南　華　大　學

資訊管理研究所

碩士論文

軟體系統複雜度評估—

一種修正的堆疊基馬可夫模式

Software System complexity measurement—
A Modified Stack-based Markov Model

指導教授：王　鄭　慈　博士

鍾　志　明　博士

研　究　生：賴　秀　女　撰

中　華　民　國　九　十　年　六　月

摘要

本文提出修正的堆疊基馬可夫模式，同時考慮了程式的大小及控制流程的複雜度。此外，更提出了一種新的計算程式大小複雜度的觀念，即除了考慮程式量的大小外，同時考慮到程式表示式的結合關係，並計算表示式中的控制流程的複雜度，以建立出更完善的評估方法。本論文除了建構一簡單評估模式，能將更多影響程式複雜度之因素考量在內，而且更以實際的管理資訊系統來驗證本論文所提出的模式。

提出的模式可作為不同資訊系統所含複雜度高低的衡量工具，而程式複雜度的高低和軟體開發成本與維護成本間又存在高度正相關的關係，所以運用本文提出之模式，亦可作為管理者或決策者，在軟體開發、擴充、選擇，或改善軟體維護工作前的決策參考依據。

關鍵字：軟體評估、複雜度評估、資訊理論、馬可夫模式、堆疊基馬可夫模式、資訊系統評估

Abstract

In this study, a modified Stack-based Markov model has been proposed. This model can represent control flow complexity and block size complexity of software. Besides, it reflects not only volume complexities but also the nesting complexity of expressions. This thesis establishes a simple metric model and it has considered the factors in the programs. Several sets of real management information systems were used to verify for this model.

This proposed model could be a metric tool while to measure the complexities in different information systems. The modified Stack-based Markov model can measure the complexity of software, which has positive correlation with the developing and maintaining cost of the software. Using this model, information theoretic measures of software can be extracted to help managers and researchers making decisions for developing, selecting, or maintaining software systems.

Keywords: information system evaluation, software metric, information theory, Stack-Based Markov Model, complexity

目錄

編號 *頁碼*

表目錄

編號 *頁碼*

圖目錄

編號 *頁碼*

第一章

緒論

隨著科技的進展，電腦技術的日趨成熟，除了硬體技術進步之外，在軟體方面也呈現蓬勃的發展。近年來軟體產業的成長呈跳躍式成長，從西元 1996 年到 1997 年，軟體產業的營業額成長了 20%。在這期間中，前十大軟體公司的市值向上推高 30%，達三千四百億美元的價值[張國鴻 2000]。形形色色的資訊系統已成為日常生活不可或缺的骨幹，其所涵蓋應用的範圍更是與日俱增，相對的，電腦軟體的設計也隨之日趨複雜化，如何設計高品質的軟體系統正是每位軟體設計師與工程師們所關心的課題。

軟體評估在軟體工程領域是個重要的學門，而在軟體評估的領域中最熱門的莫過於軟體的複雜度評估，所謂複雜度評估係指將軟體經由量化的方法，計算出一個數值，以這數值來表示軟體的複雜度。就軟體開發者或維護者的角度而言，要開發或維護軟體所最終必須直接接觸軟體程式碼，所以程式碼的可了解性代表著軟體開發者或維護者對程式碼了解程度的難易，故透由對軟體程式碼來計算軟體複雜度的高低值，這數值正可表示軟體開發者或維護者對此一軟體程式碼可了解性的難易程度高低，而適當之評估除了可以讓管理者對軟體更加地了解，還可用來預估軟體發展時間、軟體開發及維護成本以及軟體之可信賴度，適當利用軟體評估技術可提供以軟體開發者、維護者及管理者的角度來評估軟體效能或改善提昇軟體品質。

1.1 軟體評估介紹

評估(metric)一詞可解釋為針對系統元件或軟體發展過程，控制一些既定的屬性而加以量化的衡量方法[IEEE 1993a]。軟體評估是軟體工程中一個重要的領域，所謂軟體評估，正如其名，得以用於評估軟體，亦即用以測量任何型態的軟體系統、軟體開發程序或相關文件的方法，例如以程式碼的行數來測量產品大小就是一種常用的評估法。而軟體評估的主要功用在於有助於規劃或預測軟體的發展。利用軟體評估技術得以管理軟體專案之品質控制、生產力衡量，且藉由軟體評估工程能使得影響軟體系統之軟體屬性得以被進一步地了解及研究，假若軟體屬性可加以量化評估，則相較之下即可以輕易地控制軟體之品質及軟體之發展效果。因此許多研究即致力於軟體評估領域，並用以預測軟體之維護效果[Li 1993]，以及設計之品質優劣與否 [Basili 1996]。

軟體開發是極端複雜的事，大多數的大型軟體是由數百萬行程式組合而成。舉例來說，微軟視窗九五軟體就有高達十億行的程式碼。每行程式至少有一個執行指令，會與其他行的程式有互動關係，而影響到整個系統，而且其中的任何一個小小的錯誤，可能在剎那間拖垮整個系統 [張國鴻 2000] 。

依不同的觀點，軟體評估可有不同的分類方式，因評估方式所著重的觀點是在於針對軟體產品本身，亦或針對軟體開發過程的不同，而將軟體評估分類為軟體產品評估(software product metrics)以及軟體過程評估(software process metrics)二類[Li 2000]。而這二大類別之下可再加以分類出：程式的大小、程式的控制流程（control flow）、程式的資料流程（data flow）、古典資訊理論（classical information theory）[Wang 1994]。現有之軟體評估領域亦可

分為靜態評估技術(static metrics)及動態評估技術(dynamic metrics)二類，所謂靜態評估，意指運用程式文字特性，例如程式區塊、代符、運算元、運算子.....等等屬性、或程式之圖形特性，例如程式向量圖形中之節點、路徑.....等等屬性來作為衡量單位所計算的評估方法、而動態評估，則是利用實際執行程式的方式來找出重要屬性的特質或屬性相互間的關係。

1.2 研究動機及目的

在軟體評估的研究中，各學者提出之評估模式[Conte 1986, Halstead 1979, McCabe 1976, Henry 1981, Shannon 1948, Edwards 1991]，雖能作為評估之工具，但多未臻完善，目前的評估模式多未能將程式中的影響複雜度的因子用簡單模式運算，故期望建構一合適的軟體評估模式，即簡單又能同時兼顧程式的大小及控制流程的複雜度，並能作為比較不同軟體間所含的複雜度高低的衡量準則工具。

在諸多的軟體複雜度評估方法中，以學者所提出堆疊基馬可夫模式的評估方法中[Edwards 1991, Wang 1991, Wang 1994, Wang 1997]，同時考慮程式的控制流程、結構及程式大小，但其表示式及計算上較為複雜，所以擬修正堆疊基馬可夫模式評估法，轉化為一簡單的堆疊基馬可夫模式評估法，利用更簡單之模式以作為軟體評估指標。

從系統發展生命週期之總成本來看，其中系統維護成本所佔的比例從七 0 年代的 35%至 40%、八 0 年代之 40%至 60%、至今約為 70%至 80%的壓力下，此項比例還會隨著資訊系統的不斷發展將會一直的提高[林仁常 1998, 楊建民 1996]。但對管理者而言，系統的高維護費用是極不樂於預見的結果，如果一個系統的可維護性是可以評估的話，那麼相

對而言，維護系統所需耗費的資源將可預先衡量，如此一來，對組織而言，就可避免突發性的支出成本。因此，對管理者而言，程式之可維護性的度量可以作為其決策時的依據，以決定軟體系統是否應進行維護，亦或應重新設計以減少進一步的維護花費。可維護性的度量不會測量出對系統進行特定改變的花費，也不會預言特定元件是否需要加以維護，它們是基於程式可維護性相關於它們複雜度的假設，這樣的度量能測定某些方面的程式複雜度，它表示出高複雜度的程式所對應的是很難維護的軟體系統。

正因為資訊系統之開發及維護成本之高低與系統複雜度間存在顯著正相關[Boehm 1981, Walstion 1977, 林信惠 1993]，且研究亦指出軟體評估與成本估算模式間具有高度正相關，尤其以計算程式大小的評估法與成本間的關係性最高[Grable 1999]，所以管理者需要一可靠工具，作為預估資訊系統開發及維護所需之人力、經費時之衡量準則。若能準確地計算系統的複雜度，則可預估軟體系統開發的成本，故利用複雜度評估技術，可協助管理者作有效之衡量，並作為評估管理資訊系統之開發、擴充或選擇時之決策參考依據。

軟體評估在科學工程學科中尚屬萌芽階段，它在趨於成熟階段前，仍會建立出許多新的指標，但不論如何，這些現有之評估準則或尚未建立之新準則，除了是為了建立出軟體評估的準則或方法外，更需要去驗證準則的有效性，因為唯有真正能準確評估的準則，才能有助於企業軟體發展之用。

因為現今我國產業普遍對於軟體評估及軟體維護的重視程度遠不及發展軟體本身，軟體評估之研究風氣可謂方興未艾，所以引發我針對軟體評估領域進行探究的動機。因此本論文的研究除了將進行現有的各類軟體評估工具歸納整理外，並修正堆疊基馬可夫模式評估法，提出一簡單有用之

軟體評估模式，有效衡量軟體複雜度，並得以提供運用於實際企業界之資訊管理或評估使用，提供學術界與實務界做為評估管理資訊系統之開發、擴充、選擇，或改善軟體維護工作時的參考。

1.3 論文架構

本文第一章為緒論，其餘各章節內容分述如下：第二章概述軟體發展的歷程、整理學者在軟體評估領域內相關文獻的探討。此外，本章並介紹常見的評估方法。

在第三章中，針對堆疊基馬可夫模式作深入之文獻探討，介紹堆疊基馬可夫模式的基本理論，及學者針對堆疊基馬可夫模式在評估上所提出的相關文獻。

第四章中，建立本論文所提出的一種修正的堆疊基馬可夫模式，以修正後的模式建構出一較簡化、周全的軟體衡量方法。

在第五章中，將以實際的程式來驗證修正的堆疊基馬可夫模式，並提出比較數據與相關的研究結果。

在最後一章中則提出結論與建議，除了敘述本論文的研究貢獻外，並針對模式建構上的其餘未考量之處，提出未來的研究方向。因此本論文架構可以圖 1-1 所示：

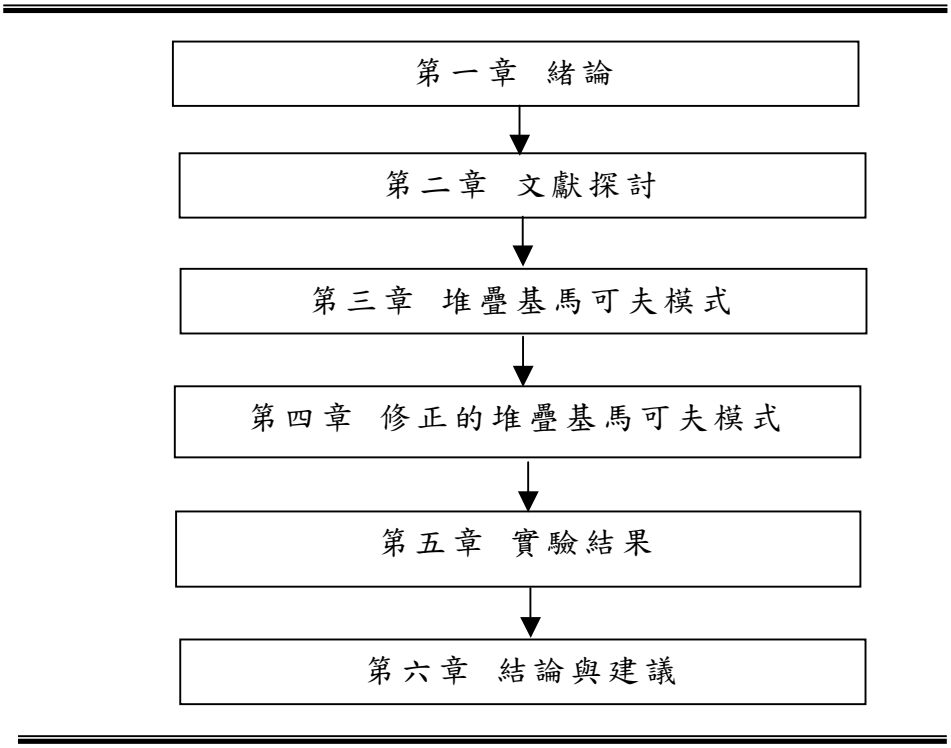

第一章 緒論
第二章 文獻探討
第三章 堆疊基馬可夫模式
第四章 修正的堆疊基馬可夫模式
第五章 實驗結果
第六章 結論與建議

圖 1-1 論文架構

第二章

文獻探討

就軟體評估領域來看，學者們已提出的評估方法為數頗多。在本章裡，將針對軟體複雜度評估之相關領域進行文獻歸納整理，先概述軟體系統發展的歷程，以及概述軟體產品評估與軟體過程評估二類評估類別。

本論文主要將針對軟體產品評估類別進行探討，並提出軟體產品評估類別中，一些主要的評估方法研究中，可約略分為以程式的大小、程式的控制流程、程式的資料流程，和資訊理論為基礎。

- 程式大小：計算程式行數多寡或程式中符號間量的大小[Conte 1986, Gray 1987, Lubinsky 1990, Halstead 1979]。
- 程式控制結構：考慮程式中的控制結構間的關係，來計算程式所含的複雜度高低，此一評估準則常以程式向量圖來作為表示或計算的方式[McCabe 1976, Ramamurthy 1988]。
- 程式資料流程：計算程式間動態資料流程的扇入(fan-in)、扇出(fan-out)數值的複雜度高低[Henry 1981]。
- 古典資訊理論：計算程式符號所涵蓋的資訊量高低，來表示程式複雜度的高低[Shannon 1948, Edwards 1991, Wang 1994, Yang 1992]。

2.1 軟體系統發展歷程

很多人認為軟體是一個現代的發明，但事實上軟體產業已有五十年的歷史了。軟體系統的發展可分為五個紀元。第一紀元，為應顧客訂製的大型軟體專案；第二紀元，獨立軟體產品出現；第三紀元，商用軟體公司出現；第四紀元，套裝的大眾軟體產品；第五紀元，為網際網路加值服務之軟體。

第一紀元：西元 1949 年到 1959 年，開始有著從事代客開發訂製軟體的專業服務公司。在美國，軟體產業的發展最初是由政府的幾項大型專案促成，但這已造就了軟體系統發展的開端。

第二紀元：西元 1959 年到 1969 年，以往一般皆認為軟體並無法單獨出售，而軟體公司也無法從販售軟體獲利。軟體是為個別客戶特別的需求而製作，要不然就是由電腦公司免費送給客戶，第二紀元不同於前階段的地方在於，為因應重複銷售給不同客戶而開出來的軟體產品出現了。但是此時的軟體產品業還是處於萌芽階段，直到 1970 年，軟體產品的銷售估計還不到二億美元。

第三紀元：西元 1969 年到 1981 年，IBM 從 1970 年開始，把軟體、服務，與硬體分開販售的政策，加強軟體產業的獨立性，企業逐漸習慣向軟體公司購買軟體，而且也願意付費購買。

第四紀元：西元 1981 年到 1994 年，個人電腦的出現帶來一種全新的軟體：個人電腦平台上的大眾化套裝軟體，微軟公司即為這階段中最成功和最具影響力的軟體公司。1980 年代，軟體產業產值急遽地以 20%成長率持續擴張。以美國

為例，其軟體產業的營業額從 1982 年的一百億美元，成長到 1985 年的二百五十億美元，是 1979 年的十倍。

第五紀元：西元 1994 年到 2008 年，網際網路，造就了一個新的軟體紀元，利用網路技術，創造出全新的軟體產品。網際網路的風行不僅帶來了軟體產業的發展，更使得軟體產業與電訊事業、傳播媒體及消費電器產品業等等的產業有著整合的可能性。

所以軟體終已從硬體分離，成為極具潛力的新興產業。軟體品質的優劣與否關係著軟體系統的成敗以及軟體系統的後續維護成本的高低，為了有效衡量軟體，軟體評估就成為不可或缺的工具 [張國鴻 2000] 。

2.2 軟體維護

對開發軟體系統而言，系統除錯測試是非常重要的，而且系統測試的費用相較於在程式運轉以後才發現錯誤時，可能要花費的維護費用來看是相當低廉的。此外，幾乎每一個要長時間使用的程式，都需要加以持續的維護。程式維護就是更新程式以使它能繼續使用下去的一種過程。對機構而言，程式維護的成本相當高。據估計，有許多機構大約有一半以上的程式作業時間都是花在維護現有的應用程式上[金子葳 1998]。研究亦指出，各種應用軟體的維護費用雖有很大的不同，但以商業應用系統為例，其維護費用大約與系統發展的費用相當，而嵌入式即時系統的維護費用則可能高於系統發展費用的四倍以上[留忠賢 1997]。而進行軟體評估正可以作為程式測試及程式維護的工具，這些工具可以用來降低程式的維護成本。

2.2.1 軟體維護的定義

軟體維護可定義為「交貨以後一個軟體產品的改正錯誤、改善效率或者其他的屬性，或者使產品適應這個修改過的環境」[IEEE 1993b]。此外，有些學者專家也提出其對軟體維護的看法：

- 軟體維護是由於一個問題或者有改進的需要，對軟體產品進行程式碼和相關文件的修正，目標是在於保存它的完整性時修改現有的軟體產品[Martin 1983]。
- 軟體維護是在於軟體系統成為產品後，所要維持軟體系統的作業與效率[FIPS 1984]。
- 軟體維護涵蓋軟體系統生命週期從安裝直到它結束為止[von Mayrhauser 1990]。

在軟體生命週期中，舊的觀念視軟體維護為軟體系統生命週期的最後一階段，新的觀念則視軟體維護是全盤性及整體性的[Layzell 1994]。

2.2.2 軟體維護的型態

軟體維護會因系統修改或更正的緣由或目的不同而有不同的型態，例如因發現系統錯誤而需要更正、因要增加新功能而修改、及因要將其功能最佳化而修改[Rochkind 1975]。所以系統維護可分為改正、適應、完美、以及預防維護四個型態，其中改正維護是指軟體在實際使用時發現錯誤，而由軟體維護人員來檢視並更正之；適應維護是指系統環境的軟硬體改變時，應用軟體通常亦需要進行修改的維護型態；完美維護意指軟體移交使用者使用之後，可能使用者會提出增加新功能或修改原有功能的要求，為滿足這些需求所作的維護工作即稱之為完美維護；而預防維護則指軟體更

正的目的在於改進軟體未來的可維護性或信賴度[Swanson 1976, 胡正國 2000]。

2.3 軟體衡量

衡量的功用在於有助於了解一些已存在或過去已發生的況狀，然而在很多情形下，衡量者也會預測一些實際上不存在實體中的屬性，例如，假設要建立一具高信賴度的軟體系統，軟體發展需花費一段時間，而管理者想在軟體發展早期就能提供系統能達到信賴度目標的保證，為了能在系統完成前就能提供信賴度指標，就需建立一能考量出影響信賴度的因素的模式，然後在系統發展前，就能根據評估者的了解，以一些可以當下衡量的屬性來預測出可能的信賴度或其他屬性[Fenton 1997]。

2.3.1 軟體評估指標

理論上來說，利用評估指標，使得我們得以預測一些軟體外在屬性的值，例如可維護性(Maintainability)、可信賴度(Reliability)、可用性(Usability)及可攜性(Portability)，這些外部屬性通常是管理者認定軟體優劣的關鍵因素。但外在屬性是在軟體開始啟用之後始能發現，而軟體的內部屬性則是可以直接從軟體本身來加以測量的軟體屬性。外在屬性通常無法直接加以測量，所以我們必須假設一些可以測量的內部屬性和我們想要了解的外部屬性間有所關係，如圖 2-1 所示。

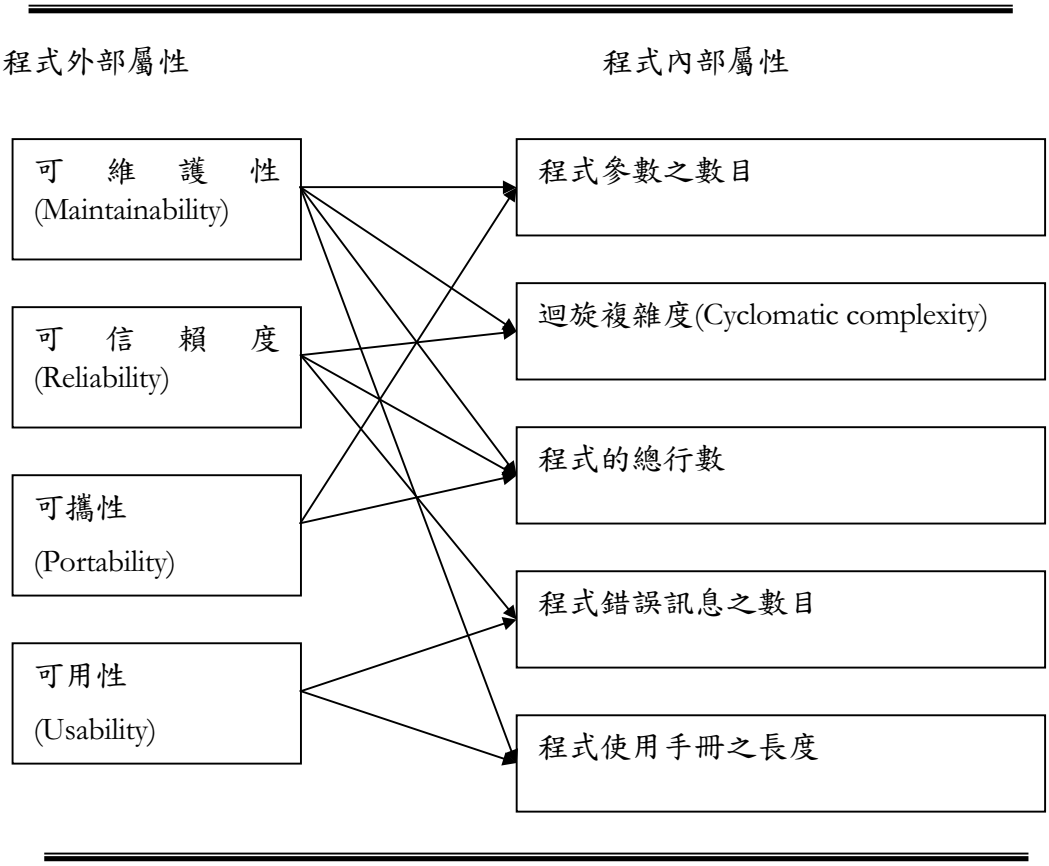


圖 2-1 程式內部屬性與外部軟體屬性間的關係圖

圖 2-1 顯示一些可能有意義之外在品質屬性，以及一些可測量而且可能與外部屬性有關之內部屬性。而程式之內部屬性要能有效作為預測軟體外部屬性的話則這些內部屬性必須能精確地測量，而且我們所能測量的屬性與外部屬性間的關係必須存在，而瞭解並且驗證過此關係後，便能夠利用方程式或是模型來加以表示[留忠賢 1997]。

2.3.2 衡量的等級類型(scale types)

衡量等級可被區分為五個主要的類型，依序分別為名義上的(nominal)等級、順序的(ordinal)等級、區間的(interval)等級、比例的(ratio)等級，以及絕對的(absolute)等級[Fenton 1997]。

其中，名義上的等級是定義出種類後，再將每一實體依其屬性的值，歸類至特定的種類中，主要的特性為：僅能區分出不同類別，但各類別中並沒有順序的關係。順序的等級是由名義上的等級中再增加順序等級上的資訊，主要的特性為：已注意到屬性間順序的關係、任何可維持順序關係的映對(mapping)是可接受的、加減或其他運算子在這衡量等級中是無意義的。區間的等級所能提供的資訊量又高於前二者，它能使我們理解從其中的一類別跳至另一類別間的大小關係為何，主要特性為：可保有順序性、可保有差異性，但無比率性、加減運算子是有效，然而乘除運算子則無效。比例的等級能提供比率上的比較資訊，其主要特性：衡量映對上可保有順序、區間大小及實體的比率性、所有的運算符號都是具有意義。絕對的等級是限制條件最嚴謹的，主要特性為實際上去計算實體集合中的元素數目值、實際的計算是唯一一種可能出現的映對類型、所有的算術分析都是有意義的。

表 2-1 衡量的類型

等級類型	轉變關係性	例子
名義上的(Nominal)	M 至 M'間有 1-1 的映對關係	標籤、實體分類
順序的(Ordinal)	M 至 M'間具有單調遞增函數的映對關係，即 $M(x) \geqq M(y)$ 意味著 $M'(x) \geqq M'(y)$	個人偏好、硬度、空氣品質、智力測驗(原始分數)
區間的(Interval)	$M' = aM + b$ $(a>0)$	相對時間、智力測驗(標準化分數)
比例的(Ratio)	$M' = aM$ $(a>0)$	時間區間、長度
絕對的(Absolute)	$M' = M$	實體的實際計算

這五大類型在定義上的限制性以後者高於前者，且有其各自的轉變(transformation)關係，如表 2-1 所示。其中映對(mapping)的目的在於能處理以數字表示的資料並且使用這結果來推斷出以觀察為依據的屬性的結論。舉例而言，堆疊基馬可夫模式計算出的複雜度值歸類為具順序性這類型中，屬性間具有順序上遞增的關係，但卻不具有加減乘除、比例性或其他運算子計算上的意義，也就是說數值間的僅具有大小的比較關係，然而不能比較彼此間是否具有相同區間或比例、倍數的關係性。

2.4 軟體產品評估與軟體過程評估

依軟體評估進行時著重觀點的不同，可將軟體評估分類為軟體產品評估(software product)以及軟體過程評估(software process)二類，所謂軟體產品評估是評估諸如軟體程式碼、設計文件、使用手冊.....等等文件之軟體產品，例如程式大小之評估即屬於軟體產品評估之類，而軟體過程評估即是衡量整個軟體之發展過程，假若一軟體系統的設計結果是深受軟體如何被設計而成所影響，而考慮或衡量這類評估屬性的方式，即屬於軟體過程評估[Li 2000]。軟體產品評估與軟體過程評估兩者間的共同性，在於都是為了找出軟體衡量方式中，那些真正顯著具有影響性的因素；而二者間的差異則在於評估方式是針對軟體產品本身，亦或針對軟體開發過程的不同觀點。

產品評估模式的衡量在程式碼完成時，就可以根據軟體程式碼及其相關文件，來進行評估工作；然而過程評估模式的衡量則需要等到軟體系統交付給顧客之前，方能得到完整的資料以進行評估工作。基於產品評估模式可在軟體發展的

較早期即獲得所需的資料，而且軟體程式碼是所有系統的開發過程中必定會產生的產出物，具有可以在不同系統間作比較的特性，所以本研究主要提出的模式，將依據軟體產品評估這類評估準則，以建立出軟體評估的模式。

2.5 程式的大小

不同的程式語言有其不同的特性，諸如程式語法或指令上的差異，但基本上仍會存在著某些共同的特性，例如每一程式均可以計算出它所包含的行數，因此，學者們提出利用計算程式所包含的行數[Conte 1986]，或計算程式中所包含的運算元及運算子的個數[Halstead 1979]。

2.5.1 行數計算法(Line of code)

在以計算程式碼行數評估方法中，有一類學者的定義方式，是將程式碼的所有行數均視為有效行數，在這定義下程式的抬頭(headers)、宣告(declarations)以及所有已執行(executable)或未執行(non-executable)之行數均需加以計算統計[Conte 1986]，以圖 2-2 中的程式 PG1 為例，其程式之總行數為 11，而另一類學者的定義則是認為程式中的命令行以及空白行，不應加以計算統計[Gray 1987]，這方式的定義是較為廣泛接受，因為這定義不計算非指令行數，所以稱之為非指令程式行數計算(non-commented line of code, NCLOC)。不論如何，行數計算評估法的優點為計算方法十分簡易，但是缺點則是它在定義上不夠嚴謹，亦即許多程式在撰寫時為了便於理解，常會加入空白位元或空白行，以及註解行(command line)，但是在評估程式效果時，這些空白行

並未貢獻出等量的效果，是否應列入衡量仍有待商榷，也因此許多學者致力於此一領域，針對程式空白行及指令行加以定義出評估準則[Lubinsky 1990, Fenton 1997]。

```
PG1(input, output)；
VAR
   a, b, c, d, m：integer；
BEGIN
   readln(a, b, c, d, )；
   a：=a + a；
   b：=b + b；
   c：=c * d；
   m：=a + b - c；
   writeln(m)
END.
```

圖 2-2 Pascal 程式範例(一)

在計算程式所含行數多寡這類方法中，除了定義行數方式的差異外，還普遍存在另一個問題，即在考慮程式複雜度之際，視程式中的每一行均有著相同的權重，然而，在實際的程式裡，程式中各行數不儘然複雜程度相同，亦即某些行數可能會含有較高的複雜度，所以程式間真實的複雜度可能無法加以比較，以圖 2-2 與圖 2-3 中的程式 PG1 與程式 PG2 來比較，二個程式之總行數同樣為 11，這是因為行數計算法只考慮程式長度之複雜度，卻未考慮程式之控制結構複雜度，所以無法比較出相同長度，但不同控制結構的程式間的複雜度，而這情形之解決辦法即給予不同權數，再加以衡量[Conte 1986]。

```
PG2(input, output)；
VAR
   a, b, c, d, m：integer；
BEGIN
   readln(a, b, c, d)；

   IF a＞b THEN
      m：=a+ b
   ELSE
      m：=b + c + d；
   writeln(m)
END.
```

圖 2-3 Pascal 程式範例(二)

2.5.2 軟體科學法(Software Science)

程式是由許多有限數量之代符(token)的集合元素所構成，所以將程式代符屬性分為運算子(operator)以及運算元(operand)二類，其中，定義 n_1、n_2、N_1、N_2，以作為複雜度評估的基礎，如表 2-2 所示。

表 2-2 軟體科學法之複雜度衡量元素定義表

符號	符號所含意義
n_1	程式中，不同的運算子的個數
n_2	程式中，不同的運算元的個數
N_1	程式中，所有運算子出現的總頻率
N_2	程式中，所有運算元出現的總頻率

由此，可以衍生出測量軟體複雜度的公式，如表 2-3 所示，其中，*n* 代表著不同運算子及運算元的加總；*N* 代表著程式中所有代符發生的總數。所以軟體科學法的複雜度衡量中，可以計算程式的程式字彙(Program vocabulary)、程式長度(Program length)、量值(Volume)、困難度(Difficulty)、努力(Effort) .....等等的複雜度評估值，其中量值以 $V=Nlog_2n$ 來表示，可以衡量出要完整表示程式時，所需的最小位元數目[Halstead1977；1979]。

表 2-3 軟體科學法之複雜度衡量表(一)

衡量值	符號	公式
Program vocabulary	*n*	$n=n_1+n_2$
Program length	*N*	$N=N_1+N_2$
Volume	*V*	$V= Nlog_2n$
Difficulty	*D*	$D=n_1/2*（N_2/n_2）$
Effort	*E*	*E=D*V*

表 2-4 軟體科學法之複雜度衡量表(二)

衡量值	符號	程式 PG1
Program vocabulary	*n*	*n*=11+5=16
Program length	*N*	*N*=23+18=41
Volume	*V*	$V= 41\log_2 16=164$
Difficulty	*D*	*D*=11/2*（18/5）=19.8
Effort	*E*	*E*=11/2*（18/5）*164=3247

以圖 2-2 中的程式 PG1 為例，可計算出其程式長度值、量值、困難度以及效果值，如表 2-4 所示。因此利用軟體科學法可考慮不同程式之程式長度值、量值、困難度及效果值，並進而能比較出不同程式間複雜度高低的情形。

然而在軟體科學法的衡量方式中，視程式的所有元素具有相同的複雜程度，但是，在實際程式中，不同的元素可能存在著不同的複雜度，所以另有學者提出利用權重作計算衡量的方式，認為在了解程式內容後，可以給予不同權數來作程式複雜度之計算[Berns 1984]。

```
PG3(input, output)；
VAR
   a, b, c, d, m：integer；
BEGIN
   readln(a, b, c, d)；

   IF a＞b THEN
      IF b＞c THEN
         m：=a+ b
      ELSE
         m：=b+ c
   ELSE
      m：=b + c + d；
   writeln(m)
END.
```

圖 2-4 Pascal 程式範例(三)

表 2-5 軟體科學法之複雜度衡量表(三)

符號	程式 PG1	程式 PG3
n	*n*=11+5=16	*n*=11+5=16
N	*N*=23+18=41	*N*=22+19=41
V	*V*= 41log$_2$16=164	*V*= 41log$_2$16=164
D	*D*=11/2*（18/5）=19.8	*D*=11/2*（19/5）=20.9
E	*E*=19.8*164=3247	*E*=20.9*164=3427

在這種以軟體科學方法為基礎來作為評估的方法中，其優點在於計算方式簡單，但缺點則是未能考慮程式中的控制流程，以圖 2-2 中的程式 PG1 及圖 2-4 中的程式 PG3 比較之，因為軟體科學評估法只考慮程式長度，而未考慮程式巢狀結構，所以無法計算比較出這二個程式間的複雜度高低，如表 2-5 所示。其中，PG1 有 11 個不同的運算子，分別是“BEGIN END”、“readln”、“()”、“,”、“;”、“:=”、“+”、“*”、“-”、“writeln”、“.”。而這些運算子出現的總次數為 23 次，所以 n_1=11，N_1=23，而程式中不同的運算元個數有 5 個，其出現的總次數為 18 次，n_2=5，N_2=18。而 PG3 也有 11 個不同的運算子，分別是“BEGIN”、“readln”、“( )”、“, ”、“;”、“:= ”、“+”、“IF THEN ELSE”、“>”、“writeln”、“.”。而這些運算子出現的總次數為 22 次，所以 n_1=11，N_1=22，而程式中不同的運算元個數有 5 個，其出現的總次數為 19 次，n_2=5，N_2=19。

2.6 程式的控制流程(Control Flow)

有些學者認為程式複雜度的高低主要是與程式的決策結構(decision structure)有關，在這類評估法中大多利用向量圖形來表示程式的控制結構，並進而計算程式控制結構中的複雜度[McCabe 1976, Ramamurthy 1988]。

2.6.1 迴旋複雜度(Cyclomatic Complexity)

McCabe 提出一個使用向量圖形理論技術的程式複雜度測量法，他的理論主張程式複雜度並不和程式大小有關，而是和程式的決策結構(decision structure)有關，並認為具有高度迴圈複雜度的元件會比低複雜度的元件需要更多的維護。

在這評估法中，向量圖形的意義在於它可以涵括程式結構，利用軟體程式碼之潛在路徑作為分析的一種評估法，迴旋複雜度評估法(Cyclomatic complexity, CC)即為一著名的控制流程衡量法[McCabe 1976, Wallace 1996]，迴旋複雜度評估法並已經擴展到能廣泛涵括一個系統的設計和架構的複雜性[McCabe 1989] ，此外，迴旋複雜度評估法亦可運用於不同軟體間之比較[McCabe 1994]。迴旋複雜度評估法其表示公式為：

$$CC = V(G) = e - n + 2p$$

$V(G)$代表程式之迴旋複雜度值；

e 代表圖形邊的數目；

n 是圖形結點的數目；

p 是指程式所包含模組之數目

換言之，假若決策點定義為圖形中結點的外擴邊數都是2 的情形，則迴旋複雜度可以利用圖形決策點數目(DE)加一作計算，也就是 *V(G) = DE + 1*，以圖 2-2 之程式為例，可計算出程式 PG1 的複雜度為 2，而圖 2-4 之程式 PG3 的複雜度為 4，故利用迴旋複雜度之衡量可比較出二程式的複雜度，以含有巢狀結構的程式 PG3 之複雜度高於程式 PG1。

V(PG1)=3-3+2=2

V(PG3)=13-11+2=4

利用迴旋複雜度評估法，優點為計算簡單也廣為人知，但它卻忽略了當程式含有不同深度之巢狀結構情形下，其複雜度比較，如圖 2-5 所示，三個向量圖形的迴旋複雜度均為4，是故無法比較程式中不同巢狀深度的差異性，亦即實際程式中，巢狀條件結構比起非巢狀條件結構 的敘述更難以瞭解。此外，迴旋複雜度評估法亦未考慮到除了控制流程以外，程式中其他因素的複雜度，例如程式大小的複雜度，因此其他學者相繼提出理論修正[Harrison 1992, Li 1987]。

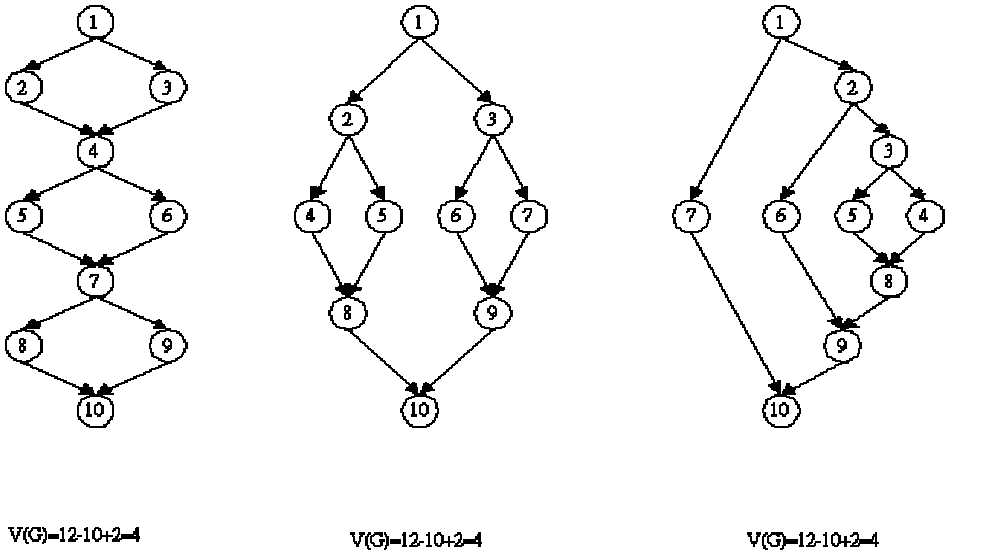


圖 2-5 迴旋複雜度的圖例

2.6.2 加權值衡量法(Weighted Measures)

加權值衡量法是從軟體科學法及迴旋複雜度評估法中所衍生而來的評估法，它的計算方式是讓程式中的運算子與運算元分別加入一個權重值 [Ramamurthy 1988] 。

在權重值評估法中，利用權重以計算出程式的控制流程複雜度，以控制流程圖來看，假若程式中無巢狀結構，則巢狀深度為 0，若在第 L 層深度出現 DO WHILE, IF THEN 或 DO UNTIL 的語法，則深度加深為 L+1。其計算公式為：

$$Nw_1 = \sum x \ in$$

$$Nw_2 = \sum x \ in$$

$$Nw = Nw_1 + Nw_2$$

$Vw=Nw*log(n_1+n_2)$

T 是模組內的所有運算子

R 是模組內的所有運算元

$\partial(x)$ 為所有 x 在 T 聯集 R 中的值域

$\int(x)$ 為所有 x 在 T 聯集 R 中巢狀深度

Nw_1 是所有運算子的加權計算

Nw_2 是所有運算元的加權計算

Nw 是程式加權長度

Vw 是程式加權量值

利用加權值衡量法來比較圖 2-2 中程式 PG1 和圖 2-4 中程式 PG3 的複雜度，程式 PG1 不含巢狀結構所以權重為 0，而程式 PG3 的深度最高為 2 層，以權重值評估法之計算可區分出有巢狀結構的程式 PG3，其複雜度會高於程式 PG1，如表 2-6 所示。其中，PG1 沒有包含決策點，所以評估值與未用權重值的評估法一樣，而 PG3 中，第一個巢狀的“IF THEN ELSE”的巢狀深度為 1，所以其迴旋複雜度為 2，其權值為 1，第二個巢狀的“IF THEN ELSE”的巢狀深度為 2，所以其迴旋複雜度為 3，其權值為 2；此外，在第一個巢狀“IF THEN ELSE”內的“＞”，其權值為 1，而在第二個巢狀“IF THEN ELSE”內的“＞”，其權值則為 2；因為程式中的其他運算子的權重值為 0，所以 $Nw_1 = N_1+6=28$。而程式 2 中的運算元，在二個“IF THEN ELSE”巢狀中的 IF 子句中，“a”有一權值為 1，“b” 有一權值為 1，有一權值為 2，“c” 有一權值為 2，所以 $Nw_2 = N_2+6=25$，則 $Nw = 28+25 = 53$。

表 2-6 利用權重值評估法計算程式複雜度

程式名稱	程式 PG1	程式 PG3
Nw	41	53
Vw	41**log*(11+5=164)	53**log*(11+5)=212
Ew	164*[(11*18)/(2*5)]=3247	212*[(11*25)/(2*5)]=5830

2.7 程式的資料流程(Data Flow)

視程式中的資料傳遞關係是影響程式複雜度的主要因素，學者提出的評估方式是依據程式中所涵括的靜態活變數(static live variable)作為估計的準則[Dunsmore 1979]，而Henry也提出根據程式的大小以及其扇入(fan-in)與扇出個數(fan-out)以計算程式複雜度[Henry 1981]，Bieman則考慮程式中資料間的關聯性[Bieman 1984; 1985]。這類依據資料流程的評估法，考慮了程式實際的動態執行狀況。

Constantine與Yourdon提出如何測量在結構圖中設計元件的扇入與扇出個數的方法。扇入個數，意謂結構圖中進入元件方塊的線之數目，本質上這就是呼叫此元件的其它元件數目。扇出個數，則意謂結構圖中離開元件方塊的線之數目，即是表示所呼叫的元件數目，如圖2-6所示。高扇出個數可說明此呼叫元件的複雜度可能很高，因為控制邏輯的複雜度需要與所屬的元件來協調。

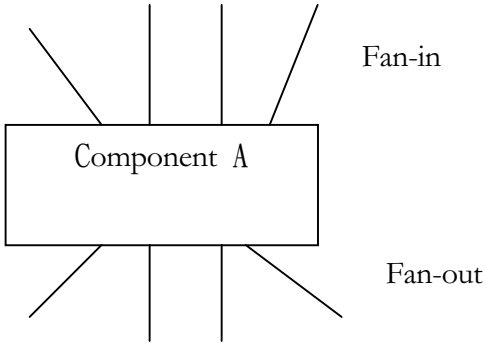


圖 2-6 結構化的扇入與扇出圖

Henry 與 Kafura 則提出資訊化的扇入、扇出格式。將程式中的資料流程定義出二種不同的基本型態，一為區間內的資料流程(local data flow)、另一則為全面之資料流程(global data flow)。

所謂區間內的資料流程，意指資料流程存在於模組呼叫另一模組，亦或模組回饋執行結果至另一模組時，以圖 2-7 所示，可解釋一區間內模組 A 之資訊流程，以參數 *x* 來呼叫模組 B，而這三者間的關係可以<A, *x*, B>來表示；而圖 2-8，則表示資訊流由模組 B，將模組 B 的執行結果傳回模組 A，可以<B, B, A>來表示。

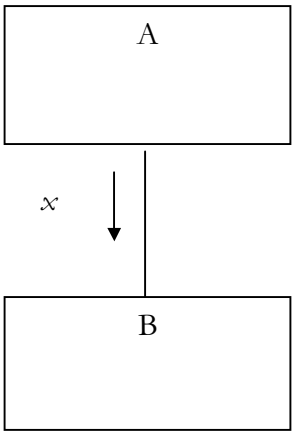


圖 2-7 區間資料流程之呼叫參數關係圖

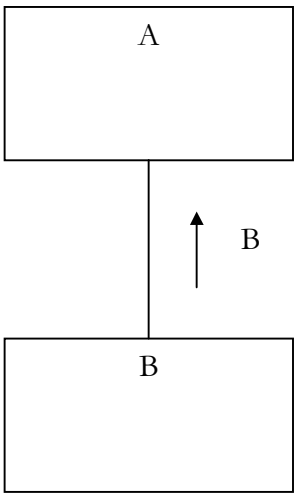


圖 2-8 區間資料流程之傳回結果值關係圖

而全面性資料流程，意謂一模組寫入資料至一全面性資料結構中，而其他模組亦可從這資料結構中讀取資料，如圖 2-9 所示，這樣的資料流程關係，可以<A, DS, B>來表示。

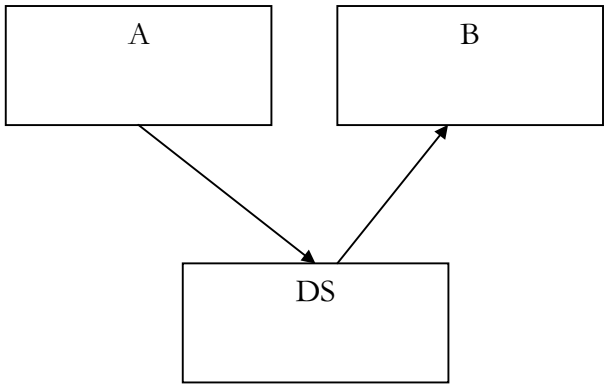


圖 2-9 全面性資料流程關係圖

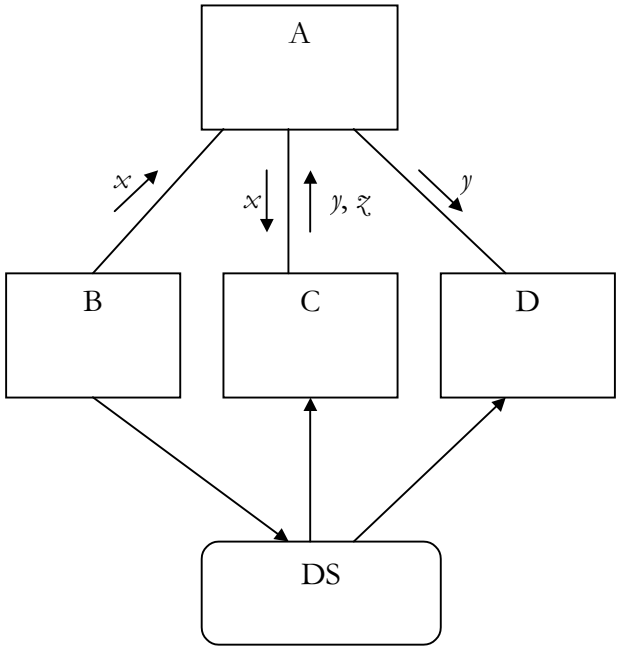


圖 2-10 資料流程關係圖分析

此外，再以一較複雜之範例來說明並計算程式中的資料流程複雜度，如圖 2-10 所示。模組 A 有三個子模組，其中區間內模組的關係為：

<A, *x*, C>

<A, *y*, D>

<B, *x*, A>

<C, *y*, A>以及

<C, *z*, A>

而全面性模組資料流程的關係則為：

<B, DS, C>以及

<B, DS, D>

由圖 2-10 所衍生出的扇入、扇出關係可分別加以計算，扇入意謂以指定模組為終端的資料流程數目，以模組 D 為例，扇入值為 2，其之資料流程分別為<A, *y*, D>，<B, DS, D>；而扇出意謂從指定模組所產生之資料流程數目，以模組 B 為例，扇出值為 3，其之資料流程分別為<B, *x*, A>，<B, DS, C>，以及<B, DS, D>。並將圖 2-10 之資料流程複雜度以公式 IF_4，加以計算，如表 2-7 所示，其中之長度可用任一類計算程式大小的評估法來計算，例如程式碼的行數。

$$IF4 = \sum_{1}^{n} (FI * FO)^2$$

$$\text{Complexity} = \text{Length} * (FI * FO)^2$$

表 2-7 資料流程複雜度評估表

模組	*Fan-in (FI)*	*Fan-out (FO)*	*FI* FO*	$(FI*FO)^2$
A	3	2	6	36
B	0	3	0	0
C	2	2	4	16
D	2	0	0	0
總和				52

由此可知，資料流程複雜度考慮了從一個模組傳到另外一個模組的參數個數，以及在模組間共享的動態資料結構。

Henry 並驗證這類測量法可確認出潛在不完美的系統程式，指出具測量指標高量值的模組，會造成系統問題的不對稱數目並且會導致很高的維護成本。但是資料流程複雜度的計算上則顯得太過於複雜而不易計算[Henry 1981, Shepperd 1995, 留忠賢 1997]。

2.8 古典資訊理論(Classical Information Theory)

古典資訊理論是另一類被運用於複雜度評估的方法。Shannon 首先將資訊理論運用於表示符號與訊息(message)之間的關係[Shannon 1948]，每個訊息所涵的資訊內容可定義為：

$$I(s_i) = -\log p(s_i) \qquad \text{(bits)}$$

$$H(S) = -\sum_{i=1}^{n} p(s_i) \log p(s_i) \qquad \text{(bits)}$$

其中 $p(s)$是訊息 s 發生的機率，I 代表訊息所涵蓋的資訊內容，H 則表示全部訊息所平均涵蓋的資訊內容，亦即計算整個訊息的熵(entropy)。

根據資訊理論的觀念，視資訊系統的開發者為一資源產生器；而軟體系統的開發則被視為一資源產生器所產生出的一連串具特殊意義的事物。這個過程可被視為一類似馬可夫過程(Markov process)，可由實驗的樣本庫中去歸納出事件可能發生的機率，而根據前述假設，則可計算出程式所涵的資訊內容，並利用資訊內容的值來表示此一程式的複雜度。Berlonger 也提出了另一種資訊內容的定義：$I=f_i \log(1/p_i)$，其中 f_i 是符號 i 在程式中出現的次數，p_i 是符號 i 出現的機率[Berlonger 1980]，陸續有許多研究提出其他以資訊理論觀念

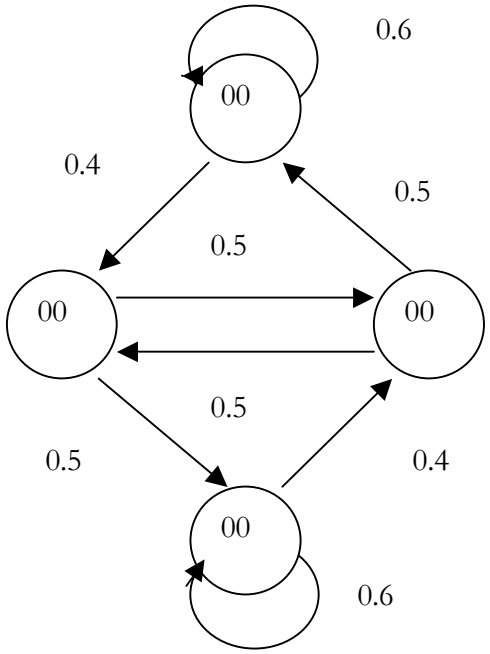
為基礎的複雜度評估方法[Mohanty 1981, Samadzadeh 1988, Edwards 1988, Patrick 1982]。

因為早期運用資訊理論基礎衡量程式複雜度的方式裡，常視資訊源為一無記憶裝置的資訊源(memoryless source)，但是在實際程式語言中，符號間彼此是不可能完全獨立的，於是學者利用有記憶裝置的資訊源，例如 n 階的馬可夫資訊源(n-th Markov source)，並運用這種有記憶裝置的馬可夫模式來計算程式的複雜度。所以 n 階馬可夫模式的資訊內容及其熵可定義為：

$$I(s_i|s_{ji,\ldots,}s_{mi,}) = -\log p(s_i|s_{ji,\ldots,}s_{mi}) \quad \text{(bits)}$$

$$H(S|s_{ji,\ldots,}s_{mi}) = -\sum_{S} p(s_i|s_{ji,\ldots,}s_{mi}) I(s_i|s_{ji,\ldots,}s_{mi,}) \quad \text{(bits)}$$

舉一狀態圖來表示出二階的馬可夫資訊源的例子，其中資訊源所發射的符號僅由 { 0,1 } 所構成，而其發生的條件機率則如圖 2-11 所示：



$p(0|00)=p(1|11)=0.6$

$p(1|00)=p(0|11)=0.4$

$p(1|10)=p(1|01)=p(0|01)=p(0|10)=0.5$

圖 2-11 二階馬可夫資訊源圖例

雖然利用 *n* 階的馬可夫模式，可以解決先前視程式符號相互間無關係的這個缺點，但是它卻存在著另一個問題，因為 n 階的馬可夫資訊源的記憶裝置是一佇列的形式，所以它無法充分表現出實際上程式語言的某些特牲，例如「if」的巢狀控制流程中，「end if」結束指令不可能出現在其相對應的「if」指令前面。於是 Edward 提出了一種以堆疊為記憶裝置的馬可夫模式來作為計算程式複雜度的評估準則[Edwards1990; 1991]。

第三章

堆疊基馬可夫模式(Stack-based Markov Model)

本論文研究將建構之模式是以堆疊基的馬可夫模式(Stack-based Markov Model, SBM)為基礎，所以本章中將先就堆疊基的馬可夫模式的觀念及其相關應用作闡述。

資訊理論基礎下，程式可以被視為一連串符號所組成，當這些成功發射的符號間的資訊源為無記憶裝置時，這些發射出的符號間存在著獨立的關係，但是我們知道，在實際的程式中，每個符號並非獨立存在，它存在某些關係。所以 Edwards 提出了一種反應程式符號關係的模式－堆疊基馬可夫模式[Edwards 1990,1991]，在這模式下，軟體系統開發者可被視為是一個資訊發射源，而開發出來的軟體系統就可視為是資訊源所發射出的一連串具有意義的事件或符號，如圖 3-1 所示，這些符號或程式的資訊內容，以及整個資訊源的熵(entropy)，可被用來反應這個軟體系統的複雜度。這種堆疊基的馬可夫模式已被應用在一般及函數的程式語言中，可表示程式中巢狀控制結構的複雜度[Yang 1992]，以及整個程式的複雜度[Wang 1994]。

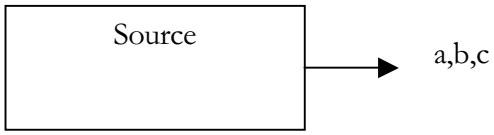


圖 3-1 馬可夫資訊源

在堆疊基馬可夫模式之中，每一個新的符號的產生是決定於其堆疊頂端的情形而定，所以這符號出現的機率也就依

這個堆疊頂端的情形而決定，而且會產生一個堆疊記憶的動作。

堆疊中的基礎動作包含把符號壓入堆疊上，以符號 push() 表示這壓入堆疊的動作、把符號彈出堆疊，以符號 pop()表示這彈出堆疊的動作，以及表示堆疊頂端的值，以 vtop()表示。條件機率 p (b| a) 可以用來表示這種堆疊情形，意指在 a 發生的的情況下，再產生 b 的機率。

最簡單的馬可夫模式是平衡括號所產生的符號，假設元素集合為{ (,) , S , $ }，則程式之控制流程複雜度可以一連串相同或不同的平衡括號來組合。表 3-1 所示為馬可夫平衡括號。在此，

當符號(發生時，會產生一堆疊動作 push()；

當符號）發生時，會產生一堆疊動作 pop()；

符號 S 表示其他符號的發生；

符號$ 表示這一連串符號的結束；

資訊內容（*I*）及程式之熵（*H*）計算式分別為：

$$I = -\sum f(x_i \mid c_j) \log\ p(x_i \mid c_j);$$

$$H = -\sum_j p(c_j) \sum_j f(x_i \mid c_j) \log\ p(x_i \mid c_j)$$

表 3-1 馬可夫平衡括號

Stack Top	Next Symbol/Stack Action			
	(/push(	) /pop(	S /none	$ /none
empty	p	0	u	1-p-u
(	q	1-q-v	v	0

舉例說明，假定一馬可夫平衡括號如表 3-2 所示，則可利用該表計算出各種情形下，符號所含的資訊內容之值。

表 3-2 馬可夫平衡括號之例子

Stack Top	Next Symbol/Stack Action			
	(/push(	) /pop(	S /none	$ /none
empty	1/2	0	0	1/2
(	1/3	2/3	0	0

I（()()()$）＝4$log$3=6.3399

I（()(())$）＝2$log$2+4$log$3=8.3399

而由馬可夫平衡括號中，可以導出下列方程式：

$$P(0,0)=1;$$
$$P(t,s)=0;\ \text{for}\ s>t$$
$$P(t,0)=(1-p-u)P(t-1,0)+(1-q-v)P(t-1,1)+uP(t-1,0);\ \text{for}\ t>0$$
$$\Rightarrow P(t,0)=(1-p)P(t-1,0)+(1-q-v)P(t-1,1);\ \text{for}\ t>0$$
$$P(t,1)=pP(t-1,0)+(1-q-v)P(t-1,2)+vP(t-1,1);\ \text{for}\ t>0$$
$$P(t,s)=qP(t-1,s-1)+(1-q-v)P(t-1,s+1)+vP(t-1,s);\ \text{for}\ 1<s\le t$$

上列之方程式中，s 代表堆疊個數；t 代表著時間。又當 t 趨近於無窮大時，如果 $\frac{q}{1-q-v}<1.0$，則 Edwards 導出下列方程式[Edwards1991]。

$$\therefore D(s)=\lim_{t\to\infty}P(t,s);\quad \text{if}\ \frac{q}{1-q-v}<1.0$$
$$D(0)=\frac{1-2q-v}{1-2q-v+p}$$
$$D(1)=D(0)\frac{p}{1-q-v}$$
$$D(2)=D(1)\frac{q}{1-q-v}=D(0)\frac{pq}{(1-q-v)^2}$$
$$D(s)=D(s-1)\frac{q}{1-q-v},\ for\quad s>1$$

堆疊基馬可夫模式，被驗證已考慮了程式中的控制流程[Edwards 1991, Wang1991;1994, Yang1991;1992]，學者更已提出堆疊基的馬可夫資訊源模式應用在表示 C 程式語言中的表示式，以驗證運算元及運算子間的關係。這個模式不但反映了表示式中量的複雜，也包含了表示式中巢狀控制結構的複雜度[Wang 1997]。此外，堆疊基馬可夫模式亦已考慮到程

式中的資料流程[Kim 1991]，而堆疊基物件導向的開發模式亦已有學者提出[Holland 1993]。利用堆疊基馬可夫模式作為軟體評估的方式，因為能同時考慮到程式的控制流程、資料流程、程式的大小.....等特性，所以使用這模式來分析不同資訊系統或用以分析不同程式語言的特性，可以清楚的區分出其複雜度的高低，利用堆疊基馬可夫模式，管理者想要的複雜度評估資訊，及研究者想要的軟體特性資訊，都可以從實際的程式中取得並加以衡量。

第四章

修正的堆疊基馬可夫模式

4.1 建構評估模式的過程

本文提出修正的堆疊基馬可夫模式，利用更簡單之模式來衡量軟體複雜度。

一些研究報告中指出[Shepperd 1995]，建構軟體評估模式之際，需針對軟體衡量以及模式建構架構的各階段有著深入的研究，而這軟體衡量建構的架構過程可被劃分為規劃(Planning)、收集(Collection)、分析(Analysis)，以及驗證(Validation)四階段，如圖 4-1 所示，依照這軟體衡量週期來執行評估，可完成軟體衡量活動，茲將軟體衡量週期之各階段目的概述如下：

1. 規劃(Planning)：包含定義軟體衡量所需涵蓋的活動、設定軟體評估之需求目標，此外，收集資料的工作前置準備、軟體評估工具的發展，亦應包含在規劃階段。
2. 收集(Collection)：為軟體衡量週期的第二階段，需依附於實際的衡量過程之中，且應含有控制機制，以確保程序的正確性。
3. 分析(Analysis)：這階段的活動在於將收集之資料加以量化衡量，並可利用統計分析等工具來協助了解建構的雛型，此外，在這些評估衡量結果應有適當的回饋機制，來確保建構模型能有效評估、比較軟體系統。

4. 驗證(Validation)：對學者或衡量者而言，在進行評估過程中，需持續關注的焦點為評估的正確性與否，所以驗證工作在衡量過程中的各階段中，皆應被視為重要的工作。

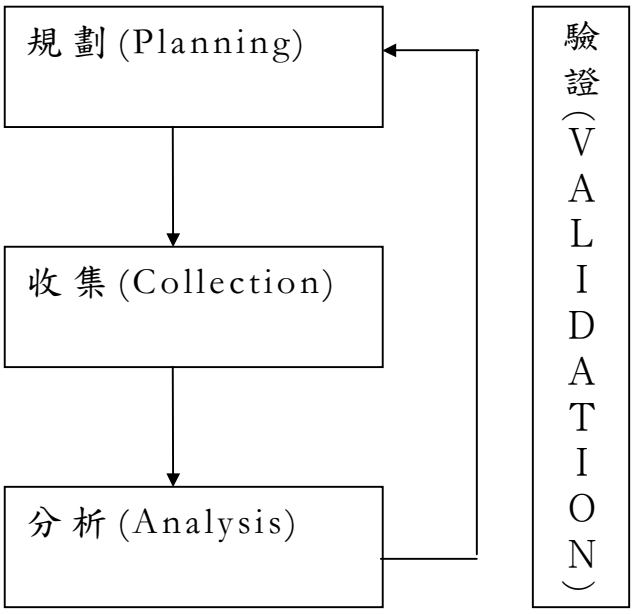


圖 4-1 軟體衡量週期的階段流程圖

而 Jacquet 則提出一量化之軟體評估方法，定義出四個基本步驟，分別為衡量方法之設計、衡量規則之應用、衡量結果之分析、衡量結果之利用 [Jacquet 1997] 。

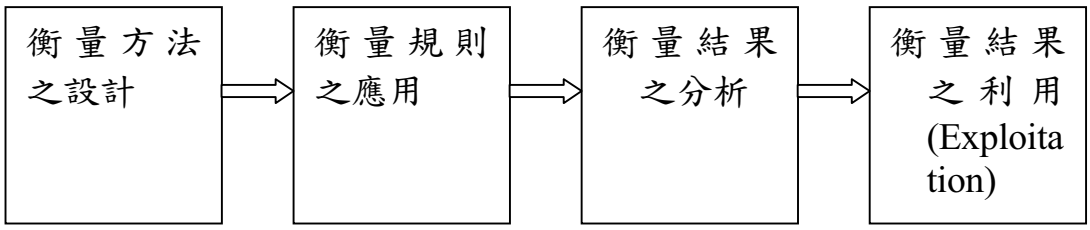


圖 4-2 衡量過程步驟圖

所以本論文建構評估模式的過程，依據下列幾個步驟進行，分別為：規劃衡量目標、收集程式資料、建置評估模式、將資料運用本模式計算其程式複雜度、驗證建構之雛型模式、將資料分析結果回饋建置模式，以作為修正模式之依據或建構出完整評估模式，如圖 4-3 所示。

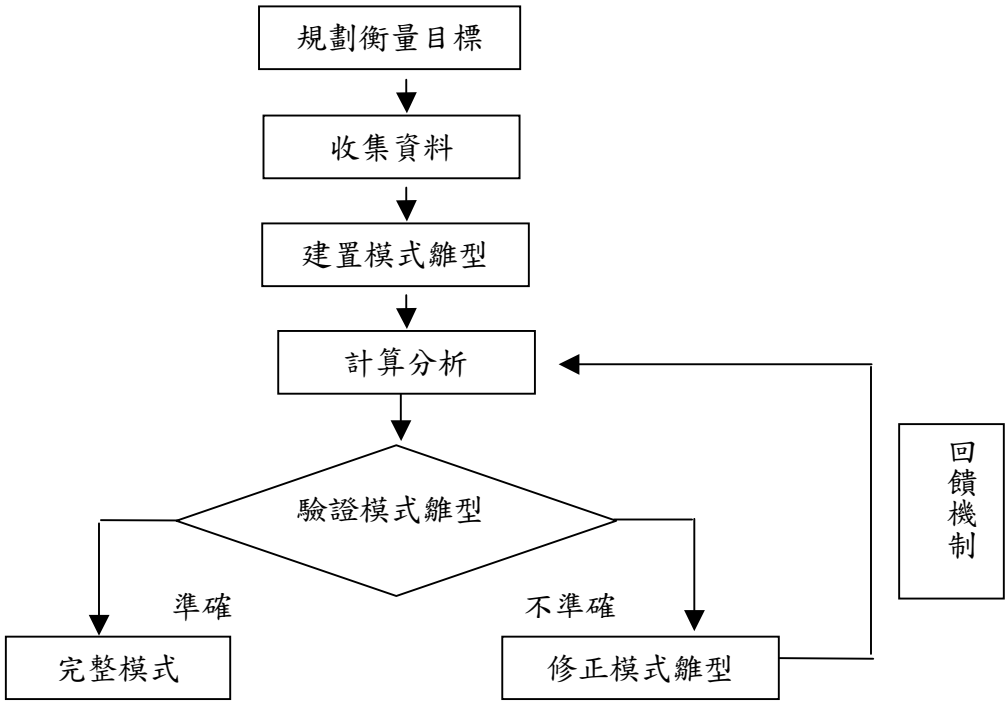


圖 4-3 本論文建構評估模式的架構

在建置模式中，期以一簡單模式來作為程式複雜度評估的工具，而建置過程中，如發現模式的缺失，則將修正建置的模式雛型，期使建置完成後的模式能最適切表達實際軟體之複雜度。

4.2 初始評估模式的建構

在初始評估模式的建置中，最簡單的馬可夫模式是平衡括號所產生的符號，假設元素集合為{ (,) , $ }，因此程式碼可被轉換為左括號以及右括號所組合之集合{ (,) }，原先馬可夫模式中的元素{ S }，可被簡化省略，因為程式中的每個運算元或運算子也都能以左右括號來加以表示，而且每個運算元或運算子都會影響堆疊層數，是故，{ S }元素在簡單堆疊基馬可夫模式中並不存在。在這假定下，初始評估模式可同時完整地考慮程式的控制流程複雜度、程式表示式中量的複雜度以及表示式的控制流程複雜度。假設修正的馬可夫平衡括號則如表 4-1 所示。在此，

當符號(發生時，會產生一堆疊動作 push()；

當符號）發生時，會產生一堆疊動作 pop()；

符號$ 表示這一連串符號的結束；

表 4-1 初始模式的馬可夫平衡括號(一)

Stack Top	Next Symbol/Stack Action		
	(/push(	) /pop(	$ /none
empty	p	0	1-p
(	q	1-q	0

而由修正之馬可夫平衡括號中，可以導出下列方程式：

$P(0,0) = 1;$

$P(t, s) = 0;$ for $s > t$

$P(t,0) = (1 - p)P(t - 1,0) + (1 - q)P(t - 1,1);$ for $t > 0$

$P(t, 1) = pP(t - 1,0) + (1 - q)P(t - 1,2);$ for $t > 0$

$P(t, s) = qP(t - 1, s - 1) + (1 - q)P(t - 1, s + 1);$ for $1 < s \le t$

而上列之方程式中，s 代表堆疊個數；t 代表著時間。又當 t 趨近於無窮大時，如果 $\frac{q}{1-q} < 1.0$，則導出下列方程式。

$$\therefore\ D(s) = \lim_{t \to \infty} P(t, s); \quad \text{if} \quad \frac{q}{1-q} < 1.0$$

$$D(0) = \frac{1 - 2q}{1 - 2q + p}$$

$$D(1) = D(0)\frac{p}{1 - q}$$

$$D(2) = D(1)\frac{q}{1 - q} = D(0)\frac{pq}{(1 - q)^2}$$

$$D(s) = D(s - 1)\frac{q}{1 - q}, \quad for \quad s > 1$$

建立初始模式後，利用簡單的平衡括號的例子計算符號複雜度的高低值，以驗證模式是否能作為準確衡量的工具，其中初始模式的馬可夫平衡括號如表 4-2，4-3 所示：

表 4-2 初始模式的馬可夫平衡括號(二)

Stack Top	Next Symbol/Stack Action		
	(/push(	) /pop(	$ /none
empty	3/4	0	1/4
(	0	3/3	0

I（ ()()()$ ）=4*log*4-3*log*3=3.2451

表 4-3 初始模式的馬可夫平衡括號(三)

Stack Top	Next Symbol/Stack Action		
	(/push(	) /pop(	$ /none
empty	2/3	0	1/3
(	1/4	3/4	0

I（ (())()$ ）=4*log*4-2*log*2=6

所以運用初始模式，就可以用簡單的模式來區分出符號間含巢狀結構的情況下，其資訊內容應該是具有較高的複雜度。但是再經由其他的實驗數據，卻發現初始模式尚無法分辨出符號中具有不同的巢狀深度的情形，再以三個簡單的平衡括號的例子來說明初始模式，雖可計算出的複雜度高低，卻仍無法分辨符號間皆含巢狀結構，但巢狀深度不同情形下的複雜度高低，其中初始模式的馬可夫平衡括號如表 4-4，4-5，4-6 所示：

表 4-4 初始模式的馬可夫平衡括號(四)

Stack Top	Next Symbol/Stack Action		
	(/push(	) /pop(	$ /none
empty	1/2	0	1/2
(	3/7	4/7	0

I（ (()()())$ ）=7*log*7-3*log*3-4*log*4+2*log*2=8.8966

表 4-5 初始模式的馬可夫平衡括號(五)

Stack Top	Next Symbol/Stack Action		
	(/push(	) /pop(	$ /none
empty	1/2	0	1/2
(	3/7	4/7	0

I（ (() (()))$ ）=7*log*7-3*log*3-4*log*4+2*log*2=8.8966

表 4-6 初始模式的馬可夫平衡括號(六)

Stack Top	Next Symbol/Stack Action		
	(/push(	) /pop(	$ /none
empty	4/5	0	1/5
(	0	4/4	0

I（ ()()()()$ ）=5*log*5-4*log*4=3.6096

由以上三個平衡括號的例子可以知道，初始模式在評估之際，雖然能考量到程式之控制流程巢狀結構，以及程式中表示式之區塊大小的複雜度。但由實驗數據卻發現，此一初始模式尚無法精準將影響程式複雜度高低的因素完全表示出來，以致於無法有效衡量比較程式的複雜度，三個平衡括號中僅能區分出無巢狀結構與含巢狀結構符號間複雜度的不同，卻無法區分出符號間含有不同巢狀深度的複雜度。

再以二個程式為例，說明初始模式無法有效比較程式複雜度的情形。以初始模式來建立平衡括號，並計算程式複雜度熵(*H*)的值，如圖 4-4、表 4-7、表 4-8 及表 4-9 所示。

```
Program 1 ()
{
    int a, b, c;
    a=1;
    a=2*a;
    c=b-a;
    if (a>b)
        c=2*a;
    else
        c=2*b;
    if (a>c) {
        a=b;
        b=c;
    }
}
```

```
Program 2 ()
{
    int a, b, c;
    a=1;
    a=2*a;
    c=b-a;
    if (a>c) {
        if (a>b)
            c=2*a;
        else
            c=2*b;
        a=b;
        b=c;
    }
}
```

圖 4-4　以 C 語言所組成的程式(一)

表 4-7 初始模式的平衡括號，以 Program1 為例

Stack Top	Next Symbol/Stack Action		
	(/push(	) /pop(	$ /none
Program1			
empty	1/2	0	1/2
(	18/37	19/37	0

表 4-8 初始模式的平衡括號，以 Program2 為例

Stack Top	Next Symbol/Stack Action		
	(/push(	) /pop(	$ /none
Program2			
empty	1/2	0	1/2
(	18/37	19/37	0

表 4-9 初始模式複雜度計算比較表

程式名稱	熵 H
Program1	0.999500045
Program2	0.999500045

由表 4-9 可以得知，利用初始模式來衡量二個程式的複雜度，可計算出相同的複雜度值，然而這二個程式間，雖然程式的行數都相同，但 Program2 卻含有巢狀結構，理論上其複雜度會高於 Program1 的結構，所以初始模式的表示方法，尚無法精準衡量程式複雜度的高低。

4.3 修正評估模式

檢討建構的初始模式，發覺原先之表示法，雖然考慮巢狀結構的複雜度，但因為控制流程或表達式中之巢狀結構的深度未加以區分，所以模式仍無法將不同深度之巢狀結構上的相異處計算區別之，因此，為了要區分不同深度之巢狀結構有著不同的複雜度，針對初始模式的缺點加以修正，將之演進為修正的堆疊基馬可夫模式，在這修正後的模式中，假設程式中不同的堆疊深度所有可能產生的情況，分別以不同堆疊深度來加以計算，並假設以{0,1,2,3,...n}來表示程式中堆疊的深度。所以衡量複雜度之際，假設產生的新符號會使巢狀深度加深的話，則堆疊記憶裝置亦會壓入另一個新的堆疊深度，此時堆疊動作的深度取 vtop()+1，並以這 vtop()+1 來作新的堆疊項端，而程式開始時堆疊頂端的深度則用 0 來表示。根據這樣的演進後，則修正的模式能更正確評估程式的複雜度，其中修正模式的平衡括號，如表 4-10 所示。

表 4-10 修正堆疊基馬可夫模式的平衡括號(一)

Stack Top	Next Symbol/Stack Action		
	(/push(vtop()+1)	) /pop()	$ /none
0	P	0	1-p
1	q_1	$1-q_1$	0
2	q_2	$1-q_2$	0
3 . . .	. . .	. . .	. . .
n	0	1	0

運用修正的堆疊基馬可夫模式，就可以分辨出符號間具有不同的巢狀深度的情形，以先前的三個簡單的平衡括號的例子來說明修正後的模式，可真正計算出符號間的複雜度高低，其中初始模式的馬可夫平衡括號如表 4-11，4-12，4-13所示：

表 4-11 修正堆疊基馬可夫模式的平衡括號(二)

Stack Top	Next Symbol/Stack Action		
	(/push(vtop()+1)	) /pop()	$ /none
0	1/2	0	1/2
1	3/4	1/4	0
2	0	3/3	0

I（ (()()())$ ）=4*log*4+2*log*2-3*log*3=5.2451

運用修正的堆疊基馬可夫模式，就可以分辨出例子中，以含較高巢狀深度的符號串列（ (()(()))$ ）的複雜度最高，符號串列（ (()()())$ ）的複雜度次之，而以巢狀深度最低的符號串列（ ()()()()$ ）的複雜度最低。所以修正後的模式可真正衡量出符號間的複雜度高低情形。

表 4-12 修正堆疊基馬可夫模式的平衡括號(三)

Stack Top	Next Symbol/Stack Action		
	(/push(vtop()+1)	) /pop()	$ /none
0	1/2	0	1/2
1	2/3	1/3	0
2	1/3	2/3	0
3	0	1/1	0

I（ (()(()))$ ）=6*log*3-2*log*2=7.5098

表 4-13 修正堆疊基馬可夫模式的平衡括號(四)

Stack Top	Next Symbol/Stack Action		
	(/push(vtop()+1)	) /pop()	$ /none
0	4/5	0	1/5
1	0	4/4	0

I（ ()()()()$ ）=5*log*5-4*log*4=3.6096

依據原始的堆疊基馬可夫模式，已考慮程式的控制流程的複雜度情形以及程式中表示式的控制流程的複雜度。但是因為程式中的表示式中，不同區塊(block size)所產生的不同組合，應該有其不同的複雜度，所以在計算修正的堆疊基馬可夫模式的複雜度時，除了考慮程式的控制流程複雜度情形，亦將程式表示式結合關係的複雜度考量在內，讓修正的模式能建構出一個簡單模式，並能將更多影響程式複雜度之因素均列入計算時之衡量因素。

第五章

實驗結果

本文將修正的堆疊基馬可夫模式，分別以不同的程式組作為測試樣本，並用實際企業界中，真正運作中的管理資訊系統來驗證此一新的評估模式的有效性。

5.1 模式的正確性

舉三個程式為例，分別計算模式的平衡括號機率及複雜度熵 H 的值，如圖 5-1、表 5-1、表 5-2、表 5-3 及表 5-4 所示。

表 5-1 修正堆疊基馬可夫平衡括號以 Program1 為例

Stack Top	Next Symbol/Stack Action		
	(/push(vtop+1)	) /pop()	$ /none
Program1			
0	1/2	0	1/2
1	5/6	1/6	0
2	7/12	5/12	0
3	4/11	7/11	0
4	2/6	4/6	0
5	0	2/2	0

```
Program 1 ()
{
    int a, b, c;
    a=1;
    a=2*a;
    c=b-a;
    if (a>b)
        c=2*a;
    else
        c=2*b;
    if (a>c) {
        a=b;
        b=c;
    }
}
```

```
Program 2 ()
{
    int a, b, c;
    a=1;
    a=2*a;
    c=b-a;
    if (a>c) {
        if (a>b)
            c=2*a;
        else
            c=2*b;
        a=b;
        b=c;
    }
}
```

```
Program 3 ()
{
    int a, b, c;
    a=1;
    a=2*a;
    c=b-a;
    if (a>c) {
        if (a>b)
            c=2*a;
        else
            c=2*b;
        a=b;
        b=c;
        a++;
    }
}
```

圖 5-1 以 C 語言組成的程式(二)

表 5-2 修正堆疊基馬可夫平衡括號以 Program2 為例

Stack Top	Next Symbol/Stack Action		
	(/push(vtop+1)	) /pop()	$ /none
Program2			
0	1/2	0	1/2
1	4/5	1/5	0
2	4/8	4/8	0
3	3/7	4/7	0
4	3/6	3/6	0
5	2/5	3/5	0
6	2/4	2/4	0
7	0	2/2	0

表 5-3 修正堆疊基馬可夫平衡括號以 Program3 為例

Stack Top	Next Symbol/Stack Action		
	(/push(vtop+1)	) /pop()	$ /none
Program3			
0	1/2	0	1/2
1	4/5	1/5	0
2	4/8	4/8	0
3	4/8	4/8	0
4	3/7	4/7	0
5	2/5	3/5	0
6	2/4	2/4	0
7	0	2/2	0

表 5-4 修正堆疊基馬可夫模式複雜度計算比較表(一)

程式名稱	Line of code (LOC)	Cyclomatic complexity (*CC*)	Entropy (*H*)
Program1	10	3	0.86278
Program2	10	3	0.88618
Program3	11	3	0.89173

因此，由表 5-4 得知，利用修正的堆疊基馬可夫模式，可清楚區分出程式中不同巢狀結構以及區塊大小的複雜度的值，其中 Program1 與 Program2，可針對程式行數相同，但控制流程的巢狀結構組合不同時，複雜度計算上仍可以加以相互比較，以 Program2 含巢狀結構之複雜度較高；而 Program2 與 Program3，則針對不同程式間，控制流程的巢狀結構組合相同，但程式行數不一樣時，複雜度上的高低比較情形。

本論文研究所提出的模式可以解決迴旋複雜度評估法中的缺點，其中迴旋複雜度的評估法的缺點在於無法區分巢狀複雜度中不同深度的組合時，程式複雜度的差異，以及無法探討程式中區塊的大小所隱含的複雜度的情形。運用本論文提出的模式，則可解決迴旋複雜度的評估法的缺點。又軟體科學評估法中，優點是考慮程式量的多寡，但缺點為未考慮程式結構，修正的堆疊基馬可夫模式則將程式中運算元與運算子組成的區塊大小亦列入考量，且可以分辨不同深度之巢狀結構。

此外，本論文提出的修正堆疊基馬可夫模式亦可以針對程式表示式中，不同運算元與運算子所構成之不同的區塊或表示式，所對應的複雜度作衡量，亦即考慮程式量的複雜度，所以表示式中的結合關係之不同，會有著不同的複雜度高低，以二個表示式 a=a*b+c+d 以及 a= c+a*b+ d 為例，說明相同的運算元與運算子組合成不同的運算式的情形下，其複雜度也會有所不同。

表 5-5 修正堆疊基馬可夫模式的平衡括號(一)

Stack Top	Next Symbol/Stack Action		
	(/push(vtop+1)	) /pop()	$ /none
表示式：a=a*b+c+d			
0	1/2	0	1/2
1	3/4	1/4	0
2	0	3/3	0

表 5-6 修正堆疊基馬可夫模式的平衡括號(二)

Stack Top	Next Symbol/Stack Action		
	(/push(vtop+1)	) /pop()	$ /none
表示式：a= c+a*b+ d			
0	1/2	0	1/2
1	2/3	1/3	0
2	1/3	2/3	0
3	0	1/1	0

這二個式子的運算元分別可用{ (,) }來表示，以修正堆疊基馬可夫模式的平衡括號表計算表示式中的機率情形，如表 5-5 及 5-6 所示。並分別用修正的堆疊基馬可夫模式與 Halstead 提出軟體科學評估法來計算程式之複雜度熵(*H*)及量值(*V*)高低，如表 5-7 所示。

表 5-7 修正堆疊基馬可夫模式複雜度計算比較表(二)

表示式	*V*	*H*
a=a*b+c+d	$9log_2 7$	0.58279
a= c+a*b+ d	$9log_2 7$	0.83442

一旦將表示式複雜度的觀念運用於整個軟體資訊系統中，則可發現系統可被簡化為一連串的平衡括號的組合，利用本研究所提出之簡單堆疊基之馬可夫模式即可計算出程式的複雜度。

5.2 實際程式的評估驗證

驗證修正的堆疊基馬可夫評估模式的正確性後，更利用六個以 Visual Basic 語言所撰寫的管理資訊系統程式作為實驗樣本，比較各程式所含的複雜度高低，這資訊系統是目前台灣中南部某製造業所使用的管理資訊系統。

表 5-8 程式行數計算比較表

程式名稱	符號總數計算 (Number of symbols)	總行數計算 (Total lines of code)
Bprogram1	18298	730
Bprogram2	42605	1536
Bprogram3	25880	987
Bprogram4	32183	1082
Bprogram5	11821	410
Bprogram6	16742	517

計算各程式的總行數，用以比較這六個管理資訊系統，如表 5-8 所示。由表 5-8 可發現 Bprogram2 的行數值為 1536，其行數是最多的，而 Bprogram5 的行數值為 410，其行數是最少的一個，而六個系統行數多寡依序分別為：Bprogram2、Bprogram4、Bprogram3、Bprogram1、Bprogram6、Bprogram5。但是僅計算總行數的評估方式並不能真正評估出系統的複雜度，故以本論文所提出之修正的堆疊基馬可夫模式來衡量各個系統的複雜度，如表 5-9 所示：

表 5-9 程式複雜度計算比較表

程式名稱	熵(Entropy)
Bprogram1	0.92322
Bprogram2	0.93062
Bprogram3	0.94255
Bprogram4	0.93131
Bprogram5	0.96767
Bprogram6	0.85216

由表 5-9 中可得知，以 Bprogram5 之複雜度最高，而 Bprogram6 之複雜度最低，而六個系統熵值多寡依序分別為：Bprogram5、Bprogram3、Bprogram4、Bprogram2、Bprogram1、Bprogram6。其中 Bprogram5 之複雜度最高，推論其原因，為該程式中，其總行數以及其字元數雖然較少，但是堆疊深度高且多，故計算出整個程式的複雜度值是最高的，而 Bprogram6 之複雜度最低，因為 Bprogram6 的行數少且其堆疊深度亦較少的緣故，故計算出整個程式的複雜度值是最低的。

由實驗數據可得知，評估程式複雜度時，不能只是考慮行數大小，而是應該將控制結構複雜度以及區塊之複雜度同時列入評估模式考量，才能計算出系統的真正複雜度。

在本章中，驗證出修正的堆疊基馬可夫模式，能以更簡單的計算方法衡量出程式所隱含的複雜度高低，所以說明修正的堆疊基馬可夫模式相較於其他評估法的考量更為嚴謹，且更能充分表示出程式的實際複雜情形。

第六章

結論

6.1 研究貢獻

本論文提出修正的堆疊基馬可夫模式，因為已同時考慮了程式的大小及控制流程的因素，所以可彌補一般評估法未周全考量影響複雜度因素的缺點。此外，本論文也提出了一種新的計算程式大小複雜度的觀念，即除了考慮程式量的大小，更應同時考慮程式表示式的結合關係，計算表示式中的控制流程的複雜度，以建立出更完善的評估方法。本論文除了建構出評估模式，更以實際的管理資訊系統驗證模式。也就是說本論文的優點，一則提出了一完整的複雜度評估模式，模式已將影響程式複雜度的因素都涵蓋在內；一則是模式具簡單性，一個好的評估模式是愈簡單愈好，論文中提出的模式除了具完整性外，更具簡單性，讓評估者易於進行評估或比較。

因為本論文研究提出的評估方法，可作為比較不同資訊系統所含複雜度高低的衡量工具，而程式複雜度的高低和軟體開發成本與維護成本間又存在高度正相關的關係，所以運用本文提出之模式，亦可作為管理者或決策者，在軟體開發、採購決策時的參考依據。

軟體評估領域中雖已存在許多衡量方法，但仍缺乏較為簡單、客觀的評估技術，以供設計者及管理者做評估之用，因此本論文便針對軟體複雜度評估技術作初步的研究。本論文除了從學理上探討軟體評估技術外，且以實際之資訊管理系統來驗證模式，最後發展出一周全的軟體評估技術，實驗

結果顯示修正的堆疊基馬可夫模式，能夠達到評估軟體複雜度之目的。

6.2 未來研究方向

未來後續研究方向，除可進一步探討程式控制結構及區塊大小對複雜度影響的權重高低，使能夠更適切的表達出客觀衡量的結果外；更可發掘複雜度高低與開發成本間的實際關係，期能透由複雜度之評估，進而模擬成程式產生器(code generator)，以作為資訊系統開發前的模擬分析，預計軟體開發所需的各項成本，藉此提供管理或決策者在選擇或開發任何管理資訊系統前的決策參考。

參考文獻

一、中文部分

[金子葳 1998] Charle S. Parker 原著，金子葳，洪秀朋編譯：*電腦概論：現在與未來*，儒林, 1998.

[林信惠 1993] 林信惠，李坤清，李明憲：*軟體開發之成本影響因素研究*，資訊管理，第一卷，第一期，1993.

[林仁常 1998] 林仁常：*軟體工程導論*，松崗圖書有限公司, 1998.

[胡正國 2000]胡正國：*我國軟體維護之現況調查研究－與其他國家比較*，屏東科技大學資訊管理系碩士論文, 2000.

[留忠賢 1997] Sommerville 原著，留忠賢譯：*軟體工程*，松崗圖書有限公司, 1997.

[張國鴻 2000] Detlev J. Hoch, Cyriac R. Roeding, Gert Purkert, Sandro K. Lindner, Ralph Muller 原著，張國鴻譯：*數位式競爭全球軟體公司的致勝策略*，天下遠見出版股份有限公司, 2000.

[楊建民 1996] 楊建民，羅正豐：*我國軟體公司成長階段與重要營運活動關係之研究*，政治大學資訊管理研究所碩士論文, 1996.

二、英文部分

[Abramson 1963] Norman Abramson, *Information Theory and Coding*, McGraw-Hill, 1963.

[Basili 1996] Basili, V.L., Briand, L., and Melo, W.L., "A Validation of Object-Oriented Metrics as Quality Indicators," *IEEE Trans. Software Eng. 10, 751-761*, 1996.

[Berlinger 1980] Eli Berlinger, "An Information Theory Based Complexity Measure," *Proceedings, National Computer Conference*, 1980, AFIPS press, pp. 773-779.

[Berns 1984] G. M. Berns, "Assessing Software Maintainability," *Communications of the ACM*, vol.27, no.1, pp.14-23, January 1984.

[Bieman 1984] James M. Bieman, "Measuring Software Data Dependency Complexity," Ph.D. Dissertation, Center for Advanced Computer Studies, University of Southwestern Louisiana, 1984.

[Bieman 1985] James M. Bieman, and William R. Edwards, "Experimental Evaluation of the Data Dependency Graphs for USE in Measure Software Clarity," *Proceedings, HICSS-18,* pp.271-276, January 1985.

[Boehm 1981] B.W. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

[Conte 1986] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *A Software Engineering Metrics and Models*, Benjamin-Cummings, Menlo Park, CA, 1986.

[Dunsmore 1979] H.E. Dunsmore and J. D. Gannon, "Data Referencing: An Empirical Investigation," *IEEE Computer*, pp.50-57, December 1979.

[Edwards 1988] William R. Edwards, Jr., "Information Content of Partitions," *19th Southeastern International Conference on Combinatorics, Graph Theory, and Computing, Baton Rouge*, Louisiana, February 1988.

[Edwards 1990] William R. Edwards, Jr., "Information Source Models for Software Analysis," *Proceedings, 13th Minnowbrook Workshop on Software Engineering*, pp.8-17, July 1990.

[Edwards 1991] William R.Edwards, Mingguey Yang, and Jong Soo Kim, "Application of the Stack-Based Markov Source to Software Analysis," *Proceedings, 14th Minnowbrook Workshop on Software Engineering*, pp.44-62, July 1991.

[Fenton 1997] Norman E. Fenton, Shari Lawrence Pfleeger, *Software Metrics: A Rigorous & Practical Approach*, 1997.

[FIPS 1984] FIPS PUB 106, "Guideline on Software Maintenance," *Federal Information Processing standards Publication*, 1984.

[Grable 1999] Ross Grable. Jacquelyn Jernigan, Casey Pogue, And Dale Divis, "Metrics for Small Projects: Experiences at the SED," *March/April IEEE* 1999, pp.21-29.

[Gray 1987] Gray, R.b., Caswell, D., "Software Metrics: Establishing a Company-wide Program," *Prentice Hall, Englewood Cliffs*, NJ, 1987.

[Halstead 1977] M. H. Halstead, "Elements of Software Science, Operating, and Programming Systems Service," *Volume 7, New York, NY: Elsevier*, 1977.

[Halstead 1979] M. H. Halstead, "Advances in Software Science," *in Advances in Computers*, vol. 18 pp. 122-129, Academic Press, 1979.

[Harrison 1992] Warren Harrison, “An Entropy-Based Measure of Software Complexity,” *IEEE Transactions on Software Engineering*, vol.18, no.11, pp. 1025-1029, November 1992.

[Hellerman 1972] Leo Hellerman, “A Measure of Computational Work,” *IEEE Transaction on Computers*, vol.c-21, no.5, pp. 439-446, May 1972.

[Henry 1981] Sallic Henry and Dennis Kafura, “Software Structure Metrics Based on Information Flow,” *IEEE Transactions on Software Engineering*, vol. SE-7, no.5, pp.510-518, September 1981.

[Holland 1993] James Allen Holland, “A Stack based Object Oriented Development Model,” Ph.D. Dissertation, Center for Advanced Computer Studies, University of Southwestern Louisiana, Spring 1993.

[IEEE 1993a] *IEEE Software Engineering standards*, Std.610.12-1990, pp. 47-48.

[IEEE 1993b] IEEE, “Standard for Software Maintenance,” Los Alamitos, *CA:IEEE Computer society Press*, IEEE std. 1219, 1993.

[Ince 1990] Ince, D.C. “Software Metrics,” *IEE Colloquium on*, 1990, Page(s): 1/1 -1/2.

[Jacquet 1997] Jean-Philippe Jacquet and Alain Abran, “From Software Metrics to Software Measurement Methods: A Process Model,” IEEE, pp.128-134, 1997.

[Kim 1991] Jong-Soo Kim, “Information Source Analysis of Data Definition and Reference in a Procedural Programming language,” Ph.D. Dissertation, Center for Advanced Computer Studies, University of Southwestern Louisiana, Fall 1991.

[Layzell 1994] Layzell, Paul J. and Macaulay, Linda A., "An Investigation into software Maintenance --Perception and Practices," *Software Manitenance: Research and Practice*. vol.6, pp. 105-120, 1994.

[Li 1987] H. F. Li, and W. K. Cheung, "An Empirical Study of Software Metrics," *IEEE Transactions on Software Engineering*, vol. SE-13, no.6, June 1987.

[Li 1993] Li, W., and Henry, S., "Object-Oriented Metrics Which Predict Maintainability," *J. Systems and Software 2*, pp. 111-122, 1993.

[Li 2000] Wei Li, "Software product metrics," *IEEE Potentials*, pp. 24-27, December 1999/January 2000.

[Lubinsky 1990] David J. Lubinsky, "Measuring Software Size by Distinct Lines," *Proceedings, 14th International Computer Software and Applications Conference*, pp. 403-407, October 1990.

[Martin 1983] Martin , J. and McClure, C., "Software Maintenance: The Problem and its solutions," *Englewood Cliffs*, NJ: Prentice-Hall，1983.

[McCabe 1976] Thomas J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no.4, pp. 308-320, December 1976.

[McCabe 1989] McCabe, Thomas J. & Butler, Charles W. "Design Complexity Measurement and Testing," *Communications of the ACM32*, pp. 1415-1425, December 1989.

[McCabe 1994] McCabe, Thomas J. & Watson, Arthur H. "Software Complexity," Crosstalk, *Journal pf Defense Software Engineering 7*, pp. 5-9, December1994.

[Mohanty 1981] Siba N. Mohanty, “Entropy Metrics for Software Design Evaluation,” *The Journal of Systems and Software 2*, pp. 39-46, 1981.

[Patrick 1982] J. D. Patrick and C. S. Wallace, “Stone Circle Geometries: An Information Theory Approach,” *in Archaeoastronomy in the Old World*, ed. D. C. Hoggie, Cambridge, 1982, pp. 231-264.

[Ramamurthy 1988] Bina Ramamurthy and Austin Melton, “A Synthesis of Software Science Measures and the Cyclomatic Number,” *IEEE Trans on Software Engineering*, vol. 14, no.5, pp. 1116-1121, August 1988.

[Rochkind 1975] Rochkind , M., “The Source Code Control System,” *IEEE Trans. Software Engineering*, vol. 1, no. 4, December 1975.

[Samadzadeh 1988] Mansur H. samadzadeh and William r. Edwards, “A Classification Model of Software Comprehension- Abstract,” *Proceedings of 21th Hawaii International conference on System Science*, January 1988.

[Shannon 1948] C. E. Shannon, “A Mathematical Theory of Communication,” *Bell Systems Technical Journal*, vol. 27, pp. 379-423, 1948.

[Shepperd 1995] Martin Shepperd, *Foundations of Software Measurement*, Prentice Hall, 1995.

[Swanson 1976] Swanson, E. B., “The Dimensions of Maintenance,” *Proceedings of the Second International conference on software Engineering*, pp. 492-497, 1976.

[von Mayrhauser 1990] von Mayrhauser T.E. , “Software Engineering-Methods and Management,” *San Diego, CA: Academic Press*, Inc, 1990.

[Wallace 1996] D. Wallace and L. Ippolito and B. Cuthill, "Reference Information for the Software Verification and Validation Process," *NIST Special Publication 500-234*, U.S. Department of Commence/National Institute of Standards and Technology, April 1996.

[Walston 1977] C. E. Walston and C. P. Felix,"A Method of Programming Measurement and Estimation," *IBM System Journal*, vol. 16, no. 1, pp. 55-73, 1977.

[Wang 1991] Cheng-Tzu Wang and William R. Edwards, "An Implementation of the Stack-Based Markov Model on Pascal Code," Technical Report 91-4-1, Center for Advanced Computer Studies, University of Southwestern Louisiana, February 1991.

[Wang 1994] Cheng-Tzu Wang, "Stack-Based Markov Model Analysis of Expressions and Data Dependencies on Programs," Ph.D. Dissertation, The University of Southwestern Louisiana, Spring 1994.

[Wang 1997] Cheng-Tzu Wang, "A Probabilistic Model of Software Complexity Measurement," *in Proceedings of the 8th International Conference on Information Management*, pp.776-782, 1997.

[Yang 1991] Mingguey Yang and William R. Edwards, "Experimental Study of the Stack-Based Markov Model," Technical Report 90-4-7, Center for Advanced Computer Studies, University of Southwestern Louisiana, February 1991.

[Yang 1992] Mingguey Yang and William R. Edwards, "Stack-Based Markov Model for Imperative and Functional Languages," Technical Report 92-5-7, Center for Advanced Computer Studies, University of Southwestern Louisiana, February 1992.