軟體系統複雜度評估—一種修正的堆疊基馬可夫模式

發表人：賴秀女/南華資管所
指導教授：王鄭慈先生、鍾志明先生

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，同時也應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。

提出的模式可作爲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，亦可作爲管理者或決策者，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

壹、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，從一九九六年到九七年，軟體產業的營業額成長了 20%。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爲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。所謂軟體評估係指利用軟體之程式碼計算出一個數值，利用這數值即可表示出軟體的複雜度，透由適當之評估可讓管理者對軟體更加地了解，並且可用以預估軟體發展時間、軟體成本以及軟體之可信賴度，適當利用軟體評估技術可以協助管理者評估軟體效能或改善提昇軟體品質。

評估(metric)一詞可解釋爲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]。

貳、研究動機

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]，雖能作爲評估之工具，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，能同時兼顧程式的大小及控制流程的複雜度，並能作爲比較不同軟體間所含的複雜度高低的衡量準則工具。

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%、至今約爲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。

正因爲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，作爲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，作爲評估管理資訊系統之開發、擴充或選擇時之決策參考依據。

因爲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爲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。

參、文獻探討

依軟體評估進行時著重觀點的不同，可將軟體評估分類爲軟體產品評估(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，在於都是爲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。

一、古典資訊理論(Classical Information Theory)

在古典資訊理論中，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爲：

$$I(si) = \log\left(1 / p(si)\right) \quad \text{(bits)}$$

$$H(S) = \sum_{i=1}^{n} p(si) \log\left(1 / p(si)\right) \quad \text{(bits)}$$

其中 p(s)是訊息 s 發生的機率，I 代表訊息所含的資訊內容，H 則表示全部訊息所涵的資訊內容的平均值，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爲一資源產生器；而軟體系統的開發則被視爲一資源產生器所產生出的一連串具特殊意義的事物。這個過程可被視爲一類似馬可夫過程(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爲基礎的複雜度評估方法[Mohanty 1981, Samadzadeh 1988, Edwards 1988, Patrick 1982]

因爲早期運用資訊理論基礎衡量程式複雜度的方式裡，常視資訊源爲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爲：

$$I(si|sj_1,\ldots,sj_m) = \log(1/p(si|sj_1,\ldots,sj_m)) \quad \text{(bits)}$$

$$H(S|sj_1,\ldots,sj_m) = \sum_{s} p(si|sj_1,\ldots,sj_m) I(si|sj_1,\ldots,sj_m) \quad \text{(bits)}$$

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，因爲 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性，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 提出了一種以堆疊爲記憶裝置的馬可夫模式來作爲計算程式複雜度的評估準則[Edwards1990; 1991]。

二、堆疊基馬可夫模式 (Stack-based Markov Model)

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爲是一個資訊發射源，而開發出來的軟體系統就可視爲是資訊源所發射出的一連串具有意義的事件或符號，如圖 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]。

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。

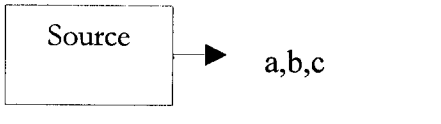

圖 1　馬可夫資訊源

堆疊中的基礎動作包含把符號壓入堆疊上，以符號 push()素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 (bl 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，假設元素集合爲{ (,) , 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 1 所示爲馬可夫平衡括號。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。而資訊內容 *(I)*及程式之熵 *(H)* 計算式分別爲：

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

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

表 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 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 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]。

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

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

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

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

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

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爲軟體評估的方式，因爲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。

肆、修正的堆疊基馬可夫模式

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、建構評估模式的過程

本論文建構評估模式的過程，依據下列幾個步驟進行，分別爲：規劃衡量目標、收集程式資料、建置本研究之模式、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，以作爲修正模式之依據或建構出完整評估模式，如圖 2 所示。

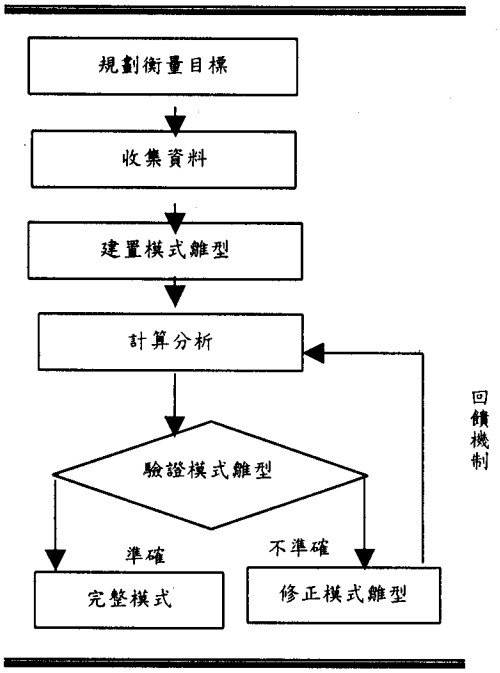

圖 2 本論文建構評估模式的架構

在建置模式中，期以一簡單模式來作爲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。

二、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，假設元素集合爲{ (,) , $ }，因此程式碼可被轉換爲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，因爲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 3 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；

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

表 3 初始模式的馬可夫平衡括號(一)

而由修正之馬可夫平衡括號中，可以導出下列方程式：

$P(0,0)=1;$

$P(t,s)=0;\text{for } s>t$

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

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

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

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。

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

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

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

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

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

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爲準確衡量的工具，其中初始模式的馬可夫平衡括號如表 4、表 5 所示：

表 4 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(()()()$) =4*log*4-3*log*3=3.2451
I((())()$) =4*log*4-2*log*2=6

表 5 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

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，雖可計算出符號的複雜度高低，但卻仍無法分辨符號間皆含巢狀結構，但卻存在不同巢狀深度的情形，其中初始模式的馬可夫平衡括號如表 6、表 7、表 8 所示：

表 6 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 7 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 8 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，但卻無法區分出符號間含有不同巢狀深度這種情形下，符號間複雜度的高低，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。

三、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，但因爲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，爲了要區分不同深度之巢狀結構有著不同的複雜度，修正初始模式，將之演進爲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 9 所示。

表 9 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 10、表 11、表 12 所示：

表 10 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 11 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 12 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。但是因爲程式中的表示式中，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外，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外，更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。

伍、實驗分析

本文將修正的堆疊基馬可夫模式，分別以不同的程式組作爲測試樣本，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。

一、模式的正確性

舉三個程式爲例，分別計算模式的平衡參數表機率及複雜度熵 H 的值，如圖 3、表 13、表 14、表 15 及表 16 所示。

表 13 修正堆疊基馬可夫平衡括號以 Program1 爲例

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 3 以 C 語言組成的程式(二)

表 14 修正堆疊基馬可夫平衡括號以 Program2 爲例

Stack Top	Next Symbol/Stack Action		
	(/push(vtop+1)	) /pop()	$ /none
Program1			
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

因此，由表 16 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。

表 15 修正堆疊基馬可夫平衡括號以 Program3 爲例

Stack Top	Next Symbol/Stack Action		
	(/push(vtop+1)	) /pop()	$ /none
Program1			
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 16 修正堆疊基馬可夫模式複雜度計算比較表(一)

程式名稱	熵 H
Program1	0.86278
Program2	0.88618
Program3	0.89173

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，但缺點爲未考慮程式結構，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 爲例，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。

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參數表計算表示式中的機率情形，如表 17 及 18 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 19 所示。

表 17 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 18 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

表 19 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	$9log_27$	0.58279
a= c+a*b+ d	$9log_27$	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爲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。

二、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爲實驗樣本，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 20 程式行數計算比較表

程式名稱	Number of symbols	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 20 所示。由表 20 可發現 Bprogram2 的行數值爲 1536，其行數是最多的，而 Bprogram5 的行數值爲 410，其行數是最少的一個，而六個系統行數多寡依序分別爲：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 21 所示：

表 21 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 21 中可得知，Bprogram5 之複雜度最高，因實際程式中，其總行數以及其字元數雖然較少，但是堆疊深度最高，而 Bprogram6 之複雜度最低，因爲 Bprogram6 的行數少且其堆疊深度亦較少的緣故。故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爲嚴謹，且更能充分表示出程式的實際複雜情形。

陸、結論

一、研究貢獻

本論文研究提出修正的堆疊基馬可夫模式，因爲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，本論文研究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外，同時也應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，

因爲本論文研究提出的評估方法，可作爲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，亦可作爲管理者或決策者，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，但仍缺乏較爲簡單、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作驗證，最後發展出一周全的軟體評估技術，研究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。

二、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)，以作爲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

[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.

[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

[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.

[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.

[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.

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

[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.

[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.

[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.

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

[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.

[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.

[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.