軟體開發－物件導向思維

◎ 陳朝英（行政院主計處電子處理資料中心設計師）

用傳統思維去駕馭最新科技的工具，開發出來的軟體與傳統工具所發展的差別不大。

壹、前言

有篇《酋長的凱迪拉克》故事是這樣的：印地安人被白人驅趕到保護區之後，一直過著貧困的生活。有一天，他們在自己的土地下發現蘊藏著豐富的石油。一夕致富的印地安酋長因此決定改變生活習慣，訂購了一部 260 匹馬力的最高級凱迪拉克加長型大轎車。

如此豪華、舒適的轎車令酋長非常高興，於是每天都坐上轎車在周邊的印第安村莊繞個幾圈過過癮。很快的兩個星期過去了，凱迪拉克業務人員依例再度拜訪酋長，以了解使用情況，酋長告訴他已經乘坐轎車走過不少路程，感覺蠻舒服的，但美中不足的是速度慢了些。業務員查看後發現里程表指針指在零。業務員百思不解，經詢問得知酋長棄馬車改搭高級轎車後，還是以馬匹拉著凱迪拉克豪華轎車行駛在道路上，當然速度比馬車還慢。於是他解下保險桿上的馬，請酋長坐進後座，車鑰匙輕輕一扭，蘊藏在引擎中的 260 匹馬力立即驅動，全速奔馳而去……。

這個故事告訴我們，處於資訊技術（Information Technology；簡稱 IT）一夕數變的年代，我們處理問題的工具與 IT 息息相關，在運用相關 IT 解決問題的過程中，是否也以較新的思維與作法來面對? 軟體開發就是典型例子。多年前結構化程式語言 COBOL 盛行時，程式設計者若沒有以結構化方式來設計，開發出來的應用系統很難具有結構化的優點。同樣地，現今許多軟體語言或開發工具，號稱具有物件導向特性，例如 JAVA、C++、PowerBuilder、Delphi、VB.NET 等，如果應用軟體系統開發者沒有足夠的物件導向思維基礎，應用這些語言或工具就不能開發出具有物件導向優異特性的應用系統。這就像是印地安酋長雖然擁有加長型的凱迪拉克，卻還是用馬匹拖著轎車前進一樣。由此可見，好的開發工具，還得配合適切的思維基礎才能發揮所長。

本文擬以傳統軟體開發的問題、遞增與循環式軟體開發流程(Incremental and Iterative Process)、物件導向思維、實施物件導向思維軟體開發的阻礙等，說明軟體開發的課題，並陳述物件導向思維要旨。

貳、傳統軟體開發的問題

十幾年來，軟體開發技術有非常大的變化，從傳統的程序導向、結構化技術，到 90 年代的物件導向方法。軟體開發技術演進起因於傳統開發方式有不足之處，我們首先就傳統開發技術隱含的一些假設與呈現的問題予以說明。傳統開發方法假設使用者了解自己的需求，需求一經確定就被凍結起來，然後才進入系統分析與設計階段，需求在設計期間盡可能不予變動。而

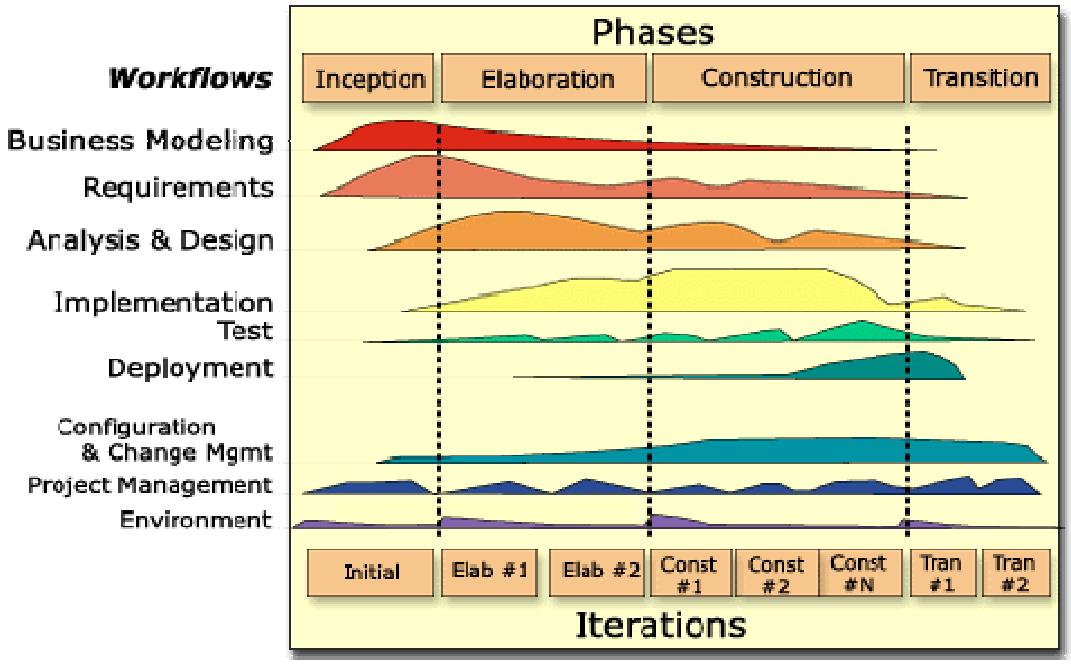
軟體開發從需求蒐集、分析、設計、撰寫程式碼、測試到驗收成一直線下去，無法回頭，故又名瀑布法。在這段期間，使用者只能從各個階段產出的文件得知進度與結果，至於系統的真正外貌、實際運作只有等待軟體完成後方能一探究竟。因爲在程式碼完全出來後，使用者才看得到系統整體外貌，假如結果不是使用者所希望的系統，因程式碼已經完成，如需重新修改系統，牽一髮動全身，耗時又耗錢，軟體開發之整體風險度明顯提高。

其次，從軟體切割與架構的角度來看，傳統開發方式，所指的軟體架構是軟體系統發展完成後呈現出來的功能架構。依據流程來切分軟體應用系統，然後由上而下分割爲樹狀的功能模組，使用者所看到的功能架構就是軟體切分的結果。實際上，流程是易變的，以不穩定的流程切分出來的模組自然是脆弱而經不起考驗的，而不穩定的模組要組成穩定的系統等於是緣木求魚。究其原因，傳統開發方式所謂的軟體架構(或稱功能架構)並不是建立在穩定的基礎上，這個穩定的基礎就是企業架構(Architecture)，或稱企業模式(Business Model)。

再者，從技術的角度來看，傳統的軟體發展方式使用功能/資料(Function/Data)的方法。功能與資料被分開來處理，針對某一項資料執行運算(Operation)的功能必須知道這些資料的內部結構，資料如何存放? 存放在什麼地方? 不同型態(Types)的資料，其資料格式稍有差異，程式中就需要一些條件子句(Condition Clauses)以驗證(Verify)這些資料型態，因此使用這種方法導致內部邏輯較爲複雜，而穩定性更是一項令人憂心的因素。因爲只要資料結構一有變動將直接影響使用這些資料的所有功能。有關係的功能與多處出現的地點必須跟著進行檢查與更新。如遇大規模的系統需要進行功能維護時，僅僅對局部的少數程式進行更改，結果是引進更多的程式蟲(Bug)，甚至一發不可收拾，修改軟體或解決問題就會成爲程式開發人員的夢魘，這就是爲何傳統開發方式發展出來的應用軟體耗費許多人力與時間於系統維護上的原因之一。

參、遞增與循環式開發流程(Incremental and Iterative Process)

循環式途徑又稱漸進式途徑，是物件導向觀念中的主要軟體開發途徑。傳統開發途徑一般區分爲需求蒐集、分析與設計、建置、最後是測試驗收與部署(Deployment)。也就是通常稱呼的瀑布式開發方法。而循環式途徑則是將整個開發流程分爲啓始、詳述、建構與轉換等四個階段。每一階段又再細分爲一到數個循環(Iteration)，每個循環所經過的程序相當於瀑布式的方法(即需求、分析與設計、建置…等)，如圖一所示：



圖一、瀑布式開發流程

一、在啓始階段(Inception Phase)進行：

- 定義專案範圍
- 找出潛在風險
- 評估資源需求，準備專案環境
- 找出主要的使用個案（Use Case）
- 得到初始的軟體架構（Architecture）
- 製作系統雛形，驗證技術可行性

使用個案(Use Case)係使用者與系統互動的過程，用來表達系統的功能需求。其中最重要的一環是依系統建置風險高低安排使用個案處理的先後順序，風險越高者優先安排在初期處理。

二、詳述階段(Elaboration Phase)：

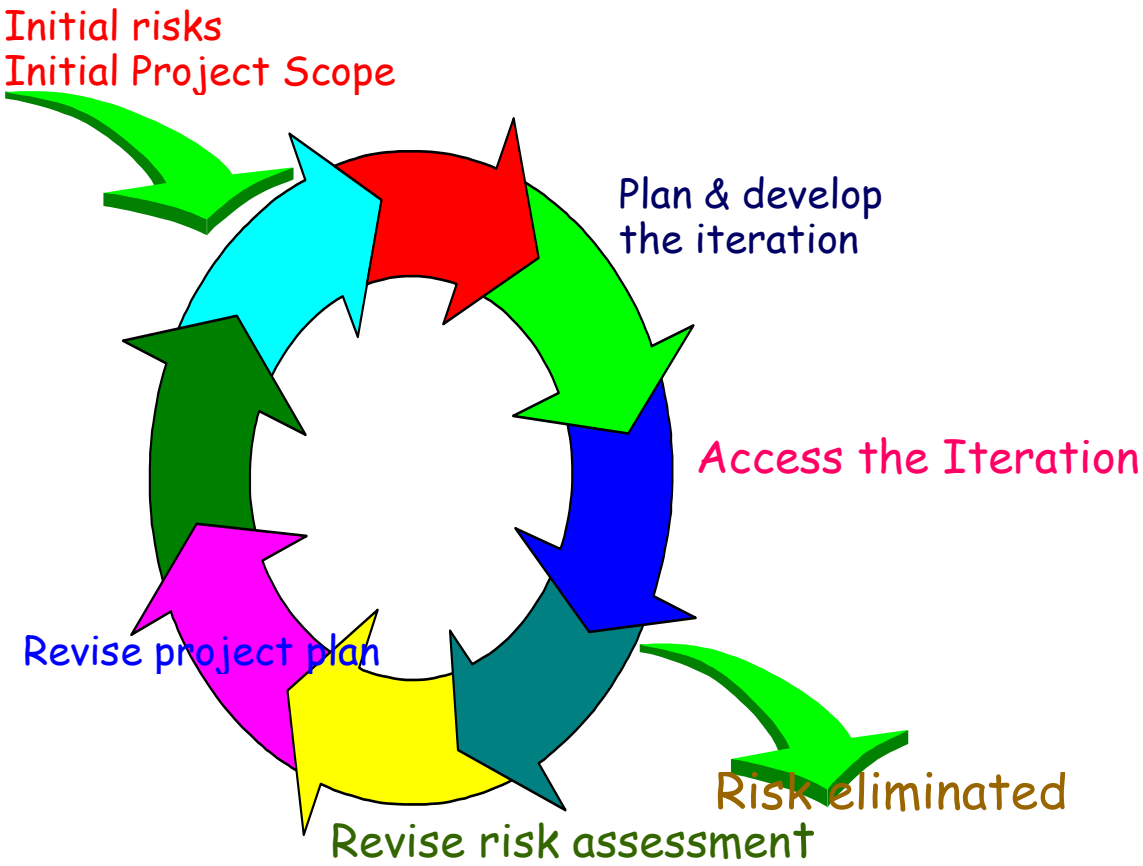
- 考慮大部分重要需求（使用個案）
- 設計出可靠的軟體架構
- 調整前一階段軟體架構，找出軟體元件
- 製作系統雛形，驗證架構的穩定性
- 詳細規劃專案的活動及所需資源

三、建構階段(Construction Phase)：根據前一階段的軟體架構，建置完成產品的所有需求，得到產品的測試版本。

四、轉換階段(Transition Phase)：

包裝(packaging)、移轉、訓練、支援、維護。

由於物件導向方法藉由封裝(Encapsulation)來產生高度模組化的物件，且具有穩定的介面，物件之間有明確的關係。基於物件間的獨立性，軟體開發人員可以先針對某些使用案例(Use Case)分析某些物件，進而設計實作之；完成後可再針對另外一組使用案例分析設計一些物件，並實作之；依序循環下去，直到整個軟體系統完成處理。所以在上面的四個階段中，每個階段都區分爲一至數個循環來處理。圖二是一個循環所要進行的工作：



圖二、Iterative & Incremental Development

遞增與循環式軟體開發流程最主要的精神在於每完成一個循環就減少整個專案的風險；並且針對整個專案的風險重新評核(Review)，透過該循環所產生的雛形軟體與使用者進行溝通討論，引導使用者對需求能夠更明確的陳述。如有需要，可以對專案計劃與需

求進行調整。除了對於每一個循環所產生的軟體進行測試之外，同時也將通過測試的該循環軟體加入整體程式碼進行整合與測試，這就是遞增與循環式軟體開發流程(Incremental and Iterative Process，簡稱 I&I)。

從 I&I 開發流程可以知道物件導向開發方式具有下列傳統開發方式所沒有的優點：

1.將較高風險的部分在系統開發的前期解決，降低專案風險；而傳統開發方式的專案風險是遞增且在程式碼完成時達到最高。

2.由於是採取循環式開發方法，允許在專案進行中針對專案計劃進行調整，變更或增加需求。由於使用者對其需求的認知是漸進的，不可能在一時之間完整地將需求表示出來，此種開發方式真正符合事實需要。

3.在開發過程中透過與使用者的互動，降低軟體不符需求的風險。

4.對於複雜度高而且大型的專案，較傳統方式易於控制軟體品質。

肆、物件導向思維

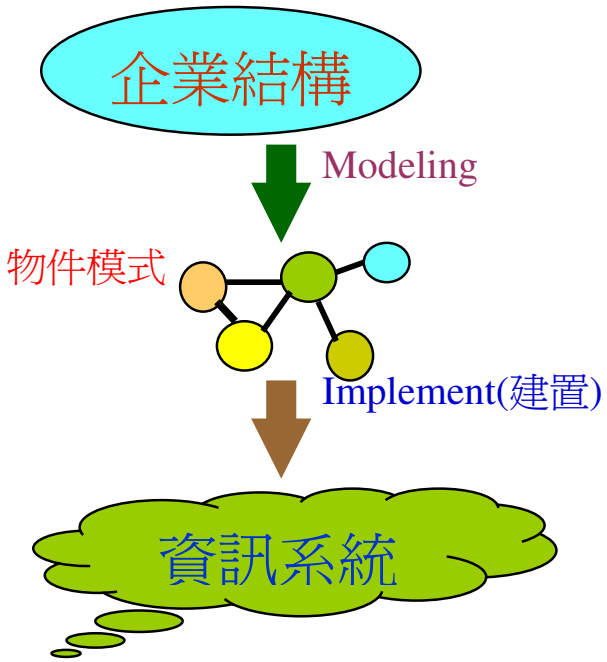
物件導向思維乃是軟體人員心智(Mental)運作過程，包括對問題的看法，與如何透過對問題的了解而導出解決方案(Solution)，然後選用合適的軟體建置。常見的物件導向思維模式說明如下：

一、實體對應(Entity Mapping)與領域知識架構(Domain Knowledge Architecture)

由於傳統開發軟體著重在系統功能上，致力於思考系統應提供那些功能，以及使用者使用這些功能的順序，而不去探究企業的實體結構。但在真實世界(Real-World)企業透過一系列的企業活動來爲顧客提供產品或服務。這一系列的企業活動稱爲企業流程(Business Process)，企業中的各個實體(Entity)經由彼此間的互助合作來執行企業流程，此種實體間的結構關係可用企業模式(Business Model)來表達，例如圖書處理系統中的書籍、讀者、出版社等皆是企業實體；又如學生選課系統中的學生、老師、學科、成績等等也是屬於企業實體。企業實體對應到企業模式中的企業物件，企業的每一項活動對應到企業物件的一項運算(Operation or Method)，每一個企業物件都有它的責任(Responsibility，也是 Operation or Method)。物件導向思維就是先了解問題領域或企業結構(又稱企業領域結構)，然後再將這種結構、實體與實體間的關係對應到物件模式中的物件，軟體開發人員再依據物件模式來建置資訊系統。整個對應關係如圖三所示。

經由物件導向思維所導出的物件模式，因爲是來自於真實世界的企業領域結構(Business Domain Architecture)，因爲企業領域架構較爲穩定，當系統需求改變時，就好像真實企業領域中的企業實體責任的變動或責任的增加一樣，以此種思維方式所切分與建置出來的軟體物件在面對變遷時的適應性與穩定性當然會比傳統方法所開發出來的軟體物件良好。有了穩定的物件，可以經由重新組合以應付不斷變動的流程；總之，由企

業領域導出穩定的物件，再由物件的重新組合以解決流程變動的問題。所以在物件導向思維下開發出來的軟體會是較爲穩定而有彈性與擴充性。



圖三、以物件模式建置資訊系統

二、抽象化(Abstraction；或稱萃取)

抽象化係將焦點放在物件的整體而非細節，重點在於表現什麼是物件？物件做什麼？忽略與目的無關的其他細節。如此，不但可以讓我們對整個問題的思考維持在一定的概括性上，不至於陷入旁枝末節的泥沼中，藉由抽象化也可以降低處理對象的複雜度。

物件導向開發過程有許多地方使用抽象化思維，例如在需求蒐集階段最主要的產出製品(Artifact)是使用案例模式(Use-Case Model)，著眼於整個系統的外部觀點，使用顧客語言來描述使用者與系統的互動過程，重點是使用者希望未來的系統是什麼樣子？具備什麼樣的功能? 當開發進入分析階段時，著眼於系統內部觀點，使用發展者的語言來描述系統，其主要的產出是分析模式(Analysis Model)，使用概念性物件以實現使用案例(Use Case)，經由與實作無關的概念性物件之間的互動以完成使用者需求。所以分析模式提供整體系統的概觀，由於與系統建置無關，有利於設計階段採用可供選擇的不同設計方案。進入設計階段，陸續將一些實體因素考慮進來，包括使用的資料庫、傳輸協訂、建置用的軟體語言或軟體工具、處理型態等等。從需求蒐集到設計，每個階段都有著重的焦點，運用抽象化以忽略當時不需要處理的細節，有效降低軟體系統的複雜度。

三、擬人化

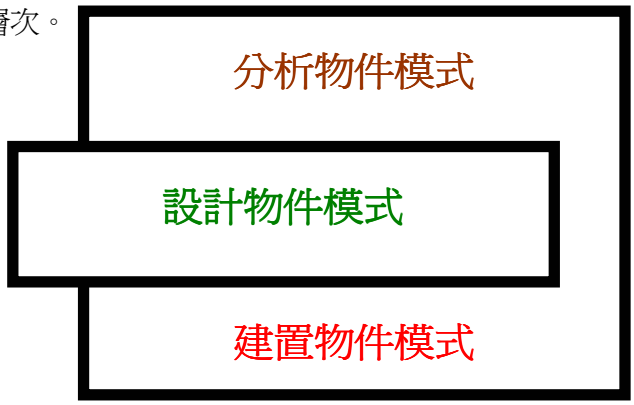
在物件導向的思維中，物件除了具有屬性與行爲(方法)外，物件與物件之間可以經由溝通與互動來完成某項任務。爲了釐清物件與物件之間的合作關係，安排物件的角色，賦予物件責任，定義物件的介面與行爲，我們需要將物件擬人化。將實體或物件看成像人類一樣有智慧、善溝通、可以互助合作。只要物件間的角色與物件本身的責任明確而清楚，有助於後續的設計與建置，同時也比較有利於物件的再利用（Reuse）。

四、以主架構爲中心的開發流程(Architectural Centric Process)

前面提到實體對應(Entity Mapping)，物件導向的思維方式就是先找出穩定的問題領域物件或企業物件，以做爲軟體開發的基礎。穩定、彈性是物件導向的主要要求，元件再用(Component Reuse)則是第二項要求，最後是追求分散式處理以平衡電腦負荷。這三個角度的軟體架構則是物件導向軟體開發時切分物件之依據。

1. 穩定及彈性(Stable and Scalability)：

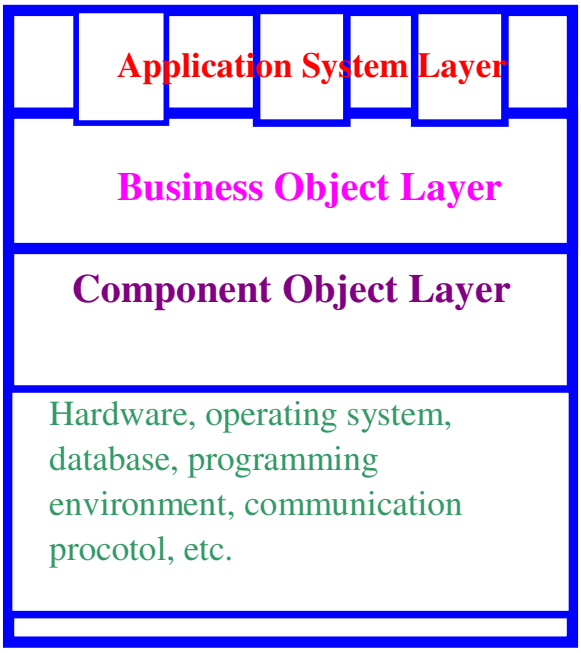
首先，由領域分析找出企業物件，分析物件間的互動與合作關係形成分析物件模式，加入實體因素就是設計物件模式、以建置語言的特性來表達就是建置物件模式，各個模式表現不同的抽象層次。



2. 元件再用(Reused)：分層架構(Layered Architecture)

將分析、設計得到的物件加以分層(Layer)，成爲一群元件(Component)，透過元件分層，以提昇其再用性，元件間有一定的介面互相溝通。而不同層間有一定的相依關係，越是大規模而複雜的軟體越有分層的必要（詳見圖四）。

例如一個銀行系統可能包含的企業物件：客戶、帳戶、貸款；而應用系統包括借款系統、貸款系統及轉帳系統等。這些系統屬於特定應用(Application-Specific)，他們提供銀行功能運作所需之企業流程。企業物件層(Business Object Layer)通常是實體物件(Entity Object)，屬於較爲穩定者。高層物件能夠使用低層物件，低層物件不允許使用高層物件。



圖四、分層架構圖例

3. 分散式處理(Distributed)：

將分析、設計得到的物件依網路架構分組，將物件區分爲呈現層(Presentation Layer)、企業邏輯層(Business Logic Layer)、資料來源層(Data Store Layer)，分別對應到硬體的三層式架構的用戶端(Client)、應用伺服器(Application Server)、資料庫伺服器(Database Server)。此種架構的分層方式，將軟體物件切分爲呈現於用戶端的物件、於應用伺服器處理的企業邏輯物件，以及跟資料庫相關的物件。如此各司其職，能讓畫面樣式的更動與中間層的企業邏輯，系統維護更方便，且與後端資料庫的連結透通性(Transparency)更良好。

五、資料與功能之物件化處理

在物件導向思維方式下，資料與功能(行爲或方法)是一體的，經由封裝(Encapsulation)機制，每個物件僅允許其他物件經由正式管道(該物件提供的方法)存取該物件所掌管的資料。使用資訊隱藏(Information Hiding)技術，其他物件只能知道該物件提供的功能，並且透過訊息的傳遞來使用該物件的功能，禁止透過非正式管道，直接存取該物件的資料或使用其內部的方法。萬一因需求變更或維護上的因素，必須更改資料格式、結構或邏輯時，只要針對該物件進行處理即可，如此就能使變動區域化(Localized)，無需牽動全局。此種方法開發出來的物件軟體也比較穩當而有彈性，降低軟體維護人員許多負擔。

陸、實施物件導向軟體開發的阻礙

以物件導向思維開發出來的軟體既然有這麼多優點，爲什麼許多開發人員還是停留在傳統的方法呢？究其原因，可以歸納下列幾點：

一、 改變思維的挑戰

物件導向思維的建立需要一段時間的學習，對於軟體設計的新手來說，問題比較小；但是對於具傳統背景的系統開發人員可能相當艱難，扭轉傳統思維的包袱，改變思考的習慣，對他們而言是一項大挑戰，因而裹足不前。

二、 組織文化

在事事講求快速的時代，任何事情都被要求速成，軟體開發在業務掛帥的組織當中，軟體專案時程由業務主導，開發時程不斷被壓縮，仍被要求在預定時程內完成專案，至於軟體品質則只有等到系統完成後才能評估。

三、 人力因素

政府機關許多資訊人力被舊有系統綁住，忙於維護或管理原有系統。在資訊界則是受承接專案工作影響，資訊人員根本無暇接受新知或學習新的思維方式，對於推動以物件導向思維的開發方法有不利的影響。

伍、結論

企業外在競爭環境變遷快速時代，爲滿足企業需求，軟體開發商和資訊部門人員都必須加速應用軟體的開發以及應用系統的維護。物件導向技術，與過去傳統軟體開發技術不同，可用以開發元件化(Component-Based)軟體，易於修改和不計次數的重複使用，而且開發出來的軟體具有良好的品質，已成爲全球軟體開發的主流。其優點雖倍受肯定，但開發人員也要具備物件導向思維基礎，才能將物件導向的技術發揮得淋漓盡致。