知識索引，且技術報告在這方面的問題更大，因為技術報告原本就是針對較細的議題進行探討，故一般以其真正探討的主題做為關鍵字則通常過於侷限，讓使用者找不到資料，但若採用較寬的領域做為關鍵字，則又會讓只想找該領域知識的

使用者讀取不符需要的知識，例如 *S. L. Martins, M. G. C. Resende, C. C. Ribeiro, and P. M. Pardalos* 的”**A parallel hybrid GRASP for the Steiner tree problem in graphs using a hybrid local search strategy”**[37]這篇技術報告，它是要處理 graph 中 Steiner tree problem，提出了一個 parallel greedy randomized adaptive search procedure（GRASP）做為解決方案，但如果僅以 Steiner tree problem、parallel greedy randomized adaptive search procedure 做為關鍵字的話，則只有使用者完全了解他的問題，直接找尋「Steiner tree problem」才可能找到這篇報告，但若加入 graph 做為其關鍵字，則所有對「graph」有興趣的使用者都可能找到這一篇探討特定問題的技術報告，這都是因為關鍵字的定義缺乏法則及關聯性所導致的問題。

由於技術報告也是「問題導向」，一篇技術報告的內容主要是在「領域」中探討特定「問題」，運用一個「論點或技術」提出「解法」，故我們同樣可以將其關鍵字分為四類，即「領域」、「問題」、「解法」、「論點或技術」，這跟我們對技術論文關鍵字的分類是一樣的，而由於其「問題導向」的本質，這樣的分類也足以對其內容進行簡單的描述。

3.1.3 書本知識本質

書本知識對軟體開發者而言也是非常重要的知識來源。由於書本知識通常是一般性介紹，故書本知識的產生是在相關議題技術論文發表到一定程度之後才會出現，並非最前瞻的知識，其探討的內容也不會像技術報告或技術論文那樣深入，相反地是側重介紹及教學的功能，這對想要了解或學習特定主題的軟體工程師而言特別有幫助，一般人對一個不熟悉的領域通常都是透過書本知識來學習的。分析書本知識的特性將能夠幫助想要學習的軟體工程師快速找到所需的教材。

以知識產生的目的來看，書本知識是以介紹及教學為主要目的。各個主題都會有所謂的入門書籍，若此主題的知識較繁多，則可能會有進階書籍。無論是入

門書籍或進階書籍，書本知識都是偏向介紹性質的。也因此，書本知識不會針對某個特定議題進行深入探討，相反地，書本知識往往是對議題進行概略的介紹，讓讀者能夠對此議題有個初步的了解。跟技術論文、技術報告的專一性相反，一本書裡可能會提到數十個議題，因此書本知識是以廣博為主，而技術論文則以深入見長。

從技術書籍的架構來分析其本質。一本技術書籍的基本架構都包括有目錄（Table of Content）、序言（Preface）、簡介（Introduction）、章節內容（Chapters）以及索引（Index）。序言（Preface）的內容通常只是說明此書著作的背景以及適當哪些讀者，並不適合用來描述書本知識的內容。簡介（Introduction）部份是對此主題做一簡單描述，適合對此主題沒概念的使用者閱讀，但並不適合用來描述此書的內容架構。章節內容（Chapters）則是書本知識的主體，這項特性有助於我們對書本知識架構的理解；但由於章節的定義並沒有制式的規範，因此我們需要更佳的表示方法。索引（Index）其實可以看成是書本知識的關鍵字（Keyword），但因為索引也是缺乏定義及關聯性，因此一本書出現了特定的關鍵字（比如 compiler optimization），其內容仍無法預測，比如此書可能是介紹 compiler optimization 有哪些方法，也可能是在談 computer architecture 的時候提到 compiler optimization 的重要性等。

書本知識是圍繞在一特定主題（subject）之上，一般而言這個主題都會是書名，例如 Software Engineering、System Programming 等。在主題之下便是此書所介紹的議題（topic），依不同的書本特性，一本書可能以一個章節（chapter）或數個章節來討論此議題，一個議題下的內容依其本質可分為三類：

(1) 介紹此議題的章節內容：在討論介紹的過程中，一個議題可能會分成數個子議題（sub-topic），子議題又可能再細分為更小的議題（sub-sub-topic），這些內容具有階層架構的特性，例如 *Roger Pressman* 的 **Software Engineering**[38]，針對 Project Management 這個議題寫了四

章，這四章可分為四個子議題，分別是 Project Metrics、Risk Management、Project Planning、Project Scheduling 等。又如 *Richard Johnsonbaug*h 與 *Martin Kalin* 的 **Object-Oriented Programming in C++**[39]一書，在 Inheritance 這個議題上提到了四個子議題：Constructor、Polymorphism & Virtual Function、Destructor、Multiple Inheritance 等。

(2) 對特定議題的解法或方法：某些議題的介紹並不是分為子議題，而是針對此議題介紹各種不同解法或方法。這種類型的內容與前兩節討論的技術論文、技術報告雖然有些不同，但其本質仍屬「問題導向」，因此可採用上述的問題與解法來加以描述。在這裡問題即是書本討論的議題，而解法則是書本談到的解法或方法。例如 *Hafedh Mili* 等的 **Reuse-based Software Engineering** [40]一書中對 Domain Analysis 提出了如下的方法：Feature-Oriented Domain Analysis（FODA）、Organization Domain Modeling（ODM）、Join Object-Oriented Domain Analysis（JODA）、Reuse Library Process Model（RLPM）、Domain Analysis and Design Process（DADP）、Domain-Specific Software Architecture（DSSA），又或者演算法的書對排序方法介紹了如下的方法：Bubble Sort、Insertion Sort、Binary Sort、Quick Sort、Shell Sort 等。

(3) 介紹此議題所提供的範例：由於書本知識是以介紹為主要目的，因此在其說明過程中會以範例讓讀者易於理解。這種內容尤其容易出現在偏向實作面的書本，例如程式語言、資料結構、演算法、資料庫等等。範例有助於讀者對議題的理解，但其實並沒有真正增加書本介紹的知識範圍，因此這部份的內容不必加入對書本知識的描述中。

經由上述的分析，可知書本知識亦可籍由良好定義的關鍵字加以描述，這個關鍵字可分為「主題」、「議題」、「解法」三類：

(1) 主題（Subject）：主題即是一本書所探討的主要內容，通常會是書名。主題可以藉由一良好的分類架構將軟體技術知識的主題分類，使其定義標準化。

(2) 議題（Topic）：一本書的主題下會包含幾個主要的議題。這些議題下可能又可細分為子議題，由於這些子議題的內容組成主要議題，故亦可看成主要議題包含數個子議題。書本知識的議題間可能會有閱讀上的順序關係，例如閱讀 Object-oriented analysis 跟 Object-oriented design 之前應先閱讀 Object-oriented concept，因此他們會有依存的關係。

(3) 解法（Approach）：如前所述，某些議題的介紹並不是分為子議題，而是針對此議題介紹各種不同解法或方法，這類型便以「解法」定義。

書後的索引（Index）雖然也具有相當的參考價值，但由於索引實在太過龐雜，若索引中的每個字都做為書本的關鍵字，反而在描述上增加許多困難，且看不出書本內容的重點。而上述的三項關鍵字分類係依書本知識的架構得來，故以此三類關鍵字便能對書本知識做有效率的描述，書本知識關鍵字可以圖 3 - 6 表示：

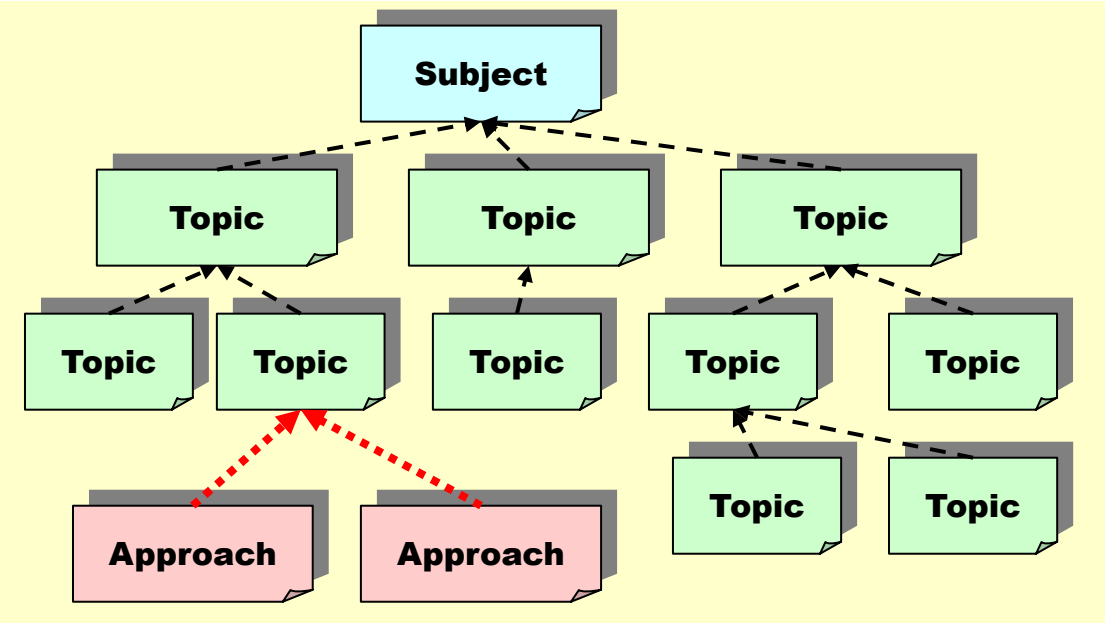

圖 3 - 6　書本關鍵字示意圖

以 *Roger Pressman* 的 **Software Engineering** 一書中針對 Project Management 這個議題的內容以上述的方式分類，便可得到如圖 3 - 7 的結果：

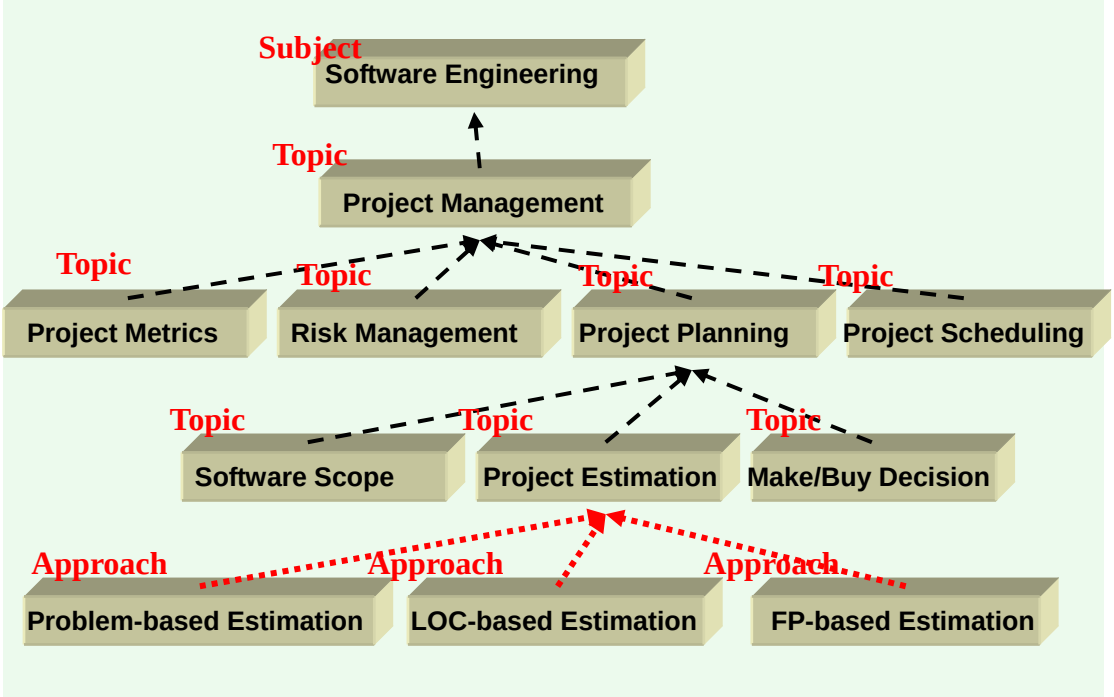

圖 3 - 7　書本關鍵字表示法範例（software engineering）

將書本知識以上述方式表示後，每個書本知識可看成一個個獨立的物件。這些物件可能因為討論的主題(Subject)相同，因而會在知識庫中分在同一類型中。此時知識庫必須整合這些書本知識提供給使用者選擇。在主題之下有書本的議題（Topic），故多本書的議題可以整合後取其聯集，讓使用者了解此主題下有哪些議題，例如介紹 Data Structure 的書，有的書提到 List、Queue、Stack、Hash Table，有的書提到 Tree、Hash Table 等，則知識庫可整合這些議題供使用者參考，若多本書談到了同一議題，則知識庫可列出各書在此議題下的子議題或解法（Approach）供使用者比較。這樣做對使用者來說非常方便，因為他可以快速掌握多本書的概略內容，找出有興趣的議題，進而取得適合的知識。

雖然書本知識與技術論文、技術報告的「問題導向」特性有些不同，但仍可將書本知識視為具有「問題」與「解答」特性的知識。因為對書本知識而言，使

用者的問題通常就是：特定主題應看哪本書？特定議題的介紹？特定議題有哪些方法？而針對這些問題的解答就是書本的內容，透過「主題」、「議題」、「解法」三類關鍵字並配合階層架構，使用者能夠知道特定主題有哪些書，這些書提到了哪些議題，特定議題下又介紹了哪些子議題，或介紹了哪些方法，因此，「主題」、「議題」可視為問題，而「子議題」、「解法」便是解答。

透過上述的表示方法，可為書本知識定義完整的關鍵字做為描述及索引，也能夠讓使用者快速掌握書本知識的內容，並可對特定主題、特定議題的書本知識進行概略的比較，增進書本知識的效率。

3.2 以問題－解法為基礎的軟體知識分類方法

傳統的知識分類方法大都依知識所談到的內容加以分類（如 Taxonomy、Faceted Classification）或描述（如 Case-based Reasoning、Ontology），但鮮少依知識特性及使用者的需求考慮。因此當知識量累積過多時，常常讓使用者無法快速取得他所需要的知識，使得知識庫缺乏效率。例如擁有大量論文的 IEEE 電子期刊資料庫、ACM 電子圖書館，雖然提供使用者知識的分類，也能夠讓使用者以關鍵字查詢，但動輒上千篇的查詢結果卻讓使用者無法快速掌握其所需的知識，使用者與系統的溝通僅能透過一些未經定義的關鍵字，因此使用者必須浪費許多時間閱讀知識後發現此知識並非他所需要的。照理知識庫擁有愈多的知識愈好，但沒有效率的知識分類法讓大量的知識變成系統及使用者的負擔，這是非常可惜的。

依上一節所分析的，技術論文及技術報告皆是「問題導向」的知識，其目的是為了提出對問題的解法。此外，使用者使用知識庫的目的可分為要了解某領域、要深入了解某議題、找尋特定的問題的解答或了解技術的運用領域。從知識的特性及使用者的需求，都能夠發現「問題－解法」是知識建構的一個重要基礎，

技術論文、技術報告是根據問題提出解法，而使用者找尋知識時也是在尋找問題的解答。

因此，我們可以利用「問題－解法」為基礎提出一個軟體知識分類方法，若我們能夠有效率地描述論文所處理的問題及其使用的解答，則使用者便可以快速地找出他所需要的知識。更進一步，這套分類方法應該能夠自動對知識進行分析，以一些準則對解法排序，避免使用者時間的浪費。此外，這套知識分類方法也要能夠描述書本知識，讓使用者能找尋特定主題的介紹性知識。

根據 3.1.2 節對軟體知識本質的分析，發現軟體知識的關鍵字是最常見的索引依據。然而目前的軟體關鍵字缺乏良好的定義法則，故顯得零散且無關聯性，由其知識本質，我們可將技術論文及技術報告的關鍵字分為四類，即「領域」、「問題」、「解法」、「技術」。這四個分類正好符合「問題－解法」架構，因為要描述一個問題，應當先描述其所在領域，軟體知識的領域是有階層性的，一個大領域可分為數個小領域，因此描述知識的領域最恰當的方式就是利用階層架構（Hierarchy-structure），故我們可以稱領域為**領域類別（Category）**，有了領域之後，我們便可以在階層架構上加入對問題的描述。這些問題便可稱為該領域下的**「議題」（Issue）**；而描述一個解法，重點就在於**「解決方法」（Approach）**以及運用的**「技術」（Technique）**，因此，我們只需利用有良好定義的關鍵字，即可對技術論文及技術報告進行有效率的「問題－解法」分類與描述。

書本知識依 3.1.3 節的分析，可將其關鍵字分為**主題（Subject）**、**議題（Topic）**、**解法（Approach）**三類，這個分類能夠與上述的分類整合。因為書本知識所探討的主題其實與上述的領域類別（Category）是很接近的。我們利用階層架構將知識領域做分類，這些分類其實都可做為書本知識的主題，例如分類上有 Software Engineering 的領域類別，也有書本知識的主題為 Software Engineering，如 3.1.3 節所述，我們可以藉由一良好的分類架構將軟體技術知識的主題分類，使書本知識的主題定義標準化，因此，只要這個分類架構具有足夠

的深度與內容，便可做為書本知識的主題。而書本知識下面的議題（Topic），雖然與技術論文的議題（Issue）意義不同，但亦可將 Topic 視為 Issue，只是書本知識的 Issue 是有主要的 Issue，底下可再細分為 sub-Issue 或 sub-sub-Issue，而技術論文的 Issue 則是領域類別下的問題，其性質不同，但都可以用議題（Issue）來代表。由於書本知識與技術論文本質上就不同，因此也不會造成定義模糊的問題。至於書本知識的解法（Approach），與技術論文的解法也有些許差異，書本知識通常是針對一個議題介紹幾種解法，而技術論文則是針對一個問題提出其解法，但在本質上仍是問題與解法的關係，故可與技術論文的分類整合。

誠如 3.1.3 節所述，雖然書本知識與技術論文、技術報告的「問題導向」特性有些不同，但仍可將書本知識視為具有「問題」與「解答」特性的知識。透過「主題」、「議題」、「解法」三類關鍵字並配合階層架構，可將「主題」、「議題」視為問題，而「子議題」、「解法」視為解答。

因此，技術論文、技術報告、書本知識三項技術知識皆可以「問題－解法」為基礎建構其關鍵字分類法，由 3.2 節對這些技術知識本質的分析，可知這樣的分類對其內容已能提供足夠的描述，在各種領域中，「問題－解法」是不變的法則，故這樣的定義是充份且可行的。事實上，定義關鍵字原本就是撰寫技術知識的必要過程，以此方式描述知識也不會增加作者的負擔，相反地，卻能讓他更容易地定義有意義的關鍵字。

技術知識關鍵字依上述的「領域類別」、「議題」、「解法」、「技術」分為四種類型後，可進一步發現這些關鍵字類型以及技術論文、技術報告、書本知識等都可利用物件導向（Object-oriented）的中的類別（Class）或是物件（Object）來表示。

首先觀察技術論文與技術報告的關鍵字，「領域類別」是階層架構的樹狀結構，每一個領域類別都可視為一個 Class，上層的領域類別包含下層的領域類別，

就像是物件導向中的包含（Aggregation），最下層的領域類別底下會有議題（Issue），即技術論文與技術報告所要解決的問題，這些問題在領域類別之下，故可視為領域類別包含這些議題。一個議題會有幾個解法（Approach）試圖解決它，這就像是物件導向中一個介面（Interface）會有一些類別實作它，因此，議題可視為介面，這個介面會有一些解法去實現（Realization）它。對特定的議題，解法可能不止一種，例如對 text categorization 的議題，有些論文可能採用 machine learning 的解法，有些論文則可能採用 neural networks 的解法，這些不同的解法，可看成不同的類別在實作議題，解法之間可能會有改良的關係，即一種解法是改良自另一種解法，這個關係就像是類別之間的繼承（Inheritance）一樣。而很多論文可能會使用同一種解法，這些論文便可視為該解法類別的物件（Object），這些論文物件因其內容不同，故運用的技術亦不同；這些技術同樣可視為不同的類別，被論文物件所使用（Use）。當論文採用相同的解法，並運用了同樣的技術，則表示這兩篇論文非常類似，新的論文能夠被期刊接受並登出，表示新的論文應有一些改進舊論文之處，故可視為新論文是改善（Refine）了舊的論文。

因此，關鍵字類型與技術知識可用類別圖（Class Diagram）來表示，如圖 3-8：

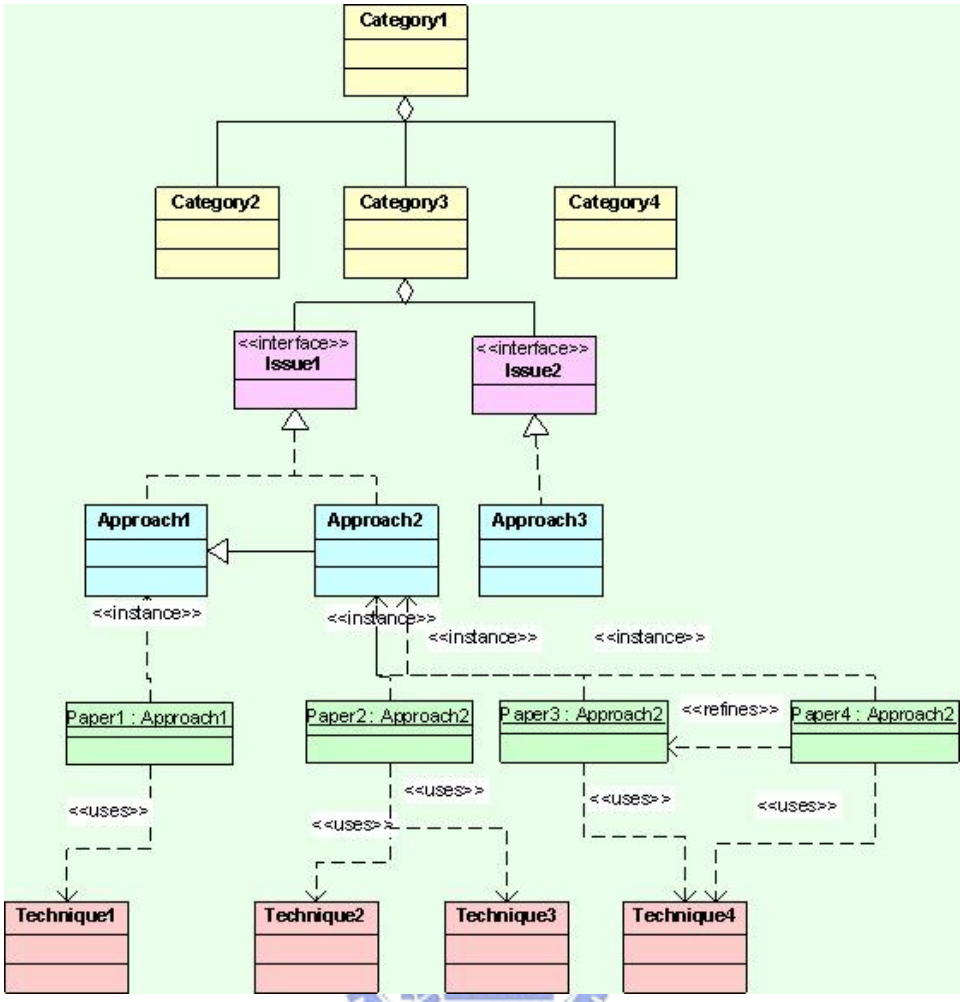

圖 3 - 8　技術知識關鍵字類別圖

書本知識的關鍵字也同樣具有物件導向的特性，如上所述，書本知識探討的主題（Subject）亦可由階層架構分類，故與領域類別一樣，上層的領域類別包含下層的領域類別，許多本書可能會探討相同的主題，這些書都可視為該類別的物件，這些物件依其內容不同，包含了各自的議題、子議題，而這些議題、子議題也都可視為不同的類別，不同的書可能會包含相同的議題，這些物件所包含的議題聯集即為領域類別上的書本議題集合，議題之間可能會有閱讀先後的依存關係，就如同類別之間的依存（Dependency）關係。有些議題書本知識是提供方法或解法，這就跟上述技術論文或技術報告的特性一樣，這時議題就可視為介面，而解法則可視為實作它的類別。

因此，書本知識與其關鍵字可以圖 3 - 9 的類別圖（Class Diagram）表示：

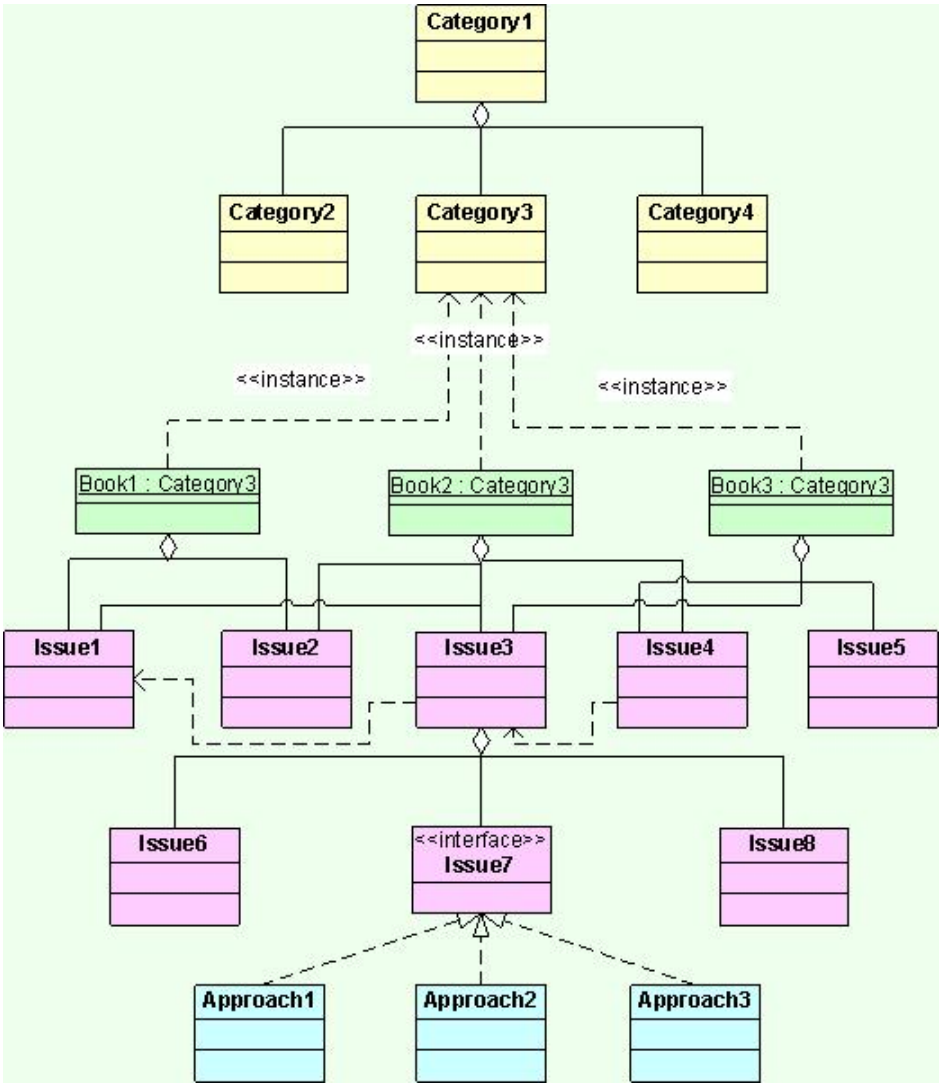

圖 3 - 9　書本知識關鍵字類別圖

由上述的分析，可得知關鍵字類型與知識具備物件導向特性，可以用類別來表示關鍵字類型，用物件表示知識。採用物件導向的方式讓關鍵字與知識之間產生了許多關聯性，藉由這些關聯性能夠讓知識庫對知識分析並增進搜尋效率。此外，關鍵字是類別，故可以靜態地儲存在知識庫中，知識是物件，故能夠自由新增至知識庫，增加了知識分類的彈性。將議題當做介面，而以解法來實作的概念，讓使用者很容易找到議題的各種解法，以及其下有哪些知識物件。藉由解法之間的繼承（Inheritance）以及知識物件之間改善（Refine）的關係，使用者可以對解法與論文進行快速而有效率的比較，直接閱讀適合的論文，減少時間的浪費。

知識的關聯性能夠讓一個個獨立的知識形成一個知識網路，一個知識能夠依

不同的關係找到其他的知識，藉以滿足使用者各種不同的需求。這些關係可整理如圖 3 - 10 所示，簡述如下：

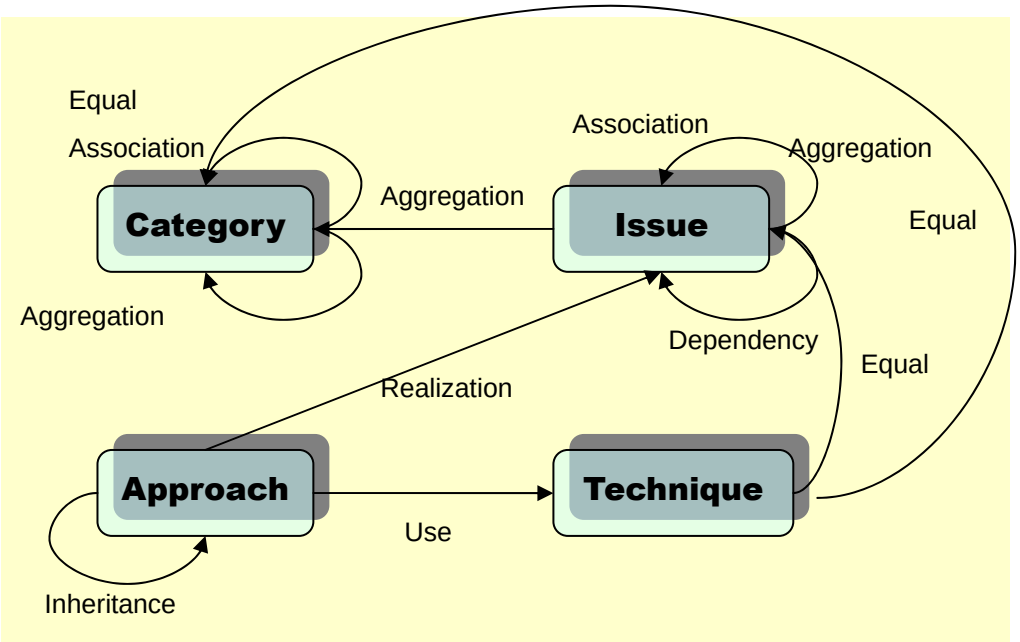

圖 3 - 10　關鍵字分類及關聯性示意圖

(1) 包含（Aggregation）：關鍵字中的類別是層狀結構，因此彼此之間有著包含的關係，書本知識的議題之間亦有包含的的關係；在物件導向中的 Aggregation 係指一個物件由其他物件所組成，在意義上是極相似的。因此利用 Aggregation 來表示類別之間的包含關係，此外也用來表示類別下所含議題的關係。

(2) 實現（Realization）：一個特定的議題會有幾個解法試圖解決它，這樣的關係就好比物件導向中一個定義好的介面（Interface），有一些物件實作它（Implement）。這樣的關係在物件導向中稱為 Realization，因此我們利用 Realization 來代表議題與解法之間的關係。

(3) 使用（Use）：一個解法會運用一些技術，這樣的關係就類似物件導向中一個類別使用了另一個類別一樣。因此我們就稱解法運用技術的關係為使用（Use）。

(4) 繼承（Inheritance）：新的解法可能會參考舊的解法，這在論文之中尤其常見，這樣的關係通常是針對舊的解法加以改進，提出更好的解法。而在物件導向中的繼承係指子類別繼承了父類別的功能，並加上一些修改以提供新的功能，這與解法與解法間的參考關係是相當類似的，故我們利用 Inheritance 來代表解法參考解法的關係。

(5) 相等（Equal）：由於我們利用了 Taxonomy 的樹狀結構來做知識分類，倘若使用者一開始選擇了一個類別，就只能找到該類別下的知識。但其實知識的分類並不是單純的樹狀，例如 Internet → WWW → Programming 與 Programming → WWW 其實是相同的，故我們的分類之中必須建立一些相等的關係，使得使用者不會因為一開始選擇的類別而找不到他所需的知識。此外，解法參考的技術也可能是一個議題或類別，這樣的關係也可利用相等來代表。在物件導向中並沒有 Equal 這樣的關係，故只能建立 Equal 關係來表達。

(6) 關聯（Association）：除了上述所談的關聯之外，在類別之間、議題之間與解法之間會有一些意義上的關聯性，例如 Graph 跟 Computation Geometry 其有某種程度的相近，但又不是完全相等，則可利用 Association 來表達這樣的關係。在物件導向之中也同樣有 Association，這是表示物件之間有結構上的關聯性，兩者都是在表示某種意義上的相關性，因此利用 Association 來表達意義上相關的關係。

(7) 依存（Dependency）：在書本知識中，議題之前有先後閱讀的關係，故可視為依存關係。而物件之間亦有依存（Dependency）的關係，用來表示物件之間互相影響，故以 Dependency 來表示書本知識中議題先後閱讀的關係。

(8) 改善（Refine）：當技術論文或技術報告使用相同的解法，並運用了相同

的技術時，藉由其發表時間，可知新的論文是自舊的論文改善而來。物件導向中的改善（Refine）指的是抽象層級的不同，如 analysis class 可以 refine 成 design class，這之間一樣是有更加深入及精鍊的意義，故採用 Refine 來表示相同解法與技術的論文其改進關係。

以使用者存取知識庫的角度來看，可分成找尋領域的介紹、深入了解特定議題、找尋問題的解答以及了解技術的運用狀況等四種需求，對找尋領域介紹的使用者而言，可藉由上述的領域類別（Category）以及類別之間包含、相關、相等的關係對領域類別進行瀏覽，找出有興趣的領域，找到類別之後便可列出相關的書本知識，進一步列出其中的議題（Issue）供使用者參考，並選擇想了解的內容進行閱讀。至於想深入了解特定議題的使用者，便可利用類別與議題之間的包含關係找到特定議題，並找出適合的書本知識進行閱讀。至於找尋問題解答的使用者則可先找到議題，再藉由議題與解法的實現（Realization）關係找到解法，並可藉由使用（Use）關係快速掌握解法使用的技術，對解法進行簡單的比較。對於想了解技術運用狀況的使用者，透過使用（Use）關係可找到運用特定技術的解法，再由實現關係便可找到議題乃至於領域，便可了解技術的運用狀況。此外，由於我們是以關鍵字做為描述的基礎，故傳統的關鍵字搜尋亦能幫助使用者，且「問題－解法」的關鍵字分類法對關鍵字有嚴謹的定義，故使用者可直接輸入有興趣的領域、議題、解法或技術，不會找出許多根本不適合的知識。如此知識庫的知識愈多，愈能夠讓使用者找到有用的知識，達到軟體知識管理的真正目的。

當使用者閱讀知識之後，也可能會想取得其他相關的知識；這種情況又可分為幾種類型：想進一步了解目前知識中提到的技術、想知道此解法被哪些解法改進或運用，這時便可利用上述的繼承（Inheritance）與運用（Use）關係來達成。

因此，上述的關係能夠讓使用者找出需要的知識，增進知識庫的效率，且能讓使用者取得相關的知識，足以達到知識管理的需求。

有了關聯性後，可以更進一步地以符號的方法來表示：

(1) 包含（Aggregation）：≻

(2) 實現（Realization）：←

(3) 使用（Use）：⊕

(4) 繼承（Inheritance）：▷

(5) 相等（Equal）：≡

(6) 關聯（Association）：⇄

(7) 依存（Dependency）：►

(8) 改善（Refine）：⇒

如此，我們便可以運用符號以及關鍵字來表示一個知識，例如前面的例子，*Steffen Heinz* 的"**Burst tries: a fast, efficient data structure for string keys**"，其內容是處理 text database 領域中的 string data structure 問題，他提出了稱為 burst tries 的解法，此解法運用了 splay trees、tries、binary trees、hash table 等技術，故可以符號表示如下：

(database ≻ text database) ≻
string data structure ←
burst tries ⊕
(splay trees, tries, binary trees, hash table)

此解法是改善 ternary search tree 的，故可表示如下：

(database ≻ text database) ≻
string data structure ←
(burst tries ▷ ternary search tree) ⊕
(splay trees, tries, binary trees, hash table)

技術報告也可以用這種方式表示，如 3.2.2 節中舉的例子 *S. L. Martins, M. G. C. Resende, C. C. Ribeiro, and P. M. Pardalos* 的"**A parallel hybrid GRASP for the Steiner tree problem in graphs using a hybrid local search strategy"**，它是要處理 graph 中 Steiner tree problem，提出了一個 parallel greedy randomized adaptive search procedure（GRASP）做為解決方案，因此可表示如下：

```
graph ≻
      Steiner tree problem ←
          parallel greedy randomized adaptive search ⊕
                (greedy, hybrid local search,
                 randomized search)
```

書本知識同樣能透過此方式進行描述，如 3.2.3 節中舉 Pressman 的 Software Engineering，其中 Project Management 的部份可表示如圖 3 - 7，為了方便起見，再將此表示圖置於下方：

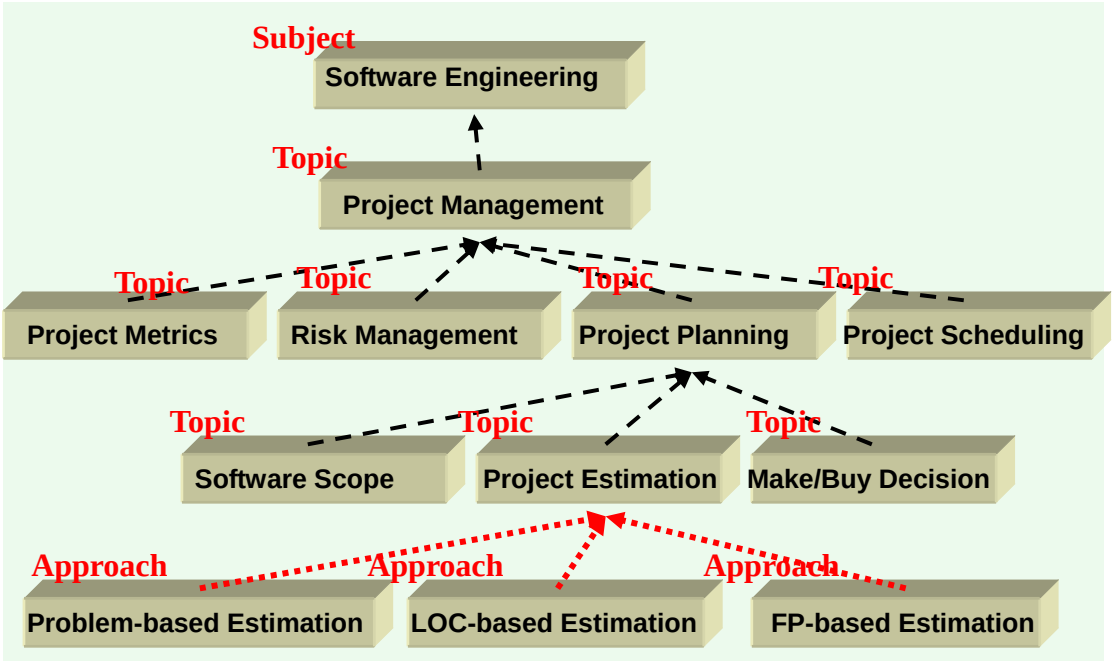

圖 3 - 7　書本關鍵字表示法範例（software engineering）

若表示成符號則可寫成：

```
software engineering ≻
project management ≻
      ( project metrics, risk management,
        project planning ≻
           (software scope, project estimation ←
              (problem-based estimation,
               LOC-based estimation, FP-based estimation)
            , make/buy decision)
         , project scheduling)
```

利用符號來表示樹狀結構對人來說雖然比較難以理解，但對電腦系統而言卻

是非常容易解析；只要定義完整的 BNF（Backus-Nauer form）以及 syntax，即可透過 lex & yacc 產生解析器（parser），對此格式的字串進行解析。

以「問題－解法」為基礎的知識分類法由於將關鍵字完整定義，故能夠提升技術知識搜尋的速度、準確性，且能依使用者不同需要給予適當的知識，且由於「問題－解法」的原則在軟體知識中任何領域皆通用，故不會侷限於特定的領域中，更重要的是這套方法能夠利用符號清楚而有效地描述知識，若技術論文、技術報告以及書籍作者都透過此方法來定義並描述其關鍵字，則所有的知識皆能夠在一個統一的架構下交換使用，知識庫之間能夠合作、交換知識，更進一步甚至能夠架構一個知識分享平台讓所有的技術知識在這平台上提供、分享、搜集、買賣知識，這個符號表示法也能夠採用 XML（Extensible Markup Language）來定義規格，則知識的分享便成為簡單的任務，大幅增進軟體知識管理的效率。

3.3 效益比較及分析

在前一章提到了四種常用的知識分類方法，包括 Taxonomy、Faceted Classification、Case-based Reasoning、Ontology。這四種分類方法各有其優缺點，在 3.1.2 節中也曾提出這些方法的問題，以下將以我們提出的「問題－解法」知識分類法與上述四種方法進行比較與分析。

與Taxonomy的比較

Taxonomy 是以樹狀架構為基礎的知識分類法，是最直觀也最為常用的方法。Taxonomy 最主要的分類是以領域為主，例如著名的杜威十進分類法（Dewey decimal classification），在第一層是以大領域分類，如自然科學、語言學、社會科學等，再下一層就更細分，如自然科學可分為數學、天文學、物理、化學等等。以軟體知識的 Taxonomy 為例，最著名的是 ACM 分類系統（ACM computing classification system），第一層是軟體知識的大領域，如硬體（Hardware）、計算

機組織（Computer System Organization）、軟體（Software）、資料（Data）、計算理論（Theory of Computation）、計算數學（Mathematics of Computing）等等，再下一層就再細分，例如軟體可分為程式技術（Programming Techniques）、軟體工程（Software Engineering）、程式語言（Programming Languages）、作業系統（Operating System），這個分類主要以三到四層為主，從上述的分類可明顯看出這是對知識領域的分類。

對知識領域的分類固然重要，但單純的 Taxonomy 只能對知識做一簡單的分類，當相關的知識愈來愈多時，相同分類下的知識就變得繁雜且難以比較。使用很難在特定領域中找出合適的知識，這對增長快速的軟體知識而言非常沒效率。像技術報告、技術論文這些知識都是日新月異，同一個問題可能會有許多不同的論文，運用不同的解法與技術來解決，純粹以領域來分類實在沒辦法滿足軟體知識的需求。

相較之下，「問題－解法」軟體知識分類法也有樹狀架構的領域類別，但在類別之下加入了「議題」，使類別與議題之間有所區隔，也可同時整合書本知識與技術論文等。此方法採用了解法實作議題的概念，並將解法定義為物件導向中的類別（Class），使得技術論文能依其特性做良好的分類，並提供了更高的彈性。此外，技術論文之間的改善（Refine）關係能夠讓使用者快速取得最合適的知識，而在論文之下尚有其運用的技術，基於技術論文的本質，這樣的描述既簡單又有效率，且當讀者對特定技術不熟悉時，還可藉由此技術找到相關的知識，讓知識不再只是在樹狀架構下固定的分類，而是可以自動分析、自由連結的知識網路，讓知識更有效率地分類及索引，大幅縮短使用者找尋知識的時間。

由上述可知，Taxonomy 方法僅對領域類別加以分類，只有「問題－解法」架構中的類別（Category）功能，缺乏議題、解法、技術等技術知識特性，而樹狀架構下的知識又缺乏關連性，因此相較之下，「問題－解法」架構對技術知識而言較為合適，且能提供更佳的知識管理效率。

與Faceted Classification的比較

Facet Classification 是利用清楚定義、完全互斥及完整窮舉的知識主題、特性來進行分類，其特色是讓知識分類具備更高的彈性及自由度，不必侷限於之前的分類架構，隨時都可以新增一個 Facet 分類。因此一項知識可依不同面向來分類，如 IBM 對於物件導向元件分類（Classification of object-oriented components），就有如演算法（Algorithm）、應用領域（Application domain）、元件大小（Component size）、開發標準（Development standard）、功能（Function）、實作語言（Implementation language）等等面向的分類。由於 Facet 的建構與維護都較 Taxonomy 繁複，且技術論文、技術報告或書本知識都缺乏明顯互斥的分類面向，因此對此未有標準的 Facet 分類系統。對技術論文而言，其 Facet 除了領域類別外，還可能包括如作者、期刊等，而書本知識則可能有作者、出版社、ISBN 等面向，這些面向雖然有助於縮小檢索範圍，但對知識內容的描述沒有幫助，而主要的領域類別僅是一個單純的 Taxonomy，其缺點如前所述，知識量增加時將對使用者造成負擔。此外，即使定義了更具意義的描述面向，Facet Classification 對使用者而言仍不方便。因為使用者必須從各個面向的大類別往下找尋，無法直接利用簡單的關鍵字，由於使用者並不清楚知識工程師的分類架構，對某些面向而言這樣的方法是相當耗力的，尤其當 Facet 愈來愈多，分類階層愈來愈多時，在這裡面找尋知識是費時費力的。

而「問題－解法」分類架構採用樹狀架構的領域類別，在其下採用了「議題」、「解法」、「技術」等三種不同類型的關鍵字，增加了對知識的描述，且使用者不必透過各種面向去搜尋，可直接利用上述類型的關鍵字找尋，例如使用者可直接找 Travel salesman problem 的解法，也可指定要找 Greedy 或 Dynamic programming 解法的論文，或更進一步指定特定的技術等，找到知識之後利用此分類架構中的物件導向關係可以找到使用相同解法與技術的論文，或找相關解法、相關技術的知識，具備高度的彈性，藉由關鍵字搜尋比起 Facet Classification 方法快速而直

觀，對使用者而言較為方便。

因此，Facet Classification 雖然可對知識定義較多面向，但無法描述解法、技術等知識，且難以將書本知識與技術論文、技術報告整合，提供各種不同需求使用者適當的知識；知識間也僅有各面向的同類關係，欠缺知識比較與技術相關性，使得知識僅是單純地以各面向分類，而無法對知識做較有意義的描述。「問題－解法」知識分類法能改善上述問題，因此是較佳的分類方法。

與Case-based Reasoning的比較

Case-based Reasoning（以下簡稱 CBR）的原則是以過往的經驗來解決新的問題，因此對知識的處理是先分析，再經過特定的程序使知識能符合其模型，當使用者對知識有需求時，透過其介面對案例（Case）進行描述，系統將此案例分析後找出最合適的方案。由於此方法牽涉到對案例與解決方法的內容描述，故其模型往往相當複雜，且有一定的領域限制，沒辦法運用在各個不同領域中。對 CBR 方法而言，各個案例都是單獨的存在，彼此之間僅有一些共通性的關聯，而沒有架構良好的分類系統。這對軟體知識而言是不合適的，因為軟體知識有非常多的知識領域，而各領域之下又有各種不同的問題，這些問題若沒有經過知識領的分類，將顯得雜亂而難以搜尋，且不容易描述與分辨。例如資料結構中的 Binary search tree 與搜尋引擎中的 Inverted file 都會有改善最差狀況（Improve worst case）的問題，這看似類似的問題，由於其領域不同，相關的解法與技術也完全不同，因此，CBR 方法以單純的「案例」對「解決方法」來描述知識，不足以對領域既廣且深的軟體知識做有效的分類。

「問題－解法」軟體知識分類法，採用了定義良好的階層架構對知識領域分類，在領域之下才定義議題與解法，並更進一步定義了各論文使用的技術，使得議題有了良好的分類，讓技術論文、技術報告與書本知識都能依其知識領域不同做最基本的分類，有效地依內容縮小知識範圍，且能透過關鍵字類別的關係找到

其他相關的知識，增強知識的重用性與延伸性，相較於僅以「案例」與「解決方法」來分類的 CBR，這個方法更適合用於複雜領域的軟體知識中。

與Ontology的比較

Ontology 係指領域知識定義一個共通的分享方式,運用正式且清楚定義的模型，籍以描述知識的意義。Ontology 也可以利用樹狀架構來分類，並搭配關鍵字以及特殊的關聯性對知識搜尋，然而 Ontology 為了要建構良好的知識模型，因此經常必須運用具有領域特性的描述方法，不同領域所定義的 Ontology 也很難通用。例如著名的 On-To-Knowledge 其文件的 Ontology 可表示如圖 3 - 11：

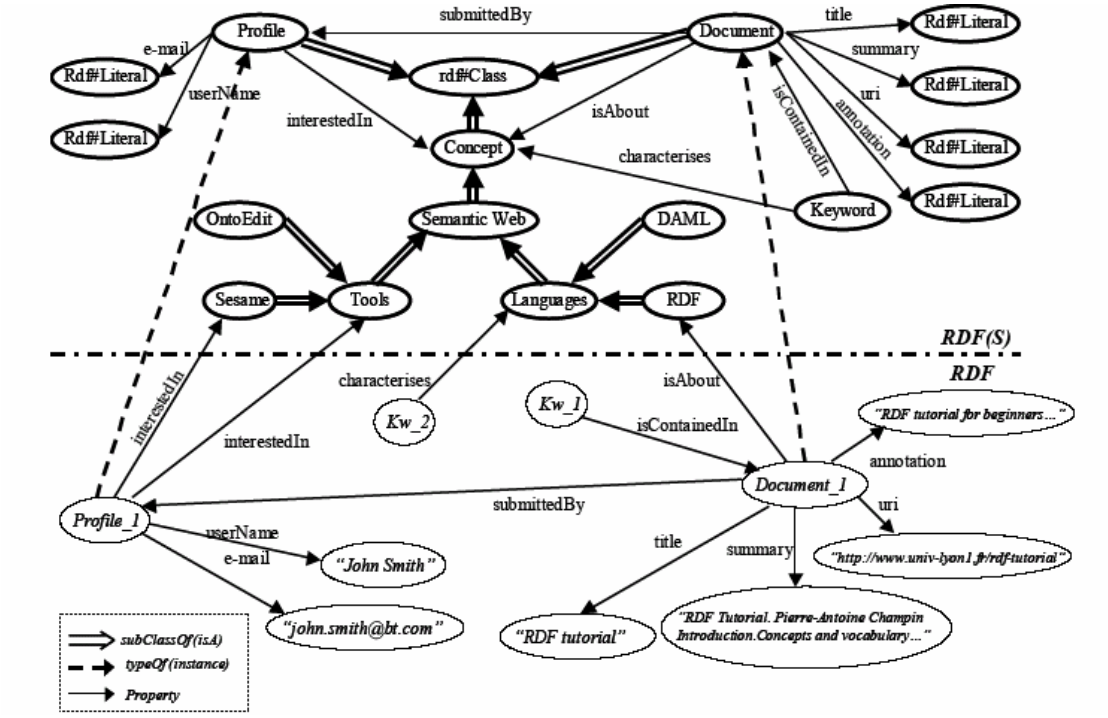

圖 3 - 11　On-To-Knowledge Ontology 模型圖

如上圖所示，其對文件做了詳盡的描述，如標題（title）、摘要（summary）、作者、相關概念、附件等，而作者也描述其名字、e-mail 等資訊，這樣的描述固然能夠改善傳統關鍵字分類的缺憾，但純粹以各類特性來描述仍有其問題。以上述「問題－解法」知識分類法而言，亦可籍由 Ontology 方式描述技術論文的類別、議題、解法、技術等特性，但這樣的描述法讓知識之間缺乏關聯性，使用者無法在閱讀知識之後快速取得相關知識，也沒辦法對知識進行比較，當知識量愈

來愈多時，單一議題下的解法非常多，Ontology 僅能提供這些解法，卻沒辦法對這些解法進行比較分析，造成使用者必須閱讀全部知識才能判斷，非常沒有效率。Ontology 為增進效率，就必須提供更詳細的知識描述模型，但這個方法的領域限制性，就無法運用在各個領域的軟體知識上，要對每個領域都要發展其 Ontology 也非常耗時耗力，且造成使用者搜尋時的困擾。

由上可知，「問題－解法」知識分類法除了對知識做良好的描述外，還建構了知識的物件導向關聯性，使知識能夠互相比較與參考，達到真正知識網路的概念，比起 Ontology 僅對知識描述更有效率。

綜合上述，可整理各分類法與「問題－解法」分類法的比較如下表：

	Taxonomy	**Faceted**	**CBR**	**Ontology**	**Problem-Approach**
關鍵字與論文關係	僅能表示論文領域	能表示論文領域及其他面向，但難以應用至所有領域	能表示論文的問題及解法，但難以應用至所有領域	依描述模型而定，可表示領域類別、使用概念等	以論文所在領域、解決議題、運用解法及技術表示其性質
對知識領域分類	樹狀架構	樹狀架構，可多面向	無	樹狀架構	七層樹狀架構
找出特定議題之解法	無，僅能找領域下的論文	無，僅能依其他面向縮小範圍	有	依模型而定	有
找出特定技術之運用論文	無	無	依模型而定	依模型而定	有
找介紹性知識	無	有	無	有	有
同性質解法排序	無	無	有	無	可依繼承、改善等關係進行比較

表 3 - 1　各分類法比較表

第四章、軟體知識的關鍵字分類機制

前一章提出了以「問題－解法」為基礎的軟體知識關鍵字分類法，本章將進一步介紹分類機制以及搜尋及排序法則。4.1 節將說明關鍵字的分類及關聯性建構方法；4.2 節則說明領域類別架構的設計概念；4.3 節描述此分類法的知識搜尋及排序法則；4.4 節中則說明符號表示法及其 BNF（Backus-Nauer form）。

4.1 關鍵字分類建構法則

在第三章中提出了以「問題－解法」為基礎的關鍵字分類法，將軟體知識的關鍵字區分為類別（Category）、議題（Issue）、解法（Approach）、技術（Technique）四個類型，當關鍵字分成這些類型後，就能夠更進一步地建構知識的關聯性。這些關聯性有助於知識庫對知識的搜尋與分析，也能讓使用者在閱讀知識後取得相關的知識內容。由於物件導向所提出的物件關係與知識之間的關係有相當高的相似程度，因此我們可以利用物件之間的關係來表達知識之間的關係。

這些分類及關聯性需要清楚定義，方能使知識作者對其關鍵字有效地分類，讓所有的知識在一個統一的架構下交換使用，使知識庫之間合作、交換知識，真正達到軟體知識管理的最主要目的：將正確的知識在適當的時機提供給需要的成員。

建構知識關鍵字時，關鍵字類型的選擇原則如下：

(1) **類別（Category）**：類別指的是知識所在的領域類別。為了讓知識能夠互相交換溝通，將建構一個公開且維護良好的知識分類，類似網景公司提出的 ODP（Open Directory Project）或是 ACM 分類系統（ACM computing classification system），供知識工程師或知識作者選取適當的領域類別，這個類別架構應提供足夠的深度與彈性，對軟體知識領域適當地分類。

有了這樣的分類架構，對於知識領域的選擇就有標準可循，分類架構將在 4.2 節中詳細討論。

選擇技術論文或技術報告的領域類別時，應依據其探討的主題，在分類架構上從大至小縮小範圍，選定類別後，再循父類別定義至根類別。舉例而言，若分類架構中有 Data structure →Tree →Binary search tree 的類別，當一篇論文所談論的內容是 Binary search tree，則其領域關鍵字即是 Data structure、Tree、Binary search tree，但這些關鍵字之間有包含（Aggregation）的關係，故知識庫能夠知道此知識是在討論 Binary search tree，不會造成模糊不清的問題。以符號方法表示，即為：

Data structure ≻ Tree ≻ Binary search tree

書本知識的類別其實就是其探討的主題，一本書只會有一個主題，只要分類架構有足夠的深度與廣度，每一本書都能夠在此分類架構上找到適合的類別，選擇類別的方法一樣是由大至小縮小範圍，再循父類別定義至根類別。

(2) **議題（Issue）**：議題關鍵字對技術論文或技術報告而言是指其解決的問題，對書本知識而言則是其介紹或探討的議題、子議題等。由於其意義不同，故定義法則也不相同。

對技術論文或技術報告而言，由於具有問題導向的本質，故找出其解決的問題非常容易。問題在於描述問題的用字，定義關鍵字雖然比直接用自然語言描述明確地多，但同義字或類似字詞仍會造成知識分類時的困擾。如前一章舉的例子中，有一篇論文的關鍵字就有 text classification、text categorization 兩個同義字，即是因為這兩個詞都有人用，所以必須將這兩個詞都定為關鍵字。為解決此問題，知識分類架構中應在類別底下加入目前已有的議題，由於知識作者必須在分類架構中

選擇類別，選定類別後便可列出議題以供作者挑選；倘若這篇論文是在探討新的議題，應經過知識工程師與該領域專家討論後將此議題加入分類架構中，如此便可讓議題的定義標準化。當議題標準化後，知識庫尚能利用同義字詞庫（如 WordNet）來改善關鍵字搜尋的同義字問題。

書本知識作者對議題的定義也應有標準可循，但因為同主題下的議題可能有很多，難以全部列舉，故可採用參考其他著作的方式。如前一章所述，每一本書都可視為該主題類別的物件，每個物件各包含其介紹的議題，故可將這些議題集合起來成為主題類別下的議題，由於介紹相同主題的書本其內部的議題一般而言差距不大，故書本知識作者可參考這些已在知識庫的議題，來訂定自己書本的議題架構。只要主議題定出來後，底下的子議題便可依書本的章節架構而定。這樣定義的好處是讓議題的架構有一共通的基礎，以利使用者快速掌握書本的議題架構。

(3) **解法（Approach）**：對技術論文或技術報告而言，解法關鍵字指的是解決問題所用的方法。例如解 Travel salesman problem，有的論文可能使用 Greedy 的方法，有的論文採用 Dynamic programming 的方法，像這些就是所謂的解法。因為軟體知識的解法能夠代表其原理，對該領域熟悉的使用者而言，看到論文的問題與解法便能對論文有一概括性的認識。論文或技術報告的作者必然知道自己提出的方法是屬於哪種方法，因此可以清楚明確地定義解法關鍵字。解法關鍵字代表了論文的核心，故具有高度彈性，作者在定義解法關鍵字前，可先至相關議題下觀看目前已有哪些解法，若自己的解法相同可直接參考，若不同則根據論文或技術報告內容直接定義新的解法關鍵字。

書本知識也有解法關鍵字，但這跟技術論文或技術報告有些不同，書本的解法是指對特定問題所介紹的方法，例如 sorting，就可能有 quick sort、bubble sort、Shell sort、insertion sort 等不同的方法，對書本知識作

者而言，可以很自然地判斷哪些章節是屬於議題－子議題的關係，哪些章節則屬於議題－解法的關係，清楚定義這些關鍵字有助於使用者理解與搜尋。

(4) **技術（Technique）**：技術關鍵字是指技術論文或技術報告所提出的解法中所運用的技術。「解法」指的是論文運用的基本方法，例如第三章舉的例子中，有一篇解決 text categorization 的論文是採 machine learning 的解法；而「技術」則是指其實際運用的技術，例如這篇論文運用了 train-and-test 以及 k-fold cross-validation 的技術。知識作者在定義技術關鍵字時，也應參考目前分類架構中已存在的技術關鍵字，一般而言，技術關鍵字應是清楚定義（well-defined），且較成熟且廣為人知的。論文當然也可以有新的技術，不過由於定義新的技術關鍵字將影響後面作者，故應經過知識工程師與該領域專家討論後再加入分類架構中。

籍由上述法則，技術論文、技術報告與書本知識的作者都能夠清楚且明確地定義其知識的關鍵字。當關鍵字依上述分類後，有些關係便會自動產生，例如類別間與類別－議題間的包含（Aggregation）、議題－解法間的實現（Realization）、以及論文與技術間的運用（Use）等。但仍有一些關聯性必須由知識作者定義：

(1) **繼承（Inheritance）**：繼承是指新的解法（Approach）是從舊的解法改進而來的，這個關係是指方法的改進，而非論文的改進。例如針對排序的問題，有一篇論文提出 bidirectional bubble sort 的方法，這是改進自 bubble sort 的，由於此方法是新的解法，因此可以定義繼承關係。這個關係並非所有的論文作者都必須定義，只有當論文作者提出了新的解法，而此解法又是從舊有的解法改進而來時，才需要定義此解法的繼承關係。這是因為解法可視為物件導向中的類別，而論文或報告可視為物件，因此繼承關係對採用相同解法的論文都是一樣的。

(2) **依存（Dependency）**：依存指的書本知識中議題的先後閱讀關係，例如閱讀 Object-oriented Design 之前應該先閱讀 Object-oriented Concept；這部份的關係建立之後有利讀者找到一個適當的閱讀切入點，且可跳過沒有依存關係的章節。當書本作者定義了書本的議題、子議題、解法等架構之後，便可定義議題之間的依存關係，此關係對作者而言非常容易，因為書本知識主要便是在介紹與說明，故其章節編排的先後原本就有依存關係的考量，作者便依此定義依存關係即可。

(3) **改善（Refine）**：當論文採用相同的解法，並運用一樣的技術時，知識庫可自動依其發表時間建構改善關係。這個關係也可由技術論文或技術報告作者直接定義，有時論文雖然用了不同的技術，卻也可能是改善自其他論文的方法，故作者可以定義此關聯性。

由上述，可以整理技術論文或技術報告作者定義關鍵字步驟如圖 4 - 1：

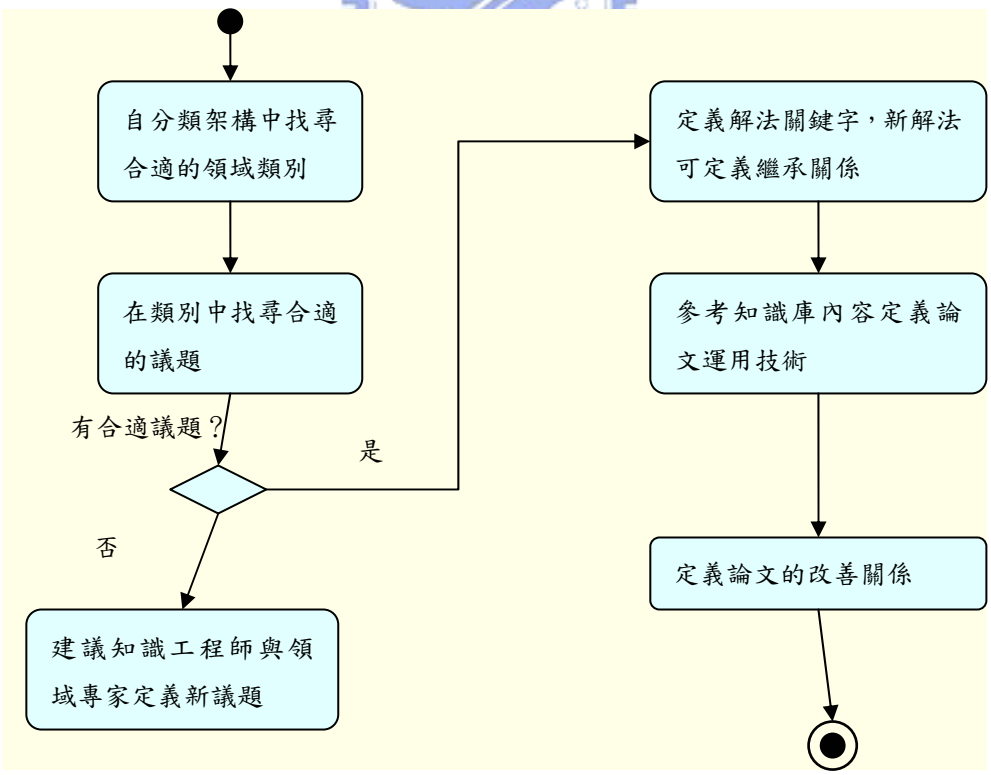

圖 4 - 1　技術知識關鍵字定義流程圖

書本知識作者定義關鍵字步驟則可整理如圖 4-2：

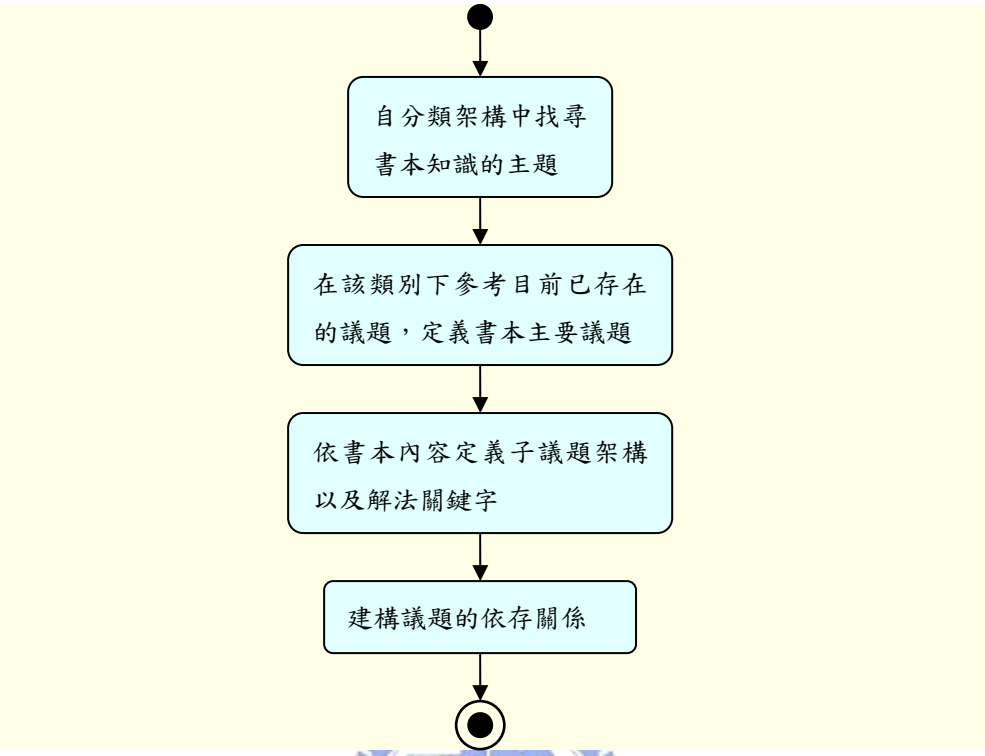

圖 4-2　書本知識關鍵字定義流程圖

依上述流程，知識作者可以輕鬆地以「問題－解法」知識分類法定義關鍵字，並建構關鍵字的關聯性，對技術論文、技術報告的作者而言，定義關鍵字是發表前必須的工作；對書籍作者而言，定義書後的索引亦是不可少的任務。採用以「問題－解法」為基礎的關鍵字法，由於是依據知識的本質所定義，因此對知識作者而言並不困難。對技術論文、技術報告的作者而言，他們一定非常清楚論文所在的領域、處理的議題、提出的方法以及運用的技術；對書本知識的作者，同樣也十分清楚書本知識探討的主題、議題、子議題或方法等，以往定義關鍵字還必須思考哪些字詞比較重要，而以這種方法定義，反而更明確且更容易。

4.2　領域類別架構設計

領域類別是「問題－解法」知識分類法中的基本架構，由於領域具有階層特

性，大領域又可細分為小領域，因此可採用樹狀的階層架構（Hierarchy structure），即 Taxonomy 的方法對領域進行分類。由於議題與解法都將建構在此領域類別之下，因此這個分類必須有足夠的廣度與深度，以及適當的彈性，方能讓使用者有效率地在其中找到合適的知識。

由於 Taxonomy 的分類非常方便且直觀，因此有非常多著名的 Taxonomy 領域分類系統。以圖書館的分類而言，最常用的即為杜威十進分類法（Dewey Decimal Classification）以及美國國會圖書分類法（Library of Congress Classification），而運用在期刊上最著名的則是 ACM 分類系統（ACM Computing Classification System），此外尚有網路上許多搜尋引擎使用的 ODP（Open Directory Project）等，這些分類方法都是對知識領域分類的階層架構，以下將對這些方法進行討論，並提出適合軟體知識的領域類別架構。

杜威十進分類法是由 Melvil Dewey 在 1876 年發明的，目前由 OCLC（Online Computer Library Center）負責維護。杜威十進分類法是利用十進位的數字來表示類別，首先將所有的知識領域劃分為十種類型（稱為 Main class），在每個 Main class 下又可劃分十種較小的子領域（稱為 Division），而在 Division 之下又再分十種更小的領域（稱為 Section），而在 Section 之下的小領域通稱為 Subdivision。由於是採用十進位的數字表示，因此每個類別至多只能分為十個子領域，但都可以無限地細分，至目前為止杜威十進分類系統已經需要四本書才能夠寫完。此分類法主要是以四層為基礎，其下可彈性地細分，其分類範例如圖 4 - 3 所示：

600 Technology (Applied sciences)
630 Agriculture and related technologies
636 Animal husbandry
636.7 Dogs
636.8 Cats

圖 4 - 3 杜威十進分類法範例

此分類法由於採用十進位編碼，故其分類數量受限，隨著各領域學門發展不同，加上某些新的領域興起，此時舊學門已將較高層的編碼用去，故新學門只能分到較低的階層，由於此方法是在 1876 年發明，故有許多分類已經顯得不合時宜。例如近年增長快速的電腦科技，是被分在自然科學→數學→電腦科技中，相當不合時宜，但因其編碼已固定，且之前的書籍也都依此編目歸類，故只能在此架構下繼續細分，十分欠缺彈性。

美國國會圖書分類法是在 1897 年由 J.C.M. Hanson 與 Charles Martel 帶領的各領域專家所研發的。此分類法將知識分為 21 個主要領域（稱為 Main class），在各 Main class 下則分為次領域（稱為 Subclass），用以表示主領域的學門或分支；Subclass 又可再細分為更小的領域（稱為 Division），Division 則可更細分成 Subdivision。此方法跟杜威十進分類法有點類似，都是用編碼的方式來表示分類，Main class 與 Subclass 以英文二十六個字母編碼，其下的 Division、Subdivision 則依其規則以十進位數字編碼，其分類範例如圖 4 - 4：

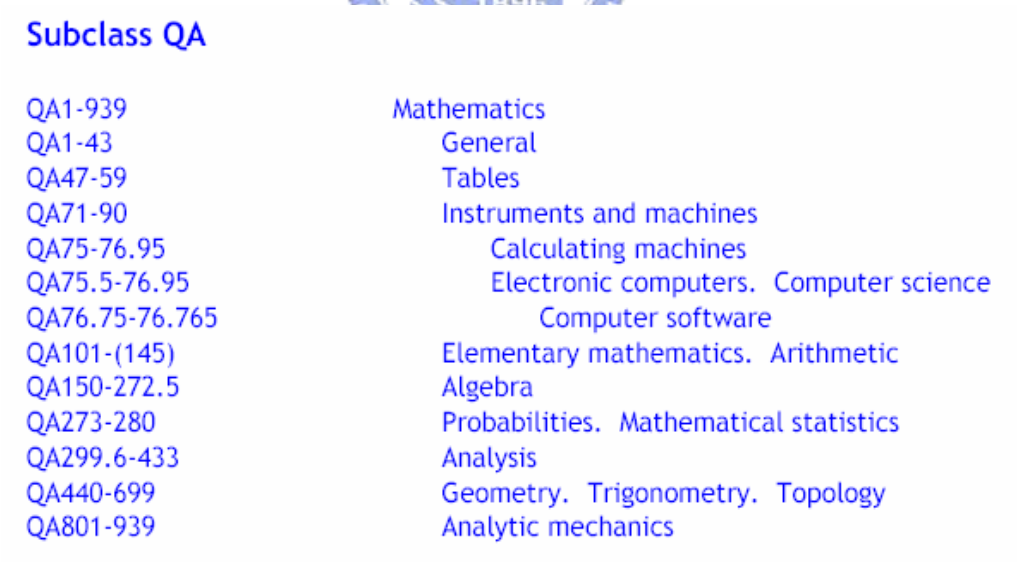

圖 4 - 4　美國國會圖書分類法範例

此方法跟杜威十進分類法類似，故其問題也是缺乏彈性，但因為之前的書籍已依此方法編目，要更動非常耗時耗力，因此只能繼續沿用。由於軟體知識可以

電子化，沒有書本編目的問題，因此可以設計更具彈性的分類架構，以符合領域學門發展的變化。

ODP（Open Directory Project）是由網景公司（Netscape Communication Cooperation）提出及管理的。此分類架構與傳統分類不同，採用了自主管理的方法，ODP 僅提供一些分類架構的準則，由一些志工對此分類的發展進行維護。由於此分類法的開放性與彈性，使其發展快速而穩定，非常多的搜尋引擎都是使用 ODP 做為其分類架構，如 AOL Search、Netscape Search、Google、Lycos、DirectHit、HotBot 等。最高層級的分類是事先設定且不能更動的，其下的分類則沒有限制其層級數量，僅提供一些簡單的分類準則，例如當一個分類下有超過二十個連結時，志工應考慮對其進行細分等。此分類法尚有一項特色，它提供了相關類別（Related Category）以及相等類別（Equal Category）來加強此分類系統的效率。所謂相關類別指的是將具有相關性的類別加以連結，如此在類別中搜尋時亦可以將其相關類別納入搜尋中，藉以增加搜尋的有效範圍，例如 Algorithm 中的 Graph 跟 Computation Geometry 就是一個相關類別的狀況。相等類別則是提供使用者由不同的瀏覽路徑，例如 Internet → WWW → Programming 與 Programming → WWW 其實是相同的，故在分類之中建立一些相等的關係，使得使用者不會因為一開始選擇的類別而找不到他所需的知識。ODP 的分類範例如圖 4 - 5：

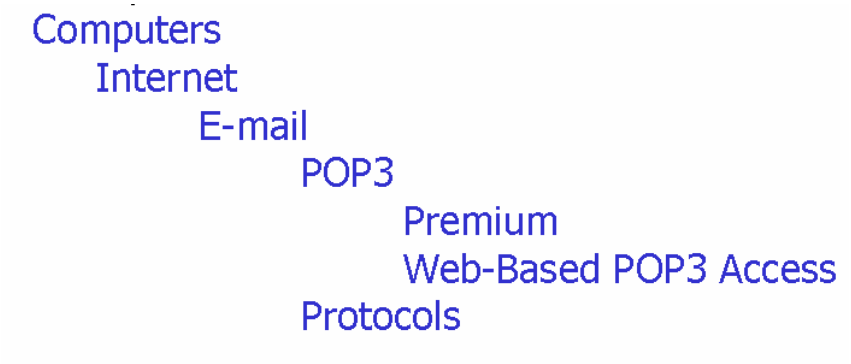

圖 4 - 5　ODP 分類範例

ODP 雖然沒有限制層級數量，但至目前為止其分類都以四層為主，這個分類的設計是以網頁資料為基礎，因此四層的分類已經足夠。但軟體知識的發展是非常深入的，四層分類將使得同一分類下的議題及知識太過龐大，且使用者也無法依此分類準確地找到適合的知識，例如近年來急速發展的 Internet，若要細分的話可能會出現下列的類別：Internet → WWW → Web Programming → Scripting Language → ASP → Database Connection，光是這些就需要六層的分類才真正能夠將知識恰當地分類。

ACM 分類系統（ACM Computing Classification）是最常運用在技術論文上的分類法，此分類法由 ACM（Association for Computing Machinery）提出，由一個專門組成的分類維護委員會（Classification Update Committee）進行修改與維護。此分類以一個四層的樹狀架構為主，第一層以英文字母編碼，第二、三層則以數字編碼，第四層由於經常更動，故不予編碼。此分類法在 1982 年提出，在 1982、1983、1987、1991、1998 都有修改。由於各領域學門發展不一，在 1998 年此委員會原本要重新修改整個分類架構，但由於此分類法已行之有年，大幅修改將造成嚴重影響，因此仍只是在第三、第四層進行修改。其分類範例如圖 4 - 6：

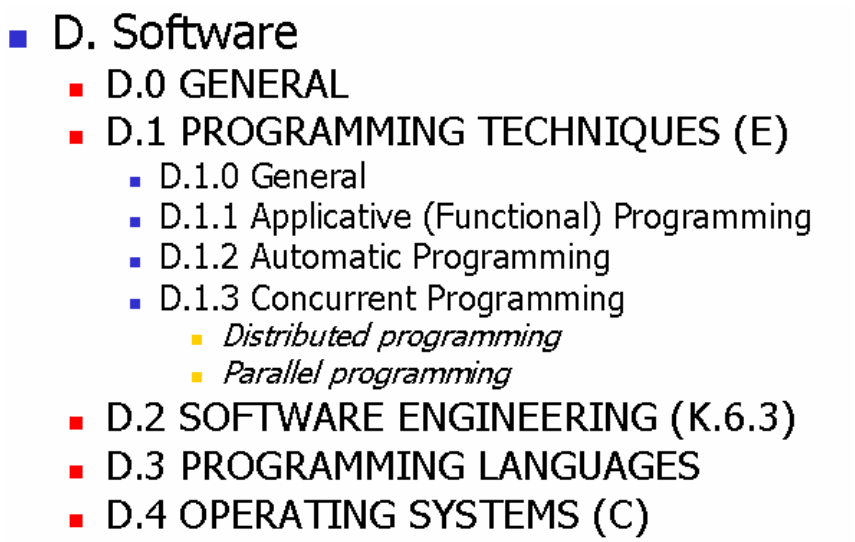

圖 4 - 6　ACM 分類系統範例

由上述可知，目前主要的分類方法中，圖書館的分類法因為歷史悠久，學科的分類跟現今的知識已有一定差距，至於 ACM 及 ODP 的分類法都大約只有四層的架構，由於軟體知識各項學門的探討都愈來愈深，四層的分類已不足以使用。需要建立一個更具深度及彈性的分類架構，以符合軟體知識分類的需求。

除了上述分類法外，生物種類的分類也是採用樹狀結構，且從十七世紀 Carl Linnaeus 發明仍沿用至今，成為生物分類的標準。此分類系統是採七層架構，從上往下依序是界（Kingdom）、門（Phylum）、綱（Class）、目（Order）、科（Family）、屬（Genes）、種（Species），雖然生物種類繁多，但這樣的層級概念已足以對所有的生物分類。

由上述可知，常用的分類方法都不超過七層架構，心理學家 George Miller 於 1956 年提出人的暫時記憶數量是七加減二，層級架構太少固然會造成分類不夠詳細，但若層級太多則會造成使用者難以從中找尋合適的分類。因此，我們提出以七層架構為基礎的分類系統，搭配分類的相等及關聯性，並使此架構具重整彈性，以符合軟體知識之需求。

杜威十進分類法的四層架構由上而下依序是 Main class、Division、Section、Subdivision，而美國國會圖書分類法的四層架構由上而下則是 Main class、Subclass、Division、Subdivision，參考這兩個著名的分類系統，定義我們的七層分類架構如圖 4 - 7：

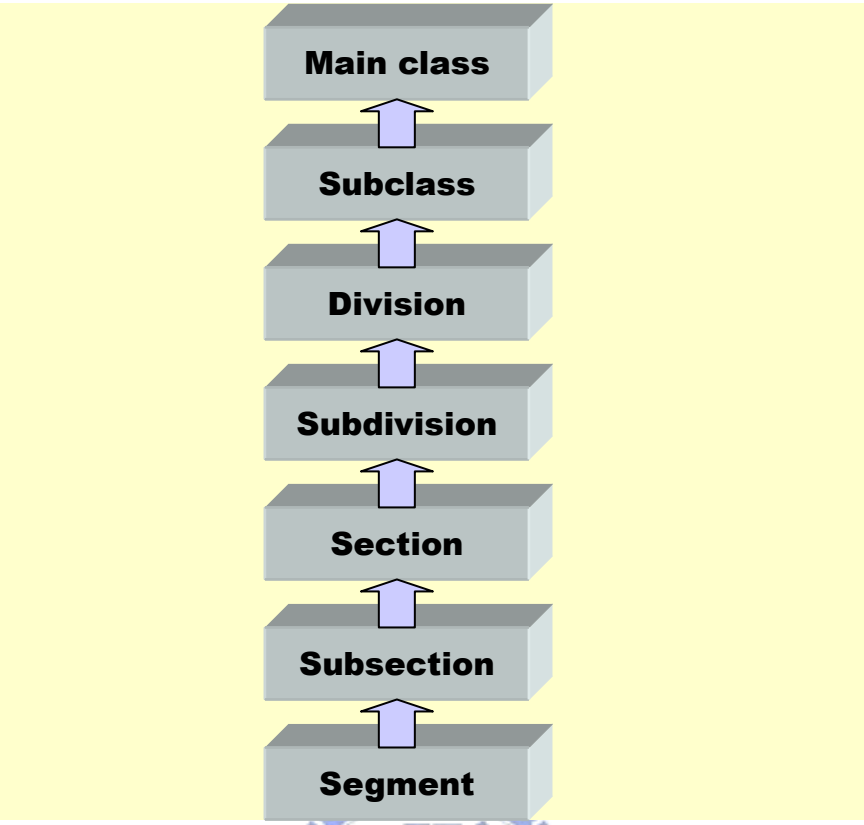

圖 4 - 7　七層分類架構示意圖

此七層架構提高了目前常用分類法的深度，讓軟體知識能夠依其討論內容恰當地分類，而七層架構又不致於太過繁雜造成使用者困擾，例如近年來發展快速的 Internet 就可以依此架構分類如圖 4 - 8：

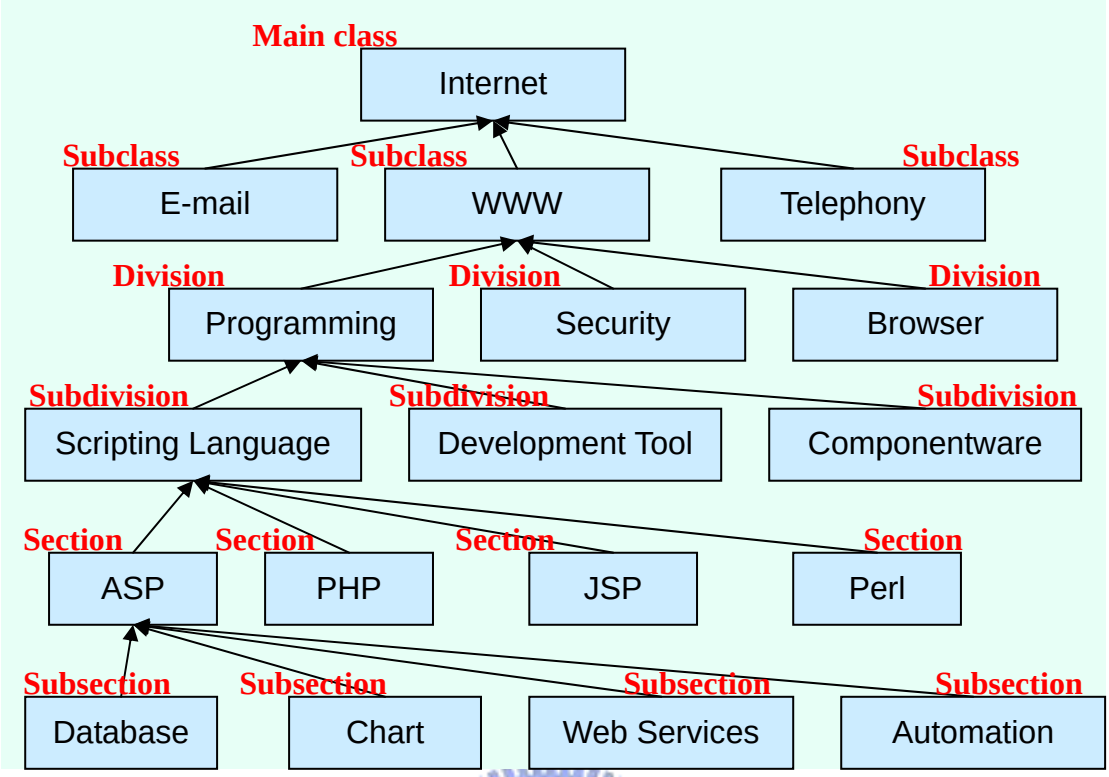

圖 4-8 七層架構分類範例

此架構必須容許重整及重新排列層級。如此當新科技出現時可先分至較低層級，隨著此科技相關知識深度及廣度增加，可將此類別繼續細分，當其下類別達到一定程度後，再將之調整至較高的層級。採用這種調整方式，無論有再多的新領域，七層架構都將足夠使用。

我們提出的七層架構分類系統，提高了傳統分類方法的深度，更適合各項學門探討愈來愈深的軟體知識，不使用圖書館分類法或 ACM 分類系統的編碼方式，使得將來類別的調整更有彈性，新的科技先分至低層級，待其相關知識討論深度增加時再逐步移至高層，讓此分類系統能不斷適應領域發展的變化。另外此分類系統將運用 ODP 的「相關類別」及「相等類別」概念，提昇使用者在其中找尋知識類別的效率。故此分類系統擷取各常用分類系統的優點，並改善其缺點，使之能對軟體知識進行更佳的分類。

4.3 知識排序法則與搜尋方法

知識排序法則與搜尋方法是知識分類法非常重要的一環。排序法則的好壞將影響知識搜尋結果的排序，當知識量愈來愈多時，排序法則將對知識庫的效率產生決定性的影響，好的排序法則能夠將使用者想要的知識排在前面，即使知識量非常大，亦不會造成使用者時間的浪費；而好的知識搜尋方法能讓使用者依其需求找尋知識。以下將先介紹知識排序法則，再進一步說明知識搜尋方法。

4.3.1 知識排序法則

根據第三章的分析，找尋書本知識是想找領域的介紹或深入了解特定議題。對領域介紹而言，書中若談到較多的子議題，往往代表其介紹的較為豐富且完整。舉例而言，介紹演算法的書很多，若有一本介紹了十種主要的演算法，另一本介紹了十五種，則介紹十五種的書對於要了解演算法的讀者而言會較有助益，因為他會能從中了解較多的演算法。對特定議題而言，同樣是介紹愈多子議題者愈有助益，例如第三章提到介紹 C++的書，對於 Inheritance 這個議題，若有的書介紹了 Constructor、Polymorphism & Virtual Function、Destructor、Multiple Inheritance，而有的書只介紹了前三個議題，則閱讀前者的讀者能多對 Multiple Inheritance 有所了解，因此幫助較大。

除此之外，書本知識的頁數也具有一定的代表性，在一般情況下，相同的議題，使用較多頁數的書本往往介紹得較為清楚且完整。這些特性將能夠運用在書本知識的比較上。

對於技術論文或技術報告而言，因為要找尋的是問題的解法，故比較的應該是何者是此問題的最佳解法。但解法的比較依領域不同，其準則亦不同，例如 Hash table 可能是比較其 Collision rate，而 Binary search tree 可能是比較其 Internal path，但我們發現論文之間的解法可能是有繼承的關係，即新的解法是從舊解法

改良而來，一般而言，新的解法對該議題而言應有較好的效益，否則不會被大家接受。此外，若兩篇論文採用相同的解法，並運用同樣的技術，表示這兩篇論文的作法類似，則新的論文應是改善舊論文作法而產生的，且其效益應較佳，假若採用相同解法與技術但效益卻沒有比較好的話，不會被期刊審訂的領域專家所接受。如第三章所述，我們利用上述關係建構了繼承（Inheritance）與改善（Refine）的關係，故可用以做為知識排序的比較法則。

此外，若一篇論文被許多其他領域的論文所引用（Citation），表示其解法相當具有學術價值，故除了上述採用繼承、改善對解法做一基本比較外，尚可利用論文被其他領域論文引用的次數來做排序。

由上述，對書本知識的排序準則依序是：子議題數、頁數；對技術報告與技術論文的排序準則依序是：解法繼承、論文改善、被引用數。籍由這樣的準則，在一般情況下能夠讓使用者找到最可能符合需求的知識。

4.3.2 知識搜尋方法

如 4.2 節所述，我們的分類將以七層架構為基礎，配合相關類別與相等類別提昇知識搜尋效率。由於軟體領域既廣且深，因此使用這樣的分類較容易幫助使用者找到符合的知識類別。如之前分析的，使用者找尋知識的需求可分為四種類型。以下對這些需求說明其搜尋方法：

1. **找領域的介紹性知識**：此狀況是使用者對特定領域不熟，想對此領域有初步了解。由於書本知識產生的目的即是以介紹與教學為主，故針對這類使用需求，應以找尋最具介紹性的書本知識為目標。使用者應利用我們定義的七層分類架構，從上層往下找尋有興趣的領域，由於他想找特定領域類別，故利用分類架構瀏覽是最容易的。找到類別之後可先觀看該領域的基本介紹以及子議題，並可進一步依上述的排序法則找出合適的書籍閱讀。

2. **深入了解特定議題**：此狀況是使用者已對領域有所了解，而想找尋特定議題的介紹。例如使用者知道何謂 graph，想了解 TSP（travel salesman problem）為何。這類需求仍以找尋介紹為主，故書本知識還是較為合適，由於使用者已對領域有所了解，故可以很容易地藉由類別瀏覽先找到領域，知識庫可該領域下的子議題列出供使用者挑選。找到議題後，使用者可先看議題基本介紹及相關的解法，也可觀看有哪些相關書本知識，籍以了解此議題。

3. **找出問題最佳解決方案**：這是使用者已對該領域與議題熟悉，想找出該議題的最佳解法。這類需求最適合給予討論該議題的技術論文或技術報告，因為這兩類知識的本質便是提出問題的解法。由於使用者對領域與議題熟悉，因此可以很快地利用分類架構由上往下找到議題，此時使用者可先觀看此議題下提出了哪些不同的解法，此外藉由前一節所介紹的排序方法，能讓使用者閱讀較佳的論文。使用者閱讀論文前可先觀看這些論文分別運用了哪些技術，以便快速掌握論文的要點。

4. **了解技術的運用情況**：使用者想了解技術運用狀況，可分為兩種類型，其一是想知道哪些論文或報告運用了這項技術，其二則是想了解這項技術被運用在哪些議題或領域上。知識庫可讓使用者直接輸入該技術關鍵字，並由使用者選擇要找論文或是領域，由於「問題－解法」分類法建構了論文運用技術的關係，故可由技術找到相關論文，並可再往上找到其議題與領域類別。

除了採用上述方法外，若使用者對此分類架構熟悉後，也可以直接使用符號表示的方法快速找尋知識，例如直接輸入：

Programming ≻ Data structure ≻ Tree ≻ Binary search tree ≻ Improve worst case

可找到 Binary search tree 的改善最差狀況議題。

4.4 關鍵字符號表示法

「問題－解法」知識分類法定義了關鍵字及其關聯性，依 3.3 節所述，這些關聯性能夠以符號表示。以符號表示關鍵字，將能使知識的描述簡化及標準化。這個描述可定義明確文法，讓電腦系統容易解析，故知識的處理、分類、索引、保存全部都能夠自動化，大幅節省知識管理中知識處理（Knowledge Processing）的時間。

在 3.3 節中，定義了八種關聯性，這些關聯性的符號定義如下：

(9) 包含（Aggregation）：≻

(10) 實現（Realization）：←

(11) 使用（Use）：⊕

(12) 繼承（Inheritance）：▷

(13) 相等（Equal）：≡

(14) 關聯（Association）：⇄

(15) 依存（Dependency）：►

(16) 改善（Refine）：⇒

依這些關係對知識的描述，可定義其 BNF（Backus Naur Form）如下：

```
<knowledge> ::= <stmtlist>;

<stmtlist> ::= <stmt>;<stmtlist> |
               <stmt>

<stmt> ::= (<category>) ≻
             <issue>←<approach>⊕ <technique> |
           <category> ≻ (<bookissue>) |
           <caterelation> |
           <issuedeplst> |
           ⇒<paper>
```

<category> ::= (<categorylst>) |
<identifier> ≻ <category> |
<identifier>

<categorylst> ::= <identifier> {, <categorylst>}

<issue> ::= <identifier>

<approach> ::= (<identifier> ▷ (<apprchlst>)) |
<identifier>

<apprchlst> ::= <identifier> {, <apprchlst>}

<technique> ::= (<techlst>) |
<identifier>

<techlst> ::= <identifier> {, <techlst>}

<bookissue> ::= <identifier> ≻ <bookissue> |
<bkissueitem> |
(<bkissuelst>) |

<bkissueitem> ::= <identifier> ← (<bkissuelst>) |
<identifier>

<bkissuelst> ::= <bookissue> |
<bookissue> , <bkissuelst>

<identifier> ::= <letter> {<letter> | <digit>}

<caterelation> ::= <identifier> ≡ (<categorylst>) |
<identifier> ⇄ (<categorylst>)

<issuedeplst> ::= <issue> ▸ (<issuelst>)

<issuelst> ::= <identifier> {, <issuelst>}

<paper> ::= <author>. <name>.
<journal>, <page> - <page>

<author> ::= <identifier> { and <author>}

<name> ::= <identifier>

<journal> ::= <identifier>

<page> ::= <digit> {<digit>}

採用上述的文法，可以將關鍵字關聯性以符號表示。此文法設計有陳述句（statement）的概念，因為知識的基本架構如書本知識的議題或技術論文的類別、議題、解法、技術必須寫成一個陳述句，而其他的關係如「改善」、「依存」及類別的「相關」、「相等」就必須以其他陳述句描述。

第五章、軟體知識庫系統之設計

本章將以「問題－解法」分類法實作一雛型（Prototype）知識庫，以證分類法的可行性。在 5.1 節首先根據「問題－解法」知識分類法整理出系統功能需求。接著 5.2 節根據這些需求，提出系統的設計架構與模組設計。5.3 節則將說明系統資料儲存設計，最後在 5.4 節介紹實作的使用者介面。

5.1 系統功能需求

此雛型（Prototype）知識庫主要提供使用者進行知識的搜尋、檢索以及知識工程師對知識內容及分類架構的維護。依「問題－解法」分類架構，關鍵字可分為「類別」、「議題」、「解法」、「技術」四種類型，對每一種關鍵字都會有不同的操作。依一般使用者與知識工程師的不同，可整理其需求如下：

(1) 一般使用者需求：一般使用者主要利用分類架構由上往下找尋知識，其中包括找領域類別介紹、議題介紹、議題解法等目的，使用者也可以找尋特定技術被運用在哪些論文或領域。此外可利用符號表示法新增知識。其需求可整理如表 5 - 1：

一般使用者需求

編號	需求	說明
R1	新增知識	利用符號表示法新增知識
R2	類別瀏覽	從分類架構中瀏覽知識類別，找到類別後可閱讀其子議題、書本知識、最新狀態等
R3	讀取議題	特定議題介紹、書本知識、最新狀態等
R4	讀取解法	讀取解法介紹及相關論文等
R5	閱讀技術論文或技術報告	閱讀其摘要、引用、運用技術等
R6	閱讀書本知識	讀取書本知識議題架構及依存關係
R7	技術運用狀況	了解技術運用論文、解法、議題、領域
R8	最新知識庫文件	取得知識庫最新狀態

表 5 - 1　一般使用者需求表

(2) 知識工程師需求：知識工程師主要工作是維護知識分類架構，其需求可整理如表 5 - 2：

知識工程師需求		
編號	**需求**	**說明**
R9	調整知識類別	知識類別的搬移、分割、整合，以及設定知識類別的關聯性
R10	管理議題	議題的新增、搬移、修改等
R11	管理解法	解法的說明以及繼承關係等
R12	管理技術	技術的新增及其相關或相等關鍵字

表 5 - 2　知識工程師需求表

除了上述功能需求外，知識庫的效能也是設計重點，因為知識庫必須儲存大量知識，故回應時間快慢及儲存空間大小必須納入設計考量。

5.2　系統架構與模組設計

此知識庫範例將提供使用者於其上瀏覽、檢索、新增知識。這些功能都是由使用者主動使用，且知識庫有集中資料的特性，因此可使用 WWW（World Wide Web）架設網站伺服器，讓不同平台的使用者能夠透過瀏覽器從遠端執行伺服器端網頁程式，我們將使用 IIS（Internet Information Server）搭配 ASP（Active Server Page）做為網站架設及系統開發平台，後端採用 SQL Server 資料庫存放資料。

由於直接存取資料庫較費時，我們會先將重要的資料及常用的查詢結果在系統初始化時從資料庫讀出，並暫存至系統記憶體中，這部份採用 COM（Common Object Model）物件來完成，另外較不常存取的資料則先自資料庫取出存至檔案系統中，以加速運作效率。這個方法在大型知識庫中將會消耗過多系統資源，由於本實作範例僅為證實分類方法的可行性，知識量不會很大，故可用此方法加速存取效率。

因此，此知識庫的系統架構如圖 5 - 1 所示：

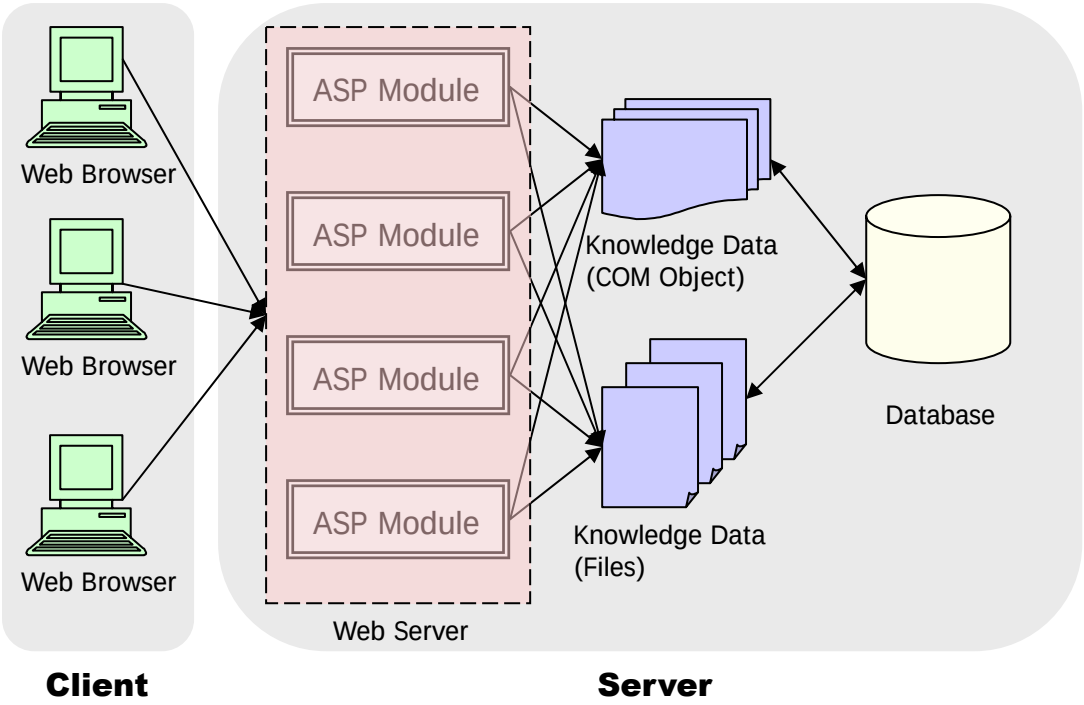

圖 5 - 1　知識庫系統架構圖

依前一節的功能需求，我們將系統分為幾個主要模組，利用層級式（Layer）的設計方法，其模組架構圖如圖 5 - 2：

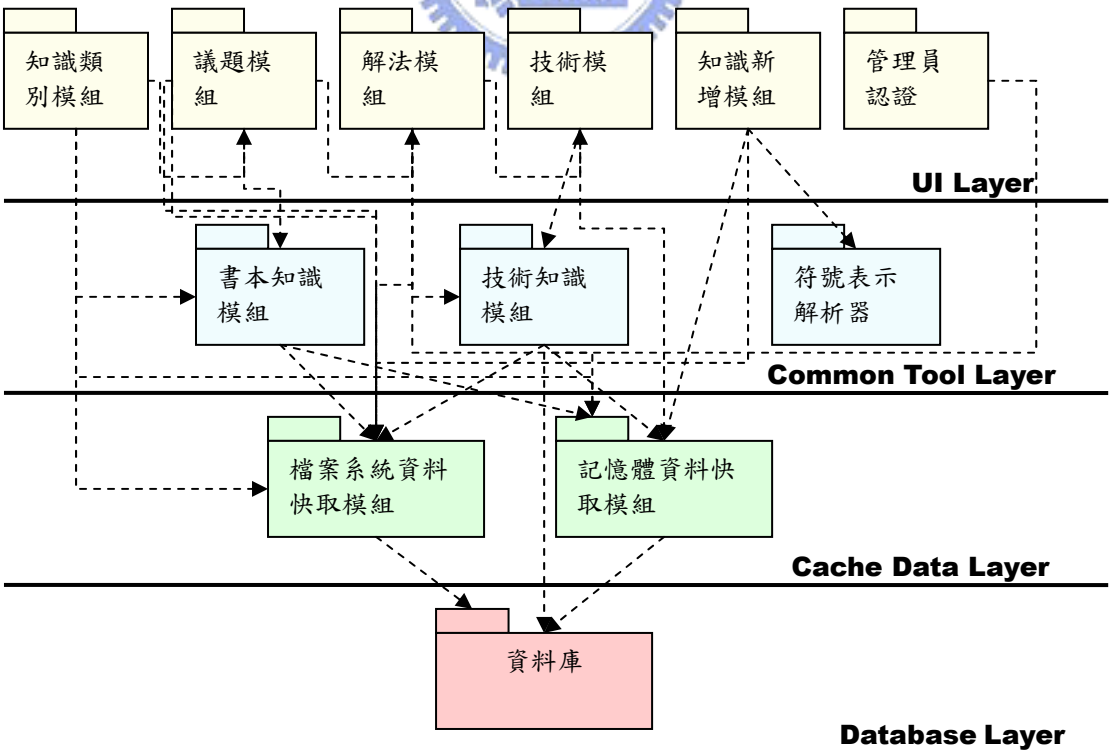

圖 5 - 2　系統模組架構圖

各模組功能介紹如下：

- **知識類別模組**：

  知識類別模組負責樹狀分類架構的瀏覽與維護，其功能包括：

  (1) 類別瀏覽：使用者點選類別瀏覽後，會產生一樹狀結構的瀏覽介面，使用者可以自由選擇想展開的類別，或直接點選類別以讀取類別介紹。

  (2) 類別簡介：顯示特定類別的簡介。

  (3) 子類別 / 子議題：若是樹狀結構中的非葉節點（non-leaf node），列出其子類別，葉節點則列出其子議題。

  (4) 類別最新狀態：列出該類別下的最新知識。

  (5) 最熱門子類別 / 子議題：依子議題數或使用數排序熱門子類別或子議題。

  (6) 相關類別：列出與特定類別相關的類別

  (7) 書本知識列表：依排序法則列出類別下書本知識。

  (8) 書本知識議題列表：列出該類別下所有書本知識子議題聯集。

  (9) 新增子類別 / 子議題：知識工程師可在特定類別下新增子類別或子議題。

  (10) 搬移類別：知識工程師將特定類別搬移至其他類別下。

  (11) 修改 / 合併 / 分割類別：知識工程師對類別進行修改、合併、分割等動作

- **議題模組**：

  (1) 議題簡介：顯示特定議題的簡介。

  (2) 父類別：顯示特定議題的父類別串，即其父類別一路至根類別。

  (3) 書本知識列表：依排序法則列出介紹此議題的書本知識。

  (4) 解法列表：列出特定議題下的各種解法。

  (5) 議題最新狀態：列出特定議題下最新的技術論文或技術報告。

  (6) 搬移議題：知識工程師將特定議題搬移至其他類別下，當類別架構修改時才會發生。

- **解法模組：**

  (1) 解法簡介：顯示特定解法的基本介紹。

  (2) 論文列表：依排序法則顯示採用該解法的論文，以及各論文使用的技術。

  (3) 父議題：顯示特定解法的父議題及其父類別串。

  (4) 修改解法繼承關係：知識工程師可對解法繼承關係進行修改動作。

- **技術模組：**

  (1) 相等 / 相關關鍵字簡介：顯示特定技術相等或相關的知識類別、議題或解法，並連結至該關鍵字下的簡介。

  (2) 運用論文：顯示運用特定技術的論文。

  (3) 運用議題：顯示運用特定技術的議題及其父類別。

  (4) 運用知識類別：顯示運用特定技術的知識類別。

  (5) 新增技術：知識工程師可新增一特定技術。

  (6) 修改技術關聯性：知識工程師可修改特定技術的相關 / 相等關鍵字。

- **知識新增模組：**

  此模組可利用符號表示法描述知識後新增至知識庫。

- **管理員認證：**

  此模組負責知識管理員的身份認證工作。

- **書本知識模組：**

  (1) 書本議題架構：以樹狀架構方式列出書本議題架構，以及依存關係。

  (2) 相關書本列表：列出介紹相同主題的書本。

  (3) 讀取書本知識：若書本知識已電子化，直接給予使用者書本知識，否則告知使用者書本知識所在位置。

- **技術知識模組：**

  此模組提供使用者技術論文或技術報告及其相關知識，功能如下：

(1) 技術知識摘要：顯示技術論文或技術報告摘要。

(2) 技術知識相關資訊：顯示技術知識作者、期刊、出版年月等相關資訊。

(3) 引用列表：顯示技術知識所引用的知識。

(4) 被引用狀況：顯示技術知識被哪些知識所引用。

(5) 運用技術：顯示技術知識所運用的技術。

(6) 改善 / 被改善關係:顯示技術知識改善了哪些知識,又被哪些知識改善。

- **符號表示解析器：**

  此模組為利用 lex & yacc 建構符號表示法的解析器（parser）。可將輸入的符號表示字串解析為各種類型的關鍵字。

- **檔案系統資料快取模組：**

  此模組在系統初始化時自資料庫取出一些較大且較不常用的資料,放至檔案系統中加速存取速度。當資料有修改時亦先修改這些資料，待一段時間後再一起寫回資料庫。

- **記憶體資料快取模組：**

  此模組在系統初始化時自資料庫取出一些常用資料,放至系統記憶體中加速存取速度。資料有修改時亦先修改此資料，待一段時間再一起寫回資料庫。

5.3 系統資料儲存設計

由於知識庫必須管理大量資料，因此其資料儲存設計將大幅影響系統效能。為了加速存取速度，如 5.2 節所述，我們採用了快取資料的方式，但此方法是存放在系統記憶體中，不具有可靠性（reliability），故仍需設計具備效率的資料庫架構做為穩定的資料儲存系統。

本節首先將在 5.3.1 節中介紹資料庫的設計，包括資料表及相關的觸發程序

（trigger）、預存程序（stored procedure）等。在 5.3.2 節則介紹快取資料的資料結構設計。

5.3.1 資料庫設計

由 5.2 節的各模組功能設計，儲存在資料庫中的資料仍以類別、議題、解法、技術等分類，此外還包括技術論文、技術知識、書本知識的資料，由於關鍵字間以及知識間有關聯性，因此必須設計對應的資料表儲存其關聯性，整體知識庫表格設計如圖 5 - 3 所示，依各資料表分述如下：

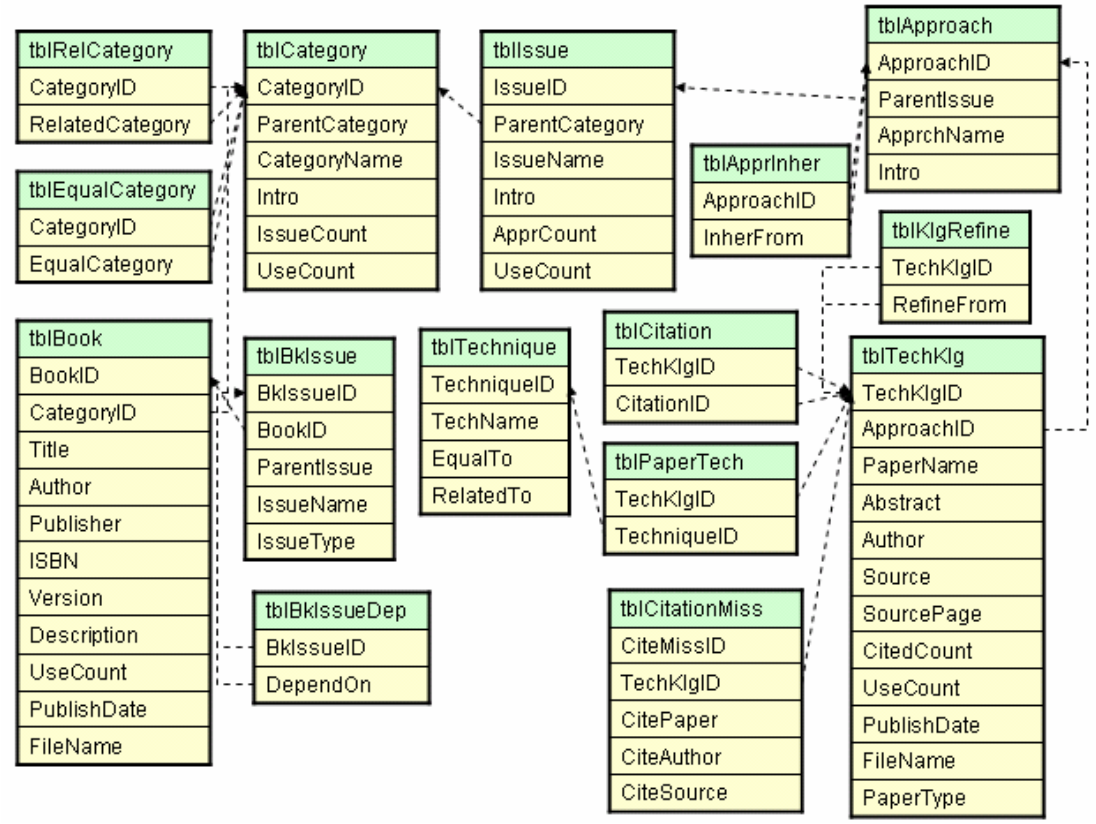

圖 5 - 3　資料表架構圖

- **tblCategory：知識類別資料表**
  - CategoryID：自動遞增的數字代碼，Primary Key
  - ParentCategory：父類別代碼，0 表示最上層目錄。此欄建立索引。
  - CategoryName：知識類別名稱

- Intro：知識類別簡介，為一指到長文字的指標。
- IssueCount：類別下的議題總數，此欄建立索引。
- UseCount：閱讀此類別介紹的次數，此欄建立索引。

知識類別為樹狀結構，因此每個類別必須儲存其父類別代碼，如此便可由程式建立樹狀結構關係，由於建構是由上往下，故此欄會做為篩選條件，故建立索引。子議題數與使用次數是用來對類別的熱門度排序，由於要排序，故必須建立索引以加快速度。

- **tblRelCategory：相關類別資料表**
  - CategoryID：知識類別代碼，此欄建立索引。
  - RelatedCategory：相關知識類別代碼。

相關性是雙向的，故此資料表會撰寫觸發程序（trigger）來維護雙向關係，當新增一筆資料時，此觸發程序會自動將 CategoryID 與 RelCategoryID 對調再寫入一筆，刪除時同樣也會一起刪除。

- **tblEqualCategory：相等類別資料表**
  - CategoryID：知識類別代碼，此欄建立索引。
  - EqualCategory：相等知識類別代碼。

相等類別也是雙向關係，故如 tblRelCategory 資料表一樣由觸發程序維護其雙向性。

- **tblIssue：議題資料表**
  - IssueID：自動遞增的數字代碼，Primary Key。
  - ParentCategory：父類別代碼，此欄建立索引。
  - IssueName：議題名稱。
  - Intro：議題簡介，為一指到長文字的指標。
  - ApprCount：議題下解法總數，此欄建立索引。
  - UseCount：讀取議題介紹的次數，此欄建立索引。

議題建構在知識類別下，所以要儲存父類別代碼，而父類別在找尋子議題時會以此欄為條件，故建立索引；議題下解法數用以對議題熱門度排序。

- **tblApproach：解法資料表**
  - ApproachID：自動遞增的數字代碼，Primary Key。
  - ParentIssue：此解法實作的議題代碼。
  - ApprchName：解法名稱。
  - Intro：解法簡介，為一指到長文字的指標。
- **tblApprInher：解法繼承關係表**
  - ApproachID：解法代碼，此欄建立索引。
  - InherFrom：此解法所繼承的解法代碼，此欄建立索引。

解法之間的繼承關係由此資料表儲存，由於搜尋時可能會找繼承自哪些解法，也可能找哪些解法繼承了目前的解法，故兩欄位都要建立索引。

- **tblTechnique：技術資料表**
  - TechniqueID：自動遞增的數字代碼，Primary Key。
  - TechName：技術名稱。
  - EqualTo：與此技術對等的關鍵字。
  - RelatedTo：與此技術相關的關鍵字。

技術資料的介紹是來自於與其相關或相等的關鍵字，可能是知識類別、議題或解法，故其表示法採用簡單編碼，以一個英文字母表示型態，接著便是其數字代碼。知識類別為 C、議題為 I、解法為 A，例如特定技術與知識類別代碼 15 相等，則表示為 C15。

- **tblTechKlg：技術知識資料表**
  - TechKlgID：自動遞增的數字代碼，Primary Key。
  - ApproachID：此知識採用的解法代碼，此欄建立索引。
  - PaperName：論文或技術報告名稱。
  - Abstract：摘要，為一指到長文字的指標。
  - Author：知識作者。
  - Source：知識來源，可能是期刊或研討會名稱。
  - SourcePage：知識來源頁數範圍。
  - CitedCount：此知識被引用數，此欄建立索引。

- UseCount：此知識被閱讀數，此欄建立索引。
- PublishDate：知識發表日期，此欄建立索引。
- FileName：知識全文檔案名稱。
- PaperType：知識是技術論文或技術報告，技術論文為 1，技術報告為 2。

技術知識包括技術論文與技術報告，由於此知識會運用特定解法解決特定議題，故需儲存解法代碼。被引用數是論文排序準則之一，此欄的值由觸發程序維護。發表日期亦可做為排序準則，這些欄位都建立索引以加速排序。技術知識的全文檔案通常用 PDF（Portable Document Format），故存放在特定目錄下，資料庫則儲存檔案名稱以便存取。

- **tblCitation：引用資料表**
  - TechKlgID：技術知識代碼，此欄建立索引。
  - CitationID：引用知識代碼，此欄建立索引。

技術知識的引用關係可做為排序準則，並可找出引用與被引用的知識。由於兩種情況都有可能，故兩欄皆建立索引以加速存取速度。

- **tblCitationMiss：引用知識暫存資料表**
  - CiteMissID：自動遞增的數字代碼，Primary Key。
  - TechKlgID：技術知識代碼，此欄建立索引。
  - CitePaper：引用知識名稱，此欄建立索引。
  - CiteAuthor：引用知識作者，此欄建立索引。
  - CiteSource：引用知識來源名稱，此欄建立索引。

由於引用知識可能不存在於知識庫，此情況便先將引用知識資料存放於此資料表，每次新增知識時便來此比對，若比對符合便將此資料改至 tblCitation，由於要比對知識名稱、作者與來源，故這些欄位都要建立索引。

- **tblKlgRefine：改善關係資料表**
  - TechKlgID：技術知識代碼，此欄建立索引。
  - RefineFrom：改善來源知識代碼，此欄建立索引。

改善關係是技術知識排序的最重要法則，採用相同解法及運用同樣技術的論

文，可利用其發表日期先後決定改善關係。由於搜尋時可能會找改善的論文或改善來源，故兩欄皆建立索引。

- **tblPaperTech：運用技術資料表**
  - TechKlgID：技術知識代碼，此欄建立索引。
  - TechniqueID：技術代碼，此欄建立索引。

  此資料表紀錄技術知識所運用的技術，由於搜尋時也可能找特定技術運用在哪些論文乃至議題、知識類別上，故兩欄皆建立索引。

- **tblBook：書本知識資料表**
  - BookID：自動遞增的數字代碼，Primary Key。
  - CategoryID：書本討論知識類別代碼，此欄建立索引。
  - Title：書本名稱。
  - Author：書本作者。
  - Publisher：出版社名稱。
  - ISBN：國際標準圖書編號（International Standard Book Number）。
  - Version：版本號。
  - Description：書本描述，為一指到長文字的指標。
  - UseCount：閱讀此書的使用數，此欄建立索引。
  - PublishDate：出版日期。
  - FileName：書本全文檔案名稱。
- **tblBkIssue：書本子議題資料表**
  - BkIssueID：自動遞增的數字代碼，Primary Key。
  - BookID：議題所屬書本代碼，此欄建立索引。
  - ParentIssue：父議題代碼，0 表示最上層議題，此欄建立索引。
  - IssueName：議題名稱。
  - IssueType：議題類型，1 為子議題，2 為解法。

  書本知識子議題是樹狀結構，議題下可能是子議題或解法，故同樣採用各議題儲存其父議題的方式讓程式能建構樹狀結構。由於建構是由上而下，故父議題代碼會做為搜尋條件，建立索引以加速存取。

- **tblBkIssueDep：書本子議題依存關係資料表**
  - BkIssueID：書本子議題代碼，此欄建立索引。
  - DependOn：依存子議題代碼，此欄建立索引。

  書本子議題之間的依存關係由此資料表儲存。由於搜尋時可能會找依存哪些子議題，也可能找哪些子議題依存於特定議題，故兩欄皆建立索引。

由於類別與議題都有「最新狀態」的功能，因此設計一特殊資料表來儲存各類別、議題的最新狀態：

- **tblLatestStatus：最新狀態資料表**
  - ParentType：關鍵字類別，此欄建立索引。
  - ParentID：關鍵字代碼，此欄建立索引。
  - SeqNum：最新狀態序號，愈小表示愈新，此欄建立索引。
  - LatestType：知識類別。
  - LatestID：知識代碼。

  ParentType 是一個英文代碼來表示關鍵字類別，可能是 C（類別）或 I（議題），SeqNum 表示序號，每個類別至多存五筆最新知識，故 SeqNum 是 1~5；LatestType 是英文代碼表示知識類別，可能是 T（技術知識）或 B（書本知識）。

技術知識的引用數（CitedCount）是由 **tblCitation** 的觸發程序控制的，當一筆資料寫入 **tblCitation** 時，會啟動觸發程序將此知識的 CitedCount 加 1，刪除時也會啟動觸發程序將 CitedCount 減 1，藉此維護知識的引用數。

當 **tblApproach** 的資料有插入或刪除時，會執行一預存程序 **spApprCount**，這個程序會重算該父議題的解法數（ApprCount）。若 **tblIssue** 的資料有加入或刪除時，會執行預存程序 **spIssueCount** 重算父類別的子議題數（IssueCount），並會以遞迴的方式計算至根類別。

當 **tblTechKlg** 與 **tblBook** 的資料有插入或刪除時，會執行一預存程序

spIssueStatus，此程序會找出影響議題的最新五筆知識（依出版日期排序），並將之插入 **tblLatestStatus** 中，並呼叫 **spCateStatus** 重新計算父類別的最新五筆知識，以遞迴的方式計算至根類別。

5.3.2 記憶體快取資料結構設計

為加速資料存取，設計記憶體快取資料以加速存取速度。此方法將資料庫的資料讀至記憶體中，採用 COM 的物件儲存，利用指標將物件連結起來。某些較耗費空間且較不常用的資料則先讀至檔案系統，直接存取檔案會比從資料庫查詢來得快。快取資料的類別關係圖如圖 5 - 4：

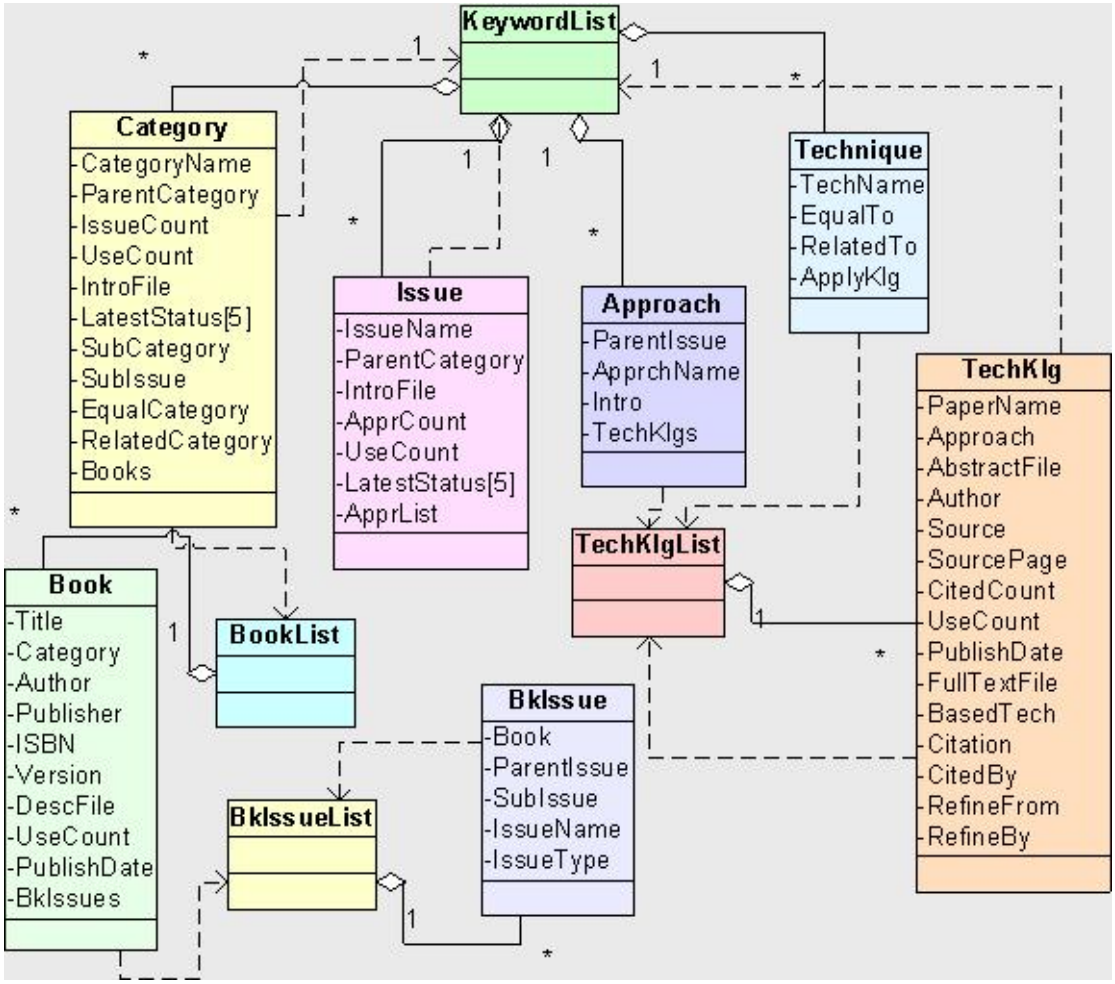

圖 5 - 4　快取資料類別關係圖

各類別設計介紹如下：

- **Category**

此物件儲存一個知識類別，並以一指標指向其父類別（ParentCategory），類別簡介存放在檔案系統中，以 IntroFile 存檔名。最新狀態存放五個指標，指向 Book 或 TechKlg，此外相等類別、相關類別及子類別都以 KeywordList 類別存取，而其下的書本知識則以 BookList 存取，其關係圖如圖 5 - 5 所示：

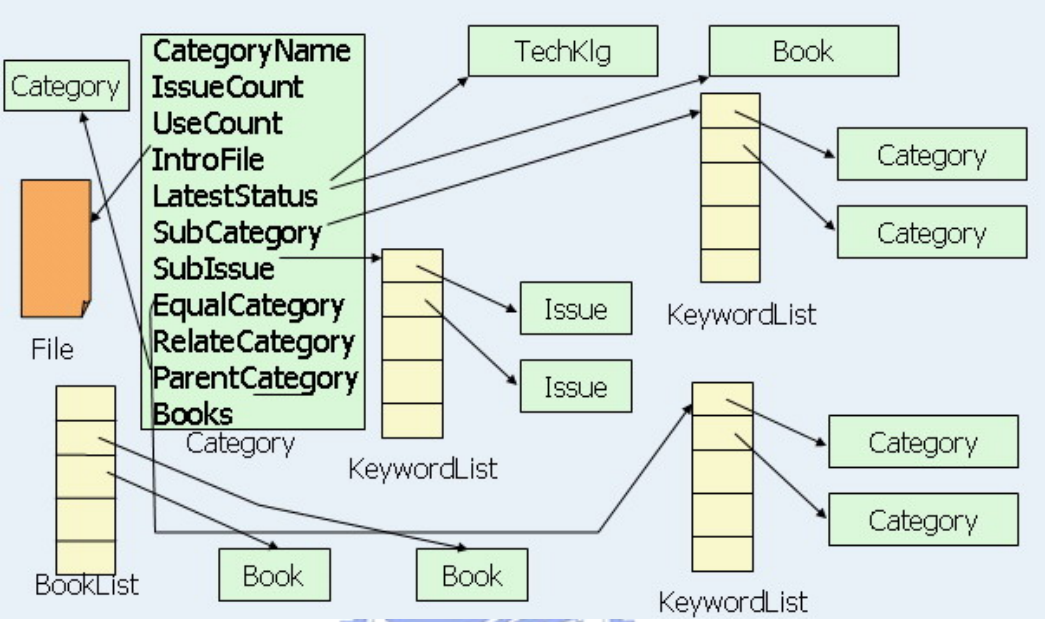

圖 5 - 5　知識類別資料結構關係圖

- **Issue**

此物件儲存一個議題，以一指標指向其父類別（ParentCategory），簡介一樣存放於檔案系統中，也以五個指標存放其最新狀態；其下的解法以 KeywordList 類別存取。其關係圖如圖 5 - 6 所示：

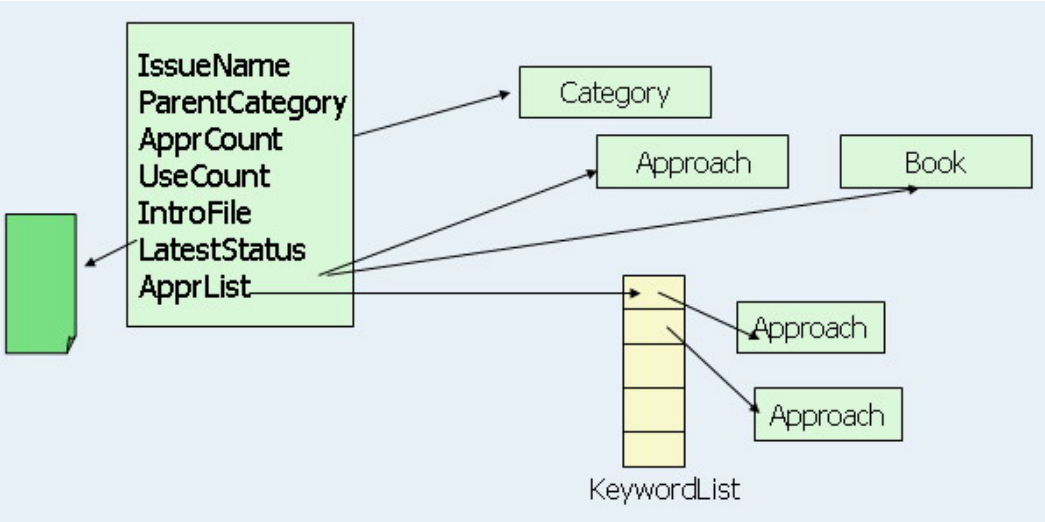

圖 5 - 6　議題資料結構關係圖

- **Approach**

  此物件儲存一個解法，以一個指標指向實作的議題（ParentIssue），並以一TechKlgList 類別存取採用此解法的技術知識，其關係圖如圖 5 - 7 所示：

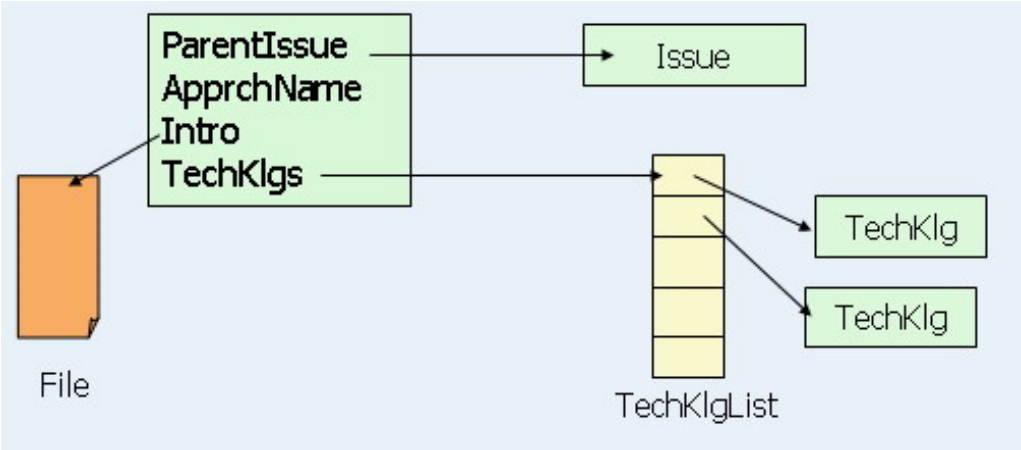

圖 5 - 7　解法資料結構關係圖

- **Technique**

  此物件儲存一個技術，分別以一指標指向與其相關或相等的關鍵字，並以一TechKlgList 類別存取使用此技術的知識，其關係圖如圖 5 - 8 所示：

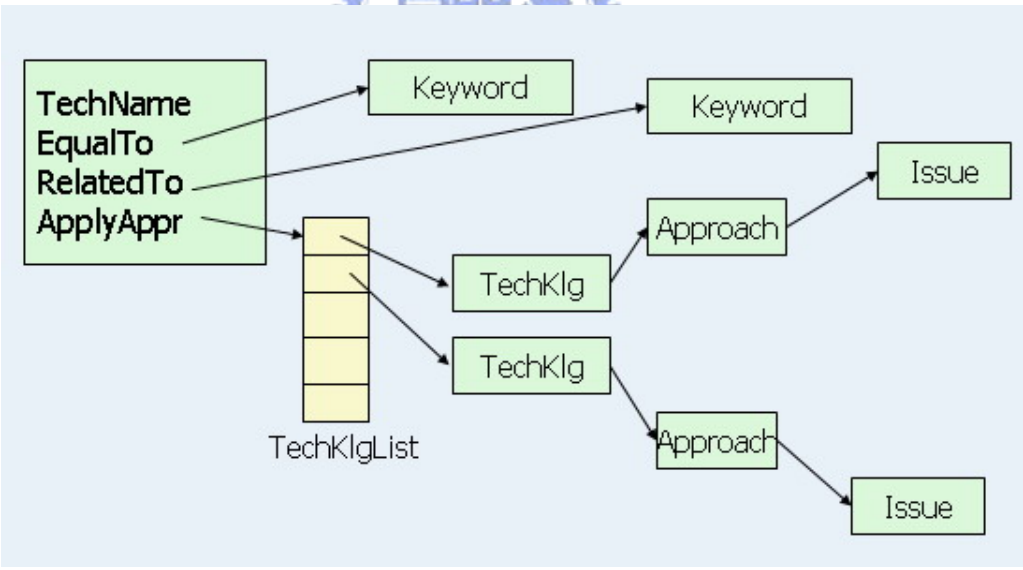

圖 5 - 8　技術資料結構關係圖

- **TechKlg**

  此物件儲存一個技術知識，可能是技術論文或技術報告。利用指標連到此知識採用的解法，並以 KeywordList 類別存取其運用的技術。此外利用TechKlgList 類別存取引用、被引用、改善、被改善的技術知識，其關係圖如圖 5 - 9 所示：

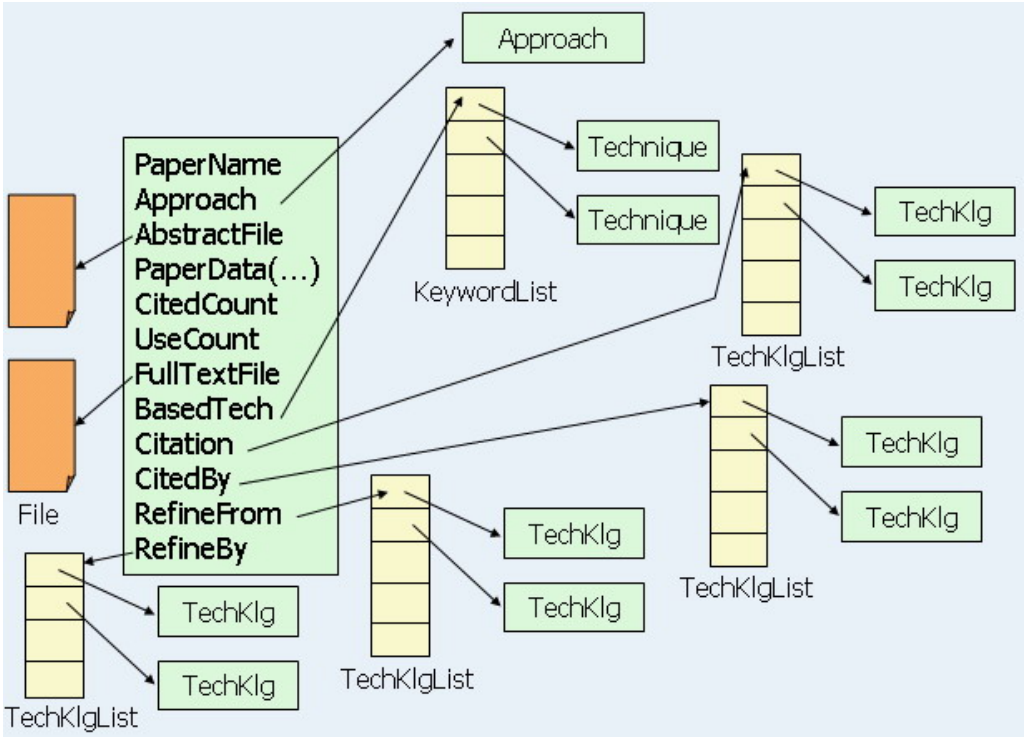

圖 5 - 9　技術知識資料結構關係圖

- **Book**

此物件儲存一個書本知識。利用一個指標指向此書本所探討的知識類別，並以一 BkIssueList 類別存取其下的子議題，其關係圖如圖 5 - 10 所示：

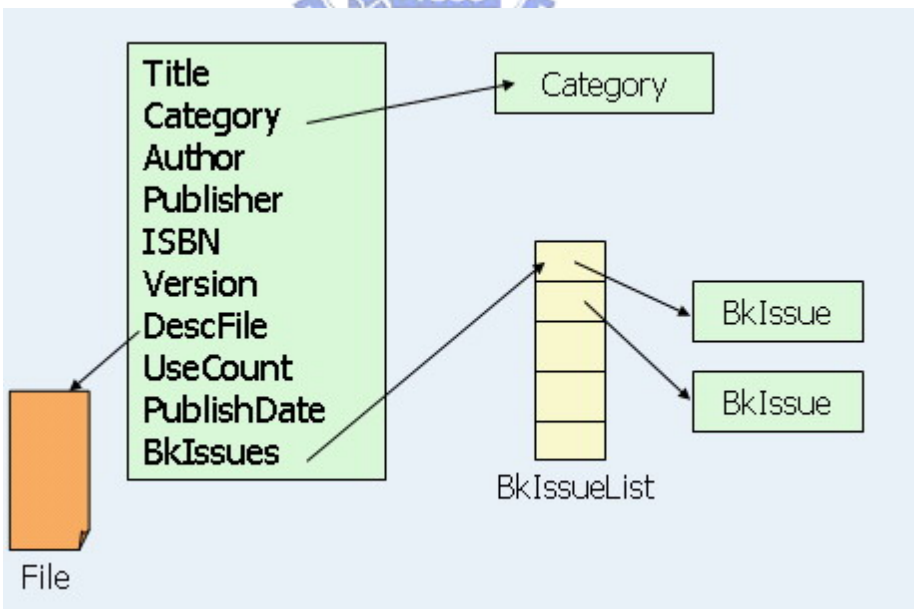

圖 5 - 10　書本知識資料結構關係圖

- **BkIssue**

此物件儲存一個書本知識的子議題。利用一指標指向該書本知識，以一指標指向父議題，並以一 BkIssueList 類別存取其下的子議題，關係圖如圖 5 - 11

所示：

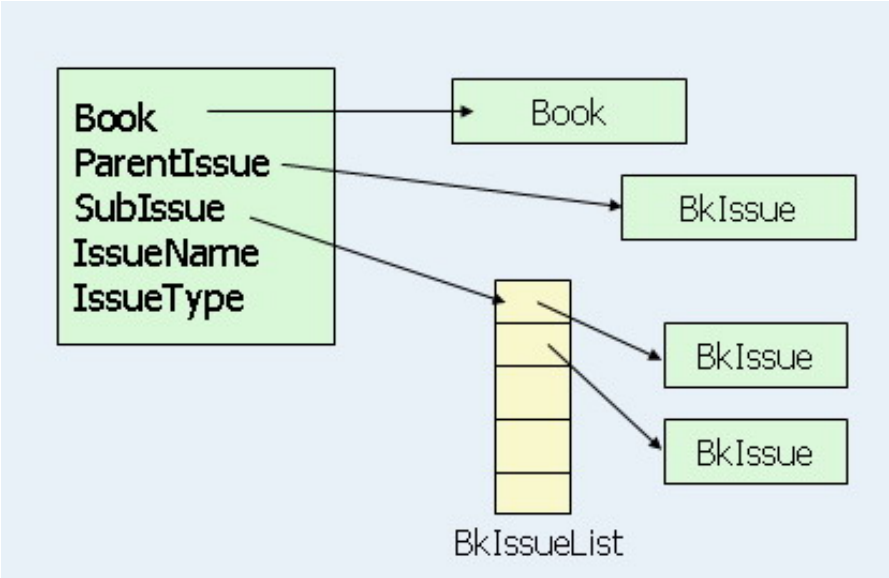

圖 5 - 11　書本子議題資料結構關係圖

上述物件資料是在系統初始化時從資料庫取出，置於記憶體中，當使用者要使用知識庫時，各相關模組便自上述物件存取資料，由於知識庫是多人使用環境，上述記憶體會讓多個處理程序（Process）共用，這時如果資料有修改，某些記憶體區段會被鎖住，此時其他處理程序將無法讀到資料。由於平行執行需要非常多的考量與控制，我們採用較簡化的方式，將所有需要修改的資料先寫入一個修改資料列表（Update List），並準備另一台機器做交換。兩台伺服器中，一台伺服器供使用者查詢及使用，另一台則將應修改的資料進行處理，一段時間後將另一台機器換給使用者查詢，原機器則依其列表處理所有應修改的資料。以這種方式，供使用者查詢的機器不必立刻修改資料，因此所有的記憶體區段都只是讀取鎖定，其他的處理程序仍可讀到資料，也不用考慮因修改導致平行程序資料錯誤的問題。

5.4　使用者介面說明

使用者介面主要分為兩部份，一個是一般使用者對知識庫進行類別瀏覽與知識搜尋，另一個是知識工程師對知識庫的分類架構及議題、技術等內容進行修改

或調整，以下將在 5.4.1 節介紹一般使用者介面，在 5.4.2 節介紹知識工程師介面。

5.4.1 一般使用者介面

一般使用者進入知識庫，首先將在下方看到知識庫的功能介紹，在右上方有四項基本功能：「新增知識」、「類別瀏覽」、「關鍵字搜尋」、「最新文件」。新增知識是讓使用者以符號表示法將知識新增至知識庫；而類別瀏覽則是最主要查詢知識庫的功能，藉由「問題－解法」分類架構逐步縮小範圍，以查詢符合需要的知識；關鍵字搜尋是供使用者直接查詢特定技術論文、技術報告或了解技術運用狀況；最新文件是供使用者了解目前知識庫有哪些最新文件。在主頁下方有功能介紹，如圖 5 - 12：

圖 5 - 12 一般使用者主畫面

新增知識分為新增技術論文、新增技術報告及新增書本知識，其中新增技術論文的畫面中，要輸入論文名稱、作者、來源名稱、頁數範圍、摘要、出版日期、符號表示關鍵字以及全文檔案，引用（Citation），如圖 5 - 13：

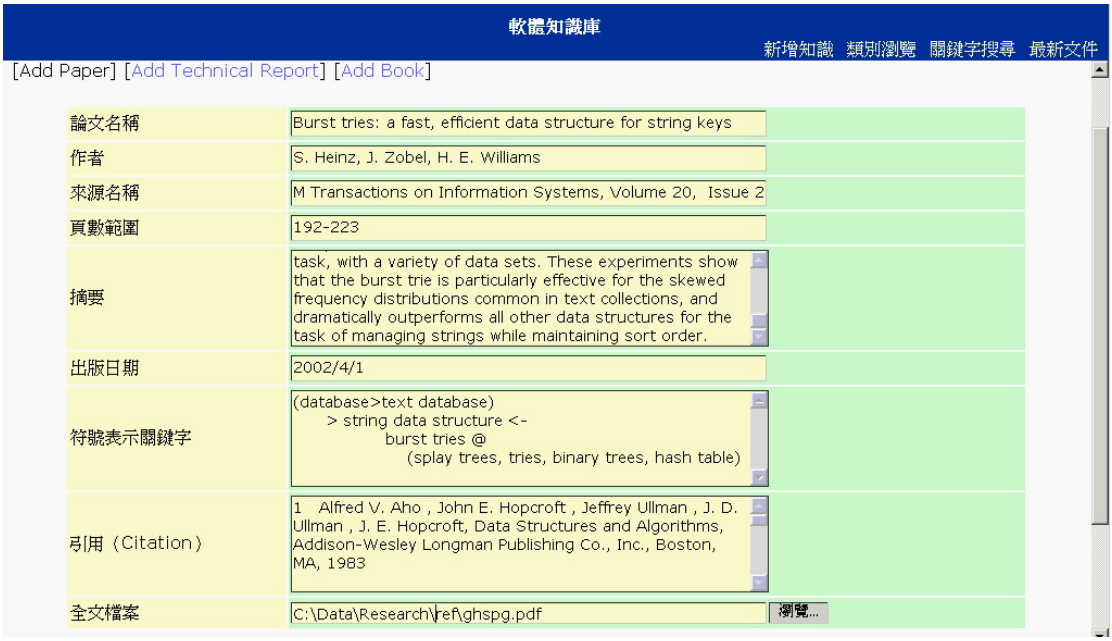

圖 5 - 13　新增論文畫面

引用的資料依論文引用固定格式，直接輸入所有的引用資料。每筆引用資料開頭要編號，結束再多換一行。由於作者數量不固定，解析時可由後往前解析，找出引用知識的資訊，這部份也可考慮採用 ACI（Autonomous Citation Indexing）系統，籍由其論文提出的演算法直接對論文全文處理，找出引用的地方並進行解析。至於符號表示法由於某些符號非標準 ASCII 碼，故輸入時將之替換如下：

(17) 包含（Aggregation）：≻ 換為 >

(18) 實現（Realization）：← 換為 <-

(19) 使用（Use）：⊕ 換為 @

(20) 繼承（Inheritance）：▷ 換為 |>

(21) 相等（Equal）：≡ 換為 =

(22) 關聯（Association）：⇄ 換為 <=>

(23) 依存（Dependency）：► 換為 #>

(24) 改善（Refine）：⇒ 換為 =>

由於符號表示法對人而言並不十分直觀，因此可設計一簡單工具，讓作者填寫資料後轉換為符號表示法。

新增書本知識的畫面中，要輸入書名、版本、作者、出版社、ISBN、簡介、出版日期、符號表示關鍵字、全文檔案等，如圖 5 - 14：

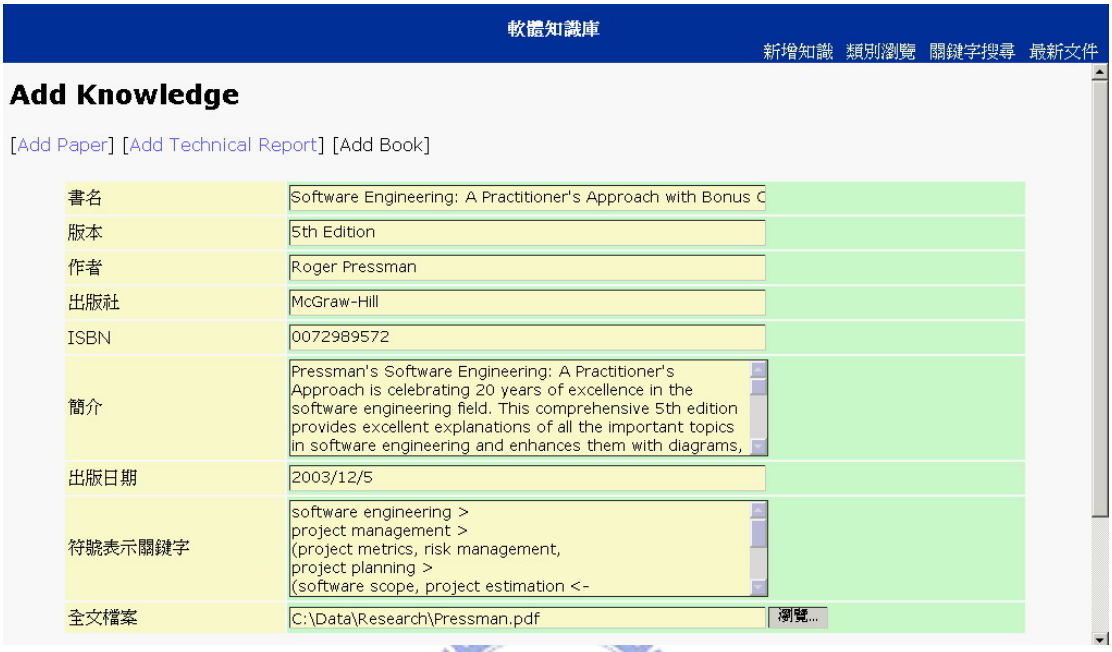

圖 5 - 14　新增書本知識畫面

類別瀏覽主畫面中，左邊是樹狀結構的類別瀏覽功能，右邊則是功能簡介。如圖 5 - 15：

圖 5 - 15　類別瀏覽主畫面

點選特定類別後，會顯示類別的簡介，對單一類別可以觀看其子類別、書本知識、最新狀態、最熱門子類別、相關類別等，其畫面如圖 5 - 16 所示：

圖 5 - 16　類別簡介畫面

其中，最熱門子議題有兩種排序方式，一種依使用次數，另一種依其下子議題數，如圖 5 - 17 所示：

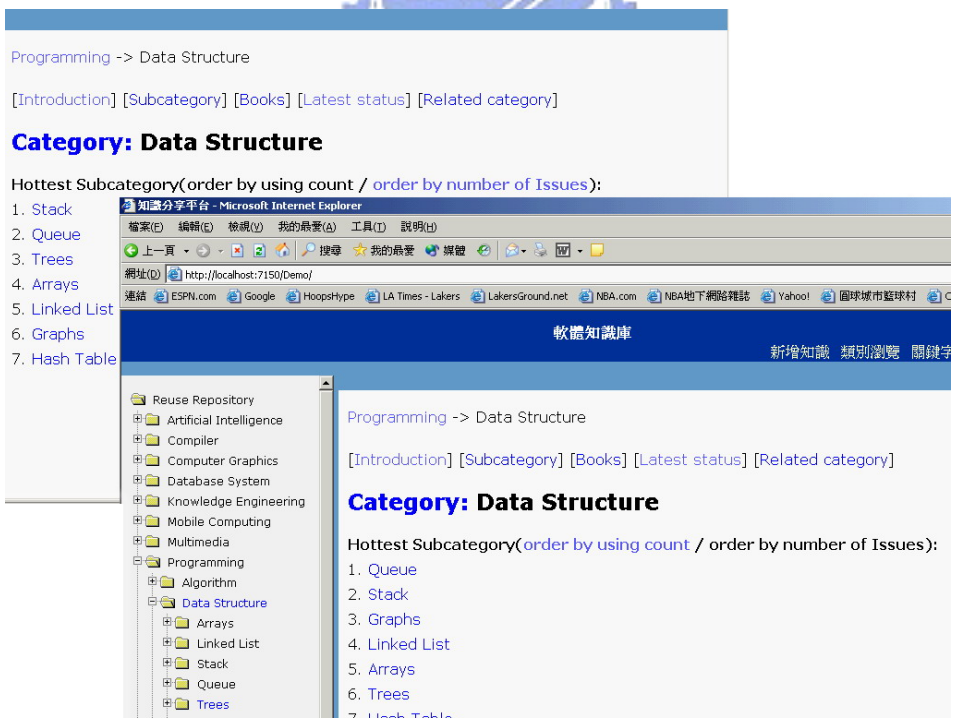

圖 5 - 17　最熱門子類別畫面

類別最新狀態會依出版日期列出該類別下最新的知識，如圖 5 - 18：

圖 5 - 18　類別最新狀態畫面

當瀏覽至最下層類別時，可觀看其子議題列表，如圖 5 - 19：

圖 5 - 19　子議題列表畫面

點選特定議題後，可看到議題簡介。使用者可選擇觀看議題最新狀態或解法列表，議題最新狀態是依出版日期列出解決此議題的技術知識，而解法列表則是依繼承關係排序列出相關解法，如圖 5 - 20：

圖 5 - 20　解法列表畫面

點選解法後可看到解法簡介，並可找出採用該解法的技術知識，解法畫面如圖 5 - 21：

圖 5 - 21　解法簡介畫面

解法下的技術知識會依改善關係、發表日期、引用關係來進行排序，點選特定技術知識後，可看到該項知識的基本資料及摘要（Abstract），並可閱讀全文，並可自由選擇觀看其技術、引用知識、被哪些知識引用、改善技術知識、被哪些

技術知識改善，如圖 5 - 22 所示：

圖 5 - 22　技術知識主畫面

引用知識與改善知識都會列出論文全名，論文引用資料如圖 5 - 23 所示：

圖 5 - 23　引用知識畫面

此外也可觀看論文運用的技術，如圖 5 - 24 所示：

圖 5 - 24　論文運用技術畫面

使用者可點選特定技術，以了解該技術的運用狀況，如圖 5 - 25：

圖 5 - 25　運用特定技術之論文列表

除了可列出運用特定技術之論文外，知識庫尚可整理運用特定技術的議題及領域類別，使用特定技術的議題列表如圖 5 - 26：

圖 5 - 26　運用技術之議題列表

除了上述針對技術知識的功能外，尚有書本知識的功能。當使用者點選了特定類別功能中的書本列表，知識庫會依子議題數列出相關書本知識，使用者可點選子議題列表觀看該類別下有哪些書本議題。書本列表畫面如圖 5 - 27：

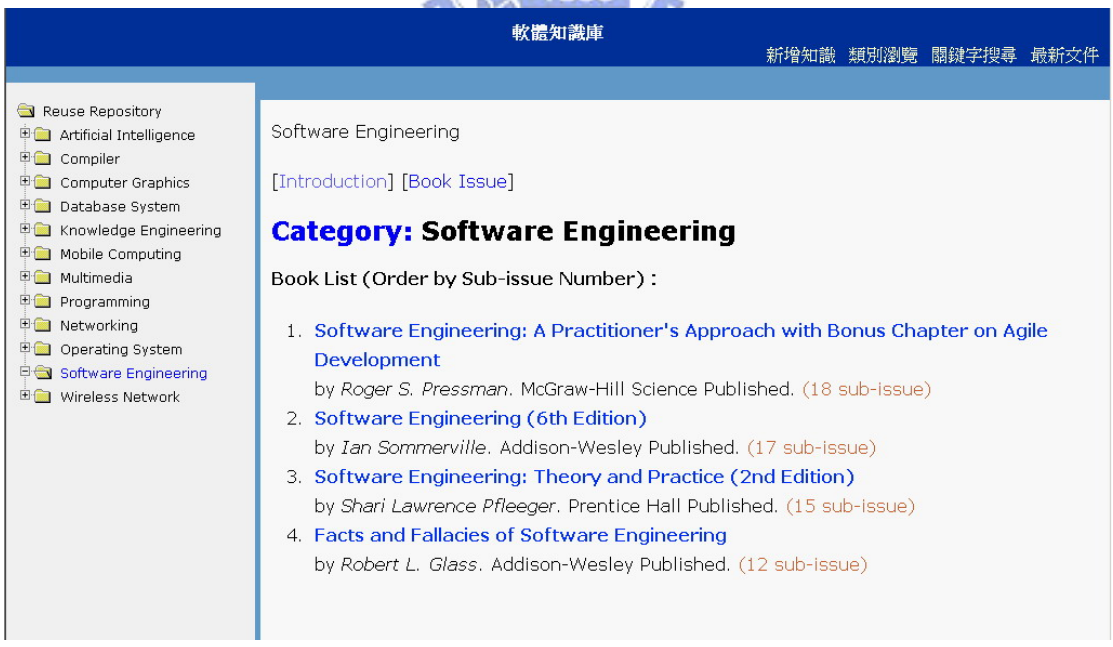

圖 5 - 27　書本知識列表

若選擇一本書後，將進入書本知識介紹畫面，其中包括書本的基本資料及簡介，此外使用者可選擇觀看議題架構，書本知識介紹如圖 5 - 28：

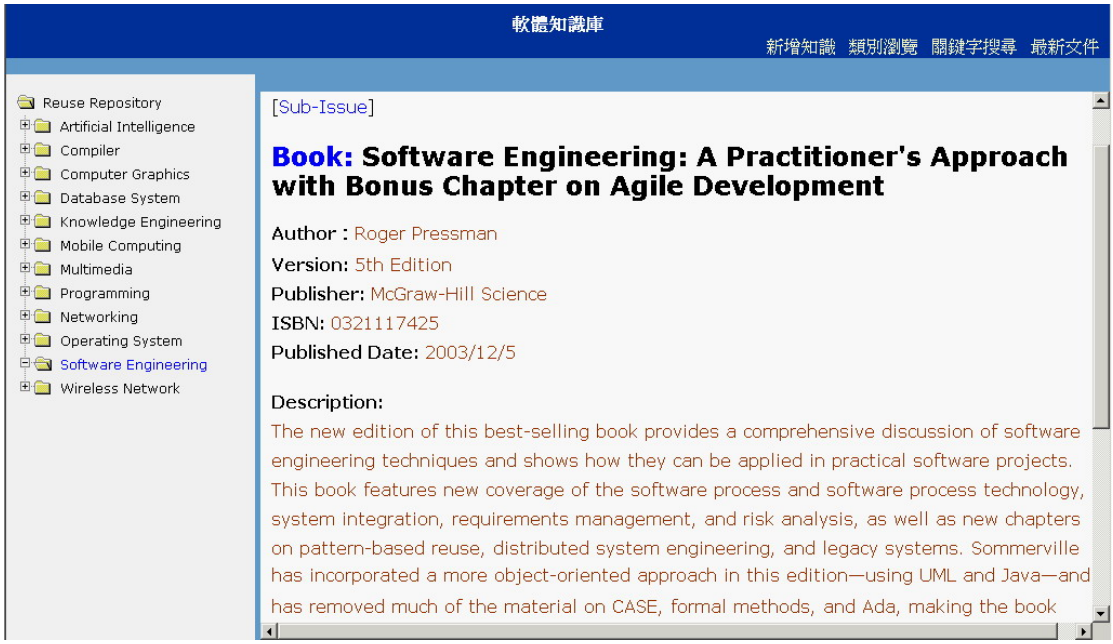

圖 5 - 28　書本知識簡介畫面

而書本知識子議題架構中，可點選子議題觀看其依存關係，如圖 5 - 29 所示：

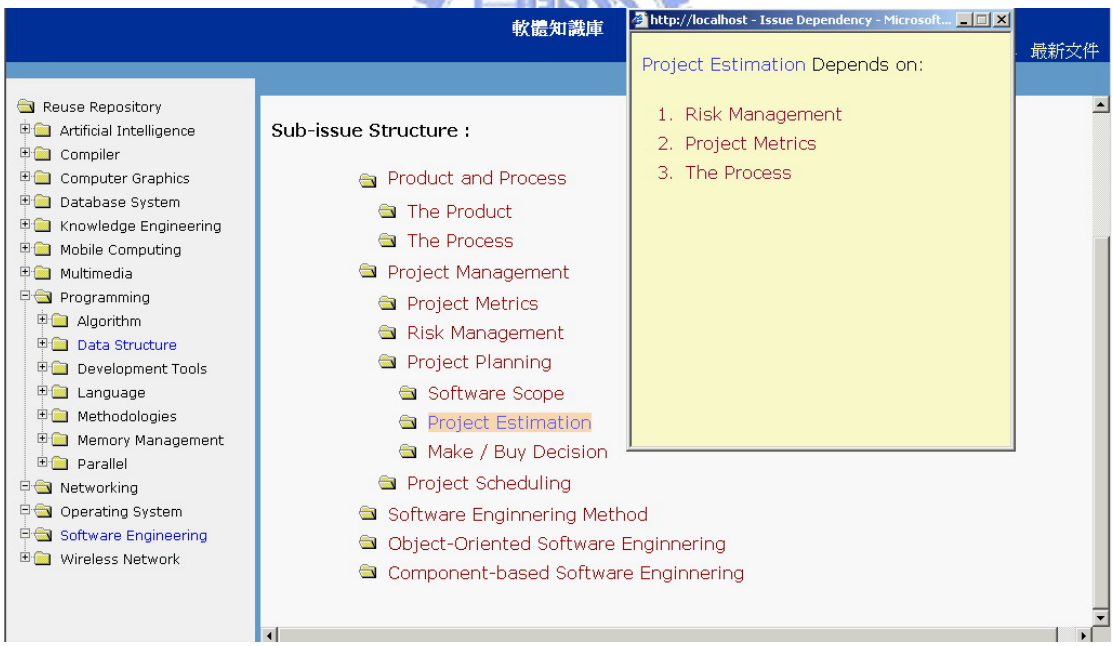

圖 5 - 29　書本子議題架構畫面

5.4.2　知識工程師介面

知識工程師具有管理類別、議題架構的功能，此外尚能管理技術（Technique）

及其關聯性等，登入驗證通過後主畫面如圖 5 - 30：

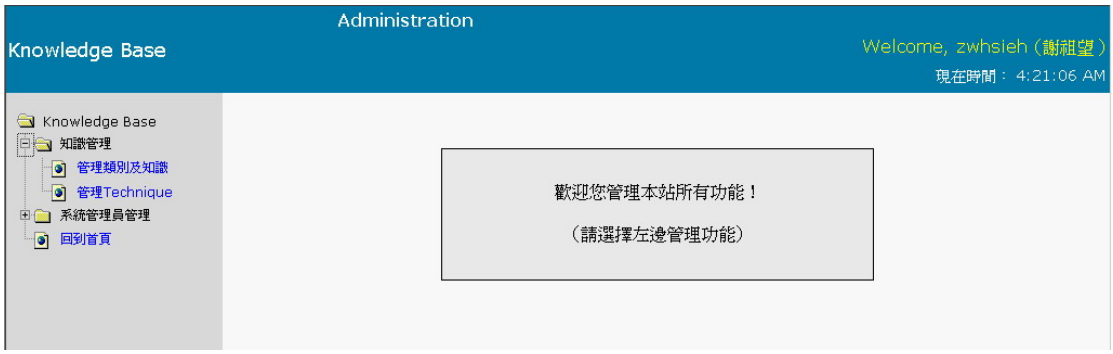

圖 5 - 30　知識工程師主畫面

管理類別架構功能可以新增、刪除、分割、合併、搬移知識類別，並可新增子議題，也可修改類別及子議題的簡介及關聯性，其畫面如圖 5 - 31 所示：

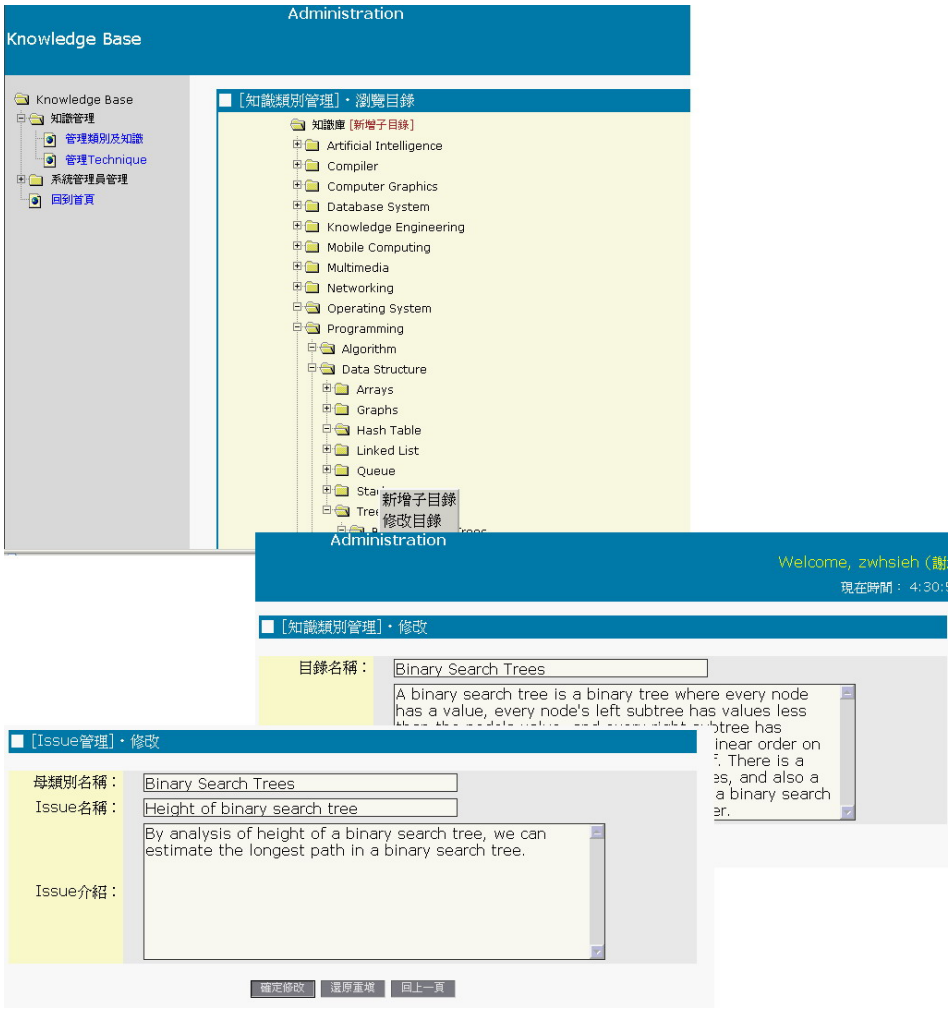

圖 5 - 31　類別架構管理畫面

知識工程師也可管理技術，以及與技術相關或相等的關鍵字，畫面如圖 5 - 32：

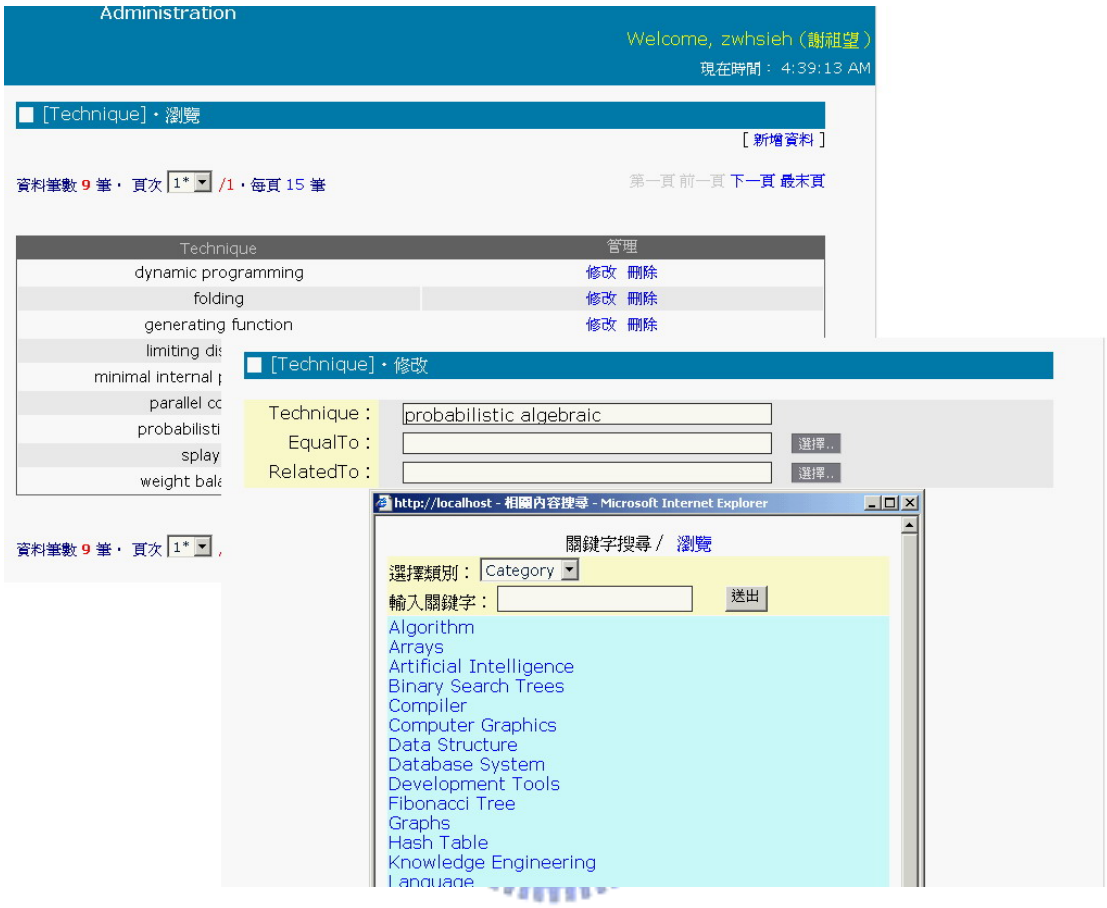

圖 5 - 32　技術管理畫面

第六章、結論

軟體產業是一項高度知識密集與人力密集的產業，必須藉由團隊的溝通、協調與合作來完成工作，這也顯示出在軟體發展過程中必須做好知識的管理，才能加速知識傳遞與學習的過程，縮短軟體開發時間，提高軟體品質，進而降低整體開發成本與加快產品的推出。

目前的軟體知識庫都面臨缺乏效率的問題。當知識累積數量增多時，知識庫的效率及正確性便會降低。知識庫的效率與知識分類方法有密不可分的關係，本研究分析目前常用的知識分類方法，發現這些分類方法中，Taxonomy 太過簡單故缺乏效率；Faceted Classification 會因知識數量增多而過於複雜；CBR 則缺乏良好的分類架構；Ontology 只做到知識的描述，而欠缺知識間的關聯性。由於這些方法都不是為了軟體知識管理而設計，故運用在軟體知識上都有其不足之處。

因此，本研究從軟體知識的本質進行分析，探討了技術論文、技術報告、書本知識等類型的軟體知識，發現這些知識都具有「問題導向」的特性。此外，為了加速知識搜尋效率，這些知識都會定義關鍵字，然而一般關鍵字都較零散且缺乏規則，且關鍵字之間沒有具體的關聯性。因此，本研究提出以「問題－解法」為基礎的軟體知識分類法，將關鍵字分為「領域類別」（Category）、「議題」（Issue）、「解法方法」（Approach）、「技術」（Technique）四類，並進一步發現這些關鍵字類型以及軟體知識都可利用物件導向的中的類別或是物件來表示。故可利用物件導向的各種關係定義彼此的關聯性，包括了包含（Aggregation）、實現（Realization）、使用（Use）、繼承（Inheritance）、相等（Equal）、相關（Association）、依存（Dependency）、改善（Refine）等。利用這些關鍵字類型及關聯性，可有效對軟體知識分類，提供不同需求使用者適當的知識，也可對知識進行比較與排序，即使知識數量增多仍能維持知識檢索的效率。

為改進常見分類法對軟體知識分類深度不足的缺點，本研究提出以七層架構為基礎建構領域類別，以符合各領域學門的深度。此外，本研究更進一步將這些關鍵字的關聯性以符號表示，並提出符號表示法的 BNF，使知識作者能利用「問題－解法」知識分類法定義關鍵字，以此方法有效地對知識進行描述。

本研究依提出的知識分類方法，實作了一雛型（Prototype）知識庫，以驗證其可行性。此系統使用 IIS（Internet Information Server）搭配 ASP（Active Server Page）做為網站架設及系統開發平台，後端採用 SQL Server 資料庫存放資料。

本研究所提出的軟體知識的分類方法具有下列優點：

(1) 能依使用者需求給予合適的知識，包括領域介紹、議題介紹、問題解法及技術運用狀況等。

(2) 對知識進行有效的分類及描述，改善傳統關鍵字搜尋效率不足的問題。

(3) 提供知識的關聯性，讓使用者在閱讀知識後可以依需求閱讀相關知識，並可依關聯性對知識進行比較及排序。

(4) 分類架構具有足夠的深度以符合各領域學門探討愈來愈深的情況，並應提供彈性使分類架構能依領域學門的發展適當地調整。

綜合上述，本研究之貢獻如下：

(1) 發現技術論文可用「領域類別」、「議題」、「解法」、「技術」來表示

本研究發現技術知識具有「問題導向」本質，故可利用所在領域、解決的議題、採用的解法及運用的技術來代表一篇論文，這樣的表示法簡短而有效地說明技術知識的核心意義，大幅提昇表示技術論文的效率。

(2) 提出技術論文的關聯性

利用上述方式表示技術論文後，可進一步發現技術論文的關係。若新解法是由舊解法，則是繼承（Inheritance）關係；兩篇論文若採用相同解法及技術，則新論文是改善（Refine）自舊論文的方法；論文可視為解

法類別的物件，故相同解法的論文可視為相同類別的物件；論文與技術間有運用（Use）關係，故可知哪些論文運用了特定技術，也可比較論文間使用的技術；解法對議題有實作（Realization）關係，故可知哪些論文解決了相同的議題。這些技術知識的關係能提供知識間的關聯性與比較，改善傳統知識分類法的缺點。

(3) 提出以「問題－解法」為基礎的軟體知識分類架構：

依前述，我們能以「領域類別」、「議題」、「解法」、「技術」表示技術論文，可進一步利用這些關鍵字類型建構一軟體知識分類架構。在最上層以七層的領域類別為基礎，在其下有各領域類別的議題，各議題之下有其解法，而技術論文又有運用的技術。依此分類架構，使用者能夠從分類架構中找尋領域類別的介紹、找出類別下的議題、找出議題下的解法，以及不同解法論文所使用的技術，此外將領域類別下也有書本知識，對領域類別不熟的使用者可找尋合適的書籍閱讀。這個分類架構能提供不同需求的使用者合適的知識，並將技術論文、技術報告及書本知識整合在一有效率的分類系統中。

(4) 提出知識關聯性的符號表示法：

符號表示法能讓知識的描述標準化。定義關鍵字後再以符號表示法對知識描述，則知識的解析、分類、儲存等工作都可自動化，提昇軟體知識處理的效率。

(5) 實作此分類法的雛型知識庫系統：

本研究根據提出的軟體知識分類法實作一個雛型知識庫系統，讓知識庫使用者能依其需求找尋合適的知識，證實「問題－解法」知識分類法的可行性。

由於知識庫系統架構龐大複雜，限於時間與研究人力的關係，此知識分類方

法在實作上仍有許多可再加強的地方：

(1) 建構知識分享平台。

依此分類法可設計並建構知識分享平台，讓所有人於其上提供、分享、收集、買賣知識，讓有價值的軟體知識也能有分享的管道，並將知識庫的功能提昇至更高的層次。設計這種大型知識庫可能必須採用分散式系統的架構設計，考慮負載平衡，並改善資料儲存架構，減少系統資源的消耗，以提昇執行效率。

(2) 建構標準化的知識描述交換格式。

符號表示法雖然能對知識描述標準化，但並非一般電腦系統的標準資料交換格式，若要達到跨知識庫整合的目標，則必須定義一套以 XML（Extensible Markup Language）為基礎的規格，讓技術知識與書本知識的基本資料、關鍵字表示法及其內容都能以此規格進行交換。

參考文獻

[1] P. Naur and B. Randell (eds), *Software Engineering: A Report on a Conference sponsored by NATO Science Committee*. NATO 1969

[2] 劉文謙, 施向玨與鍾乾癸, “軟體知識管理,” 第十三屆物件導向技術及應用研討會, 臺中縣 霧峰鄉 臺中健康暨管理學院, 中華民國九十一年九月十三日

[3] 劉文謙, 廖元誠, 施向玨與鍾乾癸, “結合知識管理與能力管理之軟體發展環境,” 第十四屆物件導向技術及應用研討會, 桃園縣 中壢市 元智大學, 中華民國九十二年九月十二日

[4] http://support.microsoft.com/default.aspx?scid=fh;ZH-TW;KBHOWTO

[5] http://www.computer.org/publications/dlib/

[6] http://www.acm.org/dl/

[7] Bloom, B.S. (Ed.) *Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain.* New York ; Toronto: Longmans, Green. 1956

[8] Wynar, Bohdan S. *Introduction to cataloging and classification*. 8th edition. p. 320

[9] A. Aamodt, E. Plaza; *Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches*. AI Communications. IOS Press, Vol. 7: 1, pp. 39-59. 1994

[10] D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag

[11] I. Rus, et. Al., *“Knowledge Management in Software Engineering: A State-of-the-Art-Report,”* DACS Report, 2001

[12] Ikujiro Nonaka and Hirotaka Takeuchi, "The Knowledge-Creating Company : How Japanese Companies Create the Dynamics of Innovation", OXFORD UNIVERSITY PRESS, 1995

[13] http://www.oclc.org/dewey/about/

[14] J.C.M. Hanson. . *The Subject Catalogs of the Library of Congress*. Bulletin of

the American Library Association 3:385-97. 1909

[15] http://www.acm.org/class/

[16] http://dmoz.org/about.html

[17] Vernon, K., and Lang, Valerie. *The London Classification of Business Studies: A Classification and Thesaurus for Business Libraries*. London: London Graduate School of Business Studies, 1979

[18] *Uniclass : Organization/Institution: Construction Project Information Committee*. RIBA Publications , 1997

[19] http://www.kmconnection.com/

[20] http://facetmap.com/browse.jsp

[21] http://xfml.org

[22] Roger Schank: *Dynamic memory: a theory of reminding and learning in computers and people.* Cambridge University Press, 1982

[23] Veloso, M.M., Carbonell, J. *Derivational analogy in PRODIGY*. Machine Learning 10(3), pp.249-278. 1993

[24] Aamodt, A. & Plaza, E., Cased-Based Reasoning: Foundational Issues, Methodological Variations and System Approaches

[25] I. Becerra-Fernandez and D. Aha. *Case-based problem solving for knowledge management systems*. In Proceedings of the Twelfth Annual Florida Artificial Intelligence Research Symposium, pages 219-223, Menlo Park, 1999. AAAI

[26] T. R. Gruber: *A Translation Approach to Portable Ontology Specifications*, Knowledge Acquisition, 5:199—220, 1993.

[27] http://www.cogsci.princeton.edu/~wn/

[28] M.R. Genesereth and R.E. Fikes: *Knowledge Interchange Format, Version 3.0, Reference Manual*. Technical Report, Logic-92-1, Computer Science Dept., Stanford University, 1992. http://www.cs.umbc.edu/kse/

[29] T.R. Gruber: *Towards Principles for the Design of Ontologies used for Knowledge Sharing*, International Journal of Human-Computer Studies, 43:907-928, 1995.

[30] Dieter Fensel and Ian Horrocks and Frank van Harmelen and Deborah L.

McGuinness and Peter F. Patel-Schneider, *OIL: An Ontology Infrastructure for the Semantic Web,* IEEE Intelligent Systems, 2001, 16, 2.

[31] Fabrizio Sebastiani, *Machine learning in automated text categorization*, ACM Computing Surveys Volume 34, Issue 1(March 2002), pp. 1-47

[32] David F. Bacon, Susan L. Graham and Oliver J. Sharp, Compiler transformations for high-performance computing, ACM Computing Surveys Volume 26, Issue 4 (December 1994), pp. 345-420

[33] W. Meng, C. Yu, and K. Liu. Building efficient and effective metasearch engines. ACM Computer Surveys, March 2002.

[34] J. Jing, A. Helal, A. Elmagarmid, "*Client-Server Computing in Mobile Environments*," ACM Computing Surveys Vol. 31, No. 2, pp. 117 - 157, June 1999.

[35] Steffen Heinz, Justin Zobel, Hugh E. Williams, "*Burst Tries: A Fast, Efficient Data Structure for String Keys*", Transaction of Information System, Vol. 30, Issue 2

[36] H. Chang and S. S. Iyengar, `*Efficient algorithms to globally balance a binary search tree*', Commun. ACM, 27, (7), 695--702 (1984).

[37] S.L. Martins, M.G.C. Resende, C.C. Ribeiro and P. Pardalos. *A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy*, J. of Global Optimization, 17: 267-283, 2000.

[38] R. S. Pressman, Software Engineering, *A Practitioner's Approach, European edition*, McGraw-Hill, 1994.

[39] Richard Johnsonbaugh, Martin Kalin, *Object-Oriented Programming in C++*, Prentice Hall, 2000

[40] Hafedh Mili, Ali Mili, Sherif Yacoub, Edward Addy, *Reuse Based Software Engineering: Techniques, Organizations, and Measurement*, John Wiley & Sons Inc, 2002