ERP 開發勝經

作者: 陳威均

Email：genie.michael@gmail.com

目錄

第一部份
人生若只如初見

第一章
緣起及目標－深度開發指南

人生若只如初見，何事秋風悲畫扇。
等閒變卻故人心，卻道故人心易變。
驪山語罷清宵半，淚雨零鈴終不怨。
何如薄幸錦衣郎，比翼連枝當日願。

清 • 納蘭容若

第一章　　緣起及目標

想寫一本深入探討 ERP 系統實務開發相關書籍的念頭已經有一段時間。一方面是因為筆者自出社會以來，就一頭踏入 ERP 產業這條不歸路，另外則是因為長期累積了一些經驗，不管是對是錯，都想留下一些紀錄，古德云:「立德，立功，立言」就是這種心態。一方面是自我反省，當然如果本書可以幫助一些想了解 ERP 的新手，或是正處在撞牆的同好，那就是意外的收穫與驚喜了。

筆者並不是想寫一本教人如何寫程式的書，雖然這是無法完全避免的。而是試圖用瞎子摸象的方式，以自身的親身經驗、理性的推理(論)及不斷的反思、歸納整理。再透過實踐、不斷的嘗試錯誤以及直接面對客戶的挑戰 (不可能的任務)等等。在這個過程中，理論知識當然是不可或缺且必須不斷的提升，但更多的是將實際的需求 (問題 issue) 尋求一個最適當的解法。

說起來簡單，但真正有切身之痛的人都知道，要解決問題通常並不難，但若要考慮到日後的維運，甚至要長遠的發展變動，例如 新版本的程式開發工具或平台 (如 Window 程式要升級為 Web 版的程式版本)，那可就不是件簡單的事了。
舉例來說：
如果貴公司有一套用 Delphi 或 VB6.0 開發的小型進銷存系統 (當然是 Window 的版本)，如果您想將這個系統改寫成 Web 版本 (B/S 版)，無論您選用 Visual Studio 20xx、PHP、Java、Python、Ruby on Rails ……等 Web 開發工具，請問程式改寫的幅度有多大？全世界有數不清的公司 (包括很多 ISV 軟體開發廠商) 都過不了這一關。因為改寫的幅度幾乎是百分之百。

將一個歷經多年千辛萬苦開發出來的系統，期間有數不清的研發投入、測試人員不間斷地測試及客戶不斷反饋使用經驗才得到的穩定版本，一旦要改朝換代，之前的努力幾乎等於付諸流水，完全從頭再來一次。只要您靜下心來好好的仔細想想，就知道這絕對是錯的，但偏偏這卻是目前業界的常態。

這也正是本書想試圖去解決的深層問題。這跟個人的聰明才智並沒有太大的關聯，反而跟您在這個舞台蹲了多久的馬步有關。如果沒有經歷過

MS-DOS 或 Linus	Windows95（xp）	Windows 7
文字模式平台	GUI 平台 Client/Server	Web 平台 Internet 普及
dBASEIII、Clipper	Delphi5、VB5 C++、PowerBuilder	PHP、ASP.NET、Java、 RoR、Python

等世代，如果沒有，是不可能去想像會有這種問題的。

當然，這跟整個 PC 產業 (包含軟硬體) 都還屬於朝氣蓬勃的新興行業，有很大的關係。第一代的 IBM PC 的 RAM 是 256K，請看清楚單位，不是 MB 而是 KB。Bill Gates 在 MS－DOS 推出時，還認為 640KB 以上的記憶體(RAM) 是非必要的。事後諸葛人人會做，所以筆者並不是在批評，而只是舉例說明，在產業發展的初期，在標準未定的年代，我們的前輩日夜費盡心思、歷盡千辛萬苦才一點一滴地建構起產業標準，並不斷推陳出新，才有目前的成果。

現在的環境當然相對較成熟穩定了，但還只是天下初定，所以筆者在本書中所提出的觀點，也僅供參考。也許在 20、30 年內，應該還有一些參考的價值。

易經為中國群經之首，將易分為三種狀態，分別是變易、簡易及不易。
其中不易指的是不會變動的事物，在稍後的章節中，筆者會不厭其煩地強調關聯式資料庫 (SQL) 的重要性。其根本的原因即在於中長期看來，資料庫都是 ERP 系統不可或缺的要角，等同易經中的不易。若以周易中的三易的觀點來看。

變易: 認為宇宙萬事萬物的變化是有跡可循的，並不是亂變。如煮開水，100 度就會沸騰。不斷變更的作業系統平台、推陳出新的開發程式工具、日新月異的開發框架及元件等屬之。

簡易: 說明周易的道理非常簡單，就是中庸之道，無過與不及，簡單的說就是陰陽對位，一陰一陽之謂道而已。
ERP＝輸入表單＋過帳等商業邏輯＋輸出報表。
這是萬變不離其中的 ERP 核心概念。

不易: 說明大自然中的人、事、時、地、物，雖然有錯綜複雜的變化，但它的本質是沒有改變的，如日出日落、四季變更是不會改變的。
資料儲存在關聯式資料庫、系統性的管理機制等開發的標準作業，這些觀念並不會隨著時間和工具的演變而改變。

筆者在本書中不斷的強調關聯式資料庫的重要性，但是並不是只將它單純的視為儲存資料的管理程式。如果這樣，就太小看了這些非常棒的工具程式，它的功能遠不只如此，但是，事實上大部分的研發人員卻只使用了它很少部分的功能，真的非常可惜。筆者在此處所提的關聯式資料庫（RMDB），是一個通用的名詞，並未指定持定的公司或品牌。雖然因為個人經驗之故，在實際的範例中，我大都是以 MicroSoft 的 MS－SQL 2005 以後的版本為主，但是相同的概念完全可以延伸運用在其他 Oracle、My SQL、Postgre SQL，FireBird 等遵循 ANSI－92 (之後) SQL 為標準的關聯式資料庫，(包含商業版本及 Open Source 版本)。

在整個 ERP 系統中，關聯式資料庫的地位絕對是重中之重，且無可取代。因為它是絕對的不可或缺。而且從最早的 DBASE III 一直到目前的關聯式資料庫，已經有近 30

年的發展歷史，幾個市場上的主力產品，仍然不斷的在推陳出新，不斷的擴增功能模組（如MS-SQL的新版本加入ReportingServices這個報表元件），表示這個產品的後市可期。更重要的是，不同的廠商推出的資料庫產品，大部份都有遵詢 ANSI-92 為標準，雖然無法百分之百的直接轉換，且轉換也有一定的困難度，但移植不同的關聯式資料庫仍然是可行的，是有可替代的選擇方案，不致被單一廠商綁死，雖然這並不容易。

筆者在前文中就曾經提到本書不是一本教你如何寫程式的入門書籍，主要的目標是希望能夠提供給計劃開發 ERP 系統的企業或是有志於此的相關軟體從業人員（如資深程式設計師、系統分析師及專案經理們），一個完整的深度開發指南。

筆者最早期開始接觸程式設計是在 MS-DOS 時代，使用 DBASE III 及 Clipper 等開發工具。那個時代常常是一個研發工程師就能搞定整套進銷存系統。那是一個百花齊放、英雄輩出的時代。如果您沒聽過或是看過類似的系統，下次到加油站加油時喵一下他們的系統，很多加油站現在使用的還是 DOS 系統，這是一種純文字模式的操作介面，大部份的系統都很單純但是非常的穩定。然後，Windows 95 出現了，圖形化操作（GUI）的作業系統成為主流，經過了一番陣痛及努力學習後，部份開發人員及公司成功的提昇到了新的作業系統及程式開發平台。在 Windows 作業系統中，最普遍被開發人員採用的程式語言如下

資料來源：http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Programming Language	2014	2009	2004	1999	1994	1989
C	1	2	2	1	1	1
Java	2	1	1	10	-	-
Objective-C	3	30	39	-	-	-
C++	4	3	3	2	2	2
C#	5	6	7	19	-	-
PHP	6	4	5	-	-	-
Python	7	5	6	25	21	-
JavaScript	8	8	8	17	-	-
Visual Basic .NET	9	-	-	-	-	-
Transact-SQL	10	27	-	-	-	-
Perl	11	7	4	3	9	20
Lisp	15	18	13	12	5	3
Pascal	16	14	82	7	3	21

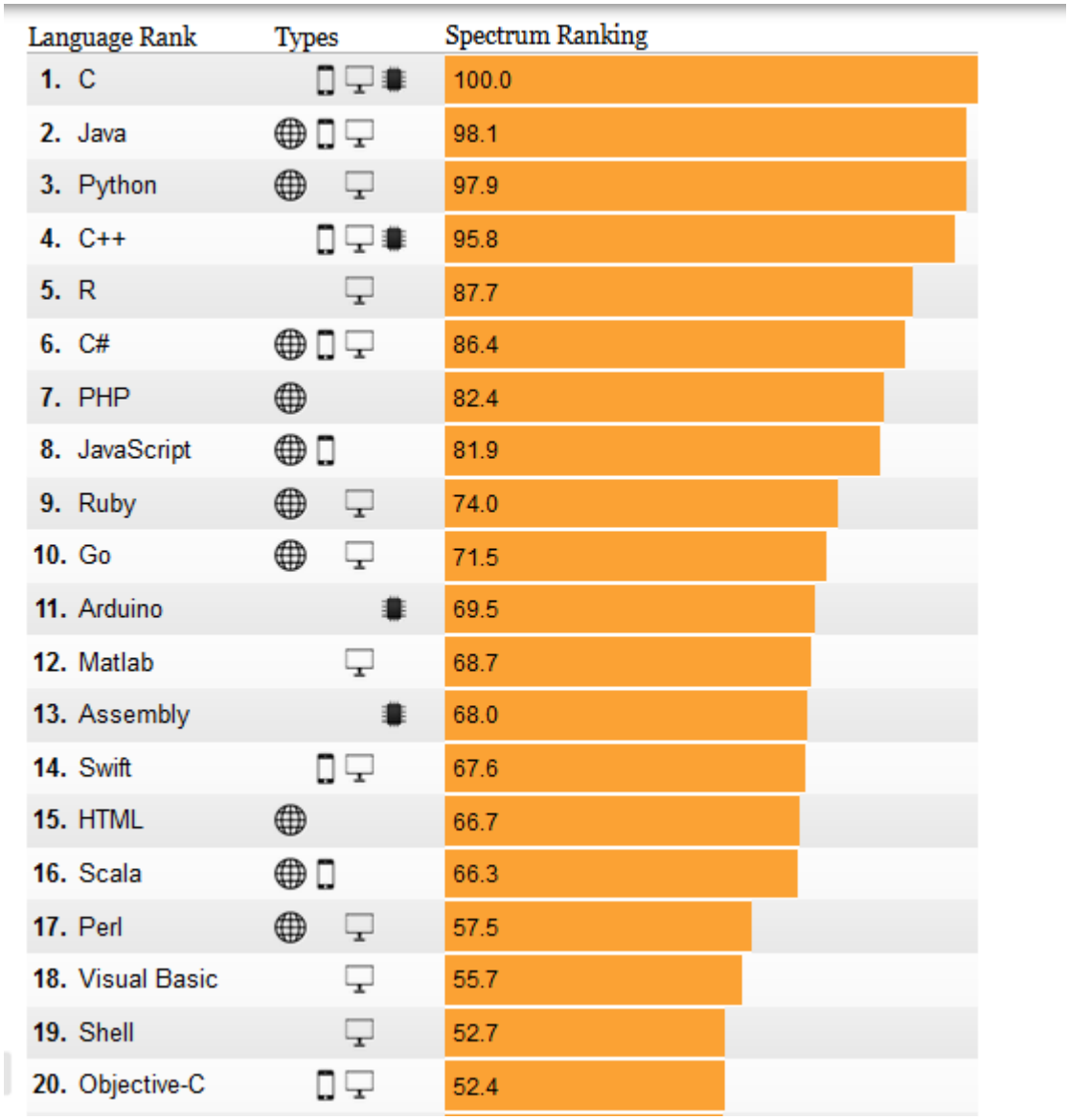
2014 年 4 月程式語言排行榜 TOP20

參考出處：http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Apr 2014	Apr 2013	Change	Programming Language	Ratings	Change
1	1		C	17.631%	-0.23%
2	2		Java	17.348%	-0.33%
3	4	^	Objective-C	12.875%	+3.28%
4	3	v	C++	6.137%	-3.58%
5	5		C#	4.820%	-1.33%
6	7	^	(Visual) Basic	3.441%	-1.26%
7	6	v	PHP	2.773%	-2.65%
8	8		Python	1.993%	-2.45%
9	11	^	JavaScript	1.750%	+0.24%
10	12	^	Visual Basic .NET	1.748%	+0.65%
11	10	v	Ruby	1.745%	-0.23%
12	17	^^	Transact-SQL	1.170%	+0.45%
13	9	vv	Perl	1.027%	-1.31%
14	52	^^	F#	0.966%	+0.83%
15	19	^^	Assembly	0.853%	+0.14%
16	13	v	Lisp	0.797%	-0.11%
17	18	^	PL/SQL	0.782%	+0.07%
18	24	^^	MATLAB	0.760%	+0.24%
19	15	vv	Delphi/Object Pascal	0.746%	-0.09%
20	35	^^	D	0.708%	+0.39%

2016 年 IEEE The Top Programming Languages 2016

http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2016



Language Rank	Types	Spectrum Ranking
1. C		100.0
2. Java		98.1
3. Python		97.9
4. C++		95.8
5. R		87.7
6. C#		86.4
7. PHP		82.4
8. JavaScript		81.9
9. Ruby		74.0
10. Go		71.5
11. Arduino		69.5
12. Matlab		68.7
13. Assembly		68.0
14. Swift		67.6
15. HTML		66.7
16. Scala		66.3
17. Perl		57.5
18. Visual Basic		55.7
19. Shell		52.7
20. Objective-C		52.4

隨著工具的完善及功能的強化，越來越多程式設計師加入了程式開發的行列。和早期單調的文字介面程式相比，各式各樣的奇思妙想、精彩絕倫的應用程式被開發出來。整個軟體產業呈現無限美好的朝氣向前衝。然後又吸引更多的公司及個人投入這個產業，形成了一個正向的向上循環。再加上 Internet 的興起，無疑是軟體產業的一個重要的里程碑。尤其是經過 2000 年的達康泡沫的洗禮，整個產業的發展更趨成熟健康，各種菲疑所思的網站服務興起，每天數以億計的新增網頁，形成以往難以想像的龐大資料庫（BIG DATA），再配合強大的搜尋引擎後，資訊變成可立即傳遞並且隨手可得。這大幅改變了世界的溝通及學習模式。

部份有遠見的個人或企業嗅到了這個千載難逢的商機，於是電子商務開始興起，從 Amazon（亞馬遜）、Dell（戴爾）、Cisco（思科）到 Yahoo、Google、日本軟體銀行再到中國阿里巴巴（淘寶網、天貓），這些網路時代的典範企業也因為搭上了這股風潮而大發利市。

不過，禍福總是相倚，企業新的需求雖然帶來了新的商機，但是對軟體公司而言，又得面對開發平台的改朝換代。這可不是件輕鬆愉快的事，一般的公司光是瀏覽器的版本升級，就必須經過反覆的測試，更別提作業系統的更新、甚至整個開發平台的更新。這個切膚之痛，只有經歷過的人才能深刻的體會。本書試圖提供一個經過實際驗證的開發模式，來協助您及您所屬的企業在開發 ERP 系統時所碰到的各式問題。只要能夠對您有些許的幫助，就不枉費筆者費盡心思的初衷。

開發ERP是一個非常龐大、複雜的工程。想在一本書中完全的詳述是不可能的任務，本書只是一個開頭，希望能拋磚引玉，讓更多優秀的作品能橫空出世。限於筆者個人才疏學淺，難免有不足或錯漏之處，敬請各位先進不吝指正，感恩。

第二章
什麼是 ERP?

泉涸，魚相與處於陸，相呴以濕，**相濡以沫，不如相忘於江湖**。與其譽堯而非桀也，不如兩忘而化其道。夫大塊載我以形，勞我以生，佚我以老，息我以死。故善吾生者，乃所以善吾死也。

莊子、內篇、大宗師

第二章　　什麼是 ERP?

2.1　ERP 的延革及歷史

1960 年代開始，企業開始使用電腦來處理日常的交易資料，以節省人力及提高資料的正確性與時效性，此種方式一般稱為電子資料處理(Electronic Data Processing, EDP)。經過多年的發展後孕育出另一新觀念，就是提高電腦的應用層次，讓電腦能支援組織內更高層次的管理活動，如管理控制與策略規劃等。於是管理資訊系統 (Management Information System, MIS)興起。這些資訊系統的來源可能是由企業自行開發、外包(Outsourcing)取得或者購買現成的套裝軟體(Software Package)。隨著時間的演進，MIS 系統功能逐步的強化，最終在軟硬體不斷的提升、軟體開發技術及相關管理知識的推陳出新下，整個資料系統在質量上有了長足的發展，這就是 ERP（Enterprise Resource Planning）系統的前世今生。

以下是維基百科關於 ERP（企業資源規劃）的定義
http://zh.wikipedia.org/zh-tw/ERP

企業-資源-計劃（**Enterprise Resource Planning**，縮寫 **ERP**），又譯**企業資源規劃**，是一個由美國著名管理諮詢公司 Gartner 於 1990 年提出的企業管理概念。企業資源計劃最初被定義為應用軟體，但迅速為全世界商業企業所接受。現在已經發展成為一個重要的現代企業管理理論，也是一個實施企業流程再造的重要工具。

起源

ERP 的概念是著名產業科技研究企業－Gartner Group 在 1990 年代初期依據資訊科技的發展及供應鏈管理所提出的觀念，以推論製造業管理資訊系統的發展趨勢。其實 ERP 並不是一個全新的系統，它是由 1970 年代的物料需求計劃（MRP），1980 年代的製造資源規劃（MRP II）所逐漸演進而成。

以推動 MRP、MRP II 著名的 APICS 美國生產及存貨管理協會在對 ERP 一詞的解釋中提到：一個 ERP 系統和典型的 MRP II 系統的差異在技術上的需求，例如圖形使用者介面（GUI）、關聯式資料庫、與客戶及供應商的整合、可視需要製作各式報告等。

簡單地說，ERP 是「一個大型模組化、整合性的流程導向系統，整合企業內部財務會計、製造、進銷存等資訊流，快速提供決策資訊，提升企業的營運績效與快速反應能力。」它是 e 化企業的後台心臟與骨幹。任何前台的應用系統包括 EC、CRM、SCM 等都以它為基礎。

ERP 主要的模組有：

生產管理	工程、材料清單（Bill Of Material）、排程、產能、工作流管理、品質控制、成本管理、生產過程、生產工程、生產流程、生產配置
進銷存貨管理	庫存、訂單輸入、採購、供應商排程、貨物檢查、付款請求處理、傭金計算
財務管理及會計計畫	總帳、現金管理、應付帳款管理、應收帳款管理、票據資金管理、固定資產管理
成本管理	賬單、時間和支出、活動管理
人力資源管理	人力資源、薪金名冊、培訓管理、員工班別和出勤管理、津貼、勞健保、績效考核
供應鏈管理	和客戶、供應商、員工之間的各種自服務介面

筆者個人的看法則是

ERP 是貫通、串接整個組織（包含政府、學校、一般企業、非營利組織、財團法人等）的電子資訊系統。此系統會收集、整理、匯總、分析組織所產生的所有資訊，最後成為整個組織協同工作的核心平台。另外可再隨著時空環境的演進，可以再整合適當的子系統擴充不足的功能，變成活的企業大腦。

從上面的這個定義來看，所有的企業、組織在面臨今天地球村的競爭環境下，都應該有所屬的 ERP 系統。不管這個系統是自行開發、委外開發、或是導入市場上現有的 ERP 產品。討論 ERP 是否已死是沒有意義的，因為 ERP 系統和企業間的關係就像是魚和水，魚是絕對無法離開水的，ERP 已經是 現代企業不可或缺的一部份，差距只是系統的規模和行業別不同的屬性罷了。

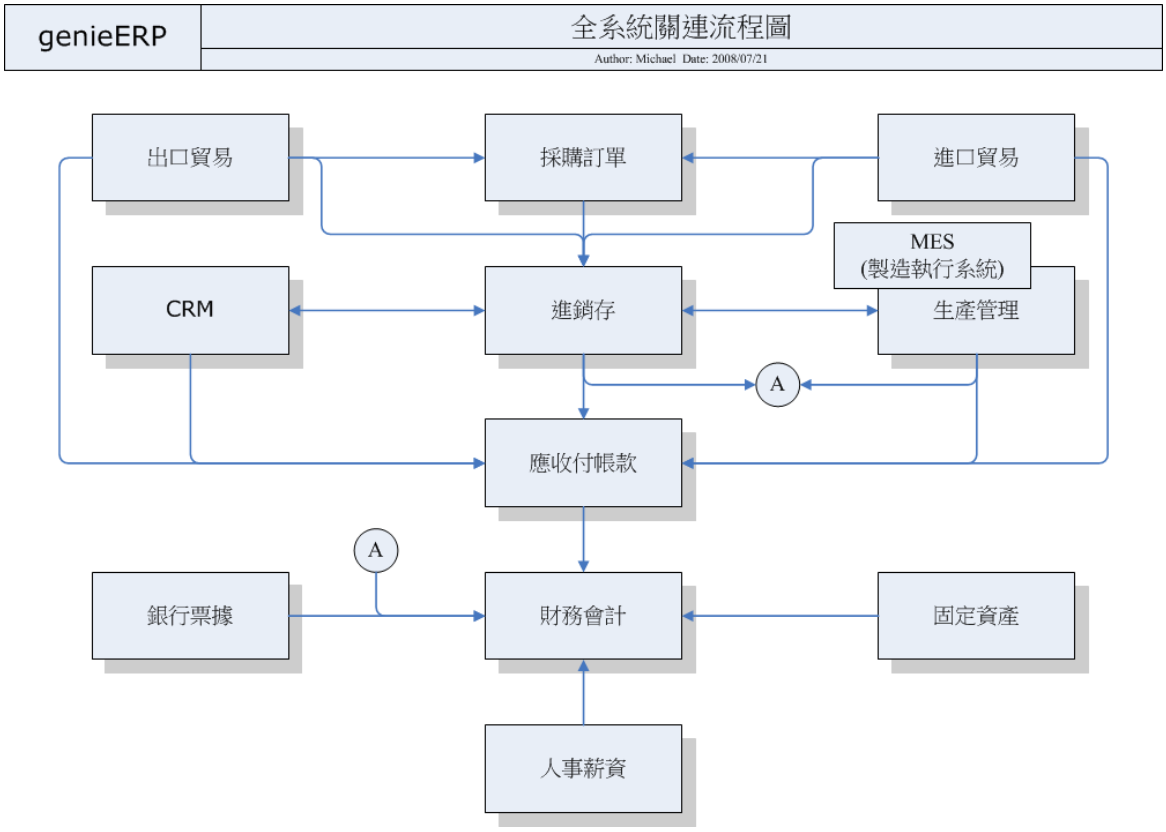
2.2 ERP 的發展歷程

	1970 年代	1980 年代	1990 年代	2000 年代
企業應用軟體	MRP	MRP Ⅱ	ERP	EERP
應用範圍	部門	工廠	企業	供應鏈
組織結構	集中組織	分散組織	分散組織	虛擬組織
資訊系統架構	大型主機	迷你電腦	主從架構	網路運算
需求重點	成本	品質	速度	協同規劃
管理重點	以生產、現場監控與物料規畫為主	強調成本、品質、效率與供料的即時性	強調研發、銷售、生產、配銷、服務與財務內部資源整合與最佳化運用	強調結合內外客戶與廠商之全球運籌管理模式

2.2　系統架構

1. ERP 並不是一個【標準】的系統：不同的軟體公司開發的 ERP 所包括的功能，與另一套 ERP 系統或多或少都有不同之處 ，因此在對 ERP 下定義時，都會傾向於強調已開發的功能及系統的特色。
2. 每一個使用者的需求不同：一家年營業額上百億的大型企業，其營運規模、組織、功能、資源等，當然要比年營業額一億的中小型企業大的多。因此，使用的 ERP 也當然是截然不同。

以下是以 genieERP 系統的各模組關連圖來說明整個 ERP 的架構。



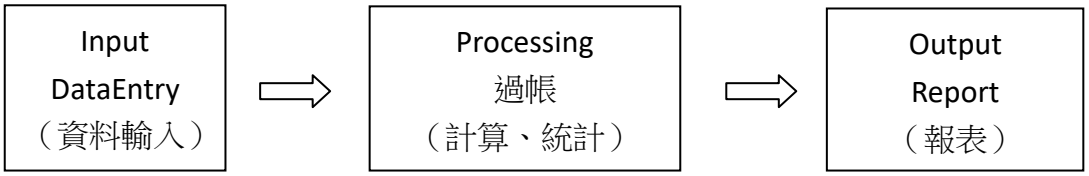
2.3　常見的模組及功能表列

無論 ERP 系統是跨國企業使用的大型程式或是微型企業使用的小型套裝軟體，基本上所有的系統大概都可以歸納分為

模組－子(次)模組－功能－報表

前一節已經用圖表展示了 ERP 的一些常見模組及模組間的關連，有一個顯而易見的重點－最終的資訊都會匯總回財務會計。這是想當然爾，因為企業經營是以獲利為主要目的，而想要瞭解目前的經營狀況，當然必須透過財務相關報表。

鴻海董事長郭台銘先生曾下定義：「系統，等於流程加表單」，雖然這屬於傳統工業時代的說法，但是這並沒有減損這個定義的價值。相對於以功能表列來看 ERP 系統，將 ERP 的每一個功能都歸類為下列三種類別



當然，ERP 這麼多的功能總會有些例外，但以這個概念應該可以涵括 80%以上的功能。以下筆者會整理一些常見的實用功能，方便各位能對 ERP 有一個快速的通盤認識。

下面是 ERP 財會模組的部份表單及功能列表

genieERP Modal Function List

會計系統(1ACT)

＊單　據　選　單＊		Table	核心版
傳票相關作業(ACTM0)			
	傳票資料維護(多幣別)	ACTSUBPOENA_M/ ACTSUBPOENA_D	●
	傳票資料維護(立沖)	同ACTM01	
	傳票常用分錄	ACTTMP_M / ACTTMP_D	●
	批次開立傳票	ACTBATCH_M / ACTBATCH_D	
	開帳傳票維護	同ACTM01	●
	傳票傳輸作業	No Need	●
預算設定作業(ACTM1)			
	部門預算設定作業	FIVACTFINAL	
	專案預算設定作業	FIVACTFINAL	
	不定期資金資料設定(部門)	ACTNONFIXDATABUDGET	
	不定期資金資料設定(專案)	ACTNONFIXDATABUDGET	
財務基本作業(ACTM2)			
	會計科目設定	ACTTITLE / ACTDEPTSEP	●
	會計類別設定	ACTCATEGORY	●
	科目年度餘額查詢	FIVACTFINAL	
財務特殊作業(ACTM3)			
	營業成本公式設定	ACTFORMULA	●
	財務比率分析設定公式		
	現金流量表公式設定		
固定資產(ACTM4)			
	固定資產資料維護		

以下是財會相關的部份報表

＊報　　表　　選　　單＊			核心版
會計基本帳冊(ACTR0)			
	傳票日報表		●
	日記帳		●
	日計帳		●
	現金簿		●
	試算表		●
	總分類帳		●
	明細分類帳		●
	科目餘額表		●
	立沖科目餘額表		
	立沖科目明細表		
	現金流量表(ERP增強計畫)		
損益資料分析(ACTR1)			
	部門別損益表		●
	預算/實際損益表(部門別)		
	營業成本表(部門別)		●
	兩期比較性部門損益表		●

接下來是進銷存的功能及表單

genieERP Modal Function List

進銷存系統(3INV)

＊單　　據　　選　　單＊		Table	核心版	程式代號
進出貨單據作業(INVM0)				
	出貨憑單維護作業	INVSTKBILL5_M / INVSTKBILL5_D	●	INVM01
	出貨退回維護作業	INVSTKBILL6_M / INVSTKBILL6_D	●	INVM02
	進貨憑單維護作業	INVSTKBILL1_M / INVSTKBILL1_D	●	INVM03
	進貨退出維護作業	INVSTKBILL2_M / INVSTKBILL2_D	●	INVM04
	非存貨出貨單	INVSTKBILL5_M / INVSTKBILL5_D		INVM05
	銷貨出庫單	INVSTKBILL5_M / INVSTKBILL5_D		INVM06
	進貨入庫單	INVSTKBILL1_M / INVSTKBILL1_D		INVM07
	進銷存年度統計查詢作業	下 Select Count 計算取得(參考過帳)		INVS0A
調整調撥作業(INVM1)				
	調整憑單維護	INVSTKBILL3_M / INVSTKBILL3_D	●	INVM11
	入庫類別設定	BASADJUSTCATEGORY	●	BASM611
	調撥憑單維護	INVSTKBILL4_M / INVSTKBILL4_D	●	INVM12
	產品成本調整作業	INVSTKBILL3_M / INVSTKBILL3_D		INVM13
	盤點清冊維護	INVCHECK_M / INVCHECK_D		INVM16
	實盤資料輸入作業	INVCHECK_M / INVCHECK_D		INVM17
	盤點調整單產生作業	無 Table		INVM18
借還單據維護(INVM2)				
	借出憑單維護	INVBORROW1_M / INVBORROW1_D	●	INVM21
	借出歸還憑單	INVBORROW2_M / INVBORROW2_D	●	INVM22

進銷存的部份報表

發票立帳作業(INVM6)				
	發票立帳調整單	INVSTKBILL3_M / INVSTKBILL3_D		INVM61
	發票立帳存貨開帳	INVSTKBILL3_M / INVSTKBILL3_D		INVM62
	發票立帳進貨單	INVSTKBILL1_M / INVSTKBILL1_D		INVM63
	發票立帳銷貨單	INVSTKBILL5_M / INVSTKBILL5_D		INVM64

＊報　表　選　單＊			核心版	程式代號
存貨狀況分析(INVR0)				
	現有庫存一覽表		●	INVR01
	歷史庫存一覽表		●	INVR02
	存貨異動明細表		●	INVR03
	低於安全存量表		●	INVR04
	庫存呆滯分析表		●	INVR05
	銷貨成本異動表		●	INVR06
	進銷存明細報表		●	INVR07
	期初期末存量明細表		●	INVR08
	批號庫存一覽表			INVR09
	批號庫存異動明細表			INVR0A
	序號進出狀況一覽表			INVR0B
單據日報明細(INVR1)				
	進貨單日報表		●	INVR11
	進貨退出單日報表		●	INVR12
	調整單日報表		●	INVR13
	調撥單日報表		●	INVR14

上面只是整個 ERP 系統的極小部份功能，但透過這幾個簡單的表列，應該能讓您對 ERP 有更清楚的認識，如果您需要完整的列表，請通過書末的連絡方式即可取得。

本章的目的是讓您對 ERP 有一個基本的認識及概念，如果您想對 ERP 有更進一步的瞭解，請參考書末的參考書目及相關的網址。

參考網址

MBA 智庫百科 ERP	http://wiki.mbalib.com/zh-tw/ERP
Wiki ERP 定義	http://zh.wikipedia.org/zh-tw/ERP

第三章
開發 ERP 的情境及方式

子曰：「君子不器。」
子曰：「學而不思則罔，思而不學則殆。」

論語為政篇

第三章　開發 ERP 的情境及方式

3.1　情境

開發 ERP 系統不外乎有下列幾種情境

1. 獨立軟體開發商（Independent software vendor：ISV）
   此類的廠商通常會開發一個標準的套裝軟體版本在市場銷售，適用於沒有太多特殊需求的企業（通常是規模不大的中小企業）。以下列舉國內部份較知名的廠商

鼎新電腦	http://tw.digiwin.biz/
台塑網科技	http://www.efpg.com.tw/efpg/
聯合資訊	http://www.uis.com.tw/
資通電腦	http://www.ares.com.tw/home/
正航資訊	http://www.chi.com.tw
亞典資訊	http://www.aden.com.tw/
鉅茂科技	http://www.datawin.com.tw/
天心資訊	http://www.sunlike.com/
凌越資訊	http://www.lyserp.com/TWindex.htm
昇峰資訊	http://www.infor-doctor.com.tw
冠理科技(江氏)	http://www.chiangsgp.com/
宣揚電腦	http://www.bethel.com.tw/cht/jp.aspx
科展資訊	http://www.simis.com.tw/index.html
益模管理	http://www.emo-5.com.tw/
捷克仕移動	http://www.gex.com.tw
頂泰興資訊	http://www.kptc.com.tw/
動量數碼	http://www.hotzsoft.com/
高格亞翼	http://www.leehem.com.tw/

優點：購買後經過基礎的教育訓練後，即可上線使用，屬於物超所值的通用版本。
缺點：如果有特殊的行業別需求，就會綁手綁腳，要增加很多的手工作業（通常是 Excel 檔的整理計算），解決的方式通常是透過二次開發，以滿足客戶的需求。

本書的大部份觀點都是以此種情境為討論的範本。在台灣，大部份的 ISV 廠商規模都不大，但是都有一定的開發能力，能夠生存下來的廠商通常在行銷及資金上相對占優勢（也就是有品牌力），新進廠商如果沒有非常突出的特點（例如開發出 Web 版或雲端版等），就很難途破重圍。即使掌握新的技術開發出新版的程式，也不保證就可以在市場上取得成功。這一點筆者個人有很深的感觸，研發出身的人往往過於輕忽或不瞭解市場，往往因非產品的因素失敗，此點僅供各位參考。

2. 企業自主開發

企業內部成立資訊部門，自行研發內部使用的資訊系統，通常大型企業或特殊的業別（如銀行、工程營造、科學園區的廠商等），會因為行業的特殊需求或基於安全性考量、業務機密等原因，才會考慮自行開發。

優點：開發的系統完全符合企業現階段的需求，且完全掌握所有的技術，不會受制於軟體廠商。

缺點：開發時程長、技術變動大、成本過高且失敗機率很大

如果自己的研發團隊能力很強，且熟悉企業內部流程，當然可以考慮此種方式。但是一般而言，面對日新月異且層出不窮的新技術，組織內部的研發人員，一般都很難與時俱進，此時適度的轉包部份需求（通常是前端 UI、或是雲端版本等）給 ISV 廠商，趁機學習新的技術，再昇級自己的開發團隊，也就是藉由外部顧問的協助，讓資訊部門不至於過於封閉，這才是王道。

針對這類型的企業，也可以參考本書所提的標準開發 SOP，擷取有用的概念，加到企業原本的 SOP，特別是關聯式資料庫的深度使用，請多用心思考，應該會有所助益。

3. 委外開發

如果企業的部分資訊系統並非核心的業務，但標準套裝軟體無法滿足需求，此時就可以考慮委托外部的軟體開發商來開發 ERP 系統。

委外開發可以分為二種方式

A. 僅程式開發外包

這種方式的主控權仍在企業本身，通常是因為內部研發人力不足、或是想引進新的技術、或是想要縮短開發的時程。至於專案進度及詳細的規格規劃等主要的管理，都仍然掌握在企業的資管（Management Information System ：MIS）部門。此時企業 MIS 的角色，主要是溝通協調及進度控管，此種情況通常會要求軟體廠商要派員工駐點，由企業指定的專案經理（Project Management：PM）安排其行程。這樣的方式在人力調派上較具有彈性。但因為駐點工程師的程度良莠不齊，而且還有流動率過高的負面效應，負責的PM要有能力解決這些issue，不過總而言之，這種委外的方式能兼顧安全性及人力調配，如能排除前面所提的負面效應，倒也不失是一個不錯的開發方式。

B. 全部委外開發

和前面提到的方式不同，大部份的中小企業受限於規模，MIS 部門的人力都不足，甚至沒有 MIS 部門，此時會由熟悉公司運作的資深人員（通常包含財會人員），用大部份口述，少部份企業規章文檔等方式，將全部的 ERP 開發工作委外開發。相對於前一種方式，全部委外的風險相對偏高，因為企業內部有非常

多的枝枝節節，很多的制度、規定及多年來延續下來的潛規則，狠多是沒有形諸文字的；這在規格書上是很難說清楚、講明白的，這也是狠多專案最終都無疾而終的跟本原因。

除非萬不得已，筆者不建議使用這種方事開發，筆者會先建議這類客戶，先導入標準版本、先上線，待內部使用者都習慣操作介面及流程後，再用【二次開發】的方式，依客戶的需求修改，也就是先導入現成的 ISV 廠商的標準版本，再改成符合企業需求的二段式上線方式。這會比完全委外的成功機率更高，而且投入的成本也較低。

不同的開發情境，專業領域知識（Domain Know How）的提供者就會有所不同，經驗豐富的顧問師，很多本身就是某個行業的資深主管轉任，對其所專精的行業，其了解程度甚至比被輔導的企業人員更專精。由其主導的專案，當然成功率就高，反之亦然。

3.2 專案開發方式

1. 瀑布式開發

5 大階段是採取線性進行的，一個階段完成後在接下一個階段。

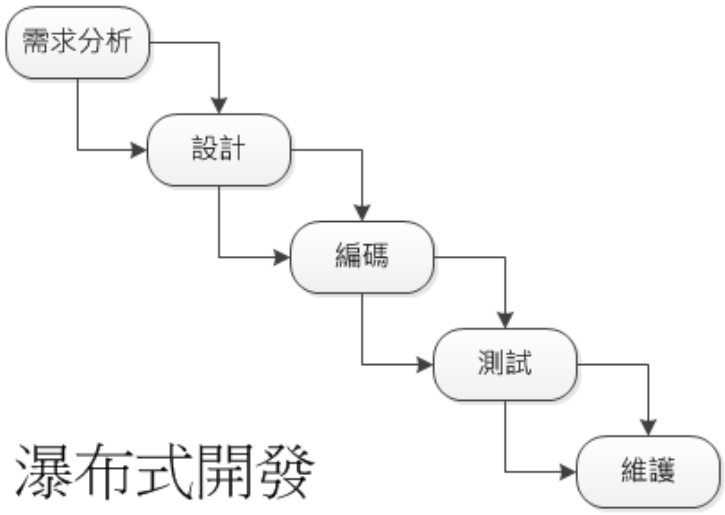
優點:

適合開發商業用軟體

一個版本一個週期

缺點:

發覺錯誤的時間點太晚，風險太大



瀑布式開發

2. 螺旋式模型

每一個螺旋都代表了一點點的需求、設計、實做、測試，沒問題才繼續做下一圈的螺旋。

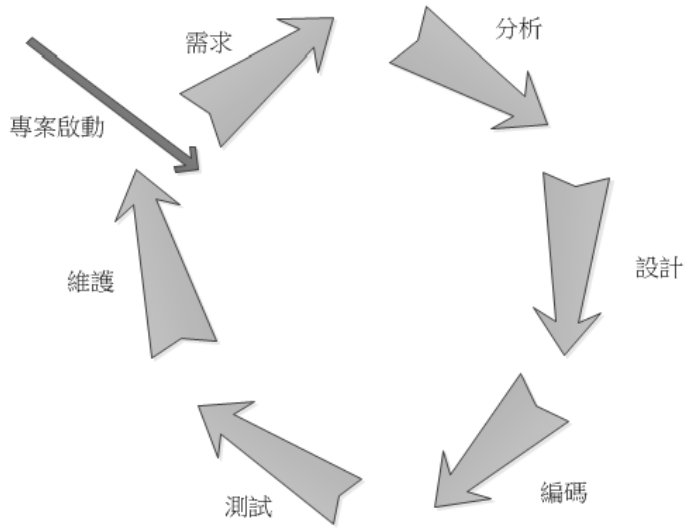
優點：

在發展初期找到可能問題，避免日後重大錯誤的發生

缺點：

發展時間過長

不適合商用軟體的開發



螺旋式模型開發

其他還有諸如敏捷式開發（Agile）等開發模型，請自行參考相關的資源學習

http://agiletalks.blogspot.tw/2014/03/blog-post.html

http://wiki.mbalib.com/zh-tw/%E6%95%8F%E6%8D%B7%E5%BC%80%E5%8F%91

http://zh.wikipedia.org/wiki/%E6%95%8F%E6%8D%B7%E8%BD%AF%E4%BB%B6%E5%BC%80%E5%8F%91

要採用哪一種開發模式，通常都是系統架構師說了算，一般而言，由於 ERP 系統的功能固定，筆者通常會採用開發初期螺旋式模型，等到確認框架後，就還是會採取瀑布式的開發模式。一來簡單，再者方便管理。當然這並不是絕對的，可依不同企業的需求自行決定。

第四章
資訊龍捲風

北冥有魚，其名為鯤。鯤之大，不知其幾千里也。化而為鳥，其名為鵬。鵬之背，不知其幾千里也；怒而飛，其翼若垂天之雲。是鳥也，海運則將徙於南冥。南冥者，天池也。

莊子、內篇、逍遙遊

第四章　資訊龍捲風

在日漸蓬勃發展的軟體產業中，基於企業內部管理需求而生的企業資訊化系統也就順勢而生。早期因為受限於電腦硬體的規格及管理的複雜度相對較低，大部份的 MIS 程式都是記錄企業交易內容（如客戶訂單、出貨單、傳票等）及報表分析為主。而隨著電腦硬體功能的日趨強大，電腦所能提供的強大運算能力，提供了軟體開發者一個更寬廣的舞台。

與此同時，關聯式資料庫（Relational Database Management System：RDBMS）開始興起，並逐漸成為企業 e 化的儲存要角。在各家廠商的良性競爭下，它的功能日漸完善及強大。更另人欣喜的事，各主要廠商原本自行開發的 SQL 語言（這是關聯式資料庫提供的一組語法，可以直接和資料庫溝通，它是一種較接近傳統的程序性語言，語法類似C），開始有了美國國家標準協會（American National Standards Institute，ANSI）所制定的標準。這讓各家軟體廠商所開發的關聯式資料庫，有了一致的語法標準，讓不同的關聯式資料庫，只要遵循標準語法，都有可能可以互相轉換。

筆者初初開始接觸關聯式資料庫時，只是很單純的把它拿來開資料表（Table）、新增些欄位（Fields）、寫幾個簡單的 View，主要是用它來儲存資料。基本上只使用了它最基礎的 20%功能。據筆者了解，這也是目前業界的常態。如果您也是這樣用關聯式資料庫，筆者就真的要為 RDBMS（如 MS-SQL、Oracle、MySQL、PostgreSQL、FireBird 等）抱屈，儲存資料當然是它的核心且必要的功能，但想要真正的發揮它的效能，就必須善用它提供的預存程序（Store Procedure）、User Define Function、Trigger 及 View。市面上有很多這類的參考書籍，筆者個人推薦楊志強老師的相關著作。楊老師的著作內容深入淺出，範例豐富且實用，實屬難得的佳作，筆者個人也從中穫益良多。

然而，也許是因為不熟悉，也許是觀念轉換有困難，真正以資料庫為核心的應用程式並不多。在稍後的章節中，筆者會就此觀點詳加說明，這也是本書的一個重要的基石。

ERP 產業是屬於資訊服務業的重要基礎核心，它的服務對象是以企業為主。近年來由於 Internet 的普及，直接導致中國逐步成為世界工廠，再加上全世界的企業都感受到地球村形成以後巨大的競爭壓力。部份嗅覺敏銳的企業開始善用新科技帶來的競爭優勢，並取得巨大的成功。如 Amazon、Dell、Cisco、Apple、FaceBook 等，這些成功的案例，讓企業導入 ERP、BI、BPM 及電子商務等系統成為一股無可抵擋的風潮，筆者稱之為資訊龍捲風。這股資訊龍捲風的影響既深且遠，全然改變了人類的生活，再加上新的管理相關學說（如六標準差、世界級管理、平衡記分卡等）的引領、相互激蕩之下，使得企業不斷的提出各式新的需求以應對新的全球競爭態勢。

這些需求被提出後，讓相關的資訊軟體廠商一則以喜，一則以憂。喜的是新的需求

若能被滿足，會給公司帶來新的獲利模式，憂的是新的需求帶給研發部門極大的壓力。因為原本的開發團隊往往無法完成這些新的需求，想到解決這些新需求，往往需要學習一些新的開發工具及程式語言（如 Java、PHP、ASP.NET、RoR、Java Script 等）及完全不同的開發框架。這在原本的架構是根本就不存在的。而在此同時，舊有的系統仍然是公司獲利的金雞母，是主要的收入來源，所以原本的研發團隊主力仍然是在維護舊系統，對於新版本只能投入有限的部份人力，所以進度當然不會太快，而且舊版本的所有功能都必然會被要求加入到新的版本中，所以會造成負責開發新版本軟體團隊的更大壓力，在人力不足、又被要求功能要完整的情況下，可想而知，新版本很容易就會胎死腹中，這也說明了很多的新技術、新產品或新的應用，都是由新創的小公司率先推出的主要原因。而且很多老牌的軟體公司也往往敗在這個關卡而黯然出局。

與此同時，有非常多的程式設計理論被提出，「物件導向」是其中影響最為深遠的理論之一，幾乎已經成為開發語言的標準。另外像物件關聯對映（Object Relational Mapping，簡稱 ORM，或 O/R mapping）、MVC 模式（Model-View-Controller）是軟體工程中的一種軟體架構模式，把軟體系統分為三個基本部分：模型（Model）、檢視（View）和控制器（Controller）等。也都成了主流開發架構。

而 ERP 系統也歷經多年的演進，由早期的記錄資訊開始走向決策支援系統（Decision Support Systems：DSS），這是一個重大的轉變。現代的企業需要的是在瞬息萬變的市場快速反應並做出「足夠好」的決策，這牽扯到戰略和戰術的層次，也就是經營和管理的的深層討論。關於這個議題，筆者不多作著墨，因為這非我所長。筆者推薦柳中岡教授所著的 2 本好書，請有興趣的同學自行研究參考

1. 漫話 ERP

   這是一本深入淺出，但引人深思的好書。柳教授用很淺白的文字，非常清楚的說明了 ERP 的觀念及和管理的關係。是一本適合普羅大眾的 ERP 入門經典。
2. 世界級管理的 28 堂課

   本書主要在探討世界級管理（World Class Management：WCM）這個可促進管理優化的技術，延續並深化柳教授先前關於 ERP 的實務經驗，它可以協助企業提升管理的位階，讓管理更合理化。本書是屬於大學高年級的教科書級別的好書，需要多次反覆精讀，必定會有很深的體悟。

企業經營本來就是以獲利為優先，這是資本主義的常態。同理，企業導入新的資訊系統也是同樣的思維。也就是新的資訊系統並不應該是個昂貴的科技玩具，而是必須協助企業強化獲利。近年來大行其道的電子商務，就是一個明顯的例子。電子商務的盛行，大幅改變了人們的消費習慣，越來越多的交易是在網上進行，中國的淘寶網和美國的亞馬遜（Amazon）就是極為成功的 2 個先行者案例。然

而因為沒有（或太晚）跟上這股潮流而失敗的企業也比比皆是。

電子商務是如此的風行，然而電子商務系統平台的背後，一定要有一個強而有力的 ERP 系統當作後台支柱。所有的生產、銷售財務、應收付帳款、訂單等都必須有一個良好的系統來管理，並即時回饋重要的資訊。這又會和「資訊整合」這個重大的議題相關。也就是「資訊孤島」的這個老問題。如果企業的 ERP 系統和電子商務系統是由不同的開發商提供，如何無縫介接又是一個另人頭痛的問題。如果有可能的話，筆者會建議統一由一個團隊來承接處理，這樣就沒有整合的問題。如果現實的狀況就是 2 個或以上的供應商，那就必須要求廠商提供完整的規格文檔及實體關係模型（Entity-relationship model：ER），在資料庫層直接透過 Store Procedure 或 Trigger 去做自動同步及發送 mail 的動作，除非資料庫層無法同步，才寫應用程式（Application Program：AP）來處理。

在商周出版 2000 年出版的【雄霸天下－思科成功的奧秘】一書中的第七章「邁向虛擬企業」中，曾經有以下的內容：

許多財務部的員工把重心放在無效率的地方，……，卡特抱怨說，組織並未有效支持公司，卡特後來將該部門多出的預算幾乎全數投入資訊科技。

資訊系統因應之道是開發線上系統，讓思科財務人員幾乎在沒有時差的情況下，就能查看最新的交易記錄，執行資訊系統（CIS）整合到思科的內部網路，對經理和主管提供思科接單、出貨及未交貨訂單的詳盡資料，同樣的，網站上決策支援系統（DSS）傳輸可另外訂作的銷貨追蹤報告給經授權的人員。要串接這些流程並非易事，……，自從財務資料連線後，只要花兩天時間就能結算一季的帳冊。

……，強力應用這些網路連結正是思科樂於大吹大擂的驕傲。實際上，思科也靠超強的連結採收累累的果實，觀察每日最新信息揭露的市場變化，思科能隨時調整銷售目標或聘請新進員工，採取行動的時間比對手早了好幾個月！

任何對 ERP 成效有懷疑的人，我都會強烈建議他們好好的仔細讀一讀這本書。思科系統（CISCO）可不是軟體廠商，他是全世界最大的數據交換器、路由器和其他網路相關設備產品的製造商。該公司市值在二ooo年三月十七日曾經打敗過微軟，一度成為全球市值最大的上市公司。

ERP 能否發揮效用，很多時候是非戰之罪，就像之前說的，企業的經營有其戰術和戰略。但是想要管理到位，ERP 勢必是重中之重。

第五章
大道至簡

有物混成，先天地生。寂兮寥兮，獨立而不改，周行而不殆，可以為天地母。吾不知其名，字之曰道，強為之名曰大。大曰逝，逝曰遠，遠曰反。故道大，天大，地大，人亦大。域中有四大，而人居其一焉。人法地，地法天，天法道，道法自然。

老子道德經

第五章　大道至簡

宇宙中的萬事萬物都有它一定的運作規則。上至日月星辰，小到如肯德基的炸雞標準流程，都是如此。ERP 系統當然也不例外。

要開發一套好用的 ERP 系統並不是一件簡單的任務，開發團隊必須對全系統的所有細節都了然於胸，這必須主其事者（系統架構師）長期浸淫在這個領域，且不斷的歸納、整理、反思，最後才能略有小成。這個學習的過程有點類似禪宗的頓悟。頓悟絕不是莫名其妙就從天下掉下來的禮物，而是長期針對某一個主題深入的研究後，在某一個時間點產生質變的進階過程。對 ERR 系統而言，就是必須透過不斷的學習、思考、靜心沉殿後，將看似複雜的系統功能，規納為簡單的決策樹圖表（或其他類似的表格等）的一個理性的歸納的過程。

您也許會想說：「拜託，Michael 你能不能講點白話文，不要讓人看的滿頭霧水？」以下就以進銷存系統為例，實際舉例說明

	數量	成本	帳款	績效利潤
進貨單	▲	○	應收▲	※
出貨單	▼	※	應付▲	○
調整單	▲或▼	○	※	※

上面的四個主題，我稱之為進銷存系統的四大天王，也就是四大要素。將看似複雜無比的進銷存數百支程式及報表，歸納為四，然後將所有的單據跟這四大要素掛勾，就形成了決策樹（Deciion Tree）。

下圖是較完整的決策樹（Deciion Tree）

genieERP 單據決策樹圖

		Flag	理論數量	實際數量	成本	帳款	業績	發票	傳票	說明
	OPO									
1	報價單	21								
2	客戶訂單	22								
3	請購單	24								
4	採購單	23								
5	採購驗收單	25								
6	採購建議清單	26								
	INV									
7	進貨單	31	＋	＋	●	●		●	●	
8	進貨退出單	32	－	－	●	●		●	●	
9	調整單	33	＋/－	＋/－	●				●	
10	調撥單	34	＋/－	撥入＋/撥出－						
11	出貨單	35	－	－		●	●	●	●	
12	出貨退回單	36	＋	＋		●	●	●	●	
13	借出單	41		－						與帳款無關, 僅進銷存內部使用代號
14	借出歸還單	42		＋						
15	借入單	43		＋						
16	借入還回單	44		－						
17	進貨入庫單	91	＋	＋	●				●	採購轉入/驗收轉入/進口轉入會有成本, 但非轉單者要自行輸入
18	銷貨出庫單	95	－	－			●(報表可選)		●	

筆者不才，搞 ERP 的第三年才整理出這張簡單的圖表，不過有了這個開始，其他模組就很快的都進入狀況，而且都有更深的體會。

筆者習練陳氏太極拳多年，有一點初淺的理解，發現 ERP 系統的學習及體會程度，和太極拳的境界提昇都有類似的概念，也許這就是所謂的「萬變不離其宗」吧！因此試著用對比的方式來說明ERP的學習進程(等級)。因要開發ERP，必須先深入的理解ERP，而真正了解 ERP 後就會發現「大道至簡」，因為所有的 ERP 模組都可以歸納為幾個主要的要素。

初學太極時，要求一招一式務必要符合規矩。無論是手、腰或腳的行進路線，甚至是眼神，都有嚴格的規範，要求基本功要紮實，雖然因為初學尚無法體會真正的鬆柔，但是練習時，還是要儘量依「逢轉必沉、鬆腰落胯、沉肩墜肘，用意不用力」等基本原則儘量去做。「立身中正、鬆腰活胯、虛領頂勁」的機本要領也要時刻牢記。此時是學習的第一個階段，「見山是山、見水是水」是完全的學習模仿階段。此階段實際練習比思考重要（當然思考也是要的，拳經、拳論也必須細讀，只是比較起來，實作更重要）。要讓身體經由不斷的練習套路，實際體會鬆柔的意義，讓身體熟練到有自然反應，讓身體放鬆不要用力，並讓身體各部位（包括眼、手、胯、腰、腿等）熟悉整套拳架的運行軌跡及定位。這是很重要的築基階段，需要花費大量的時間練習（實做）。ERP 入門的第一個階段也是一樣的道理。

這個階段學習 ERP 首要之務是熟悉所有的功能。要依 ERP 的模組別，一個功能、一個功能的學習瞭解。並一定要藉由案例的編排，實際的輸入資料，理解簡單的資料流程以及每一個表單的用途、有哪些主要欄位等。筆者初學系統時，甚至連功能表都給背下來，雖然意義不大，但也藉此說明基本功要多練。在這個階段著重在實際的模仿操作要熟記 ERP 的標準操作介面、常用功能資料建檔等事務性的操作。學習時先求廣度而不求深度，也就是先概括性的學習所有的功能（包含所有的表單及報表）。

太極的第二個階段，謂之「懂勁」。
這個階段是開始配合腹式呼吸打拳，經過了第一個階段的長期練習，拳架套路基本上已經熟練，身體已經開始放鬆，也慢慢體會出打拳時「逢轉必沉、主宰於腰、形於手指」的弧線往復動作，並開始體會週身一體的「整勁」。初始時還會有些刻意，但隨著時間日久、功力漸增，和第一階段會有明顯的差異（進步）。

這在開始練習推手後，會有更明顯的體會，初習推手時，一般同學很容易因為「不想輸」而忘了「用勁不用力」的鬆沉拳理，很容易陷入「鬥牛」的窘境。一般而言，這一定會困擾習拳者一段時間，此時老師的親身指導就會起很大的作用。在這段期間，老師會反覆的不斷的耳提面命習拳者正確的拳理，並且親身示範。當然，自己不斷的反思及和老師討論心得還是最重要的，因為「師父引入門，修行在個人」。最終會形成屬於自己的風格，到了這個階段，拳架風格也會隨之變化，會將在習練推手時的體會，反饋在拳架中，更鬆、更沉、更有整勁。經過多年的觀察，同一個老師教出來的弟子，為何

總會有一些不同（但還是看的出師承），我個人的觀察，應該都是在這個階段中，會因為每個人的職業、環境、經驗、歷練等，導致對拳理的體悟不同，因此會有一點自己的個人風格。這很好，無所謂對錯。

ERP 的第二個階段，謂之「整合」。
也就是開始由對單一功能的熟悉，轉為流程多功能的貫串。事實上，ERP 系統很少（應該說沒有）某功能是獨立於其他系統功能的，通常是一組功能完成企業職能的某個作業流程。例如業務接單出貨流程

業務報價 → 客戶下單 → 新建客戶 → 客戶徵信 → 倉管揀貨 → 出貨 → 收款

以上的流程，除了三張單據（報價單、訂單、出貨單）外，還有客戶、業務人員檔、產品、部門、倉庫等的基本資料建檔。許多個類似這樣的流程，就組成了一套模組（如進銷存），這是點、線、面的系統組合方式。這是比較基礎的整合，也比較易於理解。

不過，一般我們會特別強調「整合」，都是針對跨模組的流程而言，而且通常和財務會計有關連。這應該也很容易理解，企業經營本就是以「盈利」為最主要的目的。整個 ERP 系統自然就是以財務模組為核心，自然多數的整合都會和它有關。

最經典的「整合」範例，是『傳輸傳票』這個功能。所謂的『傳輸傳票』指的是某張會影響財報的單據（例如進貨單、出貨單、應收付票據兌現、已收帳款單等），透過此功能，可以由系統自動產生會計傳票的一個 ERP 系統居家必備的程序，以下以出貨單為例說明：

要真正搞懂 ERP，就不能不懂初級會計。而且對財務的了解越深入越好。這也是很多 ERP 顧問師都有財務背景出身的重要原因。在導入 ERP 系統時，財務部門主管通常都會是 ERP 上線委員會的重要成員，系統架構師和專案經理（Project Manager）能否用「財務語言」跟客戶溝通也是一個重要的關卡。例如當客戶詢問有一批 A 材料要認列存貨跌價損失時，系統該如何處理？（原購入成本 10，欲降為 6，數量 1000 PCS）
解法之一如下（會有多種不同的解法，只要符合會計原則即可）

1. 先輸入一張調整單（會影響成本）

2. 產生傳票

到了這個階段，對 ERP 已經有一定的認識，差別只是對更多模組及更多功能的細節更深入的理解。絕大多數的朋友終其全部的 ERP 生涯都是在這個階段。說實在的，這已經足夠了，已經算是資深從業人員了。要讓系統正常運作已經不是問題。對複雜的功能（如 MRP 物料需求展算）也都有了正確且完整的認識。既然如此，那還有下一個階段嗎？

就像太極，能進入第二階段「懂勁」，已經非常不容易了，沒有十年的苦練，都不太能體會（大部份的情況是說的一口好拳，但是做不到）。但是依然存在有更高境界的追求。

附記：限於筆者的個人經驗，並簡化範例內容，此處所提的太極拳的境界，僅限於拳架的習練，而不論內勁（內功），真正的太極博大精深，包含筆者在內的大部份同好，終吾人一生都只能不斷的追逐，很難真正的登峰造極，此處請各位讀者明鑑。

更高境界就是第三個階段－階及神明。
這是一種隨心所欲，放空無我的狀態。和禪修類似，坐到一支好香一樣的身心舒暢。但也是一樣的難逢難遇。此時身心全然的放鬆，外界的環境完全被隔絕，會進入一種心無雜念的的自然律動，實在難以言傳，只能自己體會。身體會自然的行拳、毫無刻意、身隨意轉。無怪呼太極又被稱為武禪、動禪。

筆者慚愧，追隨陳氏太極拳明師洪挺嚴老師習拳十多年，因俗務纏身，未能刻苦練習，只是偶而才能體會到這個境界。但是偶有所感，偶爾能體會到這種境界，那真是幸福。

此處借太極以譬喻 ERP，一方面是個人的經驗使然，另外一方面也希望讓讀者能更容易體會及理解所謂的層次的提升及進階的意義。否則就一個簡單的太極拳套路可以打個 30 年？看來看去都一樣，還真無法想象有何不同。

ERP 的第三個階段，稱為「彈性設計化」
這是一個對整個 ERP 系統開始反思、歸納、整理的階段。在第二個階段，功能面的規格細節大致都已經就緒，對 ERP 也已經有了一定程度的認識，即使客戶提出新的需求，在已有的基礎上也不難解決。

但是客戶（或公司內部同仁）對 ERP 功能的需求是永無止境的。特別是如果貴公司是屬於 ISV（獨立軟體開發商）的話，那面對的客戶的需求更是光怪路離、千奇百怪。

一開始，客戶合理的建議（一般而言，客戶大部份的需求聽起來都很合理），我們一般都會安排時間加入到系統中，有經驗的系統分析師更會利用擴充參數，利用參數設定來控制某些功能是否能被某客戶使用，來達到單一版本多客戶使用的目的。但是日積月累下來，看似永無止境的二次開發需求，終究會讓開發團隊筋疲力竭。那會是一場災難，相信我，你不會希望碰到的。那怎麼辦？在這種壓力下，人類進步的原動力－偷懶，就會驅使架構師開始思考，如何才能解決這個困境。

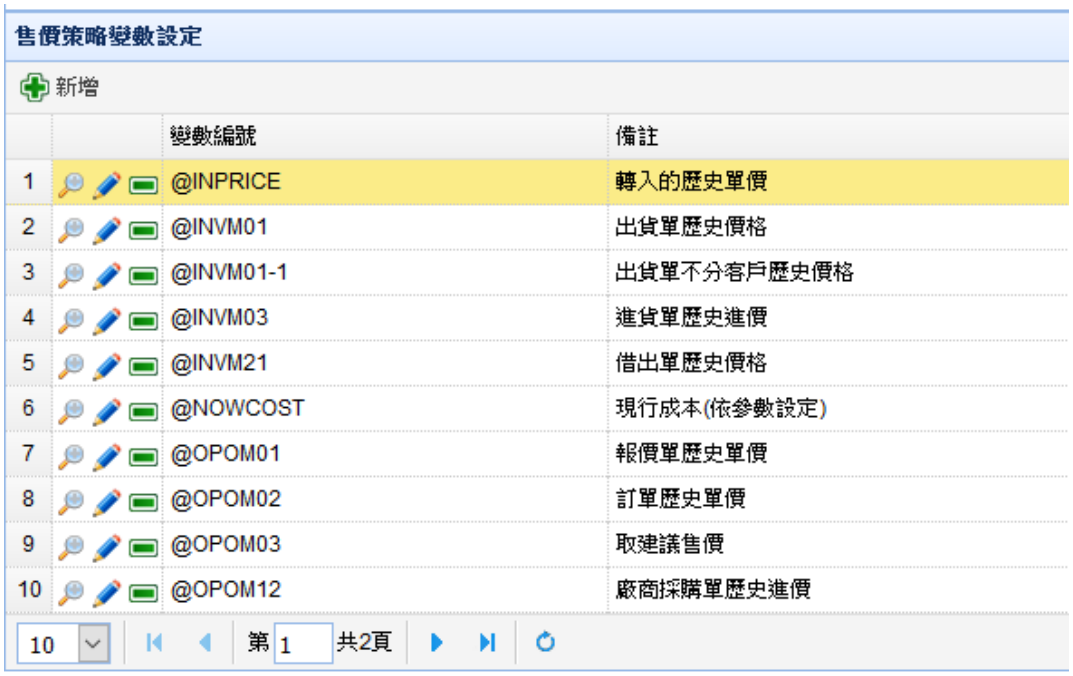
以筆者親身經歷為例，前後共參與四個不同版本（公司組織）的 ERP 開發，及大大小小上百個客製專案。系統都不是很龐大，但是功能基本上是很完整的（麻雀雖小、五臟俱全，四個版本的功能，隨著對 ERP 了解的逐漸深入，而逐步增加）。但即使如此，一直到最後一次開發 WebERP 系統的後期，才逐步的將某些常用且經常有擴充或自行設定需求的功能，著手整理為可以彈性設定、自行擴充的機制化作業，並實作為系統的一部份。細節作法，筆者計劃在後續的幾本書中再詳細的討論說明，此處先列出幾個已實作的功能供各位參考。

1. 將所有的報表（包括單據列印）機制化
2. 登入時，系統自動依 LoginID 提示待辦事項，且所有的提示內容，皆能自行設定來源，每個人看到的內容都不相同
3. 行事曆可自設定顯示資料來源
4. 傳輸傳票內容可自行設定
5. 薪資變數自定義設定作業，幾乎無所不能的公式設定
6. 所有訂單、進銷存單據的單價。可以自行設定計算公式，並可調整優先次序

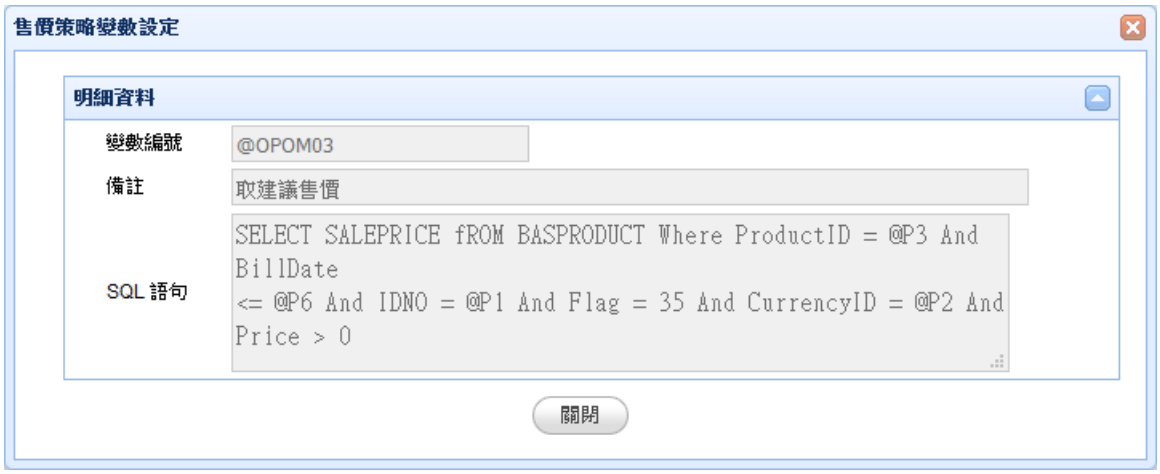
等等。

恐口說無憑，下面就以進銷存及訂單系統中，不同單據新增時，能依設定自動取出單價為例，簡單的實做一次加以說明。

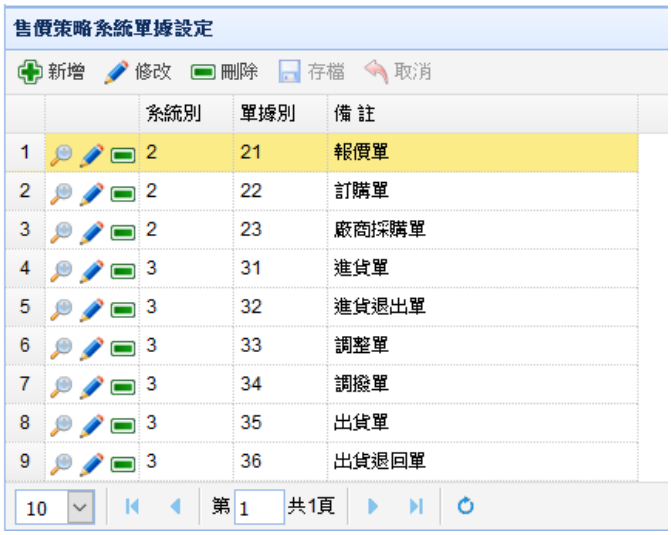
1. 首先，設計一個單價變數的作業畫面，例如，產品檔的建議售價、或是此客戶上一次的出貨單價。



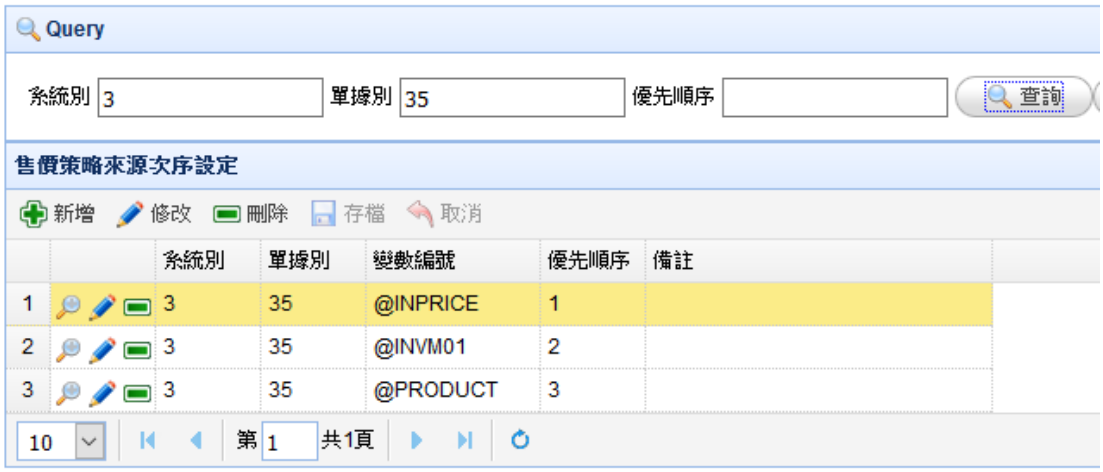
下面是其中一個變數設定(產品檔的建議價) 的內容，看的出來是簡單的 SQL 語法組成。



2. 再來，定義系統中有那些單據需要定義



3. 接著，就是定義每一張單據取的單價的次序，其內容為步驟一中定義的變數



然後，只要在程式中，透過呼叫一個統一的 function，傳入不同的參數，就可以取得單價，如果計算的邏輯有變動，也只要透過修改設定，而不是修改程式，就可以輕易地達成。請相信我，如果你有多年的維護系統經驗，你會愛上這種寫法的。

```
45
46      function doneGetBack() {
47          $('#JQBatchMove1').BatchMove('Move');
48          return true;
49      }
50
51      function getPrice(dataRow) {
52          uBatchGetPrice("dataFormMaster1", "IDNO", "CURRENCYID", "INVOICETYPE", dataRow.PRODUCTID, "35", "dataGridDetail",
53      }
54
55      //現有庫存查詢
56      function openStockGrid() {
57          initInfoDataGrid($("#dataGridFivinvfinal"));
58          uOpenDialog("JQDialogFivinvfinal", "dataGridDetail", "PRODUCTID", "PRODUCTID");
59      }
```

只要透過前面的設定，就可以完全客製化出客戶所需要的計算邏輯，重點是不必不斷的修改程式，更新版本。

這些設定和做法，都和筆者在書中反覆提及的「以資料庫為開發核心」的思惟一致。事實上，大部份的設定內容，都是在寫 SQL 語法。凡事有利必有弊，這也是這種寫法的一個限制，因為並不是所有人（包含程式設計師）都熟悉 SQL 語法。

但是這樣的寫法，無論是由程式碼的後續維護，或是毋須修改原始程式，即可無限擴充資料來源的觀點來看，都不是太大的問題。花點時間投資在學習 SQL 語法，中長期看來都是回報率很大且很久的好投資。

也許這樣的說明還是有些空洞，筆者再以先前曾舉過的傳輸傳票的功能為例，以程式的實際行數來說明兩種方式的差異。在第二階段時，筆者曾經統計過「傳輸傳票」的程式碼，大約是 9000-10000 行左右（用 Delphi5 開發，大約有 20-25 張單據要寫傳輸的程式）。因為當時的寫法是每一支表單程式（如出貨單、進貨單、應收付票據等）就寫一段程式，每一支程式大約都只有幾百行程式碼，但是在不知不覺中，程式碼就長大到

了近萬行。不但如此，每次有新增表單，如維修單時，就得再寫一段類似的程式，真是永無止境。

第三階段的寫法，是在仔細分析過第二階段的程式碼後，發現傳輸作業都是讀取來源單據的某些內容，然後寫到傳票檔去。了解了這個規律，所以就將讀取來源單的程式碼，改為 SQL 語法的設定。如此一來，程式碼的核心就只剩下讀設定檔，然後填入參數，再寫入傳票的這壹段固定程式，大約不到 500 行的程式碼。更重要的是，日後白有新增單據時，不需要修改程式，只需要在設定檔中加入應設定的 SQL 語法即可。

新寫法的優點如下

1. 程式碼更精簡，程式易維護且不易出錯
2. 當有新的需求產生時（新的單據要產生傳票），不需修改程式，也不必更新程式版本（因為跟本沒改程式）
3. 因為傳輸的內容改為 SQL 設定，所以可以依不同客戶的需求修改內容，而無需修改程式。

其他的幾個範例，大概也不脫離這樣的概念。不要把程式代碼寫死在程式中，而是透過適當的設定，讓程式去讀設定內容，然後再執行。這樣的概念理論上可以在全系統的所有類似功能執行。但是在實務上，我們只會選擇部份較重要的功能來做。因為上述的作法等於是把部份程式碼（SQL 也是程式碼），存放在資料庫設定中，凡事有利就有弊，它可能的負面效應如下

1. 會增加程式除錯的複雜及困難度，因為 SQL 語法不易除錯，不像程式設計有除錯工具可使用。
2. 過多的資料庫存取，會一定程度的影響執行的效能（但影響並不大）。
3. 部份 SQL 程式碼存放在資料庫中，有一定的安全疑慮（可透過加密解決）。

筆者在這個章節中討論了 ERP 的進階三層論，主要的目的是在提醒各位看倌，千萬不要在沒有深入了解要開發的資訊系統前，就急著開始寫程式，否則後患無窮。而所謂的深入瞭解，則是必須長期對

1. 產業的 Domain Know how
2. 程式碼合理化

保持關注，並時刻反思及不斷的歸納。最重要是，不要閉門造車，要多聽多看客戶及市場的意見。

當您真的達到第三個層次，就會發現「大道至簡」，整個程式的邏輯都在腦中清楚明白，因為 ERP 的核心概念並不複雜，不要被它外在的過多功能及報表所迷惑。一旦掌握到這些核心概念，就像是頓悟，只是這是一個大躍進，您會有「一覽眾山小」的感慨。至此，恭喜您，您會是一個稱職的系統架構師。

第二部份
邁向成功之道

第六章
精通行業別專業知識

岱宗夫如何？齊魯青未了。
造化鐘神秀，陰陽割昏曉。
盪胸生層雲，決眥入歸鳥。
會當凌絕頂，一覽眾山小。

杜甫 望岳

第六章　精通行業別專業知識

開發 ERP 系統時，最難掌握的部份並非程式撰寫，而是規格的確立。而能否順利結案，也是依交付的程式，功能是否和規格確認書一致為判斷的依據。話雖如此，但是仍然有部份的軟體開發專案，會面臨專案的尾款很難收回的囧境。原因通常是在驗收時，部份功能與客戶所預期不同，除了部份是程式的問題外，很大的部份是所謂的「灰色地帶」，客戶常會說

「你們是專業的軟體公司，這麼基本的功能，本來就應該有」 或是
「這是我們這個行業本該就有的基本功能，為什麼你們沒有考慮到」
「就是因為我們不懂電腦，你們比較專業，所以才找你們，怎麼可能連這麼簡單、基本的功能，都要我們一一的描述」

這類問題很難完全避免，特別是專案比重愈高的 case，越容易有類似的爭議。這個問題的根源，就是本章的主題，「行業別專業知識（Domain Know how）」。

俗話說「三百六十行，行行出狀元」，現在的行業別早就遠遠的超越以往，三千六百行都不止，分工更加的細緻。而每個行業都有其特定的管理重點。單就以買賣業的進銷存管理為例，鞋子和衣服類的產品，就要比一般的產品多了「尺寸」「顏色」要記錄管理。而鋼材業則是以重量（噸或公斤）或長度（公尺或碼）來管理庫存及應收付帳款。有些產品甚至有雙單位的需求（如一箱 2 支）。一旦基本的單位不同，各種單據就會有不同的欄位需求，相關的分析報表就會有所不同。

除了計量單位外，還有不同的國家或地區的稅制不同，在台灣，就有二聯式、三聯式等等多樣的發票聯式。而發票稅率大部份都是 5%，少數特殊的品項是免稅或零稅。但是在中國大陸則是不同的產品可能會有不同稅率的問題。甚至連螢幕上的字型大小、報表列印的字體等，也會是個麻煩。如果在加上一些慣有的軟體操作習慣，如按「Enter」跳到下一個欄位等，有太多的「眉角」要注意。雖然說起來都是一些小問題，但很多的小問題匯整在一起，就可能會是大災難。以上所述都還是比較簡單、普遍的例子。實際上不同行業別的差異會因為不同的業別有更多的細節待處理。

當然，通用型 ERP 套裝軟體會儘可能透過參數設定來解決這些問題，所以大部份的企業，只要選購標準版本即可，最多是加入部份二次開發的專案功能即可。但是如果差異過大，或有安全性的考量，那就可以依本書所建議的方式開發符合企業需求的 ERP 系統。

解決之道：

1. 一定要先有一個標準的核心版本。和客戶的使用者討論時，務必先跟客戶確認，以

該版本的操作介面為準，就可以避免許多操作介面的爭議。

2. 利用 Wizard 的優勢，先產生雛型程式（ProtoType），用來和客戶確認規格。所謂的 ProtoType 是指利用 Wizard 產生出第一個版本的可操作程式，類似早期的程式產生器。通常是由專案經理(PM)或系統分析師(SA) 在了解客戶的需求後，將規格輸入到 Wizard 程式中，然後產生程式碼。在這個階段中並不須要使用研發單位的任何資源。而且產生的程式碼經過客戶的確認後，會交給程式設計師繼續維護，增添功能後，就是日後交付的正式版本，並不僅僅只是 Demo 展示用。而更完整的 Wizard 程式會更進一步的產生相關的規格文檔。如此一來就可避免先前一般的專案開發，都是用 Word（或 Excel）文檔和客戶確認規格，但是客戶並無法實際操作來確認是否正確，光憑文檔上的描述，一定會有溝通的盲點。當然，規格文檔還是必要的，雛型程式和規格文檔互相配合，就可以避免很多的負面效應。

3. 開發與測試並行

　　絕大多數的疾病，如果能「即早發現，即早治療」通常都有很大的機會可以治好。一般專案的驗收期都在專案的結案日期前，也就是專案的後期，如果一切順利那就還好，一旦在驗收期才發現大問題，那就會有大麻煩了。正確的作法，應該是事先跟客戶溝通好，在開發期間就應該備好一台測試主機，經內部測試無誤的程式，應定期更新到測試主機上，並請客戶配合測試，如果有問題，可以直接填寫 issue 到專案管理模組中的 issue Tracking 系統。客戶可以隨時掌握目前的進度，不會因為資訊不足，導致在專案後期才發現重大的問題。當然，驗收期還是必須的，不過此時的驗收重點是在測試數字的正確性，而非操作介面的問題。

4. 測試案例的編寫

如何才算驗收合格？如何才能確認雙方對規格的認知沒有誤差及錯漏？

要確定上述的問題，測試案例（Test Case）應該是一個很好的工具。（關於測試，筆者會在第 12 章有詳細的說明）。測試案例最好由客戶依某一週或月的實際工作內容編寫，務必要包含全部專案的絕大部分功能。範本的格式則通常可以由獨立軟體開發商提供。但是通常十之八九客戶都沒有辦法獨立完成，所以這個工作通常是以

A. 客戶負責提供完整的測試資料內容及應有的重要報表數字（含報表格式）

B. PM 或 SA 依前述的內容編寫測試案例內容

C. 雙方在初期頻繁的討論測試案例的內容，試圖發現是否有漏失重要的流程及功能。

的過程來共同完成。

測試案例的最大優點，在於可以透過收集來的資料，確認對客戶的需求是否完全理解，因為在串接客戶資料流的過程中，如果有滯礙難行或不合理的地方，通常在這個過程會很快被挖掘出來。不過跟據筆者個人的經驗，大部份的客戶都會延遲一段時間才給，甚至給的殘缺不全，因為提供資料可能會涉及不同的部門及人員，而且提供資料者本身就有很多業務待辦，反正就是會有一堆聽起來很有道理的藉口。

無論如何，請相信我，一份完整的測試案例會讓專案在進行中，避免很多溝通的缺漏及誤解，無論再怎麼困難，都必須搞定它。這當然很大程式的考驗 PM 的功力以及平常是否和客戶的關係打好，PM 和客戶方的 MIS 窗口是否保持良好的關係及溝通，往往會在關鍵時刻發揮重大的功能，千萬不要小看這些細節。

Domain Know how 是否熟悉了解，對整個系統的成功與否佔極重要的角色。基本上它就是要完全的分析理解客戶的所有規章及業務流程，然後將其轉為資訊化作業。在這個過程中，甚至會幫客戶盤點出很多的不合理（或矛盾）的規定或制度，或是沒有規範到的例外情況，這也算是另類的企業診斷。

第七章
程式開發工具集的整合

金樽清酒鬥十千，　玉盤珍羞直萬錢。
停杯投箸不能食，　拔劍四顧心茫然。
欲渡黃河冰塞川，　將登太行雪滿山。
閑來垂釣碧溪上，　忽複乘舟夢日邊。
行路難，行路難，　多歧路，今安在？
長風破浪會有時，　直掛雲帆濟滄海。

李白 行路難

第七章 程式開發工具集的整合

ERP 系統動輒由數百、甚至上千支程式組成，是非常複雜的軟體工程。而且隨著企業間的競爭加劇及軟體工程技術的持續進步，開發程式所使用的工具及程式設計所須的專業知識也越來越龐雜。先前筆者曾經說過，早期的 DOS 及 Windows 時代，程式開發人員只要精通諸如 Visual Basic、Delphi、Power Builder、Visual Foxpro 等單一開發程式，幾乎就可以完成所有的程式開發。當然這和 1、20 年前的環境有關，那時候沒有人會想到 Internet 會如此深遠的影響幾乎所有人的生活，也沒想到智慧型手機(SmartPhone) 和平板電腦會如此普及。因為當時的系統相對單純，外在環境也不複雜，最多就是拉個內部網路，大部份的系統都是在內部網路上安裝執行，所以單一工具就可以打遍天下無敵手。但是當進入了 Web 時代以後，這一切都已經變的大不相同

目前開發 Web 程式有三大主流技術

1. MicroSoft .NET 平台

   http://msdn.microsoft.com/zh-tw/vstudio/aa496123.aspx

   http://zh.wikipedia.org/wiki/.NET 框架

2. Oracle Java 開發平台

   https://java.com/zh_TW/

   http://zh.wikipedia.org/wiki/Java

3. PHP 、ROR 等 OpenSource 開發平台

   http://php.net/

   http://zh.wikipedia.org/wiki/PHP

其實隨便詢問某個稍有經驗的程式設計師，他們都可以輕鬆地為自己專長或喜愛的程式語言，列舉出數個優點(然後順便舉出別種語言的缺點.....)，例如 .NET 易學難精，又不能跨平台；Java 能跨平台，在大型企業及政府組織的應用多，相關人才市場需求多，但學習門檻相當高；PHP 入門門檻低，支援的社群或開發者工具又多又成熟，然而由於是直譯式語言，執行效率相對較差，據統計 75% 的網站均使用 PHP 開發，包括全球知名平台如 Facebook、Wikipedia、Yahoo，Zynga，及全球最熱門的部落格系統 WordPress。

當然還有很多其他各具特色的技術，但因使用人數較少，此處先不多提。前述的三

種開發平台各有優缺點，這也是在開發系統前，內部要優先討論的議題，通常這和系統架構師的背景及喜好有關，主觀的成份居多。等決定好要使用哪一種技術後，再來就是要整合相關的元件及工具了。

要想開發一個具有後端資料庫功能的動態網站，必須熟悉並了解下列的工具及元件

1. 純網頁物件：基礎的網頁元素

HTML	http://www.w3.org/ http://zh.wikipedia.org/wiki/HTML
XML	http://zh.wikipedia.org/wiki/XML
css	http://zh.wikipedia.org/wiki/Css

2. 動態網頁的開發語言：依資料庫儲存的內容顯示網頁

ASP.NET	http://www.asp.net/
PHP	http://php.net/ http://zh.wikipedia.org/wiki/PHP
JSP	http://www.oracle.com/technetwork/java/javaee/jsp/index.html
Ruby On Rail	http://rubyonrails.org/

3. 程式語言

C#	http://zh.wikipedia.org/wiki/C#
Java	https://www.java.com/zh_TW/
Ruby	https://www.ruby-lang.org/zh_tw/ http://zh.wikipedia.org/wiki/Ruby
Python	http://zh.wikipedia.org/wiki/Python https://www.python.org/

4. 前端介面框架 User Interface

jQuery	http://jquery.com/
jQuery EasyUI	http://www.jeasyui.com/ http://www.trirand.com/blog/
ExtJS	http://www.sencha.com/
Prototype	http://prototypejs.org/ http://zh.wikipedia.org/wiki/Prototype
YUI	http://yuilibrary.com/
MooTools	http://mootools.net/ http://en.wikipedia.org/wiki/MooTools

5. 開發框架

NHibernate（ORM）	http://nhforge.org/
PHP	http://framework.zend.com/ http://www.codeigniter.org.tw/ http://www.yiiframework.com/
Ruby On Rails	http://rubyonrails.org/
Struts	http://zh.wikipedia.org/wiki/Struts

6. 報表工具（或元件）

Reporting Services	http://msdn.microsoft.com/zh-tw/library/ms159106.aspx
Crystal Report（SAP）	http://www.crystalreports.com/
telerik	http://www.telerik.com/devcraft
devexpress	https://www.devexpress.com/Products/NET/Reporting/

7. 第三方（Third-Party）元件，通常和使用者介面有關

telerik	http://www.telerik.com
devexpress	https://www.devexpress.com
訊光科技	http://www.infolight.com.tw/WebClient/
遠綠資訊	http://www.doublegreen.com/Default.aspx
新人類科技	http://www.newtype.com.tw/

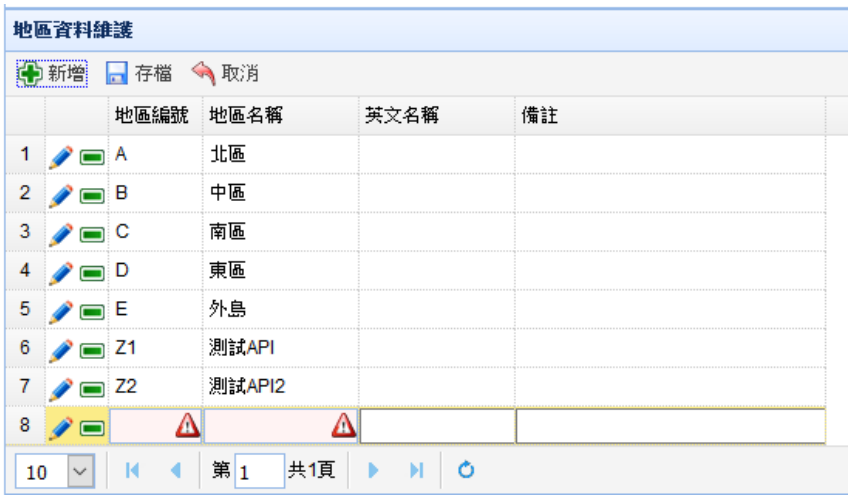
除此之外，還有為了 UI 的美觀及更好的互動性，必須加入的美編及排版等。以上每一種工具的熟悉都要花費研發人員很多的時間及精力去學習。所以大部份的研發人員都只能精通部份工具，往日一個程式語言搞定全系統的好日子已經一去不復返，所以就形成了開發 Web 程式版本的第一個門檻－各具所長（十八般武藝）的研發英雄好漢。請千萬別小看這個問題，在台灣，大部份的軟體公司規模都不大，而且熟悉 Web 開發的程式設計師相對較少，所以這會是一個大問題。考慮的更完善的系統，甚至還會自行開發一些 WebServices 或 API 供外部的程式呼叫，這個部份的技術也不斷再更新，由於可以使用的技術、工具變多變複雜了，所以開發前期的實作測試，就變的非常的重要。

一般狀況下，開發企業的解決方案，並不會選擇使用最新的技術，一方面因為最新通常意謂著變動及不確定，還有會的人也少，而 ERP 之類的系統則是以穩定為第一要務，數字正確且操作不出錯優於一切。所以還不確定的技術通常不會被採用（但是也有在別無選擇的情況下，只能冒險使用某些元件的狀況，但這只能偶一為之，風險太高了）。

先期的開發測試，主要會先以幾支規劃好的樣版程式的完整開發為基礎。關於這個部份內容，會在第十九章中詳細說明，此處僅簡要的說明樣版程式的概念。樣版程式（Template）顧名思義，就是日後要正式開發系統時的標準範本程式。通常有經驗的架構師會先將整個系統的所有表單分門別類，例如

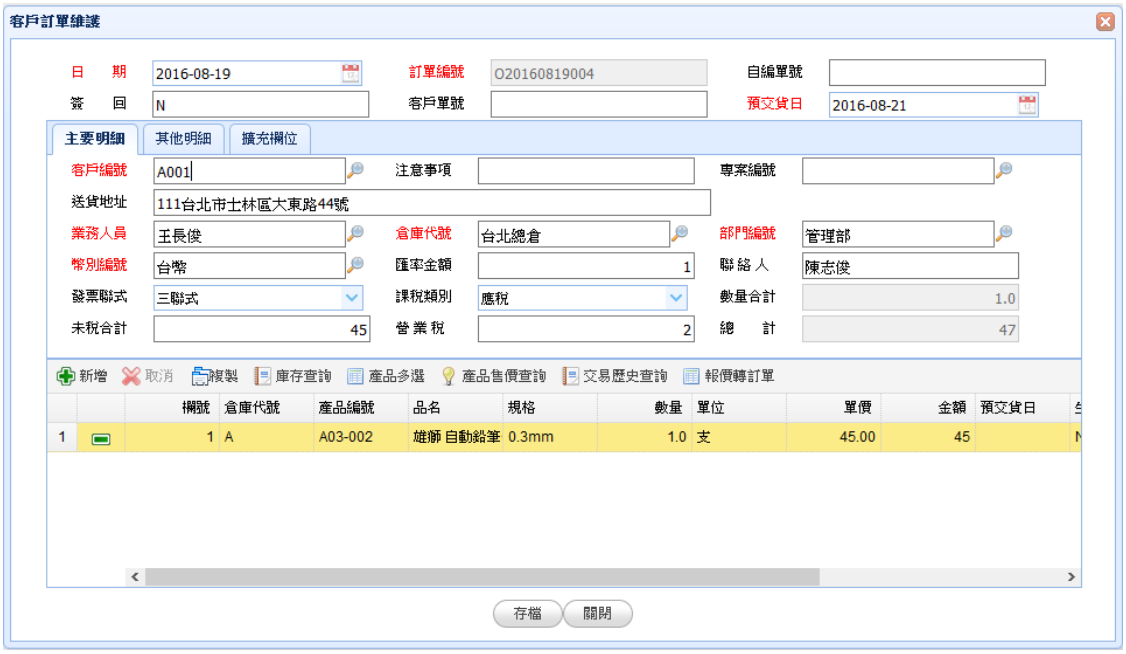
1. Single Entry：單檔單筆資料維護

2. Single Grid：單檔多筆資料維護



3. Single Tab Entry：單檔多頁簽資料維護

4. Master Detail Entry：單據資料維護表單



然後請資深 SD（System Design）開始手動開發程式（一般而言，因為開發出來當作範本，所以都會讓全公司寫程式的最高手高高手來寫），每一種樣版程式，都會選取一支具代表性的程式。所謂具代表性的意思是說，該程式具有全部或大部份該有的功能。例如

檔案上傳功能、圖片上傳顯示功能、

開窗查詢功能、多筆資料 DataGrid 顯示、

日期元件、超連結元件、查詢後多欄位取回元件

等等的操作，所可能會用到的元件或操作方式，在過程中不斷的加入及完善。並建立共用程式庫。

樣版程式之所以重要，是因為後續開發 Wizard 時，會分析這些樣版程式的程式碼，然後依據其程式內容寫成 Template 檔。之後所有程式的第一個版本，都盡量用 Wizard 來產生，如此一來就可以確保所有的程式碼都系出同源，都擁有相同的架構，日後維護起來，會比較有一致性，在撰寫樣版程式的過程中，為了要滿足系統架構師所提的各種 UI 操作風格，SD 系統設計師會不斷的嘗試整合各種元件來達成架構師的要求。舉一個實際的例子，筆者在 2009 年開始開發 Web 程式時（使用 Visual Studio 2008），一開始是使用原生的 DataGrid 當作多筆資料的顯示及編輯元件，但是原生的 DataGrid 功能實在太陽春，顯示還行，用來編修資料可真是 xxx 了，連最基本的寬度拖拉及欄位順序對調都做不到，所以就開始了一段 Google 找元件 － 測試 － 找元件 － 測試的過程，試過很多元件後，發現大部份 ASP.NET Server 端的元件，功能大部份都可滿足，但是效能極差。原本都要放棄了，最後終於找到 ExtJS 這個 JavaScript 框架，一試之下，驚為天人。它的美工有一定的水準，效能一級棒，雖然還是有 js 檔過大的問題，會導致第一次載入

較慢的問題，但仍在可接受的水準，於是就開始整合。由於我們還用了國內工具大廠－訊光科技的 EEP2008 開發平台及元件，所以資料存取層就必須寫一些共用程式來介接，再加上 ExtJS 有不同版本和不同瀏覽器上的一些小問題（例如 ExtJS 3.0 在 IE9 執行會報錯，解決之道是只能在 web.config 設為 IE8 相容模式）等等非常多的細節要調整、要解決。其中的艱辛，實不足為外人道，何況除了 ExtJS，還要整合其他很多不同的第三方元件，這個過程非常的辛苦，甚至多次萌生退意，但是，天無絕人之路，最後終於整合成功，雖不儘如人意，但也頗具水準。而且一旦整合成功，就可以用 Wizard 快速的產生包含報表在內的所有程式（第一個待改的版本）。所以樣版程式的好壞，直接影響日後整套系統的成敗，不可不慎呼！

至於如何將寫好的樣版程式，轉為 Wizard，那就不是本書所要討論的主題。筆者計劃另寫一本書，專門來討論 Wizard 這個主題，敬請期待。

第八章
建立開發平台

故其疾如風，其徐如林，
侵掠如火，不動如山，
難知如陰，動如雷震。

孫子兵法 軍爭篇

第八章　建立開發平台

開發企業 ERP 類的系統是一個龐大的工程，所以更應該像所有的工程開發一樣，一定要先做好事前的規劃與進度的控管，否則就可能會變成一個耗盡資源的無底洞。

在真正進入開發前，首先應該是確認要使用的技術平台，詳細的內容已經在前章說明。每一種技術都有其優缺點，沒有哪一種技術平台是特別完美，選擇哪種技術通常是高階的核心幹部的直覺和喜好有關，因為本書主要是「方法論」，跟您使用哪一種技術無關，又因為筆者反覆強調的以「資料庫為開發核心」的開發方法論，所以原則上開發工具主要都是在處理人機介面（User Interface），也就是資料的輸入及畫面的顯示，並儘量精簡程式碼。因為網頁版本的作業方式，是 Web Server（例如 IIS、Apache 等）會將程式轉為 HTML 後回傳到 Client 端，然後利用 Client 端的瀏覽器（IE、FireFox、Chrome、Safari 等）在客戶端的電腦顯示，所以如果網頁過大，網路傳輸速度若不快，就會影響使用者的操作體驗。這也是所謂 Thin Client 的程式寫法。

所以整合好適當的工具集是重中之重。一旦整合好開發工具後，接下來就是本章要討論的主題－建立開發平台。

圖 8-1

所謂的開發平台，早期筆者稱為空架構（框架），事實上就是一些所有模組都會共用的程式集合（大部份的程式通常都會歸屬到 SYS 模組），還有共用程式庫。

要了解為何在開發系統前必須先「建立開發平台」，讓我們就從 Login 登入作業開始說起

1. 輸入網址，如 erp.gex.com.tw
2. 輸入帳號／密碼後，登入系統，顯示主頁面，此時左方的主選單會依登入 ID 不同，而顯示不同的 ID 所能使用的功能及報表
3. 可以依個人喜好，設定不同的主選單樣式及字型大小
4. 進入 SYS 模組
    a. 權限相關設定

b. 參數設定作業

c. 其他設定

5. 點選任一功能，本處以 OPOM02 訂單維護作業為例

a. 小數位數顯示，受參數控制

b. 開窗查詢的共用介面

c. 表身的產品多選介面

d. 轉單介面功能

上面簡單的說明了登入系統及一些常用的共用操作介面，上面的這些程式，都屬於開發平台的一部份。這些機制大部份都是實作經驗的累積，而實作通常是由使用者的實際需求所歸納產生。初次開發 ERP 系統最常發生的問題，是當程式開發到一個階段，寫了幾十支程式後，發現有幾個共通的功能沒有寫，於是只好回頭修改。如此反覆多次之後，有的程式有改到，有的程式漏改，程式的版本開始出現分岐，但是開發的進度很緊，又不斷的寫新的程式，而新的機制需求又會部斷的被要求加入。在這種情況下，初期寫的程式，通常會出現和稍後版本的程式碼機制不一致的情況。這種情況很難完全避免，即使是非常有經驗的系統架構師，也難免掛一漏萬，這也就是通常 V1.0 版都還有改善空間的主要原因。

和行業別專業知識不同，Domain Know How 通常指的是行業別的專業知識，通常都和商業流程有關，而開發平台則是一個程式開發的框架，一個標準的程式模式。一個企業資訊系統，動輒幾千支程式，百萬行程式碼，如果沒有訂立開發標準，任由程式開發人員自由發揮，這一定會造成其他人員無法維護的大問題，造成日後改版、維運的困難，這絕對是重要的管理議題，絕不是「程式開發是藝術」這種無知的言論所能輕輕帶過的。

所以，在正式啟動專案開發前，務必要審慎的做好框架程式的建立，並配合標準樣版程式的測試與開發，儘可能的考慮周詳，在上述的工作沒有完成前，千萬不要急著寫程式，「吃太快、弄破碗」這句台灣諺語是千真萬確的。與其事後不斷的修改，倒不如事前的準備工作要儘量的完善，這也是系統架構師責無旁待的重任。

下面我會列出一些常見的空架構內容，您可以依實際的需求增減

1. 權限管理機制

   一般都會依 User 及 Group 設定，更周詳的大型系統還會加入

   a. 資料存取限制機制

   依 LoginID 限制所能查看的資料內容，例如一般員工只能看跟自己相關的資料，而部門主管可以查看所屬同仁的所以資料。

   b. 資料被用保護機制

      已經被使用再單據中的基本資料（如產品編號，員工編號等），不能被修改及刪除。又如在關帳日前的單據，也不能新增、修改、刪除。

2. 選單管理機制

   可以透過系統維護功能新增、刪除功能選單，配合權限的設定，可以限制 LoginID 能使用的功能選項。

3. 參數管理機制

   可以經由設定參數，微調系統的一些顯示內容及特殊功能的開關，善用參數設定，可以讓系統的彈性變大，而且可以讓系統更加適合公司的需求。例如進銷存的程本使用方式（月加權、移動加權、標準成本）

   圖

   或財務的存貨成本計價方式（永續盤存制、實地盤存制）

   圖

4. 底層共用程式庫開發

   如前所提的「速查」「多選」「轉單」等共用操作機制

   還有和資料庫溝通的共用程式

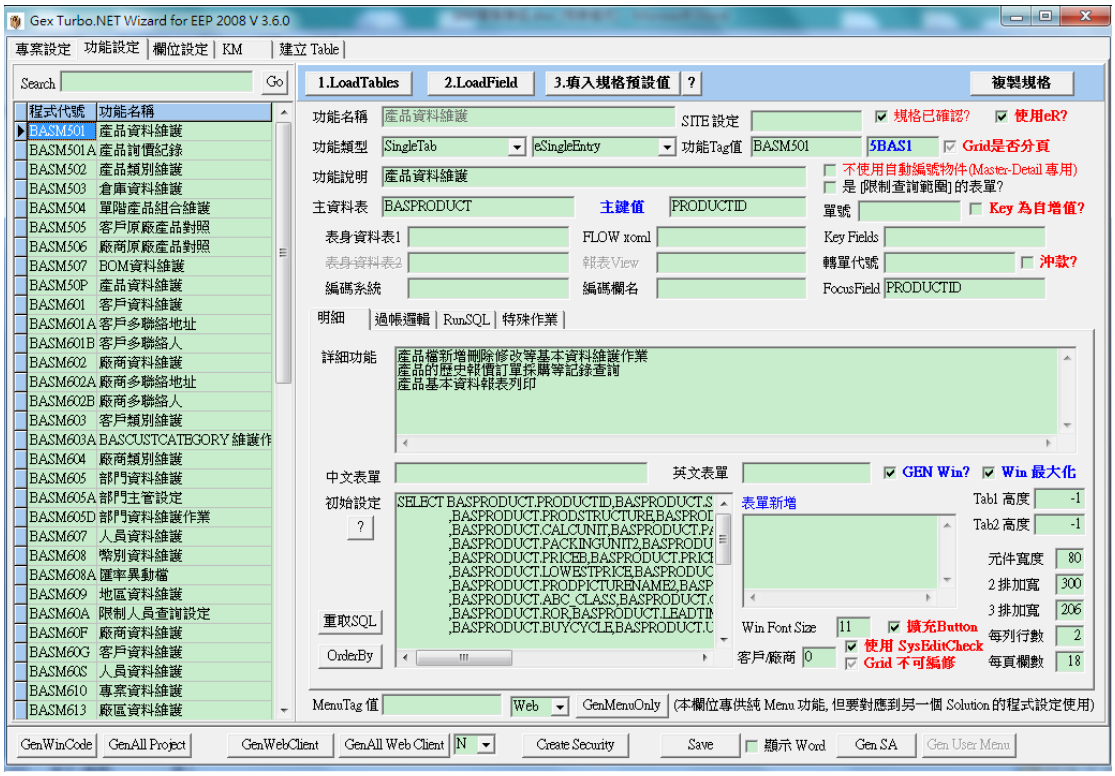
   Excel 匯出、轉入機制

   單據的單號編碼機制等等

5. Wizard 程式產生器的開發（選項）

Wizard 應該是影響專案進度及品質最大的一個機制。程式產生器已經是一個古老的概念，早在 2,30 年前，Clipper 時代就已經有一個很有名的 Stage（如果我沒記錯的話）程式產生器。近期也有 CodeSmith 這個好用的工具，只是因為對筆者而言，因為有太多特殊需求機制，此用市場上的 Code Generator 總是有一種隔靴搔癢的感覺。一方面產生的程式碼並不是依造自己的理解產生，所以很難理解，另一個原因是就算產生出來了，可是需要手工修改的程式碼也蠻多的，而且也不好維護。基於以上的原因，筆者認為還是有必要自行開發程式產生器。為了和之前的作法有所區隔，所以改稱為 Wizard（精靈）。

套句之前負責開發 Wizard 的同事名言，「Wizard 是一塊很難啃的骨頭」，的確，自行開發 Wizard 是有很高的門檻的。顧名思義，Wizard 就是在輸入程式規格後，程式就依規格自動產生的一支程式。而且是和整個開發平台完全整合的可執行程式，程式開發人員也是在這個程式版本的基礎上，繼續新增新的功能。



自行開發 Wizard 雖然艱辛，但如果成功，其回報也是巨大的。這支程式可以交給 SA 或 PM 來使用，快速的產生程式跟客戶溝通，而且大幅度的提高程式設計師的生產力，不必再去處理既無趣又費時的前端使用介面上。

因為 Wizard 的工程浩大，本書僅能做些簡單的介紹，筆者預計在系列叢書中，會有一本書專門來討論這個主題，敬請期待。

6. Log 的記錄及管理機制

   a. 使用記錄查詢

   b. 錯誤發生時的記錄（除錯用）

7. 多國語系機制

第九章
發揮資料庫的強大功能

萬物之始,大道至簡,衍化至繁

春秋・老聃

第九章 發揮資料庫的強大功能

「以資料庫為開發核心」是筆者根據多年來多次開發企業資訊系統所累積的經驗總結。當然，這是筆者的個人體會，並不表示這是唯一的開發方法，只不過依此邏輯去推論，且經實戰的驗證，是一個可重複操作、彈性頗大且可跨開發平台的作法。所以就大著膽子寫下來，算是個人一些淺見的心得分享，請各位先進不吝指正。

想要真正的發揮「關連式資料庫」的效能及強項，真正的難點並不是 SQL 語法的學習，絕大部份的軟體研發人員都略懂 SQL 語法，因為這是和資料庫溝通的標準語言。但是多數負責寫程式的工程師，通常都是透過開發工具（如早期的 VB6、Delphi、PowerBuilder，或是近期的 Visual Studio 2008、2012、2015 等），所提供的元件，透過簡單的設定，並撰寫一些簡單的 SQL 語句，就可以和資料庫溝通。做一些簡單的查詢、新增、修改、刪除等作業。這也是大多數軟體工程師使用 SQL 的方式。

這當然沒錯，而且也是必要的程式寫法，但是如果只是這樣使用關聯式資料庫，就會像倚天屠龍記中的張無忌，雖然身負九陽神功，內力之強舉世無雙，一旦被困在光明頂上，還是得透過小昭的協助學會了乾坤大挪移，理解了巧力的運用法門，才得以破關而出。關聯式資料庫就像是九陽神功，博大精深且功能強大，而關聯式資料庫中的 Store Procedure、Trigger、user defined function 等，就像是乾坤大挪移，每練深一層，功夫就更上一層樓，更懂得使力的技巧。

學習 SQL 的過程，的確和練乾坤大挪移的過程非常類似。如果沒有九陽神功的深厚內力基礎，張無忌就會和歷代的明教教主一樣，窮其畢生的精力，也只能初窺門徑，但永遠無法登堂入室。SQL 的基礎相對就複雜一點，除了資料庫的必備基礎知識之外，還必須對整套企業資訊系統的資料表結構及關連非常熟悉。

基本資料庫是由

資料庫 -- 表(Table) -- 欄位(Fields) 及 View 所組成，這當然是必須先理解的基本概念。本書一再強調 SQL 的重要性，但是 SQL 教學並非本書的重點，筆者個人推薦楊志強老師的相關書籍，讀者可自行參閱相關的書籍內容。

如果本書的概念能夠得到您的共鳴，筆者計畫會再寫幾本以主題式的概念，另外出版一系列的 SQL 實務應用的書籍。

例如 電子簽核的重要核心程式，全部以 SQL 指令完成，甚至可配合 web Services 和 Mobile(行動)總合的完整機制。

廣告時間結束，讓我們再回到本章主題。

因為 SQL 的內容都是針對表(Table) 做 [增/刪/改/查] 的批次作業，對象當然都是儲存在資料庫中的資料，所以這當然也是重要的基礎之一，再來就是寫程式時，在觀念上要「捨迴圈而用批次」。

就筆者個人的學習經驗來說，這一點反而是較難克服的，SQL 的語法並不複雜，相反的，她簡潔、容易理解，對絕大多數的軟體工程師而言，都很容易就上手使用，但是一般的研發人員，常基於以下的問題，而不曾深入的使用

1 因為早期的學習經驗（尤其是學校的教育）
2 程式語言的特性
3 寫程式較容易追蹤、除錯的觀點

習慣性的使用迴圈式的程式語法去解問題，這在資料筆數少時當然沒問題，但是隨著使用者每日大量的建檔輸入資料，當系統使用了一段時間後，使用者會開始反應操作速度變慢，更有甚者，某些需要大量運算的作業，會慢到幾乎當機。如 物料需求展算（MRP）、成本重置作業等。而當研發人員去檢查程式的時候，會發現程式寫法並沒有問題，有些獨立軟體開發廠商（ISV），甚至會推給客戶的網路環境有問題（當然有時候真的是網路的環境問題），因為其他很多客戶並沒有這些問題。

每家軟體公司或是資深的軟體工程師，都會在他們漫長的寫程式生涯中，碰到過很多難解，甚至是無解的疑難雜症。大部分都和作業系統、開發工具、及網路環境的設定使用有關。以上的問題當然和軟體工程師無關，但是的確也有很多問題，是所寫的程式碼邏輯觀念本身有瑕疵所造成的，只是受限於當時的專業及經驗，無法找到適合的解決方案。

愛因斯坦曾經說過：「當我們碰到問題時，不要直接去找解決方法，因為在原有的思考水平下產生的問題，是不可能用原來的思考來解決的，否則這個問題也就不會產生了。」 那該怎麼辦？向上提升是唯一的解決之道，就像乾坤大挪移有九層，每向上一層都是質的提升，可以透過各種途徑，如

1 公司內部的經驗交換及深入討論
2 谷歌查詢論壇上各種討論
3 直接找外部的研發顧問協助

也就是透過外部的協助，提升自己的眼界及觀念，才能真正的解決問題，要知道學習真的是永無止盡的。而這也正是很多的軟體，要到第 2 版以後（甚至更高的版本）才能運作順暢的原因。

筆者個人是在寫程式的較晚期，認識了一位 SQL 大神，透過提問後，直接看他寫的程式碼，再比對自己的寫法，結果相同但效能差異巨大。才親身感受到 SQL 的強大，當初求解的問題，就是 物料需求展算（MRP）。但即便如此，原專案也因為大部分的程式都已經完成，所以只能選擇其中幾隻較複雜、效能有問題的程式重寫，其他大部分的程式仍舊是原先的寫法，這就是既有產品的包袱。沒有人敢冒著 「原本沒錯，只是較慢，而改過後結果錯誤」的巨大風險。所以筆者雖然已經感受到了 SQL 的強大威力，也只能在另起一個新專案時，再透過不斷的嘗試，及大量的閱讀反思，才慢慢有更深層的體會，所以經驗的累積是非常重要的。

無論是主從式架構（Client / Server），或是網頁程式的開發，基本上我們都會建議將 SQL 關聯式資料庫放在獨立的網路主機上，然後使用者由 Client 端連到 SQL 主機，當然有些程式架構的程式，會架設一個應用程式主機。這個做法是讓應用程式主機(AP Server) 擔任交通警察的角色，統一處理所有使用者端(Client) 對 SQL 資料庫的所有資料存取動作。理論上這是較好的做法，但是實務上則需要審慎評估。因為一旦應用程式主機發生問題或是不穩定，會造成整個系統的當機，反而造成非程式錯誤的系統問題。在這樣的架構下，系統的效能就必須考慮網路通訊的問題。

以下我們使用一個簡單的計算現有庫存量的例子，來說明何謂迴圈式寫法、何謂批次處理。
下面的圖表是某產品的進銷存明細表
(略)

迴圈式的程式寫法如下
(略)

批次處理的寫法如下
(略)

這只是一個非常簡單的例子，簡單到您看完程式碼後，會笑說這麼簡單的程式，需要寫什麼 SQL?

如果資料庫中只有數千筆交易單據，以目前電腦的速度執行，無論哪一種寫法，都不需要 0.1 秒，應該就可以計算出來。但是如果是一百萬筆資料呢? 更多的資料筆數呢? 如果用批次處理的寫法 10 秒內應該就可以計算出來。迴圈式寫法呢? 以下我們用幾個簡單的資料，實際來測試看看
(略)

上面的例子，只是單純的讀取資料，如果再加上回寫資料到關聯式資料庫，上面的執行效率差異就會更加巨大。

最後，我們整理一下以資料庫為核心開發方法方法論的優缺點，當作本章的總結。供各位讀者參考。當然這只是筆者的一家之言，一些淺見，希望對您能有所助益。

優點：

1. 商業邏輯寫在 SQL 中，不會因為開發平台的轉換，而需大幅改寫程式。
2. 執行效能極佳。
3. 不同的 SQL 關聯式資料庫，大都會遵循 ANSI 92(以後) 的標準，可以自行轉換 SQL 而不會受限於特定的軟體開發商。

缺點：

1. SQL 寫的程式，除錯較困難。
2. 邏輯寫在 SQL 中，有安全性的疑慮
3. 更換資料庫時，轉換語法畢竟還是有相當的難度。
4. 程式版本的變更較難控制管理。

第十章
專案進度的管理自動化

知人者智，自知者明；
勝人者有力，自勝者強；
知足者富，強行者有志；
不失其所者久，死而不亡者壽。

老子。道德經

第十章 專案進度的管理自動化

要開發一套完整的 ERP 系統是一個很大的工程，要想如期完工，牽扯到很多的人、事、時的因素。如果用人工管理，很明顯一定是很沒效率寫錯誤百出，除了累死專案經理，實在沒有其他的優點。

要想讓開發 ERP 的工作能順利進行，就必須有一套系統化的管理程式。這套工具至少應該包括以下兩個主題

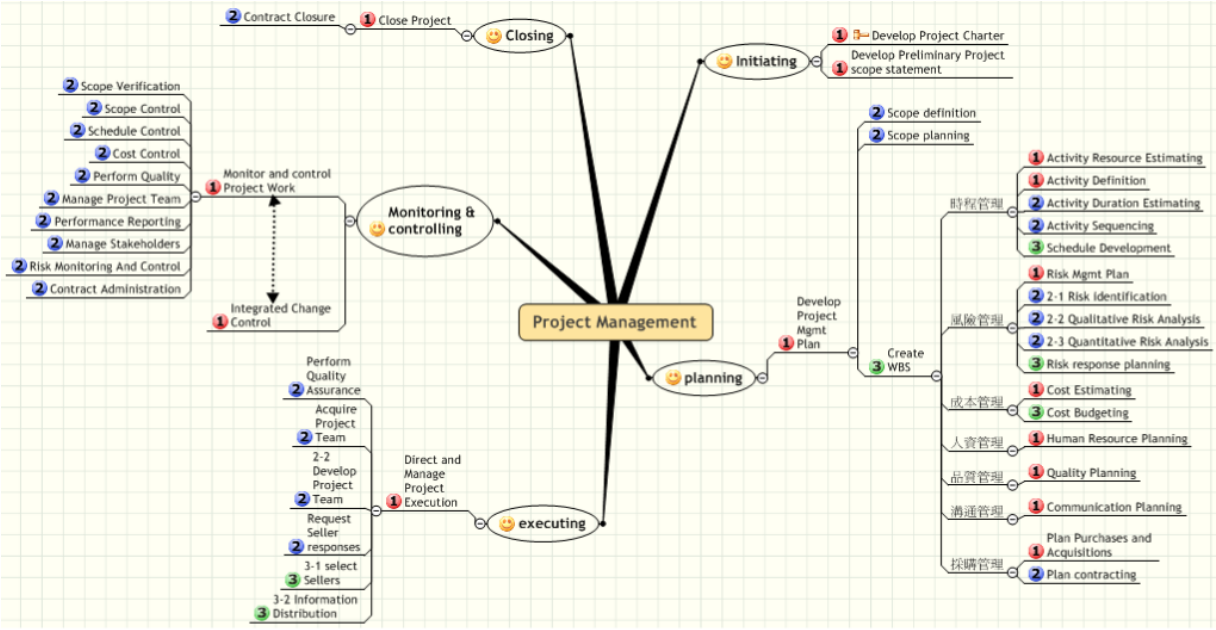
1 專案進度控管(含 WBS 及 Issue Tracking)

2 協同工作系統

一、首先，我們先來談談專案管理(Project Management)，筆者早期參與的 ERP 相關專案，規格都不是很大，頂多百來支程式，所以不覺得專案管理有什麼太大的用處(事實上是不懂、無知)，往往只是很隨意的訂立一些工作進度後，就開始寫程式。

直到後來有機會參與某筆電代工大廠的現場製造建置專案，在這個長達 9 個月的專案執行過程中，親身體會專案管理的成效及重要性，自此我就成了專案管理的信徒。

完整的專案管理是一個很龐大的課題，簡單說明大致的範圍



摘自: 專案管理架構, 鄭庠宏, PMP PMBOK第三版, PMI

PMBOK 內詳細說明專案管理所依循的五大程序群組 (Process Groups) 和九大知識體系領域 (Knowledge Areas)各定義。
其中

五大專案管理程序群組為：

起始程序 (Initiating Processes) - 授權開始一個專案或進入專案的下一個階段。
規劃程序 (Planning Processes) - 定義及琢磨目的並選擇最佳行動做法以達到專案的所要滿足的目的。
執行程序 (Executing Processes) - 協調人員及其它資源來實行專案計劃 (project plan)。
控制程序 (Controlling Processes) - 由定期監督及量測專案進度來鑑別與專案計劃的差異，必要時採取矯正行動，以確保達成專案目的。
結案程序 (Closing Processes) - 正式化一個專案或其階段的接受，使其有條理地結束。

這五大程序群組不僅適用於一整個專案，且適用於一個專案中的每一個階段。
起始程序
計劃程序
執行程序
控制程序
結案程序

九大專案管理知識體系領域為：

專案整合管理 (Project Integration Management) -
確保專案中不同的要素相互協調配合，含專案計劃書發展 (Project Plan Development)、
專案計劃執行 (Project Plan Execution)、及整體變更控制 (Overall Change Control)。

專案範疇管理 (Project Scope Management) - 確保專案包含所有及僅含有所需的工作項目以成功達成專案目的，含起始 (Initiation)、範疇規劃 (Scope Planning)、範疇定義 (Scope Definition)、範疇確認 (Scope Verification)、及範疇變更控制 (Scope Change Control)。

專案時間管理 (Project Time Management) - 確保專案如期完成，含活動定義 (Activity Definition)、活動排序 (Activity Sequencing)、活動期程估計 (Activity Duration Estimating)、時程發展 (Schedule Development)、及時程控制 (Schedule Control)。

專案成本管理 (Project Cost Management) – 確保專案如核准的預算完成，含資源規劃 (Resource Planning)、成本預估 (Cost Estimating)、預算編列 (Cost Budgeting)、及成本控制 (Cost Control)。

專案品質管理 (Project Quality Management) – 確保專案能滿足需求，含品質規劃 (Quality Planning)、品質保證 (Quality Assurance)、及品質管制 (Quality Control)。

專案人力資源管理 (Project Human Resource Management) – 最有效地運用與專案有關的人員，含組織規劃 (Organizational Planning)、人員獲得 (Staff Acquisition)、及團隊發展 (Team Development)。

專案溝通管理 (Project Communications Management) – 確保專案能適時及適當地產生、蒐集、散佈、儲存、及最終處置專案資訊，含溝通規劃 (Communications Planning)、資訊發佈 (Information Distribution)、績效報告 (Performance Reporting)、及結案作業 (Administrative Closure)。

專案風險管理 (Project Risk Management) – 敘述專案風險的界定、分析、及回應的程序，含風險管理規劃 (Risk Management Planning)、風險界定 (Risk Identification)、定性風險分析 (Qualitative Risk Analysis)、定量風險分析 (Quantitative Risk Analysis)、風險回應規劃 (Risk Response Planning)、及風險監控 (Risk Monitoring and Control)。

專案採購管理 (Project Procurement Management) – 敘述執行自組織外取得商品及服務的程序，含採購規劃 (Procurement Planning)、邀商規劃 (Solicitation Planning)、邀商作業 (Solicitation)、商源評選 (Source Selection)、合約管理 (Contract Administration)、及合約終結 (Contract Closeout)。

看完上面的範圍定義，如果你沒有頭昏腦脹，我只能說，大哥! 我服了。
至少對我這個半吊子而言，這真是令人望而生畏。

但是，以開發中小型的 ERP 企業資源管理系統來看，並不需要全模組都導入使用。主要應該導入幾個關鍵的主題即可，其他主題可是實際的需求納入
1. 工作任務(Work Break Structure，WBS) 的建立及控管。
2. Issue 的追蹤管理

以下是 PMBOK 中對 WBS 的說明
Work Breakdown Structure (WBS). A deliverable-oriented hierarchical decomposition of the work to be executed by the project team to accomplish the project objectives and create the required deliverables. It organizes and defines the total scope of the project.

嚴格說來，WBS 和所謂的工作清單（Activity List）是很不一樣的概念。WBS 指的是拆分的工作項目，而 Activity 則是為了完成該工作項目，而必須採取的行動。簡而言之，WBS 是把需求與專案範疇畫成明確的圖像，把大目標拆解成數個容易管理的小目標。

WBS 的重要性除了展現在釐清需求與範疇外，另一個目的就是工作分派。工作拆分出去了，下一步就要監督有沒有被做好，這就是 Issue Tracking 系統主要的管轄範圍。

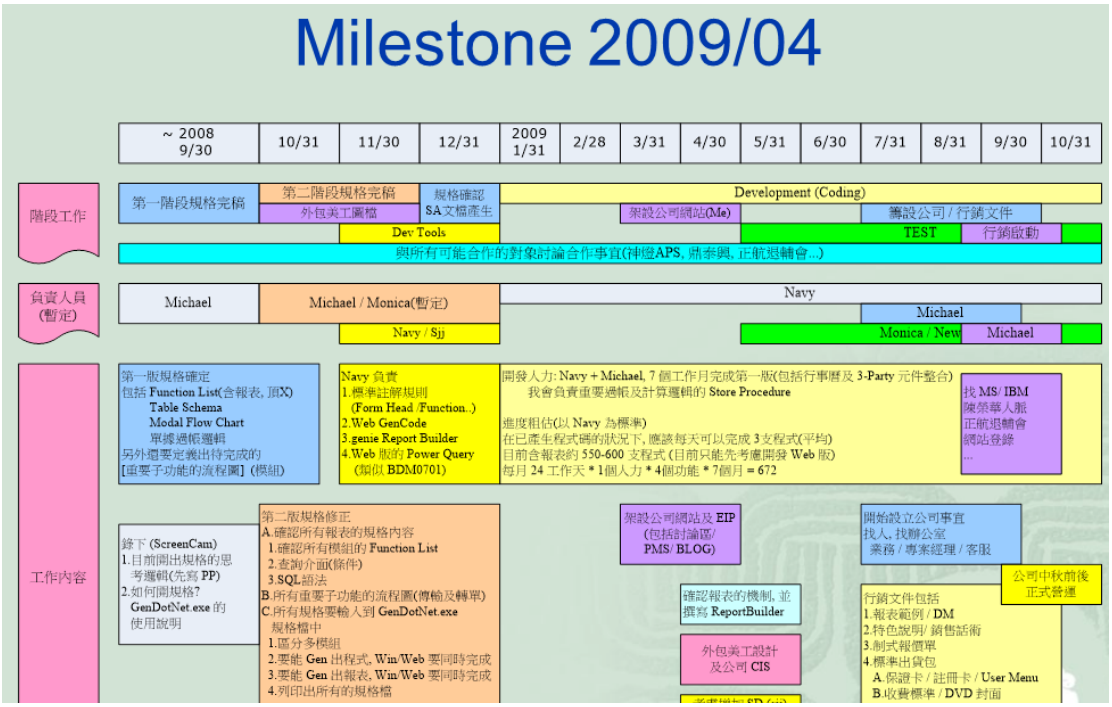
想要將一個專案制訂出一套好的 WBS 其實需要不少經驗累積，確實有些專業門檻。最好的方式是透過師徒制的協助，如果沒有資深人人的帶領，就只能在實務中打磨了，相信我，這不容易。

下圖是一個 WBS 的簡單範本，通常是類似

		Task Name	Duration	Start	Finish	Predecessors	Resource Names	% Complete
1		客戶關係管理系統	52 days?	Mon 12/5/7	Tue 12/7/17			15%
2	✓	需求收集	7 days?	Mon 12/5/7	Tue 12/5/15			100%
10		撰寫規格書	15 days?	Wed 12/5/16	Tue 12/6/5	9		39%
11		撰寫SA規格	7 days?	Wed 12/5/16	Thu 12/5/24			55%
12	✓	建立SA規格撰寫規	2 days?	Wed 12/5/16	Thu 12/5/17		[redacted]	100%
13	✓	確認撰寫規範	1 day?	Fri 12/5/18	Fri 12/5/18		[redacted]	100%
14	✓	模組A規格撰寫	2 days?	Tue 12/5/22	Wed 12/5/23	13		100%
18		模組B規格撰寫	3 days?	Tue 12/5/22	Thu 12/5/24	13		0%
19		規格4	1 day?	Tue 12/5/22	Tue 12/5/22		[redacted]	0%
20		規格5	1 day?	Wed 12/5/23	Wed 12/5/23		[redacted]	0%
21		規格6	1 day?	Thu 12/5/24	Thu 12/5/24		[redacted]	0%
22		模組C規格撰寫	1 day?	Thu 12/5/24	Thu 12/5/24	13		0%
23		規格7	1 day?	Thu 12/5/24	Thu 12/5/24		[redacted]	0%
24		規格8	1 day?	Thu 12/5/24	Thu 12/5/24		[redacted]	0%
25		撰寫SD規格	15 days?	Wed 12/5/16	Tue 12/6/5			30%
26	✓	建立SD規格撰寫規	2 days?	Wed 12/5/16	Thu 12/5/17		[redacted]	100%
27	✓	確認撰寫規範	1 day?	Fri 12/5/18	Fri 12/5/18		[redacted]	100%
28	✓	模組A規格撰寫	3 days?	Thu 12/5/24	Mon 12/5/28	27,14		100%
32		模組B規格撰寫	6 days?	Fri 12/5/25	Fri 12/6/1	27,18		0%

除了常見的 Excel，通常會整理出類似甘特圖的 milestone

如果有好用的專案管理系統的話，通常只要將 WBS 匯入，就可以整理出很多所需的圖表



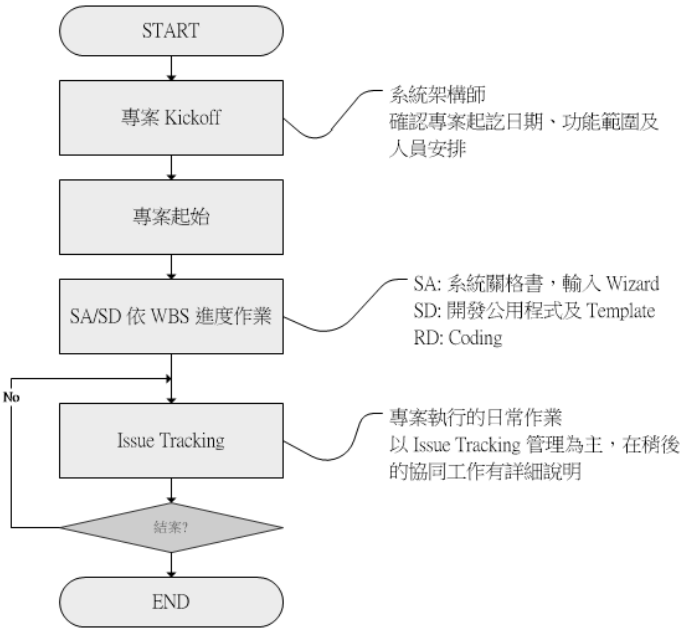
WBS(Work Break Structure) 的建立

主要是要確定專案的範圍(Scope)，在開發 ERP 系統時，最大的變數就是規則反覆的變動，規格變動是不可能完全避免的，但是必要的取捨是很重要的。這和專案經理的職責有關，專案經理的角色有兩個重點

1 和內、外部相關的部門溝通、協調及進度控管

2 規格變動是否納入行程？若是，則要考慮納入哪一個版本？是目前版本還是下一個規劃中的版本？

依以上的說明，筆者整理了一個簡單的實際作業流程圖



專案管理的主要目標是
1 如期：在規定的期限內完成專案的所有工作。
2 如質：完成既定的工作程式並保證品質如規格書內容。
3 如預算：用預算內的資源，包括人力、物力、財力。

專案管理的首要任務是
1 設定方案的範圍(Scope)
2 然後將專案的作業，細分項目到日，就是 WBS 的建立。
3 再來就是依計劃檢討成效，通常以日為工作進度，以週為單位檢討進度。
如果進度有延遲，再適度調配相關的資源，以追上落後的進度。

如果發現進度延遲已無法避免，就先看看緩衝時間(buffer)是否足夠？實在不行就要重新調整工作任務的時程。重點是不要當鴕鳥或是試圖刪除測試等日後必成大患的作業。要勇於面對、早期發現，即使進度稍有落後，只要事先告知客戶，一般而言，都可以達成協議。

聽起來很簡單？都是一些 common sense 老生常談，但是由誰來執行？會不會因為檢討會議太多，反而延誤了專案的進度。工作任務的預估行程和實際的執行進度是否一致？會不會太過樂觀和悲觀？
如果延遲該如何追上進度？加班或參加人手？

這些點點滴滴的問題，如果沒有經驗豐富的專案經理處理，肯定會造成專案的崩潰。

以上絕大部分都是專案經理的職責所在，所以跟專案經理當然要負專案的成敗責任。事實上，就因為專案經理的工作負擔非常重，所以更必須倚賴專案管理系統(Project Management System，PMS) 來紀錄追蹤管理 WBS 及 Issue Tracking。

專案的所有成員，只要隨時將目前的工作進度，輸入到系統中，所有人員的進度就一目了然，如果有異常的狀況
1 系統會自動提示
2 相關主管可隨時查詢
如此就可確保專案進度不致失控。

接下來 讓我們繼續討論第二個主題

二、協同工作管理

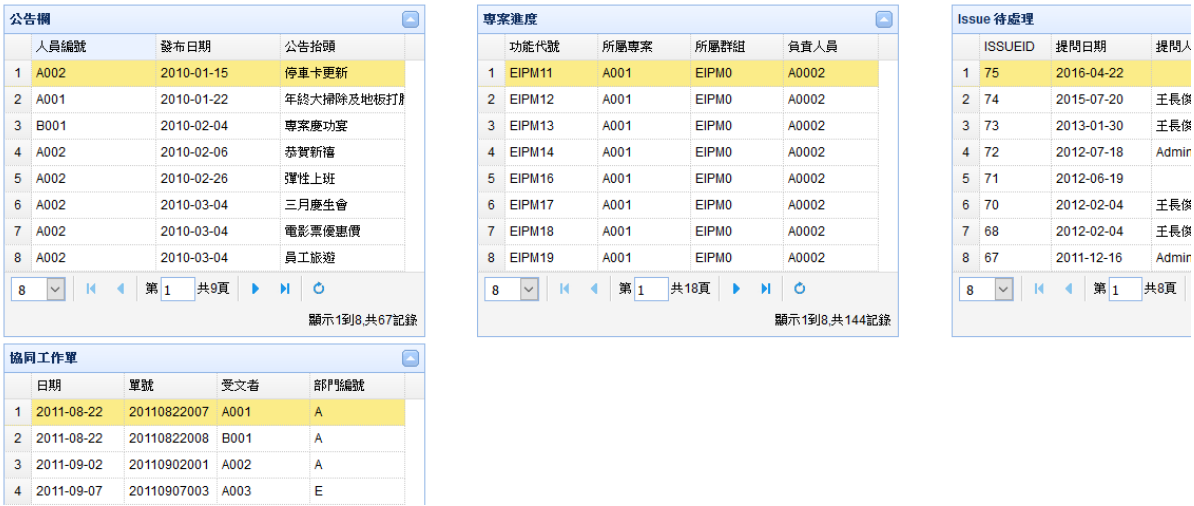
協同工作是相對於傳統的集權式管理的概念，它的要點是因為組織中，一件工作的完成，通常需要多位成員的協同合作才能完成，所以每個專案成員，除了做好自己的工作外，並要隨時關注其他相關人員的進度，並自動接續完成相關的工作(程式)，也就是讓工作流順暢無礙，只要有擋路的石頭，就會很快的被檢出並排除。

受限於早期資訊系統並不完善，及網路環境尚不成熟，每個人在工作完成工作後，都只能回報給主管，再由主管指派工作給下一個執行人員。所以主管是資訊的擁有者其控制者，工作非常繁重(這也是其價值之所在)，但是有這個必要嗎？

協同工作軟體的概念，就是在解決這個問題：讓資訊自動且即時的傳送到負責該作業的人員手中。不再需要透過管理者來控制所有的進度，效率自然會加快。當然，專案經理也可以隨時查詢專案診進度。但是當有異常發生時，系統就應該自動提出警示，也就是只做『異常管理』。

也許會有人質疑，這樣怎麼能確保工作的內容服務？這可分兩方面來討論
1 即時回報給主管，主管也不可能自己去測試，那不是主管的工作，是測試人員的工作。
2 協同工作的重點在協同工作，前提是每個人的本職學能都要有一定的水準，也就是協同工作要有高效，必須建立在小而美的團隊。就像最近很流行的敏捷開發，他們所建議的開發組織大小 是在七加減兩個人就是不到 9 個人。

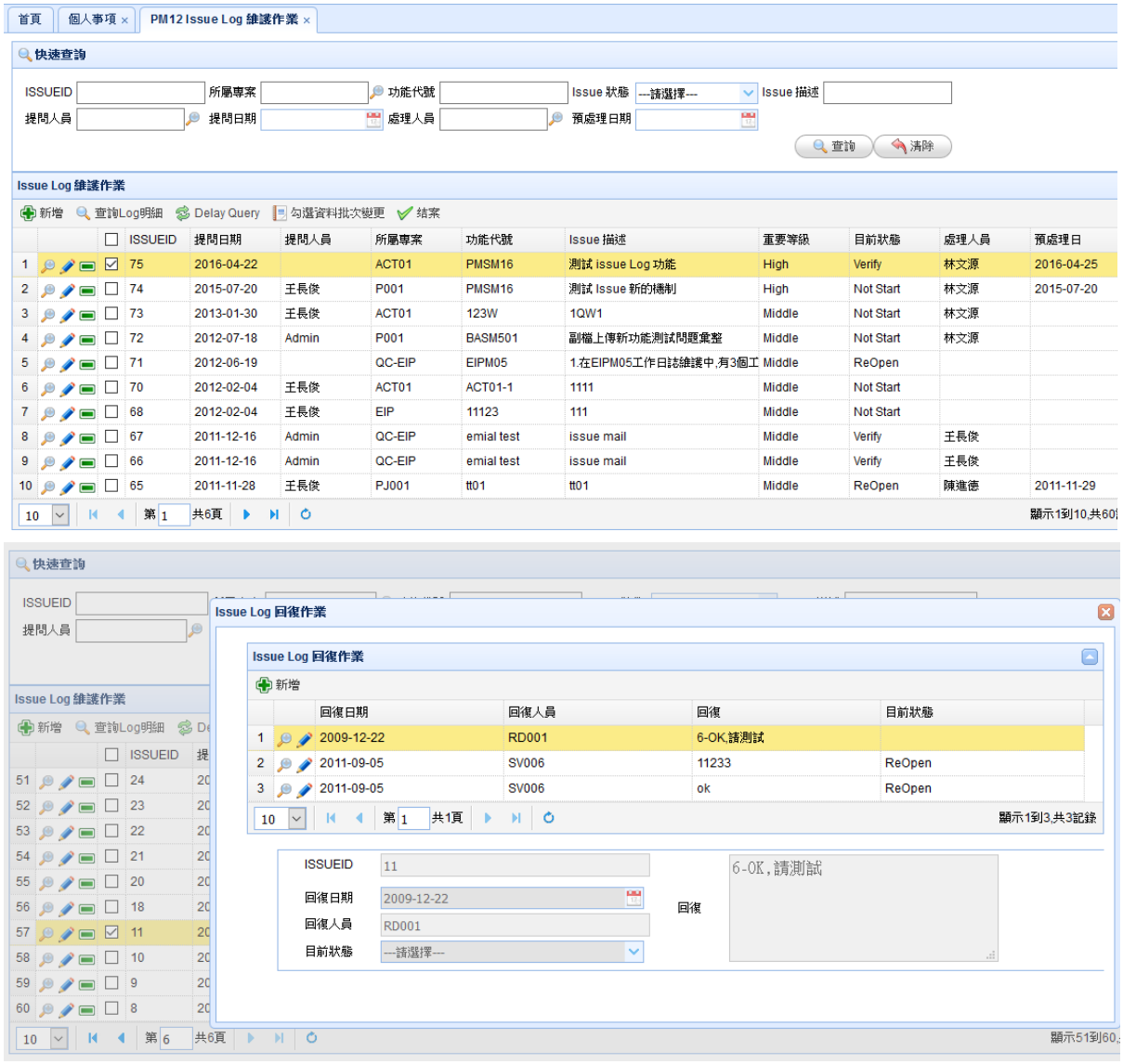
協同工作軟體，可以藉由下面的介面呈現
1 在登入系統時，自動顯示待辦



2 或是，前一項工作完成時，自動發送 mail 給下一位人員，該人員就可以經由

Hyper Link 或相關的提示自動返回系統，自動登錄到代辦的工作單據。

3 如果是 Issue(含 bug)，只要每個人員在工作完成後，輸入已完成的內容及變更狀態 (如 Vertify/Closed)。系統就會自動依 Issue 的狀態，自動將工作派送給下一個處理的人員。



4 也可以透過 EMS 系統(Event Management System)，自動派送應完成但尚未完成的工作項目(這屬於異常管理)，通知該人員及其主管。

如此一來，工作流就自動且順暢的傳送相關的工作到指定的人員手中，一關接著一關，資訊一旦透明，效率就自然加快，而且偷懶的人員就無所遁形。

大部分的企業入口網站系統，都具備有協同工作的相關功能，另外也有一些專用的軟體可以導入，請自行參考下列網站的說明

下面這個 blog 網址有提供幾個不錯的專案管理軟體，僅供參考
http://dreamerslab.com/blog/tw/comparing-5-project-management-tools/
詳細的使用說明，請自行 Google 查詢即可。

同專案管理系統，筆者也在自行開發的系統中，自行開發部份協同工作的程式，小而美、但夠用。主要也是著眼於和 ERP 及 PMS 專案管理系統的整合，方便且無縫。至於貴公司，就請自行考量要自行開發或導入已完成的現有套裝軟體。

筆者在本章中，主要是討論專案管理自動化這個課題，目的是要確保專案在執行的過程中，不會有非程式的相關問題產生，導致延遲。因為光是確保專案本身的進度會一直在正確的軌道上運行，就已經要耗費專案經理絕大部分的時間跟精力，如果還有一堆管理的工作，那就真是屋漏偏逢連夜。

專案管理系統及協同工作軟體，都有現成的產品可以使用。有商業版本，也有開放原始碼版本、也可以自行開發。重點是要專注在專案進度的有效控管，有經驗的專案經理都知道，專案只要不逾期，週週有進度，基本上，就不會有大問題。通常較能確保專案可以如期的結案。

第十一章
系統化的測試

有物混成，先天地生。
寂兮寥兮，獨立而不改，周行而不殆，可以為天下母。
吾不知其名，字之曰道，強為之名曰大。
大曰逝，逝曰遠，遠曰反。
故道大、天大、地大、人亦大。
域中有四大，而王居其一焉！
人法地，地法天，天法道，道法自然

老子。道德經

第十一章 系統化的測試

測試是一個非常重要，但卻常被忽視的工作。基本上，沒有經過完整測試的程式，就不能算是已經完成的版本。話雖如此，但是一旦碰到專案行程已經嚴重延遲，或是人力吃緊時，最常被犧牲(壓縮)的通常就是測試的行程。為什麼？因為有沒有測試，是無法由城市的外觀看出來的，另一個主要原因是因為，即使有測試，也無法保證程式就完全沒有錯誤。

作者自己起筆寫著一章時，也是思之再三、卡彈良久。但是測試這個議題，是整個軟體開發過程中，絕對無法迴避的重要課題。所以筆者還是盡量分享個人的一些淺見及經驗給各位參考，如有疏漏不足之處，尚請各位先進不吝指教。

在本章中，作者嘗試改用另一種寫法，用自問自答的方式來說明測試的一些重要內容及課題。因為本書並非測試的專門書籍，所以大部份的陳述還是圍繞著企業資源規劃系統為主軸。

問題一：為何需要測試？程式設計師不能就直接寫好沒有錯誤的程式嗎？
說明：在所謂內行人的眼中，看到這個問題，就會用不屑的語氣說，能寫出沒有錯誤程式的程式設計師，全世界只有兩個人：一個還沒有出生、另外一個已經死了。這當然是去玩笑話，但這也很接近事實，因為程式是由一連串的邏輯指令所組成，其中牽扯到的方方面面實在太多，要想寫出完全沒有錯誤的程式，雖然理論上可行，但是實務上會面臨

1 成本太高
2 難度太大
3 開發時程會拖很久等

所以除非是像太空梭、大型主機的作業系統核心等這類超級無敵重要的系統，否則都是以一般正常狀態下操作無誤為首要目標，而不強求完全沒有錯誤。
話說如此，但是要求研發人員盡可能寫出錯誤很少的程式是有必要的。因為如果工程師交出來的程式，本身的問題就多，或是改了 A 錯誤但是卻爆出 B 錯誤，這種情況如果層出不窮，測試人員沒有@@xx 就算他修養很好。

話說得簡單，但要如何才能做到？除了研發人員本身的寫程式功力要持續精進、並對所負責撰寫的程式規格要確實深入了解外，一定還要有一套標準的開發模式，才能確保程式開發有一定的品質。

主要的方式稍後章節會有詳細說明，此次簡單列舉幾個重點

1. 程式應該盡可能由元件(Component) 組成

2. 要先開發完整的公用程式庫(Library)，可供呼叫。
3. 每支程式的第一個版本，盡量都是透過 Wizard 工具產生，確保程式的一致性。

這個部分，請參閱
1 第七章：程式開發工具的整合
2 第二部分：建構開發標準

因為經由 Wizard 所產生的程式，基本程式的問題就會比較少(當然，前提是樣板程式本身，需先千錘百鍊的測試過)。應用程式工程師是在這支品質有一定基礎的程式上，加上樣板程式沒有產生的新需求的程式，也就是說工程師只要專注在擴充程式碼的撰寫及測試，問題當然就會減少許多、程式的品質自然就比較好。

所以這個問題乍看很白，但卻很有深意。本來就應該朝 Bug Free(無錯誤) 努力，如果工程師已經習慣『有 bug 是正常，發現趕快改』的思維，就無法再進步，僅依賴測試人員的測試，還是非常有限。但是如果能轉換思維，朝此方向努力，即使沒有辦法百分之百達到完全沒有錯誤，但是程式的品質還是會有大幅度的提升。

問題二：測試就能保證 Bug Free(完全沒有錯誤)？
說明：在前一個問題的討論中，筆者事實上已經針對此問題做過說明。要完全沒有 bug 是有很大的難處。這源於程式本身是由邏輯指令所組成，以一支常見的出貨單的程式為例，在存檔前可能就有諸如
1 倉庫存量不足檢查
2 客戶信用額度不足警告
3 售價低於最低售價(或現行成本) 的檢查
4 售價要依售價策略的定義取得(不同的客戶會有不同的定義)
5 客戶帳款/票據的餘額超限檢查
等等

這麼多的檢查程式(這還只是存檔前的檢查片段而已)，而且這和表單上輸入的資料有關，一般的測試方式，會依上面的規格編寫為測試案例，這是標準的測試程序。只能保證在一般正常的操作情況下，程式是正確的。但是如果發生諸如輸入的產品編號不存在、或是交貨日期不合理(如 2014/2/30) 等特殊的例外的情況 ，這可能就不在測試案例考慮的範圍內。上面只是一個非常簡單的案例，真實的狀況會更五花八門。

所以要做到完全測試到完全無錯，不僅在邏輯推論上難以達成，即使可以做到，其花費的成本可能數倍於開發成本。因此採用以測試案例為測試主軸，在佐以當

異常狀況發生時，再加入防錯程式，這在實務上是較為可行的作法。

問題三：測試有哪些方法和理論？
說明：

問題四：應該何時開始測試？和專案期末的驗收進度有關嗎？
說明：

測試的時間越早開始越好，這是稍有經驗的專案經理都知道的常識。但是 ERP 系統比較特殊的地方，是在於需要數個功能組合起來(一個作業流程)，才能完整的測試及使用。所以在編訂測試案例時，就會以一個作業流程的程式完成後，再安排一個測試的工作任務。當然，這同時也要配合測試案例(Test Case)的編寫。

稍後我們會針對測試案例有詳細的說明，簡單的說，測試案例(Test Case)是在模擬一段日常作業的完整資料。通常我們會依如下的程序編寫測試案例

1 單一功能測試：針對新增、修改、刪除、匯入等一般的操作功能測試，這屬於基本的標準功能操作測試。
2 作業流程測試：如果單據間有串聯，需測試單據的諸如複製及轉單作業是否正確。
3 輸入完後，比對相關的報表數字是否正確(測試案例會事先手工計算並編輯完成)。

依此而論，測試也會是工作項目(WBS)的一部分，和撰寫程式一樣都是由專案經理依進度安排，之所以會特別有此一問，是因為很多專案經理，會誤以為專案的驗收期等於測試期，所以在編寫工作任務時，測試的工作全部都安排在專案的後半段，專案進行時，工程師就不斷的寫程式，測試工作在程式幾乎全部寫完後才開始。這樣安排的風險非常高，一旦在後期才發現較重大的錯誤(bug)，根本就來不及修改。而且如果測試的時間和程式完成的時間間隔太長，程式中若有特殊的規格和需求，工程師要花費許多時間去回想原先的程式寫法邏輯。所以如果能在寫完程式就開始測試，有問題發生工程師就能立刻修正，這會是較有效率的做法。所謂防微杜漸，再問題剛發生的初期，就迅速的解決，這看來沒有效率的方式，才是最有效率的。

至於專案的驗收期，這段時間是在和客戶確認，所完成的欲交付的程式，是否和規格確認確認書內容相符。它必須是已完成內部測試的版本，而不是才要開始測試的版本，相信我，客戶不會想看到一個看起來已完成，但是卻沒有測試過的版本。這兩者之間有非常巨大的差別。

很多功能會卡在驗收無法完成，通常的原因是

1 進度延遲，程式尚未完成，這是最慘的。

2 程式已完成，但未經完整測試，錯誤太多。

3 程式已完成且符合規格書，但客戶認為規格書不明確(或是使用者發現某部分程式與現有狀況不同)，要變動規格。

碰到上述這些情況，專案想要順利結案，就會有很大的困難。但是有經驗的專案經理都知道，這是專案在執行結案時的常態。所以筆者才會不斷地強調：確認規格時，除了文檔外，還必須要有雛形系統可讓使用人員操作，當然，這和開發方法和使用的軟體工具有直接的關係，我會在第四部份討論更完整的細節。在執行專案的過程中，就應該先架設好測試用主機，如此才能盡可能地確確保專案能如期完成。

問題五：測試案例的內容格式為何？該如何編寫？

說明：

測試案例(Test Case)是筆者目前用來測試程式版本是否正確運作的主要方法，只要測試人員依案例內容測試過，即可保證在大多數正確操作的情況下，程式都能正確的運作。所以測試案例的重要性當然就不言可喻。

測試案例並沒有特定的格式，您可自行依需求定義格式內容，但內容至少應包含下列的原則及內容：

1 程式代號：

每一支程式都應該給予一個獨一無二的代號(function tag)，代號應依編碼原則給定，以下是一個簡單的參考範例

OPOM02 客戶訂單維護作業

OPO 表示採購訂單模組(此處固定是 3 碼，可以自訂)

M 表示這是一隻資料輸入類的表單

0 表示第一階功能表的序號

2 流水號

2 程式名稱：欲測試程式的中文名稱

3 測試人員：負責測試此功能的人員

4 動作(Action)

測試時的動作行為要點，例如

新增：連續新增 3 筆資料

修改：修改其中第 2 筆

刪除：刪除第三筆

匯入：將事先準備好的 excel 檔案轉入資料庫

查詢：查詢名稱中有 [科技] 的資料(應該有 2 筆)

傳單等特殊的作業：將 A 報價單部分轉為 B 訂單
等等

5 測試資料
事先編好的測試資料。要測試的程式需要配合對應的資料內容，通常只會先寫部分較重要的欄位資訊，因為有些表單欄位過多，實務上非必要的欄位可不必編寫在測試案例中。測試資料一定要事先有條理的編寫，因為測試並不僅僅的功能正確操作的確認而已，一般而言，測試的問題都不在基本操作，而是商業邏輯的錯誤，例如傳票過帳後，資產負債表數字不正確，或是出貨單存檔後，現有庫存正確，但是客戶的應收帳款沒有增加(或是錯誤) 等。這類的問題才會讓人心力交瘁。
6 測試檢核點(Check Point)
主要測試的目的，通常是比對某些報表的數字、或是開窗後應顯示的內容。這些結果一定要先整理成圖檔或 excel 數字，才能方便測試人員有效率的測試。
7 備註(Memo)：其他特別需要注意的測試重點
8 狀態(Status)：測試的結果狀況 如 Finished 或 Reopen 等
9 錯誤代號(Issue ID)：如果測試有錯誤，應填寫錯誤單(Issue)，方便日後追蹤。

測試案例(Test Case) 通常是寫成 Excel 和 word 文檔，每次測試前先列印數份，然後填上此次的測試日期及相關的測試人員，並將結果更新回狀態欄。

目前筆者自己並未將使作業程式化，仍維持檔案記錄的方式，日後是否需要程式化，請依您自己的需求自行決定。程式化最大的優點是可以和 Issue Tracking 系統整合。

問題六：測試時當有錯誤發生時，該如何下手查問題？還是只能全憑測試人員的直覺？
說明：程式發生錯誤，通常可大略分為兩種狀況
1 一般的操作問題：這類問題通常是程式的錯誤，只要記錄完整操作的流程，並回報研發人員，通常就可以解決。
2 數字不符：在測試階段，甚至日後系統上線開始使用，都可能會發生這種情況。因為企業資訊系統非常強調資訊整合，輸入某些單據後，會自動過帳影響多個期末值，下面舉幾個簡單的例子
a 出貨單存檔後，庫存量會減法，應收帳款會增加。
b 訂單轉出貨後，已訂未轉出的數量會減少 等等。
這一類的問題一旦出錯，因為可能和整個流程的單據都有關係，加上很多程式是呼叫 SQL 的資料庫程式(如 Store procedure 或 Trigger) 處理，這會增加解決問題的難度。

測試人員最基本的責任，事實上只要能察覺出數字有錯誤並回報即可。找出錯誤並修正當然是研發人員的責任，不過筆者通常會告訴較資深的測試人員，如果想更進一步的了解企業資訊系統，就該更進一步的比對相關的明細表及統計表，並能指出哪張報表的哪個數字有問題。也就是不但能夠由測試案例的標準數字發現錯誤，還能經由不同報表的比對，發現問題的所在。因為測試案例不可能無所不包，所以測試人員要有主動找出問題根源的能力，這會幫開發人員很大的忙。比對報表是一個標準的查帳方式，另一個更強的方式是透過下 SQL 語法，直接確認報表數字的正確性，這是較進階且能深入了解企業資訊系統的最佳方式。通常這是系統分析人員及專案經理才具備這個能力。

所以資深的測試人員，如果能學會 SQL 語法並運用到測試工作上，也就表示他開始具備了系統分析師的基本能力。當然對他未來的職場生涯有正面的幫助。

問題七：測試是誰的責任？需要有專門的測試部門及人員嗎？
說明：廣義的測試，是所有可能會用使用到系統的人，都要協助測試。在使用系統時如果發現問題(包括建議)，就應該登錄到專案管理系統的錯誤追蹤功能(Issue Tracking System) 中。
『每個人都有責任，很容易就會變成每個人都沒有責任』
所以，只要資源沒有問題，就還是應該要有獨立的測試人員編制。但筆者認為測試人員的編制應該屬於研發部門，不需要有單獨的測試部門，而測試人員應該要有三種來源
1 專職的測試人員
2 新進的研發工程師：在正式接系統開發及維護前，應該先當測試人員一段時間。主要是讓新進人員認識系統功能，並能從使用者的角度來理解系統。
3 客戶服務部門的資深客服：最好是略懂一點常用的 SQL 語法。

這三種測試人員，可依狀況安排所負責的功能。大部份的中小型獨立軟體開發商(ISV)，由於資源不足，大多會以第 3 種測試人員為主力。
因為此類人員熟悉系統且溝通無礙，但是由於客服工作本來就很繁忙，如果可能的話，在專案開發期間，最好能讓她們能專注在測試工作上。

一旦有專職人員，當然測試就是測試人員的責任。他們應該聽從專案經理的指示，依 WBS 及 Issue Tracking System 作業，盡力做好手啄木鳥的角色，全力找出企業資訊系統中的所有臭蟲(Bug)。

對於將測試工作外包，筆者是認為如果人力上可，最好還是自行測試較佳，外包測試通常只能依測試案例測出一般的操作問題，這個部分應該可以採用自動化的

測試工具來協助處理。

關於自動化的測試工具，我會在下個單元簡單的說明。

問題八：有自動化的測試工具可以使用嗎？
說明：測試是一個非常重要的工作，但是測試人員常會有『成者功在研發，敗者責在測試』的感嘆。一旦有了專職的測試人員，大部分的人都會『有這完全是你的責任』、『你要測出所有的錯誤』、『你不是測過沒問題嗎，怎麼還有 bug？』 等等的想法。

但是在前面已經反覆討論過，要測試到完全 bug free 是有困難的，測試人員只要能做到依測試案例測試，就可以了。當然，如果能更進一步，測試得更深入，甚至能主動發掘沒有寫在 Test Case 中的問題，自然是最好。

真正在測試階段，同一個流程功能可能要反覆測試多次，而且到了測試後期，就會整合更多的功能測試，如果沒有自動化的測試工具輔助，測試人員會有『日復一日，monkey test』的無力感。

因為在測試期，程式版本會不斷地發布更新。測試人員就必須不斷的重頭輸入資料，久而久之，測試人員也會心生無力。所以引入適當的測試工具是很有必要的。大部分的測試工具，原理大致相同，都是先錄製好一份腳本，例如輸入一張完整訂單，然後轉為出貨單，把操作行為腳本存成檔案。測試的時候就等於在播放一次，系統就會依照錄製的內容重新再輸入一次(當然，通常在測試前會先準備好一個空白的資料庫)，只要第一次多花點時間將要測試的內容錄好，就可以反覆測試測試案例的內容。

這種方式，主要是取代『猴子測試 monkey test』，讓測試人員專注在報表及數字的正確性，避免測試人員彈性疲乏。以下列出幾個相關的網站供各位參考

胡百敬老師的影音教學說明，淺顯易懂
https://www.youtube.com/watch?v=-B_Q07Mm_3E
軟體測試工具詳解(內有各式測試工具的列表及簡單說明)
https://kknews.cc/zh-tw/tech/ya6gxkb.html
軟體測試工具與流程文章彙整 (效能 + 自動化測試 + 安全測試 + 環境佈署)
http://www.qa-knowhow.com/?p=2442
20 個軟體測試工具大放送
https://read01.com/4xOJML.html

認識 Selenium
https://learngeb-ebook.readbook.tw/intro/selenium.html
Microsoft 的 Visual Studio Web 效能測試(壓力測試工具) 簡易流程
https://dotblogs.com.tw/milkgreenteaprograme_c_sharp/2017/01/11/225514
HP QuickTest Professional (QTP)
https://saas.hpe.com/en-us/software/uft
上海澤眾軟件
http://www.spasvo.com/

官方的雲端開發測試平台
https://www.cloudopenlab.org.tw/index.do

第二部分
中興以人才為本

第十二章
從上而下的建立組織

天將降大任於是人也，必先苦其心志，勞其筋骨，餓其體膚，
空乏其身，行拂亂其所為，所以動心忍性，曾益其所不能。

孟子

第十二章 從上而下的建立組織

軟體產業是標準的腦力密集產業。最終的產品是以數位的格式儲存在電腦中，而讓他誕生的則是正在閱讀本書的你—程式開發人員。

早期的軟體開發方式，帶有太多的藝術家事的強烈個人風格，過份的強調個人英雄主義。這在小團隊開發時，是個快捷有效的方式。但是，如今的時空環境已有很大的不同，開發系統所需的技能越來越多、越來越複雜，而且客戶的需求也因為整個經營環境的劇烈變動，而更加難以滿足。所以就必須要改為正規作戰，要組織一個完整的開發團隊，才能正面迎戰，確保專案能夠順利結案。

現在的大型開發團隊，動輒成百上千人。這個時候，適當的管理及規範就非常的有必要，況且經過長期累積的經驗證明，軟體系統的維護營運成本，遠高於開發成本。適當的訂出開發的標準作業流程(SOP)，小至變數的命名規則、函式的命名規則、程式的註解規範等，大至元件的生命週期管理等等，統一的標準規範要求所有開發團隊的成員遵循，就非常重要。

標準的研發團隊，應該至少要有下列七個角色職能

1 系統架構師：如同人類的大腦
2 專案經理：如同人類的心臟
3 資料庫管理師：如同人類的經絡穴道
4 系統分析師：神經系統
5 系統設計師：骨架
6 程式設計師：人的四肢
7 測試人員：如同人類的免疫系統

各職能的工作內容大致說明如下

1 系統架構師(System Architect)

系統架構師是企業資訊系統的大腦，負責專案範圍的設定、模組及功能的初稿，以及開發平台(框架 framework)的定義。他必須對整套企業資訊系統的所有細節規格都了然於胸。所以，架構師最好是在技術部門內由

程式設計師→系統設計師→系統分析師

一路歷練多年，精熟技術底層且有實務經驗，然後再擔任專案經理，培養和客戶及研發單位等其他人員的構通能力，才能勝任此職務。當然也有其他非技術人員出身的例子，如由資深顧問師升任，不過有技術背景一般而言更加。

一個專案，如果沒有好的系統架構師，專案的成功率就會大打折扣。關於系統架構師的重要性，筆者會在第十四章中詳細說明。

2 專案經理(Project Manager)

職務說明：擔任軟體工程的規劃、主導、協調專案的管理者

- 參與軟件工程系統的設計、開發、測試等過程；
- 協助工程管理人保證項目的質量；
- 負責工程中主要功能的代碼實現；
- 解決工程中的關鍵問題和技術難題；
- 協調各個程序員的工作，並能與其它軟件工程師協作工作。

專案經理顧名思義，是管理整個專案所有相關大小事務的人。他是開發企業資訊系統的心臟，負責驅動主導專案的進行，並盡一切努力讓專案能在結案日前如期完成。

專案經理最好要懂技術，但並不是絕對必要。他主要的工作重在溝通協調及監控專案進度的進行。

對內：他必須要隨時掌握開發團隊的動態以及進度。

對外：要定期跟客戶回報進度，並維持良好的關係及溝通管道。

專案經理貴在人和，他是否稱職會影響整個專案的成效甚鉅，所以不可不注重。

某些資源不足的小型獨立軟體開發商(ISV)，專案經理還會兼任系統分析師(System Analysis) 的角色。如果專案的規格不是很大，無妨。但前提是不能延誤本身的工作，筆者待過的公司規模都不大 (事實上，台灣本土地的軟體開發商，大部分規模都不是很大)。所以會特別說明，如何在人力不足的情況下，如何善用有限的資源。

3 資料庫管理師(DBA)

職務說明：負責資料庫系統的設計、開發、管理及維護

- 設計整體數據庫的架構，數據庫系統的建立
- 制定數據庫備份和資料恢復的工作流程與相關規範
- 監督數據庫系統的安裝
- 對數據庫的空間進行分析，有效管理數據庫的運作

資料庫管理師的主要工作，是資料庫的管理和維護。

這當然包括 SQL 語法的調教，讓關聯式資料庫能順暢的運作。專案開發的過程中，資料庫管理師的大部份工作是，依系統分析師開立的規格，撰寫相關的資料庫程式，包括複雜邏輯及所有報表的取資料(值)作業。再交付給系統設計師及程式設計師，在程式中呼叫使用。在我們的框架中，所有的報表，最好都是由程式產生器直接產生前端相關的程式，至於後端資料，就直接呼叫寫好的資料庫 SQL 程式(如 Store procedure 等)。

因為本書一再的強調『以資料庫為開發核心』的開發方法論，所以資料庫管理師負責撰寫的 SQL 相關程式，就是核心的重點，與整個企業資訊系統的效能有直接的關係，是重要的關鍵角色。

4 系統分析師

職務說明：系統分析員（System Analyst，簡稱 SA，又稱系統分析師）。

系統分析師必須具備基礎的軟體開發與質量控制的相關知識素養，並須熟練掌握軟體開發流程(SDLC)，運用於軟體開發的基礎工作。就性格特質而言，具有良好分析能力、組織能力與邏輯思辨能力者，較適合勝任系統分析師的工作使命

- 分析、類比、評估系統體系結構、功能、性能和效益
- 整體規劃系統的設計與開發管理工作，如客戶需求分析、專案可行性分析，定義產品功能，原型開發等
- 全面管控專案的開發流程，並隨時對系統進行測試調整
- 提供技術指導，使系統操作技術和解碼編程能夠更有效使用
- 參與客戶的系統軟體、硬體平臺的安裝與執行
- 依據客戶的需求，提供故障診斷服務與技術諮詢

系統分析師的工作主要是依循(承接) 系統架構師定出的開發大綱，依專案經理排訂的時程，撰寫規格確認書。大部分的軟體開發商和企業內部在開發系統時，都會先寫好規格確認書，不論是否完整。

但是有一個很奇怪的現象，就是規格確認書和實際執行的開發程式中，有一道鴻溝。因為規格確認書通常是 Word 或是 Excel 等文檔。上面圖文並茂的詳述規格需求。系統分析師完成規格確認書並和客戶確認後，就會和系統設計師及系統工程師討論說明，然後依規格書開始寫程式。

您會問說，哪裡不對？我們一直以來都是這樣做的啊。是的，這並沒有錯，可是你不覺得這樣很沒效率、很浪費工程師的時間嗎？由豐田汽車管理提出的 TPS，主要的管理重點之一就是要『消除所有的浪費』

為什麼辛辛苦苦寫好的規格書，還需要研發工程師從頭理解、重新寫程式，難道就不能直接把規格書的內容，直接產生所需要的程式碼嗎？
誠如筆者不斷的重申，『想偷懶是人類進步的原動力』，程式產生器(Wizard) 就因此需求順勢而生。關於程式產生器，筆者已在第七章中有詳細的說明，請自行參閱。

我的實際做法是，專案經理和系統分析師，都必須學會使用程式產生器，只要將規格輸入到程式產生器中，Wizard 會負責產生

1 程式的第一個可執行版本
2 規格確認書的初稿 (通常是 Word 格式)
3 將雛型程式上傳到測試主機，可以讓使用者直接操作使用介面

然後系統分析師，就會以程式產生器產生的初稿為基礎，在加入客戶要擴充的規格內容，一方面就可以和客戶確認規格是否正確，另一方面，軟體工程師則會直接拿到產生的程式版本，再根據增修後的規格書，來開始撰寫程式。如此一來 ，所有人員的工作都簡化，且沒有多餘的浪費，原本的鴻溝也消失了，流程也順暢，這才是正確的工作方法。

5 系統設計師

職務說明：依據系統分析結果，功能規格所訂之內容，從事資訊系統架構之設計，以確定系統之型態及作業方式

- 根據產品部門或客戶需求，對新產品、工具或方法進行研究、設計、開發等
- 對系統進行分析與規劃設計，確保與硬體系統的結合
- 持續對產品進行優化與升級

系統設計師通常是資深的程式設計師升任，通常都是拔擢技術能力較強的設計師高手擔任，他主要的工作內容有三點

1 開發平台的建立，含權限、主選單等以及共用表單
2 共用函式庫及元件的撰寫。
3 較高複雜程度的程式的撰寫

簡單的說，系統設計師是擔任拓荒者的角色。負責架構開發的底層程式，方便程式設計師能直接引用。來快速完成程式開發，同時也確保程式都有一定的水平，當程式設計師碰到技術上的難題時，系統設計師也應該負起訓練的角色，協助程式設計師解決問題並提升其專業能力。

6 程式設計師

職務說明：從事電腦軟體的程式設計、修改、安裝及維護

- 參與軟體工程系統的設計、開發、測試
- 協助工程管理人保證專案的品質，負責工程中主要功能的代碼實現
- 解決工程中的技術難題和關鍵問題
- 教導、指導程式工程師
- 協調各個程式工程師的工作，並能與其他軟體工程師協作

程式設計師，負責企業資訊系統所有程式的實際撰寫工作。他等於是人體的四肢、是實際的執行者。根據專案經理排定好的工作任務的行程及系統分析師撰

寫好的規格書及已經產生好的程式，然後呼叫系統設計師寫好的共用函式庫及元件。 然後，再加上自行撰寫的程式碼，最終完成程式的開發。寫完程式後並不是馬上就交付出去，而是應自行測試一次，這是最重要也最基礎的 Alpha 測試。然後再將完成的程式上傳至工作測試主機，並加入主選單，然後回報到專案管理系統。在接下來，就是配合測試人員的測試報告 (通常會紀錄在 Issue TrackingSystem 中)。如果有錯誤發生，依錯誤處理機制處理，直到該程式結案為止。

程式設計師必須遵循標準的程式開發程序，包括註解、變數的命名規則等，可以參考第五部份的說明。務必要做到任何一個程式設計師寫的程式，隨時可以交給另外一個程式設計師維護、修改的地步。這才真正達到標準化開發的境界，這才是符合標準開發程序的團隊。

7 測試人員

職務說明：針對軟體開發流程中，各階段的產品性能進行測試，找出設計上的錯誤或問題予以改正解決

- 建立可靠實用的測試環境
- 匯整實際組網環境方案，建立自動化測試方案
- 發展、維護與更新適合於應用產品的自動化測試版本和結構
- 分別在模擬環境和實際環境中對產品進行測試，並將測試結果回饋給相關部門和人員
- 針對分析測試提出報告
- 撰寫測試文檔，協助生產人員撰寫使用說明書

測試人員的工作是依據寫好的測試案例 (Test Case)，找出程式中所有可能有錯誤的程式碼。關於測試的細節請參考第十一章的詳細說明。
測試案例主要是由系統分析師撰寫，並依專案經理排定好的工作進度進行。將測試結果記錄在測試報告中，如果測試有錯誤，就輸入到 錯誤追蹤管理系統 (Issue Tracking System)，並配合程式設計師修改程式、在測試，如此重複一直到結案為止。

上述的各項職務，各有所司、並各司其職。只要每個角色都盡其所能去完成所交付的任務，專案自然就能順利的結案。不過要讓這樣一個開發團隊能溝通無礙、並能順暢的協同工作，並不是件容易的事。專案經理當然要扮演最重要的主導角色。關於組織有效率運作的議題，筆者將在第四部分有專章說明。

第十三章 組織架構及多種專案管理

在前一章中，筆者已詳述研發單位應有的角色及其主要的工作內容，並輔以人體的機能組織來譬喻。能否讓整個研發部門以一個分工合作的整體，順暢來運作，是關係到專案能否順利結案的重要關鍵。
下圖是筆者公司目前採用的組織架構

看起來和傳統的組織差異不大，仔細瞧瞧，最大的差別是以專案經理來取代傳統的研發經理。

您也許會質疑說，這不是換湯不換藥嗎？如果公司內只有一個專案在執行，的確是雷同的。所以最大的不同，是在於當有多專案在同時執行時，研發團隊是動態的組成開發團隊。當然，有些角色可能會有多專案共用的情況，例如系統設計師、 資料庫管理師和測試人員。

上述這些人力資源的分配，則是由專案經理提出計畫，然後由研發副總核准。(當然，如果組織架構有差異，你可以適當的調整)。

大部份採用研發經理制的軟體開發商，都會面臨一個難解的問題：當有多個專案同時在進行，很容易會顧此失彼，總會有部分專案因無暇顧及而難以結案。通常是以客戶抱怨的力道，來決定資源的優先分配次序。

您也許會想 『Michael，你說的應該是純專案的獨立軟體開發商，但是我們公司是套裝產品的軟體公司，我們只有一套標準的產品，專案經理智應該並不適合我們。』

軟體產業是個很有趣且多變的產業。早期標準的套裝資訊系統，可以滿足許多規模不大的中小企業的初次電腦化需求。所以套裝軟體的企業資訊系統軟體，市場一度非常的蓬勃發展。然而這個市場目前面臨很大的挑戰，由於大部分的客戶已經有過不止一次的電腦化經驗，所以標準化的套裝軟體，已經無法滿足這些客戶的需求。

但是純客製開發的資訊系統，又會有以下的問題

1 開發時間太長
2 費用太高
3 失敗風險太高

等等的不確定因素，所以註定無法成為主流。

因此，目前企業資訊系統，應該都會朝向『套裝版本加二次開發』，這種混血的開發模式發展，既擷取兩者之長又避其短版。
更有甚者，如果客戶有足夠的預算及人力，也可以考慮『協同開發』這種更緊密合作的方式。

協同開發方式的特點為

1 軟體廠商先依客戶的需求開發大部分的程式
2 客戶可以在開發的早期，也一併投入部分人力一起加入軟體公司的開發。
當然，也可以在驗收階段再加入，何時加入可由雙方討論後執行。
3 然後，軟體廠商再技術轉移所有的程式碼及相關文檔給客戶，讓客戶可以接手維護。
4 客戶投入開發人員人力學習維護系統，並加入部分新功能到系統中。
並在過程中，軟體廠商的程式設計師，應該提供直接的協助，包括撰寫程式碼等，讓客戶的開發人員能盡快上線。
5 除了原始程式碼(Source Code) 外，客戶應一併導入類似本書所主張的開發方法論，利用程式產生器產生第一版的規格文檔及程式。而非只單純的購買原始程式碼，單純的程式碼事實上更多的是一種負擔，而非資產。維護的成本過高且不利新功能需求的撰寫。

無論是上述的哪一種方式，多專案同時進行勢必是未來主要的潮流，這也是為何近年來 PMP 專案管理會如此炙手可熱的主因。因為想要同時管理好多個不同客戶的專案執行，用傳統的組織架構：研發經理制，是絕對不足夠的，一定要導入專案管理。

專案管理能力，絕對是未來軟體開發商決勝的關鍵，多專案管理事實上代表了軟體開發商是否有能夠有足夠的彈性，運用公司有限的資源，以滿足更多客戶的需求。這是一個 Win-Win 雙贏的策略，客戶的需求得到滿足，而軟體開發商贏得市場。

導入專案管理，絕不是僅僅在原有的組織中增設專案經理就可以了，沒有導入專案經理制，經驗的企業，通常會拔擢內部資深的人員擔任專案經理，但往往都沒有適當的加以培訓。或是只看過幾本相關的書籍，就直接上線，美其名是做中

學。事實上，這樣的專案經理很容易戰死沙場，也許會有少數專案經理，在歷經多個專案的摧殘後，能僥倖存活下來，但這畢竟是少數，代價過高。

並不是所有的人都適合當專案經理，軟體業的專案經理必須具備以下的能力

1 良好的溝通能力及 EQ
2 具備基礎的資料庫概念 (精通更佳)
3 良好的執行能力
4 具備一定程度的程式設計能力 (非必要)

技術能力並不是絕對必要，但是具備技術能力的專案經理，更容易掌握研發細節，才不會被程式設計人員誤導，甚至可以提供技術問題的解決之道。當然，沒有技術背景的專案經理，只要具備良好的溝通能力，也可以彌補此點不足。

專案經理較佳的培訓方式是師徒制
實戰當然是必須的，但必須有資深的專案經理，可以適時提供適當的協助。在新訓練專案經理的過程中，碰到諸如

1 專案進度延遲
2 人力資源不足
3 研發工程師要離職或心情不佳
4 客戶不斷地要求變更規格

等等千奇百怪的問題

每個問題都讓人煩不勝煩，但這也正是專案經理日常生活的寫照，所以專案經理制的優點是顯而易見的。但想要培養一名稱職的專案經理，可就不是件容易的事情。也正因為如此，所以多專案的執行能力，才會成為獨立軟體開發商，面對居烈競爭時，勝出的關鍵因素。

即使貴公司是自行開發企業內部使用的資訊系統，我也會建議適度的導入專案經理制，他的確能提高管理的效率，讓工作更聚焦。

第四部份
成功的關鍵因素

第十四章
架構師－成功的關鍵因素

天下之事，常發於至微，而終為大患；
始以為不足治，而終至於不可為。

西方諺語：The devil is in the details

方孝孺〈指喻〉

第十四章 架構師－成功的關鍵因素

要當應一名稱職的架構師，永遠必須牢記西方諺語：The devil is in the details。『細節裡隱藏著魔鬼』。

要寫一套企業資訊系統，程式並不難，基本上只要具備前幾章說明的人員組織及能力，開發出產品並不困難。但是企業資訊系統的規格有小有大，小的資訊系統也許只有幾十個功能及報表、大的複雜的企業資訊系統，功能就龐大許多，而越是複雜細緻的企業資訊系統，越到開發後期，或是寫到某些特殊的功能，如

1 物料需求展算 (MRP)
2 生產排程計算 (MPS)
3 月加權成本的批次計算
4 產品期初期末的明細異動及統計表

等等等許多棘手的問題

這些功能之所以會成為問題，根源的原因通常都和系統分析師對這些複雜功能的來龍去脈及所有的細節，是否透徹的了解，有直接的關係。

以筆者自身的經歷為例，年輕時第一次寫物料需求展算的經驗，簡直是惡夢一場。由於先前沒有物料需求展算的相關實務經驗，加上規格並不明確，僅憑著簡單看過幾本相關的書籍，及系統分析師並不精確的口述說明 (當時公司內部，沒有人懂這一段程式的完整邏輯)，就開始寫程式。

結果當然是慘不忍睹，不但數字錯漏、邏輯不嚴謹、計算效能低下，程式改版改到爆，最終雖然勉強能計算，但其中的程式碼，充滿了例外及特殊，造成日後維護及營運極大的困擾。

上面的例子當然不是常態，而是屬於較特殊的特例。而且通常發生在規模相對較小的公司，希望往中大型企業資訊系統發展的過程中，輕忽了行業別專業知識的難度及複雜程度，所常會犯的錯誤。

要解這個問題，事實上並不困難，只要找到一個有足夠實戰經驗的系統架構師即可。如果他能夠精通 SQL 語法，那更是絕佳人選。

說系統架構師是資訊系統的大腦，一點都不為過。因為整套企業資訊系統所有的資料表、欄位、流程、表單及相關數字的來龍去脈，架構師都必須了然於胸。對於所有的程式細節及設定，都必須清楚明白，也就是說，他就是一部關於企業資訊系統的活的百科全書。

說實話，這實在不是一件容易的事情，因為他必須同時精通軟體技術及商業邏輯知識。例如總帳會計、成本會計、生產管理等等。跨領域的人才，不管在哪個行業，都是鳳毛麟角。

前面筆者曾舉首次撰寫物料需求展算為負面範例，說明任何沒有深思熟慮、半調子的、不成熟、不嚴謹的規格，都不應該急著寫程式，輕率的結果只有四個字可以形容：後患無窮。

正確的做法應該如下

1 用戶端（可以是客戶和業務單位反應的市場需求），提出功能需求書（如物料需求展算）。
2 系統架構師確認是否要納入開發列表及預定開發時程，並確認規格框架。
3 責成系統分析師，依規格框架完成規格確認書，經架構師確認無誤後，交由資料庫管理師寫成 SQL 的預存程序(Store Procedure) 供呼叫。
4 程式設計師依規格文檔，呼叫使用並測試數字是否正確。

筆者認為，企業資訊系統的開發，應該是以資料庫為核心，這和目前的開發主流，是以開發工具為主，例如

1 Java
2 Virtual Studio 20xx
3 PHP
4 Ruby on rails
5 Physon

等等為核心的做法，是大不相同的。

筆者在本書中會不斷的大力提倡這個概念，諸位可以多加體會，自行思考其間的利弊得失。

關於系統架構師，由於其角色至關重要，關係到整個專案的成敗，以下整理列出應由其負責的相關工作內容

1 確定開發平台應該具備的機制
   A 主畫面的 UX 安排
   B 權限設定機制
   B 主選單及權限相關的系統檔案設計
   D 參數設定的相關作業
   E 限制查詢、已被使用不得刪除資料等等的共用機制

2 確認樣板表單的類型

a Single Grid：單檔多筆資料，適用於欄位少於 10 個以內的基本資料維護
b Single Entyy：單檔單筆資料維護
c Single MultiPage：單檔單筆資料維護，且欄位數大於 40 個以上的資料
d Master-Detail：單據表單
及表單基本操作功能和 UI 的配置、如配色、字體、字型大小等。

3 確認開發系統的功能範圍
4 對於企業資訊系統中複雜的進階功能，提供完整而詳細的解決方案。
5 規劃系統所有的檔案結構及資料流、過帳關聯等。
6 決定所要提供的報表列表，並負責極複雜報表的計算邏輯詳細背景知識。

當然，一般的細項工作，應該交由系統分析師、資料庫管理師等，實際負責執行，但是過程中所有的細節，都應由系統架構師掌握，執行專案的過程中，如有遇到任何突發或難以處理的狀況，都應由架構師決定解決方案，並由其負成敗之責。

經由以上的說明，對於系統架構師這個角色，各位看倌應該都有一定的了解。但是在現實生活中，一般規模不大的軟體開發廠商，並不容易找到這樣的人才，大多數的情況，創辦人之一通常會兼任這個腳色。這會造成 2 個可預見的問題
1 初期不會有問題，但長期擔任此職務，又要負責公司的營運，壓力比山大。
2 通常此創辦人會形成公司的 Domain 天花板，公司的專業知識會受此限制。

有心想打破這種局面，但是一方面人才難尋，另一方面，自行培養耗費的時間實在太長、緩不濟急，且公司常會變成職業訓練所，會遭遇對手挖腳或自行創業的問題。

缺少重量級系統架構師，最直接的影響，就是軟體產品的功能，一直只能停留在某一個等級位階，這就是著名的天花板理論。想要再向上提升，依現行的架構是不可能的，而就算是空降系統架構師，也會有企業文化及組織原有成員 (特別是資深人員)，能否融合的大問題。這在在處處都考驗的經營者的智慧。

所以筆者把系統架構師，當成成功關鍵因素之首。也就是想要成功開發好一套企業資訊系統，必須先有優秀的系統架構師，否則一切都只是鏡花水月一場空。

針對系統架構師難覓，筆者認為可以考慮以下的做法
1 拔擢公司內部資深的系統分析師或顧問師，當見習架構師。
2 禮聘外部顧問直接以師徒制的手把手訓練方式，訓練見習架構師。經過幾個專案的歷練，及自我的精進，應該有機會訓練出企業內部的系統架構師。

3 長期和外部顧問合作開發，藉由外部顧問的專長，來彌補企業內部的不足。

總而言之，優秀的系統架構師會帶你上天堂，反之則會帶你走迷宮、陷入泥淖。軟體公司的經營者，應時刻關心公司內部的重要成員，對於系統架構師應多所禮遇，並應隨時注意是否有優秀的新進人員，注重內部的培訓。畢竟千軍易得、一將難求。

當然，雞蛋千萬別只放在一個籃子中，平常就應多安排相關的教育訓練，讓企業內部成為學習型組織，不會因為任何單一的個人異動而傷筋動骨。

第 15 章
務實而有效的專案管理

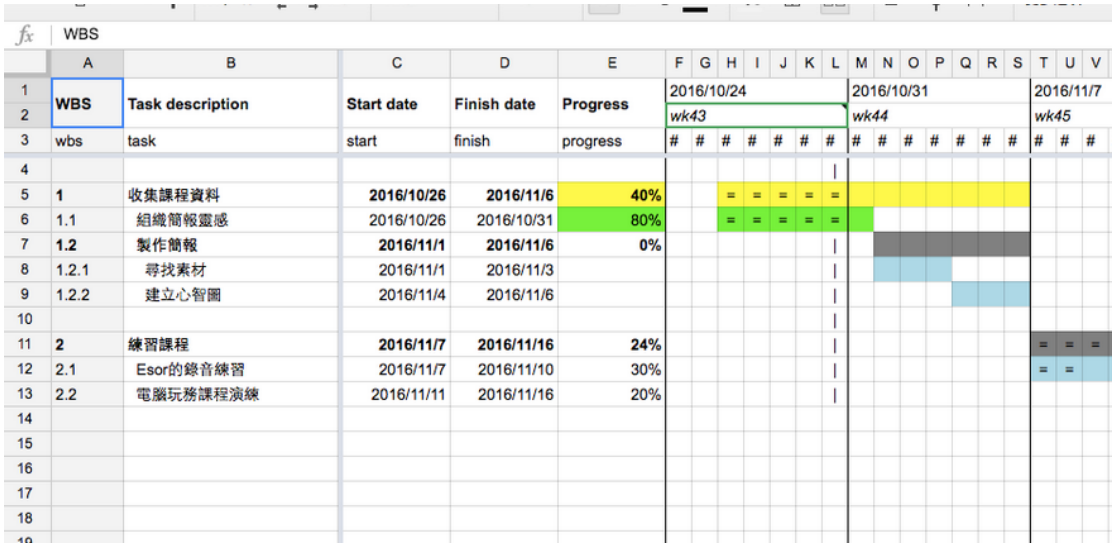
今操已擁百萬之眾，挾天子而令諸侯，此誠不可與爭鋒。孫權據有江東，已歷三世，國險而民附，賢能為之用，此可以為援而不可圖也。荊州北據漢、沔，利盡南海，東連吳會，西通巴、蜀，此用武之國，而其主不能守，此殆天所以資將軍，將軍豈有意乎?益州險塞，沃野千里，天府之土，高祖因之以成帝業。劉璋暗弱，張魯在北，民殷國富而不知存恤，智能之士思得明君。將軍既帝室之胄，信義著於四海，總攬英雄，思賢如渴，若跨有荊、益，保其岩阻，西和諸戎，南撫夷越，外結好孫權，內修政理;天下有變，則命一上將將荊州之軍以向宛、洛，將軍身率益州之眾出於秦川，百姓孰敢不簞食壺漿以迎將軍者乎?誠如是，則霸業可成，漢室可興矣。

諸葛亮、隆中對

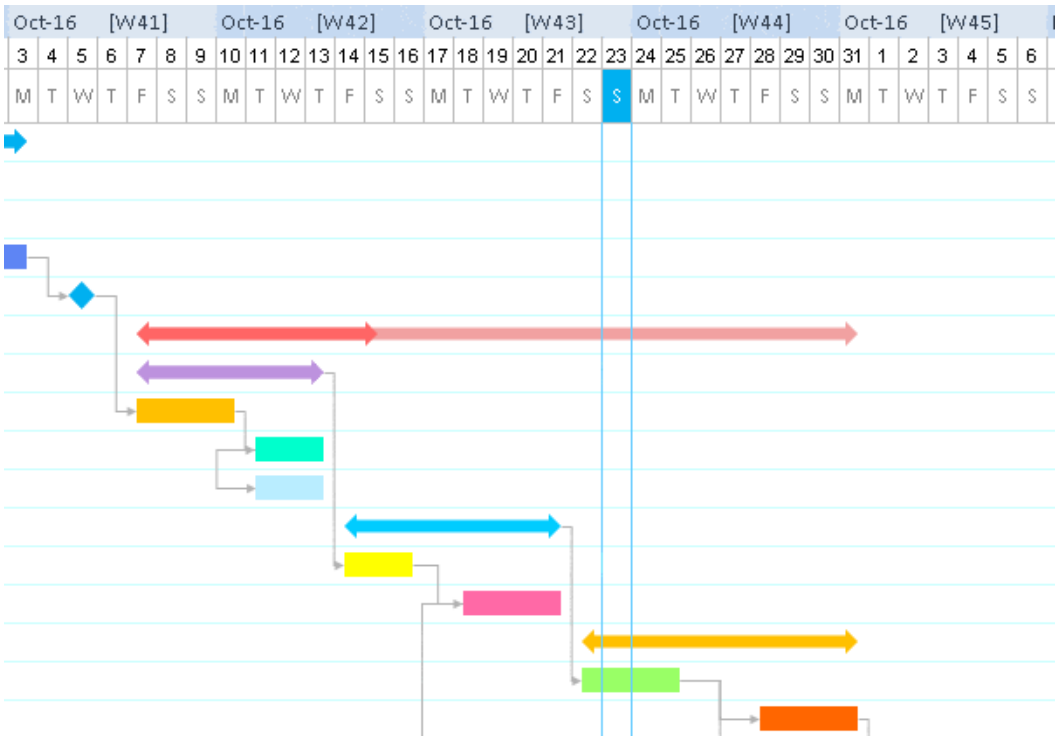
第 15 章 務實而有效的專案管理

要開發一套完整的企業資訊系統，其中包含了數百(千)支以上的程式，及上百萬行的程式碼。說實話，這是個很龐大的工程。如果沒有有計劃的管理，往往會顧此失彼，造成進度延遲及時程 (Schedule) 失去控制，完成的行程遙遙無期。要想如期的完成，就必須仰賴專案管理。

近幾年來，PMP 專業管理儼然成為一門顯學，考取 PMP 證照也成為一個重要的課題。然而，專案管理並不是一門新的學科，自二次大戰後期就廣泛被使用的甘特圖，就是專業管理中常被使用的工具(下圖範本)。



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1						2016/10/24							2016/10/31							2016/11/7		
2	**WBS**	**Task description**	**Start date**	**Finish date**	**Progress**	*wk43*							*wk44*							*wk45*		
3	wbs	task	start	finish	progress	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
4																						
5	**1**	收集課程資料	**2016/10/26**	**2016/11/6**	**40%**			=	=	=	=	=										
6	1.1	組織簡報靈感	2016/10/26	2016/10/31	80%			=	=	=	=	=										
7	**1.2**	**製作簡報**	**2016/11/1**	**2016/11/6**	**0%**																	
8	1.2.1	尋找素材	2016/11/1	2016/11/3																		
9	1.2.2	建立心智圖	2016/11/4	2016/11/6																		
10																						
11	**2**	**練習課程**	**2016/11/7**	**2016/11/16**	**24%**															=	=	=
12	2.1	Esor的錄音練習	2016/11/7	2016/11/10	30%															=	=	
13	2.2	電腦玩務課程演練	2016/11/11	2016/11/16	20%																	
14																						
15																						
16																						
17																						
18																						



專業管理之所以重要，並不是因為他有什麼神奇的魔法，好像只要導入 PMP 專案管裡，就萬事成矣，這是很大的謬論。它只是綜合及總結許多前人在實際執行專案的過程中，被證明所使用的工具及方法是有效的，而將這些驗證的方法匯總而成。所以他只是實用的科學，主要的理論並不深奧，注重在實際解決問題。事實上，大部分的管理科學都是如此。

早期的 ERP 專案在執行時，或是因為專案的規格較小，或是主事者不認識專案管理的重要性，所以很大部分的專案都是土法煉鋼。成敗很大的成份都是靠點運氣或是主事者個人的超強能力。直到近期很多軟體界的有識之士，因為先前的痛苦經驗，開始尋求外部的協助，以解決專案執行失敗機率過高的問題。而和日漸盛行的 PMP 專案管理一拍即合。但是和其它的管理科學一樣，因為試圖將各種工具及理論都納入，想要建立一個完整的架構及理論體系，其結果就是變得非常龐大及臃腫。

這是個兩難的局面。筆者個人的看法是，只要截取 PMP 中符合現行狀況的需求，拿來運用在專案開發即可，不必為了 PMP 而 PMP，硬要將教科書的東西全盤照抄。筆者在前章節中已有建議，主要使用

1 WBS

2 Issue Tracking System

這二大主題，就已足夠大部份的軟體專案管理使用。

專案管理的主要核心目的，是要能

1. 如期(行程)
2. 如質(品質)
3. 如預算(人力及其他資源)

的完成專案的交付內容。

這是標準的知易行難。專案執行時，最難克服的就是莫非定律。永遠在你料想不到的時候，狠狠的給妳一擊。

專案失敗的原因很多，常見的狀況如

. 客戶在驗收階段不斷的變更規格
. 太多本來就應該有的需求，但是規格書都沒有
. UAT 測試時，問題層出不窮
. 專案進行中主要人員異動
. 客戶對某些功能規格內容翻來覆去
. 主要的 mile-stone 功能進度未能趕上

表面上看起來都是因為逾期(因為沒有如期完工)，也就是無法準時交付合乎品質

驗證的產品。這當然是主要的原因，但是深究其根源，就會發現是專案管理中，所謂學生症候群在作祟。這才是造成延遲 Delay 的主要原因。
學生症候群，顧名思義是說：在學的學生，當老師要求交付作業的時候，無論老師原來要求的交期為何，學生都會說來不及，要求多給一點時間。而等到爭取到充分的時間後，最後也只會在交期的前幾天開始熬夜趕工，大部份的前置時間，都被浪費掉了。
聽起來很耳熟?
當然，因為每個工程師或多或少都有學生症候群。

研發人員總有忙不完的事情，研究不完的新技術，參加不完的技術研討會。
而最重要的兩點是

1. 喜歡鑽牛角尖，不知變通.
   也就是太執著，這通常表現在花費多時解決單一問題(bug 或 issue)，碰到看起來並無錯誤的程式碼，只想找到原寫法的錯誤，而不會轉換思考邏輯，改用新的寫法(當然，如果原寫法有誤，改正錯誤是必然的)，導致花費較預估多數倍的時間，這當然也會影響開發的進度。

   正確的做法，應該是適時的換腦袋，當怎麼找都找不到問題時，就應該找 PM 或 SA 討論，向其說明目前的程式邏輯及問題描述。經多次實驗證明，在說明問題的同時，真正的錯誤原因就會自動浮現，並找到解法。
   如果不行，SA 就應該思考別的解法邏輯。筆者一貫主張以資料庫為開發核心的開發方法論。如果碰到較複雜的運算邏輯或是計算，就該由 DBA 或 SA 寫好 SQL 的 Store Procedure (預存程序)，讓 RD 直接呼叫使用。一則分工明確，再者，有誰會比 SA 更清楚規格？RD 本應專注在 UI 和簡單的邏輯判斷上。這才是最有效的人力分工。

   RD 的主要職責，是在開發平台的標準架構下，使用標準的函式庫，快速的完成程式。不要因為單一 ISSUE 而延後進度。

2. 對於規格的難度過於樂觀
   也就是輕忽了程式本身的複雜程度。

   這通常是較資淺的 PM 或工程師常犯的錯誤。在專案開始啟動時，PM 就會朝開 kick-off 會議，並列出專業時程及 WBS(Work Break Structure)，在 WBS 中會列出所有的工作項目及預訂時程 (通常是以天為單位)。這中間有一個很大的難點，也是所有 PM 心中的痛，就是如何訂立每一個工作項目的工時？這和負責該 WBS 工程師的專業程度有關、和該程式的難易度有關，更和運氣有關。

專案管理屆，常流傳的一個不好笑的笑話：專案訂出日程的目的，就是用來Delay(延遲)的。

這是個很無奈的事實，所有的軟體從業人員都很想模仿硬體產業的做法。將所需的功能都封裝成一顆顆的 IC 元件，然後用主機板將所需元件組裝為成品，所以軟體界也開始將常用功能封裝成元件。

有些 third-party 的元件廠商也取得了一定的成果，但是絕大部分的元件都是在處理 UI(USER Interfice 使用者介面)，這當然解決了一些問題，提高了開發效率，但是仍然不足夠。

20/80 法則也適用於 ERP 軟體開發。
大部分的功能都是一般的維護表單及報表，這些程式的時程一般都不會有太大的差異，通常 PM 會安排資深 RD 試寫幾個功能，計算大致的開發時間，並以此為基礎，設定專案的工時。至於 20%較複雜的功能，則只能依 SA 或 PM 的經驗去猜測。

再來就是適度的安排 buffer(緩衝)時間，但是緩衝時間的加入是一門大學問，前面曾經提過的『學生症候群』就是專案 Delay 的重大因素。但是不可能完全沒有 buffer，解決之道是

A 定時加入緩衝(buffer)，例如 5 個工作天只安排四天半的工作量。
B 只在重要的里程碑 (Mile stone) 才加入
但是絕對不能在每個 WBS 都加 buffer。

所謂定時是每個成員的周行程中，都安排半天到一天的 buffer，這是較簡易的做法。至於在重要的 mile-store 加入，是在重要的時程檢查點上，加入幾天的 buffer，讓 RD 在進度落後時，可以有時間趕上。

專案經理和系統架構師是整個開發 ERP 專案的重要角色。前章節已說明系統架構師是大腦，那專案經理就像是人體的心臟、是主要的發動引擎.。

專案經理『對外』要和客戶(或是公司內部的 Power User)溝通規格及進度。定期開會回報目前的進度(次/週)，並和客戶的對口單位(通常是 MIS+power User) 搞好關係，建立交情。事實上，如果專案搞砸，對方窗口的相關人員，也不會太好過，所以雙方是共存共榮，而非對立。如果專案進度順利，那還沒有大礙，但是天有不測風雲，一旦專案 Delay 進度，內部當然是加班趕工 (適度的加班是必要的，但過度則有百害而無一利)，對客戶則必須清楚說明現況及補救措施，如有

必要則可要求延後行程。這就有賴於專案經理平時對客戶關係的經營是否成功，彼此的信賴度是否良好。

總之，專案經理是 ERP 專案的驅動引擎及舵手，時刻緊盯二大指標(WBS 及 Issue Tracking) 及搞好客戶關係是專案經理無可規避的責任。

專案經理『對內』則要定期 (通常是每天)追蹤管理專案每一個成員每天的進度狀況，並要定期的抽查程式碼的內容(品質及是否依 SOP 撰寫)。如果有特別的案例，更要不定期的召開內部檢討會議，並更新 WBS。WBS 的進度表並非神聖不可侵犯，她是需要時常更新的，專案最怕逾期，所以專案經理的重責大任是在於緊盯每一個成員的 WBS 是否有 Delay 的狀況並即時提供協助。

除了 WBS，在專案起始後，測試的工作就會同步展開，如果測試有問題，就應列入專案管理兩大指標中的 Issue Tracking System (Buy 及建議追蹤管理)系統。專案經理可以經由相關的報表查詢 WBS 及 Issue Tracking 的目前狀況，這最好要有系統可以協助。市面上有一些現成的專案管理系統(包括 Open Source)可以自行選用。

筆者多年前，在某 NB 代工大廠駐點時，他們只使用 Excel 也可以達到相同的管理目的。不過並不是每個專案經理都熟悉 Excel 的巨集指令。筆者個人是參考該大廠的做法，無後自行開發一套簡易的專案管理程式，功能雖陽春，但自用足矣。

接下來，我們來談談開發人員的管理：
軟體開發人員的管理一直都是個難題，因為當你看到一個 RD 在發呆，並不表示他在混，而是正在解一個問題時，想到出神。相反的，你看到一個 RD 很認真的在寫程式，也有可能是在寫非專案進度的程式，或是已經鑽入牛角尖誤區而不自之。這也許是 Google 為何會允許員工能自主運用 20%工作時間的原因。

管理問題是一個大課題，不在本書討論的範圍。在此處特別提出來，是因為專案經理如果想要帶好團隊，就必須有棍棒和蘿蔔(獎懲的權力)。一般而言，常用的方法不外乎

獎勵：
專案結案獎金，公司由專案金額撥出 10-15%，交由專案經理依成員對專案的貢獻，決定每個人的獎金。如果是企業內部開發自用的系統，也可以考慮撥一筆固定金額當作獎金，方式同前，另外也必須和完成的時間掛勾。
例如準時或提前完成，金額全額發放，但每 Delay 一個月，扣發 10%等等方式。

罰則：

有賞必有罰，相對於獎勵，也必須訂出對工作較不利的員工訂出罰則。一般而言，專案經理的工作負擔已經很重，不應再加太多的文管作業。所以最好的方法還是寫幾支報表程式去撈資料內容。這個部份筆者建議可以設立幾個 KPI 指標，例如: WBS 及 issue 的準時完工達成率

WBS 及 issue 的 Delay 次數統計

Issue 被 ReOpen 的次數統計等報表

然後依上面報表列出的 KPI 報表，訂出適當的罰則。

如果沒有權責，專案經理無從管理研發團隊，則以上所談都是空談。當然也要有機制能隨時觀察專案經理是否有失職，或不稱職的狀況，這應該是研發副總的責任。這個主題咱就此打住，不在往下談了，否則就沒完沒了。

第十六章
有效運作的組織

行路難，難於山，險于水。
不獨人間夫與妻，近代君臣亦如此。
君不見左納言，右納史，朝承恩，暮賜死。
行路難，不在水，不在山，
只在人情反覆間。

李白、行路難

第十六章 有效運作的組織

現代的企業較之早期職能複雜了許多，也更加的彈性。不過基本的產、銷、人、發、財還是主要的部門。其中的『發』指的是研發部門，也就是本書所討論的焦點。在本章中，我們先將問題單純化，先討論研發部門內部的協作，至於跨部門的協調暫且不談。

在第 12 章中，我們用人體的機能來譬喻研發部門的七個角色:

系統架構師	大腦	
專案經理	心臟	PM
資料庫管理師	經絡穴道	DBA
系統分析師	神經系統	SA
系統設計師	骨架	SD
程式設計師	四肢	Programmer
測試人員	免疫系統	Tester

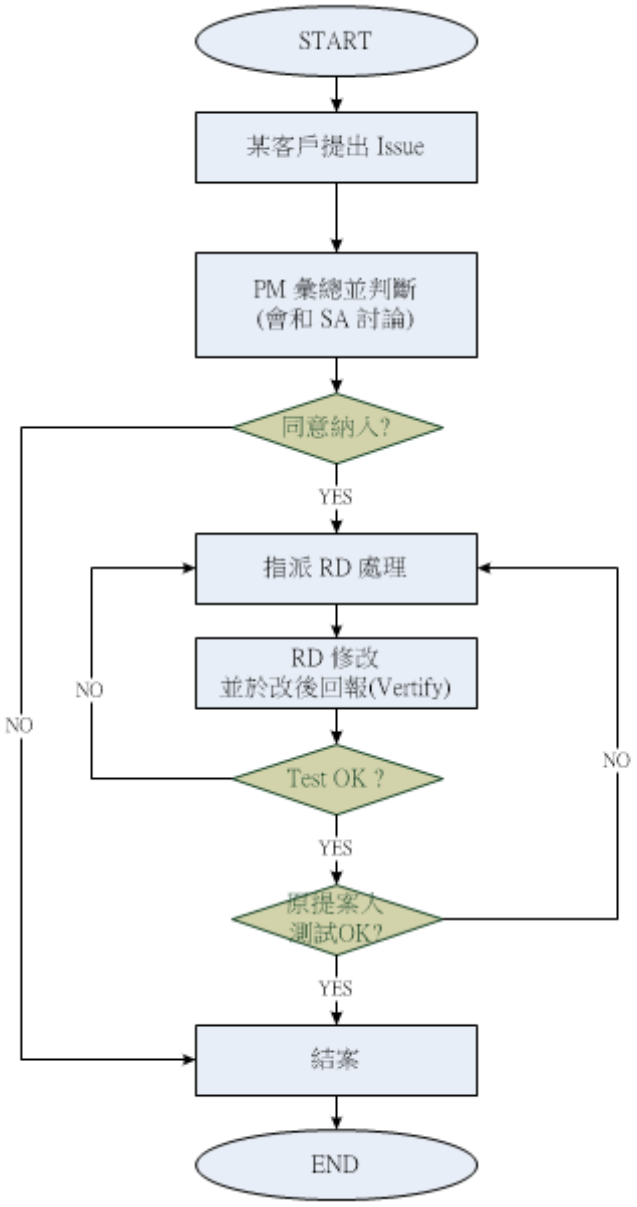
就是在強調，整個研發部門是一個分工合作的整體。不特別強調個人英雄主義。

一個健康的人體，是各司其職，且運作順暢的小宇宙，一旦某個部位出了問題，身體就會出現警訊。就是生病了。生病了怎麼辦？當然是找醫生治療，您可以看中醫，也可以去看西醫，甚至求助民俗療法。只要能治好病，就是有效的治療方式。

同樣的道理，研發內部的協同工作也是一樣的道理，當內部 SOP 已經被建立起來，所有人員就該依循標準作業程序來執行。
例如:
. 程式中變數的命名規則
. 註解的規範
. Issue 的提出，Test Case 及結案
. 文件的改版程序等

現在的管理主流是協同工作，而非早期的集權式管理。當然，協同工作並非放任，專案經理的角色就類似集權式管理的主管，只不過專案經理更注重在溝通、協調及資源調度，監督的角色次之。

協同工作的重點是每個成員，除了做好自己的工作外，並要關注其他相關人員的進度，並接續完成相關的程式。我們舉一個 issue 由客戶反映提出後，一直到完全結案為止的程序來說明協同工作的概念



一個 Issue 由提出到結案，會和下列人員有關

. 客戶
. 專案經理 PM
. 系統分析師 SA
. 測試人員 Tester
. 工程師 RD(Programer)

等都有相關，由於 ERP 的系統龐大，每天都會產生新的 Issue，而且工作是有次第的(沒有經過測試，就不能結案)。當 RD 修改好程式後，測試人員(Tester)就會測試程式是否已改好，接下來原提 Issue 人，就要確認無誤後結案。

以上的流程有點類似 Work Flow(工作流)的概念，早期受限於沒有資訊系統的支援(或是功能較陽春)，資訊不透明，很難得知其他人員的進度。所有人完成工作後，只能回報給主管，所有的資訊只掌握在主管的手中，所以完全要依賴主管的交辦。要將公司轉型為協同工作，首要之務就是資訊要公開透明(當然是對有相關權限的人員透明，並非全面開放)。所以必須要有協同工作相關軟體的支援(大部份的 EIP 系統都有此功能)。

一旦資料庫有相關的資訊，就可以透過 e-mail 內部的訊息通知(IM)或工作儀表板(Dash　Board)等程式，自動通知下一個待辦人員，而不須等待忙得不得了的主管指派，如此效率自然就會加快。

專案管理和協同工作軟體是 ERP 系統開發時所必備的二大管理工具，其目的都是讓大部份的管理自動化。每個專案成員只要進入系統或是透過 e-mail 就可以知道每天的待辦事項。完成工作後，系統就會自動將工作指派給下一個相關人員，如此工作自然就會順暢的進行。

當然，人生不如意事十之八九，事情絕不可能如此順利無礙。莫非定律早就告訴我們，意外是常態，總是會有諸如專案相關人員請假、離職、心情不佳或是共用程式有 bug，元件無法達到客戶的要求.....等等五花八門的問題。這也是為何要特別強調有效運作組織的重要性，一般而言，專案 kick-off 時，專案的所有成員都應該就位，並了解專案的背景及時程。每一個成員都應該了解自己所要負責的工作及進度。

專案經理是掌舵者，負責讓專案日日依進度進行，所以他必須掌握專案所有成員的近況及專長。當進度有 Delay 的狀況發生，短期加班的效果會比增加新的人手有效。但是如果逾期的情況已經惡化，專案經理就必須深入了解情況，換人或加人都必須納入考量。不過，如果逾期的情況太嚴重，而又有明顯的訊號發出(如客戶透過管道多次表達不滿)，那很明顯是專案經理失職(當然也有可能是資源完全不足所致)。雖然換 PM 的代價非常高昂，但是放任不管會更糟。此時唯有壯士斷腕，不過專業若是已經到了這種狀況，大部份都會以失敗告終。除非新上任的專案經理超級厲害，屬於超人類。否則，只能朝最小損失來想辦法結案。

軟體專案的失敗率並不低，導入專案管理主要的目的本來就是想要解決這個問題。但是專案管理本身並不會產出任何程式碼，他只負責監控進度，並提出警訊。所以透過協同工作軟體，讓專案成員能順暢無礙的協同工作，發揮最大的效能，才能確保開發進度能和喻期相同，如此自然就能順利的結案。

而一旦進度出現警訊，PM 就應該盡快了解狀況
. 是原先的估算過於樂觀？
. 或是成員的程度有問題？

若是前者，就有必要重新檢視 WBS 的設定，並重新安排
　　後者，就該再起啟專案的初期更換不適任的人員。否則會在後期造成很大的負面效應。

　事實上，問題如果能越早發現，就越能保證專案能準時結案，專案只要不逾期太多，問題就不會太大。效能及穩定度相關的 Issue，在有經驗的人員手中，都有很大的機會能夠排除，加上 DBA 負責將複雜的邏輯寫成 Store Procedure 供 RD 呼叫。如果整個專案能夠在這種有效率且又分工合作的研發部門執行，當然結案就不是難事。

第五部份
建構開發標準

第十七章
開發四化

行路難，行路難，多歧路，今安在？
乘風破浪會有時，
直掛雲帆濟滄海。

李白《行路難》

第十七章　開發四化

前面的章節主要談的都是觀念，主要的目的是讓您對開發整套的 ERP 系統，建立一個完整的概念。就如同 explorer(探險者)要先掌握完整的地圖，才有機會找到寶藏一樣。

接下來，我們就要開始進入實戰階段。將前面討論的內容，化為實際的作法。筆者預計分四個主題來談

1. 標準化---樣板程式
2. 元件化---底層共用程式
3. 自動化---Wizard
4. 系統化---用程式自動化管理，不要有過多的人工管理。

統稱之為開發四化。

1. 標準化

   所謂的標準化，主要是從下面幾個面向來討論

   a. 樣板程式

      每支程式的第一個版本，都是透過 Wizard 依據樣板程式產生，然後再由程式設計師(programer)，增修完成交付的版本。

      這樣做的優點是

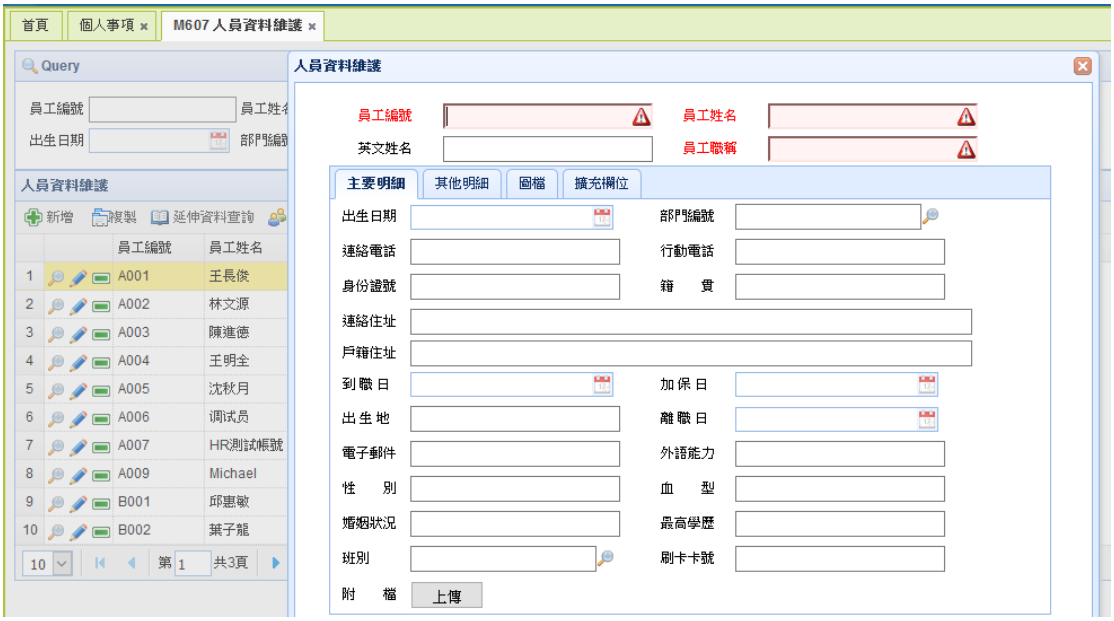
      (1) 確保每支程式的程式架構一致，任何受過訓練的程式設計師(programer)都能互相維護彼此寫的程式。

      (2) 程式是由專案經理(PM)或系統分析師(SA)，使用 Wizard 產生。並可依該版本程式和 User 確認規格，可避免溝通不良導致的規格不明確。

      (3) 輸入規格到 Wizard 程式後即可產生第一版程式，大幅提高開發生產力，而且程式也避免人為撰寫程式的疏失。

SingleGrid (單檔多筆資料維護)

SingleEntry (單檔單筆多頁簽)



Master-Detail (單據類表單)

b. 呼叫共用程式或 SQL 的預存程序(Store Procedure)

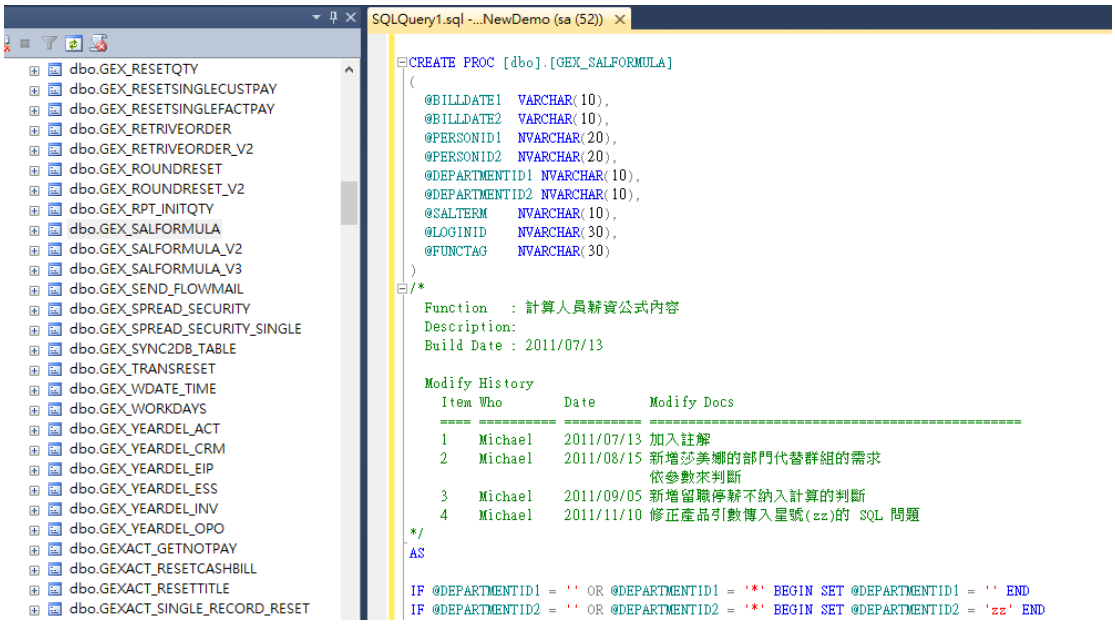
用 Wizard 產生的程式碼，只能產生大部份的通用標準程式，還是會有部分程式碼需要手工撰寫。而程式會出問題，大都是出在這個部份，所以程式設計師(programer)在手動撰寫程式碼時，務必依循下列標準的作法。

(1) 如果有共用函式(function)，請呼叫標準程式庫中的 FUNCTION。如果沒有，請和系統設計師(SD) 討論是否要寫新的 function 加入共用函式庫中，除非是很特殊的需求，一般都請寫成 funcion。

```
167  //重寫jquery.infolight.js中的infoFileUploadMethod()方法
168 function uInfoFileUploadMethod(infofileUpload, onAfterUploadSuccess, onAfterUploadError)...;
267
268  /** @description: 檢查瀏覽器及版本 */
269 function uBrowserCheck()...
305
306  /** @description: 設置Theme，于uInitTable中使用*/
307 function uSetTheme()...
317
318  /** @description: 執行SQL 語句
319   * @param {string} sql : 需要執行的SQL語句
320   * @param {string} returnData : 是否要傳回數據, Y表示傳回數據(可以在uAfterExecSQL方法中對數據操作)
321   * @return {string}
322   * @Sample : uExecSQL("Update YearFinal Set AMT = 1000","N");
323   */
324 function uExecSQL(sql, returnData)...
346
347  /** @description:初始化Table中的tr、td
348   * @param {string} markName : 基準元件
349   * @param {number} tWidth : table寬度
350   * @param {string} tColumns :參數分別：元件名稱，寬度，元件類型，colspan，同一個TD，align，rowspan
351   * @return {string}
352   * @Sample : var tColumns = [["btMultiQuery,100,button,2,Y"]];
353                     uInitTable("queryPanel", 1100, tColumns);
354   */
355 function uInitTable(markName, tWidth, tColumns)...
420
421  /** @description:給數字格式化
```

(2) 如果是複雜的運算邏輯，且和資料庫的內容有關，可以和系統分析師(SA) 討論後，請資料庫管理員(DBA)，寫成預存程式(Store Procedure)，

供程式設計師呼叫。.



(3) 如果都不是，則可以由程式設計師自行撰寫程式，但仍應將程式碼封裝為 function，且需依循開發 SOP 的相關規範。

c. 程式碼的風格一致化

此部分主要當然是針對手工加入的程式碼所做的規範。主要的目的是讓程式碼易於閱讀及維護。

(1) 程式碼一定要加入註解，且註解必須有固定的格式(依 SOP)

```
318  /** @description: 執行SQL 語句
319   * @param {string} sql : 需要執行的SQL語句
320   * @param {string} returnData : 是否要傳回數據, Y表示傳回數據(可以在uAfterE:
321   * @return {string}
322   * @Sample : uExecSQL("Update YearFinal Set AMT = 1000","N");
323   */
324 function uExecSQL(sql, returnData)...
```

(2) 程式新增的變數必須遵循命名規則

(略)

標準化的目的，是讓程式碼易於維護。

如果沒有相關的規範，不同的程式設計師，根據自己的喜好，寫出風格完全不同的程式碼，這對後續維護的程式設計師而言是個大災難。稍有經驗的開發人員都知道，程式完成後的維護成本，事實上是遠高於開發成本的。因為開發成本是一次性費用，但維運卻是長期的成本。

一套 ERP 系統，使用的期限通常都很長(至少 5-10 年)。除非有重大的變動，否則不會輕易更換系統。而且隨著企業內外部環境的變動，ERP 系統也必須配合而作出必要的調整及修改，所以易於維護(擴充)，絕對是讓 ERP 系統能長期使用的重要因素。

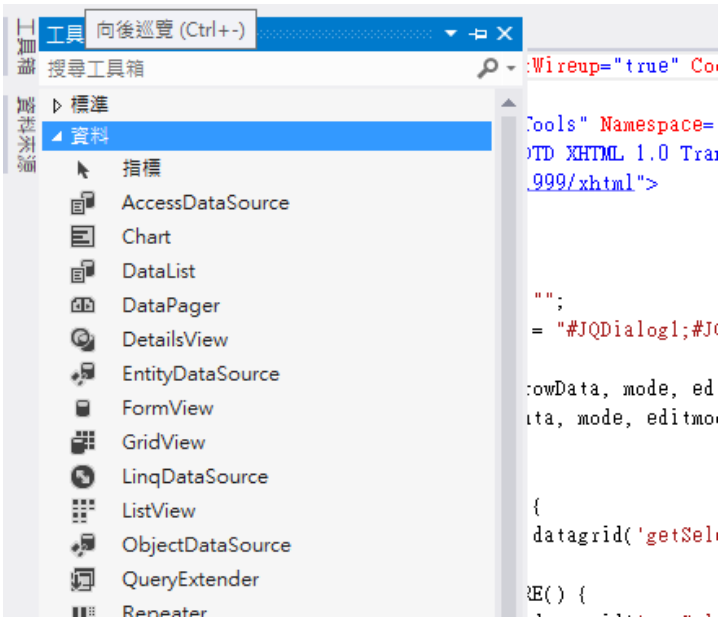
2. 元件化

此處筆者所謂的元件化，事實上泛指兩個概念

(1) Cpmponent (元件)

(2) Library (共用函式庫)

a. Comeponent：可以利用開發工具，如 Delphi 或 Visuai Studio2012 或 Esclipe 等…直接拖拉元件盤上的元件到程式中，透過設定相關的 property(屬性)及 Event(事件)後，即可運行。常見的 Label、TextBox、DataGrid、ComboBox 等一系列的 UI 元件都屬於此類。



另外大型一點的元件，如: iCale 行事曆元件、Reporting View 報表元件則是更高階，功能更豐富的元件。使用這些元件可大幅強化系統的功能，而無須投入太多的人力資源，所以善選好的元件也是非常重要的。

最新的技術通常就表示還有待時間的考驗，選用 third-party 元件也是同理。選用元件的第一守則是一定要選擇已被驗證可行的元件及技術(如已有完整的文件庫及廣大的使用者，或已有大廠的支援)，才可以考慮整合到開發平台中。開發平台的整合，本來就是件勞心費力且帶點運氣的工程。在專案起始的初期，大部份的心力都會花在平台元件的整合。除非已經確定開發平台都已整合且測試無誤，否則就不應該邁入功能開發階段。

b. Library 共用函式

(1) 框架底層的共用函式

通常是配合 Wizard 產生程式時所需呼叫的共用函式庫。

例如，我們常會將連接資料庫的連線程式(DB Connection)，寫成共用程式。一般而言，程式不應直接呼叫，大都是一般較高階的共用函式呼叫，或是 Login 時就已建立好，這類程式皆屬之。

(2) 一般共用程式

大部分的共用程式(function)皆屬之。

舉凡限制查詢條件用的共用程式，或是記錄使用者功能的 log、印表時擴充取 xml 定義的 function 等。

這類共用函式大都隨著開發的進行而逐步加入、擴充，是共用程式庫的主要構成元素。

通常，元件的來源有三

1. 開發工具內建

   這是最大宗的來源，如 Visual Studio 2010 就提供一系列完整的元件供使用。

2. third-party 廠商提供

   購買 third-party 廠商提供的商業化版本。

   此類元件通常功能非常豐富。另外，現在開源程式(open source)，也有非常多好用的元件，也可以考慮採用，但必須注意其授權方式，以避免商業使用時的糾紛。無論是哪一種來源的元件，都必須謹慎的選用及測試，才能納入標準的開發平台。

3. 自行開發

   如果找不到合適的元件，例如首頁顯示時所需要的數位儀表板元件，就只能自行開發。原則上如果能找的到現行的元件，就盡量選用，避免投入太多的人力到元件開發上。

元件化的好處非常多

1. 首先，程式碼大幅的精簡。
2. 再來，程式的可靠性大幅強化。

程式設計師只要查看相關的文檔，就能使用功能強大的元件，等於大幅提高功力一甲子，並從中學習元件的架構的規劃。當累積足夠的使用經驗後，即可嘗試自行開發元件。

軟體工廠一向是所有開發人員的夢想，而元件化則是實現軟體工廠的最重要基石。

什麼是軟體工廠?

軟件工廠的真正意義是希望軟體的開發能像資通訊硬體產業的設計組合方式一樣。如果您打開您的 PC、NB 或 Smart Phone 的外殼就會發現其內部最主要的構成機件是(主機板+ICs+其他被動元件)

而影響硬體效能最大的因素莫過於使用的 CPU。舉例而言，早期電腦的 CPU 多使用 Intel 的 286→ 386→ 486→Pentum 等

手機晶片則有高通、聯發 的多款 ARM 架構。中國近幾年來以低價高品質而創造出所謂的白牌手機(或山寨板)，之所以能掀起大風潮，主要是拜聯發科的(一條龍公板)策略所賜。

原本手機開發技術只掌握在少數國際級大廠的手中，有極高的進入門檻，一般廠商根本不得其門而入，只能望門興嘆。但是聯發科的破壞性創新策略，憑藉其優異的手機晶片及公板，讓山寨手機廠商能快速的設計並推出功能強大的平價手機，並快速佔領市場。

這就是元件化的優點，如果沒有聯發科，中國所謂的山寨市場就難以形成。

如果軟體設計能像硬體一樣，設計好主機板，然後有取之不盡的 IC 可以組合使用，然後就可以設計出一套新系統，光用想的就程式設計師們無限神往，從此不用日以繼夜的加班加到爆肝，再也不會被客戶追得滿街跑。

不過，這畢竟是 I have a dream，雖然已經有很多的國內外工具廠商投入大量的資源研發，但是目前的元件化技術，還是只能有限度的達到 ISV(獨立軟體開發商)的需求，還是有很大程度的程式碼需要手工開發。

早期 MS-DDS 時代，國內工具大廠訊光科技曾經推出 DBTOOLS 這套超強工具。這套系統當初造福了不知多少 ISV 廠商及企業，它就很接近筆者理想中的 Wizerd。

到了 Windows 時代，另一家工具大廠聯銓資訊，以 Delphi 5 為基礎開發的 VDP，則是另一個令人驚豔的產品。這些國內廠商開發的工具，大幅度的提高了軟體開發的品質，並降低軟體開發的門檻，可以說對國內的軟體產業做出了很大的貢獻，筆者這本書中的部分觀念，也借鑒了這兩套產品的優點，可見一

個好的產品影響有多深遠。

總而言之，元件化是整個開發平台的核心，就像想要蓋 101 大樓，就必須菸打好地基。想要寫出一套優質的企業資訊系統，就必須先將元件化整合好，這是整個開發團隊必須先建立好的共識。

3 自動化(Auto Matic)

就像硬體廠商會使用機械手臂或其他類似的機器設備，來大幅提升生產的效能。軟體廠商也希望能有類似的工具，能大幅縮短開發的時程。

前一節所談的元件化，就是自動化的重要基石，而本單元要談的主題，則是精靈 Wizard。

Wizard 是較近期比較常用的名詞，早期大部分都稱之為程式產生器。顧名思義，就是能【依輸入的規格，自動產生程式碼】的工具程式

程式產生器並不是什麼新的概念，早在 MS-DOS 時代，就已經有好幾套非常好的程式產生器，如 Stage、訊光科技的 Dbtools 等，到了 Windows 時代，則有聯銓的 VDP 及迅光的 EEP 等。還有國外著名的 Code Smith。

近期新的開發工具大廠，也開始在開發工具內建 Wizard 的部分功能。
例如微軟的 VS 20XX 就提供了很多的樣板程式，程式設計師只要選取所屬的型板(Template)，就會產生完整的程式框架，這也減輕了程式設計師的工作負擔。
從這個觀點來看 精靈的確會成為未來程式開發的主流技術。

筆者會在本書的第六部分，逐步且詳細的說明 Wizard 的使用流程，接下來就簡單得來談談精靈的設計概念。

設計精靈程式並不是一件容易的事，相反的它是一塊非常難啃的骨頭。懂得設計精靈的人不多，且大部分會視為核心機密而不外傳。筆者不才，不敢敝帚自珍，願意將心得分享，如有謬誤，也懇請諸位不吝指教。

因為開發 Wizard 是有一定的難度，所以並不是每家企業都必須開發自家的版本，也可以考慮導入現成的套裝產品。不過如果真有決心也有足夠的資源，想要開發完全符合自家需求的精靈程式，可先參考本書的相關說明，如需協助，筆者也可以提供顧問諮詢的相關服務。

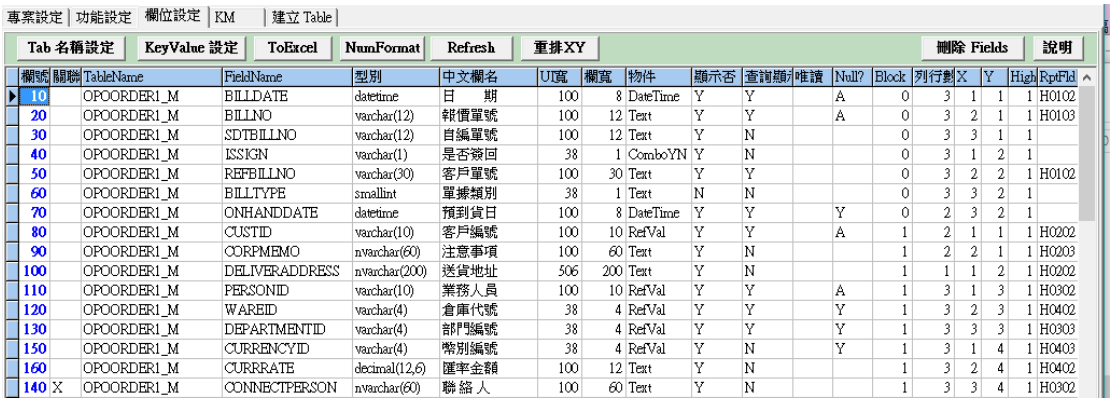
在第七章中，筆者已有重點說明大致的流程及觀念，請先參考相關的內容。

想要開發精靈程式，大致要依循下列的步驟：

1. 架構師先構思好系統的框架，並確認表單及報表的樣板(Template) 及使用者介面。
2. 評估並確定使用的開發工具，
   如 VS2012、PHP、java、ROR 等等
3. 架構師應責成專案經理，組織一個主要由資深且程式設計實作能力較強的系統設計師及資料庫管理師，共同組成【拓荒小組】，專責依架構師思的概念，實作手工打造開發平台框架、底層共用程式、表單及報表預存程式的樣板程式，及研究整合相關的第三方元件。
   這是專案起始的第一個重要工作，而且這個階段的成效，也會成為整個專案能否成功的關鍵指標。
4. 在前一點，筆者提到表單及報表的樣板程式製作，其概念說明如下
   a 首先，要求拓荒小組全手工開發出完全符合架構思規劃的樣板程式，這會有多隻不同 Style 的功能完整程式，樣式及所需功能完全由架構時規範。
   b 經嚴格測試無誤後，負責開發精靈的主要負責人(通常是系統設計師中的王牌程式設計師)，開始分析原始程式碼的內容。
   c 將每一支原始程式碼中固定不變的內容，仍維持原內容，而會依不同程式內容改變的部分，如表單畫面上的欄位，則整段程式改為 @xxxx@ 的變數代表符號。依此步驟重複，將整支程式都改為 【多個變數+程式碼】的格式內容，將其另存為樣板檔，稱之為 【樣板程式】。

```
16          }
17          document.onkeydown = keyevent;
18
19          var fullWindowDialogs = "";
20          var centerWindowDialogs = "#JQDialog1;@FSD97@";
21
22          function openForm(fid, rowData, mode, editmode, keys) {
23              uOpenForm(fid, rowData, mode, editmode, keys);
24          }
25
26          $(document).ready(function () {
27              @FSD79@
28          });
29
30          @FSD31@
31          @FSD32@
32          @FSD35@
33          @FSD33@
34
35          function openReport() {
36              uOpenRDLC("dgView", "@TableName@.@PrimarkKey2@", "cbSingleChoice", "Te
37          }
```

   d 重複上面的作業，將每一支程式都改為樣板程式。
   e 精靈(Wizard) 的概念很單純
      (1) 依定義檔的內容，讀取規格定義檔中的介面定義以及樣版程式檔的內容，如下圖


專案設定 | 功能設定 欄位設定 | KM | 建立 Table |

Tab 名稱設定 | KeyValue 設定 | ToExcel | NumFormat | Refresh | 重排XY | 刪除 Fields | 說明

欄號	關聯	TableName	FieldName	型別	中文欄名	UI寬	欄寬	物件	顯示否	查詢類	唯讀	Null?	Block	列行數	X	Y	High	RptFld
10		OPOORDER1_M	BILLDATE	datetime	日　　期	100	8	DateTime	Y	Y		A	0	3	1	1	1	H0102
20		OPOORDER1_M	BILLNO	varchar(12)	報價單號	100	12	Text	Y	Y		A	0	3	2	1	1	H0103
30		OPOORDER1_M	SDTBILLNO	varchar(12)	自編單號	100	12	Text	Y	N			0	3	3	1	1	
40		OPOORDER1_M	ISSIGN	varchar(1)	是否簽回	38	1	ComboYN	Y	N			0	3	1	2	1	
50		OPOORDER1_M	REFBILLNO	varchar(30)	客戶單號	100	30	Text	Y	Y			0	3	2	2	1	H0102
60		OPOORDER1_M	BILLTYPE	smallint	單據類別	38	1	Text	N	N			0	3	3	2	1	
70		OPOORDER1_M	ONHANDDATE	datetime	預到貨日	100	8	DateTime	Y	Y		Y	0	2	3	2	1	
80		OPOORDER1_M	CUSTID	varchar(10)	客戶編號	100	10	RefVal	Y	Y		A	1	2	1	1	1	H0202
90		OPOORDER1_M	CORPMEMO	nvarchar(60)	注意事項	100	60	Text	Y	N			1	2	2	1	1	H0203
100		OPOORDER1_M	DELIVERADDRESS	nvarchar(200)	送貨地址	506	200	Text	Y	N			1	1	1	2	1	H0202
110		OPOORDER1_M	PERSONID	varchar(10)	業務人員	100	10	RefVal	Y	Y		A	1	3	1	3	1	H0302
120		OPOORDER1_M	WAREID	varchar(4)	倉庫代號	38	4	RefVal	Y	Y		Y	1	3	2	3	1	H0402
130		OPOORDER1_M	DEPARTMENTID	varchar(4)	部門編號	38	4	RefVal	Y	Y		Y	1	3	3	3	1	H0303
150		OPOORDER1_M	CURRENCYID	varchar(4)	幣別編號	38	4	RefVal	Y	N		Y	1	3	1	4	1	H0403
160		OPOORDER1_M	CURRRATE	decimal(12,6)	匯率金額	100	12	Text	Y	N			1	3	2	4	1	H0402
140	X	OPOORDER1_M	CONNECTPERSON	nvarchar(60)	聯絡人	100	60	Text	Y	N			1	3	3	4	1	H0302

(2) 將樣板程式檔的內容，轉換為程式碼，其重點是將前一個步驟中的變數，要個別寫一段程式替換其內容。

(3) 當所有的變數內容都轉換為程式碼後，程式就正式產生完成。

表單及報表程式都是重複以上的步驟即可。

原理及實作看起來並不難，但只有真正寫過精靈程式的設計師，才能深深的體會 這塊骨頭有多難啃。但是一旦完成精靈程式，整個專案的品質及生產力即可大幅提升，所以如果專案規格夠大，筆者會建議應自行開發精靈程式，反之，則尋找類似的工具使用即可。

4 系統化

前面的三個化，都和開發程式有直接的關係及產出，而本節要談的系統化，則是和專案進度管理有關。

筆者在

第十章 專案進度的控管

第十五章 不實而有效的專案管理

第十六章 有效運作的組織

這三章中都有談過相關的主題內容。

筆者相信 Simple is King 簡單為王、大道至簡，如果要先搞一大本厚厚的手冊，才能做好所謂的管理，那就注定失敗的命運。

簡單的說，所謂的系統化，指的是透過整合使用 【專案管理程式】+ 【協同工作軟體】，建構一個專案進度控管平台，只要是專案成員，就可以透過此平台，隨時得知

a. 專案的進度
b. 錯誤及建議的處理狀況

c. 以及每個人員的待辦事項等等

也就是資訊透明，並會自動派送工作給相關人員，大幅降低專案主管的工作負擔，並讓工作流順暢，效率自然提升，專案經理只要專注在例外狀況的管理及資源的調配，讓專案一直在正確的軌道上順暢的運行。

所謂的開發四化，指的是在實際開發企業資訊系統時，所應遵循的一套嚴謹的規範，分為四大主題討論，開發系統不能再像早期隨心所欲的藝術家式的開發模式，而必須向工業管理一樣，依循標準作業程序動作，如此才能確保程式的品質及日後的維運。

軟體開發本就該有嚴謹的規範，因為企業資訊系統是企業經營的核心資訊，唯有高品質、可靠的系統，才能確保資訊的正確性。也唯有如此，軟體產業才能真正的形成一個產業，而不再只是一個尚未成熟的【新興行業】，雖然前路崎嶇迢迢，希望有志於此行業的菁英們共勉之。

第十八章
開發企業資訊系統的十大守則

人法地，地法天，天法道，道法自然。

老子、道德經

第十八章 開發企業資訊系統的十大守則

原本這個章節是想討論標準開發程序(SOP)，不過筆者仔細想想，最後還是改變心意，因為事實上，本書從頭到尾都是在談標準開發程序，實在沒有必要多此一舉。

那談談哪個主題好呢？苦思甚九，忽然靈光一閃，想到多年前曾看過的一本書中，曾經提過一篇好文 『國華十守則』，其立論精闢、發人深省，筆者從中受益良多。幾經深思後，決定就來談談『開發企業資訊系統的十大守則』

所謂的十守則，只是將多年來專案執行累積的經驗，依個人的經驗，累積整理為十個常會犯錯的行為。這其中有很多都是個人體悟，基本上帶有很濃的個人主觀色彩，不過是野人獻曝，各位先進可以依自己的狀況自行套用即可。

首先，十守則條列如下

1 口說無憑
2 凡走過必留下痕跡，一定要寫註解及文件。
3 沒有測試，就不算完成
4 程式改動前，必定要先有文檔，一切依文檔作業。
5 做好程式版本的控管。
6 專案管理導入自動化工具
7 優秀的系統架構師是成功的關鍵
8 定期召開內部技術研討會
9 規格不得隨意變更，改必有因、並須遵循標準程序
10 當斷則斷、不適任就換人

下面就逐條說明十守則的內容

1 口說無憑

在開發系統的過程中，包括

. 客戶的電腦化小組成員
. 我方的業務人員
. 專案經理

等等，會有非常多的正式及非正式的會議，討論專案的相關議題。很多的好點子會在此時被激發出來，但是，提供意見的人往往會認為：『我已經說得這麼清楚了，其他是你們研發的事』，而研發就會想說『你就隨便提了一個想法，說的又不清不楚，這怎麼能寫成程式？』

要解決這種衝突，最好的方法(應該也是唯一的方法) 是『書面化』。每當有人有好的想法 (idea)，不論是誰，經討論後，都應該要新增到 Issue Tracking System (錯誤及建議追蹤系統) 中，這樣可以確保該建議一定會有人回覆處理。Issue 原意是 『議題』，包括程式的錯誤 (bug)、建議、或是一些重要的備註都是。

將好的建議，建立到資料庫中的好處是可以隨時查詢。尤其是 Issue，一旦建檔就不能任意刪除，只能異動他的狀態。所有人都可以查詢任何 Issue 的異動歷史，也可以針對 Issue 回覆並加註意見，當成一個簡單的討論區。如此一來，好的意見就可以在專案經理的安排下排入行程管理，或是因為進度的關係，暫時移到改版建議區，等待下次要更新版本時再納入，這樣就不會有遺珠之憾。

想要達到此目的，就一定要要求提出建議的人，要將想法文字化，並上傳到 issue Tracking 系統中。有相關經驗的人都知道，在將想法整理成文字的過程中，原本的想法會更有條理、更準確。雖然好像多了一些工作，但是仍應該堅持，如果有必要，建立一個簡單的獎勵制度並不是件太難的事。這就是十大守則的第一條，簡單又不複雜。

2 凡走過必留下痕跡，一定要寫註解及文件

許多程式設計師喜歡寫程式，但是很討厭寫註解或文件。甚至認為這是浪費時間、是一個苦差事，只要程式寫得好，寫文件是沒有必要的，但真的是如此嗎？只要多經歷過幾個專案，就會知道，程式交付並不是結束，相反的，艱辛的旅程才剛開始。一套資訊軟體系統，一般而言至少會有 5-10 年的使用期限。在這個漫長的使用期間，就算軟體的品質再好，也總會有一些小問題，或是因為營運的環境改變，需要增加或修改系統的某些功能。如果沒有註解或文件，即使是原作者，有時也要花很大的工夫，才能搞懂當初的寫法邏輯是怎麼回事，如果換人接手維護，那更是一場大災難。

所以，在專案進行中，專案經理和系統分析師，一定要花時間做 Code Review，要不定期的抽查程式設計師寫的程式碼，並不斷的訓練及要求程式設計師一定要寫清楚註解及文檔。為什麼要如此刁難？主要原因有二

1 為了日後其他程式設計師，能較輕鬆的接手維護
2 為了讓六個月後的自己，還能看懂程式寫法並維護程式

前面已多次提及，程式碼 (Source Code) 就是一連串有點像英文的邏輯指令，隨著程式設計師程度的提升，三個月前隨手寫的程式碼，如果沒有註解，很有可能連自己都看不太懂，何況是要給其他程式設計師維護。

所以好的程式設計師，不僅程式碼要寫得好，也有責任將程式邏輯及重要的程

式碼片段說明清楚。修改程式前，如果是因為規格有變動，一定要先修改規格文檔 (依標準作業程序處理)，如果只是程式的邏輯變更，那只需要在程式碼中加入註解說明即可。

規格文檔和程式註解是不同的內容。規格文檔的內容是源於客戶需求及 UI 操作介面的呈現方式。而程式註解則是記錄說明，要實作規格文檔程式，程式邏輯的解法。
如果以法律條文比喻，規格文檔是主要的法律條文，而程式註解就是施行細則。所以只要規格和程式邏輯有變更，一定要有相對的文檔紀錄，否則就不能修改程式，如此才能保留程式歷次修改的歷程，方便日後的維運修改。

3 沒有測試就不算結案
『測試』是筆者在本書中反覆強調的一個重點，也是筆者的一個短板。之所以要強調，主要原因是他並不受重視。一般專案進行時，通常都長期處於趕進度的狀況居多，能準時完成程式就已經要阿彌陀佛。測試通常只是聊備一格。這樣完成的系統，通常在上線初期，都會有一段黑暗期，程式問題會層出不窮，客戶和軟體公司互相抱怨，雖然即使有完整的測試，也無法保證就不會有問題，但至少會在可控制的範圍內。

所有的程式，在程式設計師寫好程式後，都必須經過測試才算完工。這個簡單的概念一定要不斷的深植在每個開發團隊成員的心中並落實執行。唯有經過嚴格測試的程式版本，才能對外發布，如此才能確保程式的品質。

當全公司都建立好這種品質至上的觀念後，那就何愁大事不成？一個優秀的企業資訊系統，當然就可以橫空出世。

4 程式改動前必先有文檔
任何一段程式要修改前，必須先有完整的文檔說明，如果沒有文檔，就不能隨意修改程式，通常會修改程式不外乎因為
a 程式有錯誤
b 原程式有效能問題，必須改寫程式邏輯改善
c 程式要求加入新的功能，或是有規格變動

如果是前面 2 點，因為規格並沒有變動，所以並不需要修改規格文檔，只要在程式加入註解說明即可，但若是第 3 點，因為規格有變動，所以要先請系統分析師修改規格文檔，然後程式設計師再依據修改後的系統分析文件撰寫程式。同樣的，也一定要在程式中加入註解說明，方便之後接手人員維護系統。如果省略這些看似多餘的動作，常會造成日後維護人員，看這一段莫名其妙，沒頭

沒尾的程式碼，不曉得到底是為什麼會有這麼一段神奇的程式碼？只能再發揮想像力，再加一段更神奇的程式碼。如果他也沒有寫註解，這樣多折騰幾次後，新鮮的義大利麵程式碼就熱呼呼的上桌了，這段程式碼就會變成所有人的惡夢，直到不知道什麼時候為止。

5 做好程式版本控管

如果是軟體開發廠商所開發的套裝標準版本，在第一個商業化版本上市後，就會開始面臨一個棘手的問題：客戶的客製化需求。

前面的章節已經分析過，套裝版本 加 二次開發，會成為企業資訊系統商業版本市場的主流趨勢，因為這是對客戶最有利的商業模式。但是如果沒有做好版本控管，那日積月累下來，程式版本就會非常的混亂。甚至會影響標準版本的穩定。因為程式只要有改動，就會有產生新錯誤的風險。

通常會用下列三種方法，來管理二次開發

a 若客戶所提的規格，是屬於通用型的需求，那就直接修改標準版本。這種方式通常視為『改版建議』的方式來處理。

b 如果客戶所提的需求，需要在某個表單加入特殊的功能：例如出貨單要加入一個查詢的按鈕，但是程式改變的幅度不大，此時可以用參數設定的方式，修改原程式。但異動的部分程式碼，則加入一段判斷參數是否有設定包括起來。

c 如果客戶所提和原功能差異頗大 (如特殊行業別的出貨單)，那就必須『分支』(Branch)，複製原程式另存為另一支新的分歧版本，程式代碼通常是 xxx_A 以和原程式有所區別。這種方式稱做為 Branch。將不同客戶的需求，寫在不同的分歧版本中，如此一來就不會影響標準版本的穩定度，又能符合客戶二次開發的特殊需求。不過這個方法如其名，分歧會有同一支程式多版本且和標準版本隨著時間而差異變大的問題。通常只要不是 bug，都會視為不同的分歧程式來分別處理。

6 專案管理自動化

要完成一個企業資訊系統的開發專案，事實上是一個非常艱難的挑戰。除了數不盡的表單功能及報表之外，還有許多複雜的商業邏輯要處理。要想如期完成 ，就一定要導入專案管理的工作任務及 issue Tracking System 的機制。

如果沒有系統協助，將管理自動化，要用人工處理這麼龐雜的專案相關資訊，專案經理不忙中出錯是不可能的。所以不管是自行開發，還是導入現成的系統，都一定要有系統來自動化管理，讓整個專案的所有進程及資訊，可以隨時讓專案相關人員隨時查詢，並依設定自動派送作業給相關的人員。

讓工作能順暢的進行且不必再多花心思在無謂的管理相關作業上，如此才能確

保專案能如期完工。

7 優秀的系統架構師是成功的關鍵

企業資訊系統和其他軟體系統最大的不同點，是在於它是『資訊技術』和『商業邏輯』的整合體。而能完全具備兩大領域專業知識的人才—系統架構師，更可謂鳳毛麟角。

優秀的系統架構師是絕對的不可或缺，他是

a 第一個規格版本的製定者
b 全套系統操作介面的主設計者
c 底層架構 (含主選單及資料權限) 的規劃者

同時也必須對整套企業資訊系統的規格細節瞭若指掌。

正因為企業資訊系統的特殊性，所以如果系統架構師的行業別專業知識不夠周延，必然就會產生企業資訊系統很多的缺漏處，這對號稱企業營運資訊核心的企業資訊系統來說，是無法接受的重大瑕疵。所以優秀的系統架構師，自然就是企業資訊系統能否成功開發的關鍵因素。

8 定期召開內部技術研討會

系統設計師及程式設計師等相關開發人員，在日復一日的寫程式生涯中，很容易會產生工作倦怠，總認為一天到晚都在寫程式改 bug，都沒有進步，也沒有學到新的技術。久而久之就很容易會有人事異動的狀況發生，他們的心聲是事實，而且長遠來看，並不符合公司的利益。最好的解決方式，是透過一系列的教育訓練 (包括內部跟外部)，來提昇開發人員的能力。

最簡單有效的方法，是不定期召開內部技術研討會，可以針對特定的主題：如 SQL 的預存程序的某些特殊語法 (如報表的多選機制)、或是 ERP 企業資訊系統的某些較複雜的商業邏輯說明，當然也可以討論新的技術以及新的元件等等，在這個過程中，加強人員各方面的專業知識。一旦開發人員對系統的了解越深入，程式的品質和效能也會更好，對公司及開發人員都是雙贏，而且可以內部提拔優秀人才，對公司長遠發展是非常有利的。

9 規格不得隨意變更，改必有因、並須遵循標準程序

『規格變更』通常是發生在純專案的系統或是套裝版本的二次開發專案需求上 ，無論是哪一種狀況，當面臨客戶要求變更規格時，務必要先釐清，是原先談規格時，系統分析師沒有確實弄清楚客戶的需求，還是客戶自行變動原本已確認的規格。

如果是系統分析師的疏失，內部檢討當然是有必要的，更重要的是系統分析師

必須清楚的說明，為何會失誤？以及日後該如何避免。

搞過專案的開發人員都知道，小幅度的規格變動是無法完全避免的，所以筆者不是談『不得變更規格』，而是要嚴謹的面對規格變更，任何規格變更都必須經過系統分析師、專案經理及客戶方的電腦化資訊小組，仔細的討論後，簽署規格變動確認書，才能由專案經理排入行程並責成系統分析師修改規格書。一定要無比嚴謹的面對，才能避免永無止境的規格變動，才能確保專案能順利結案。

10 當斷則斷 不適任就換人

軟體產業是完全的腦力密集產業，企業資訊系統專案的成敗，百分之百是取決於整個開發團隊的努力，這當然同時包括客戶端的電腦資訊化小組。

企業資訊系統開發的過程，很類似大航海時代的探險家們勇於尋找新天地的冒險精神，要想成功的找到通往印度的新航道或是意外的發現美洲新天地，除了事先詳細的規劃及大膽的行動外，船員們的應變能力也是不可或缺的要角。也因此，在系統開發的過程中，如果發現有不適任的人員時，專案經理應適當的調整人力的配置，如果真的不適任，就應該立刻換人，免得影響專案的品質及進度。甚至，如果是專案經理不適任，也應該盡早更換，在專案經理這個層級，如果不適任，通常是人的問題沒有搞定，趁早更換也許能讓專案重新步入正軌。

開發企業資訊系統的專案，細節多如牛毛、問題千頭萬緒，當然不會只有區區的十個守則，筆者只是簡單的整理之前的經驗，提出一些重點供各位參考，如果能確實落實這十大守則，並依循本書的建議做法，要開發一套好用的企業資訊系統，就不會再是一件難事了。