

# 遊戲程式設計初學者常遇之疑問

※ 原文刊登於遊戲設計大師 1999 年四月號

作者：陳昇瑋（原名 陳寬達）

網址：[http://www.iis.sinica.edu.tw/~swc/index\\_c.html](http://www.iis.sinica.edu.tw/~swc/index_c.html)

## DirectX 的角色

DirectX，這讓初學者最易搞混學習重心的罪魁禍首。

DirectX 在遊戲開發者的歡呼聲中帶著一身眾人的期許誕生後，由於它的版本更新速度遠比作業系統還快，因此，使用者不得不自行至網路上下載或由光碟來安裝 DirectX runtime library，DirectX 就是由此吸引使用者的目光而得聲名大噪，就如同 ActiveX，COM，OLE 一般，即使不曉得它究竟是什麼玩意，至少也能琅琅上口，輸人不輸陣嘛。

緊接著各大書局的電腦書區開始成批出現打著 "Games SDK" 及 "DirectX" 名號的書籍，讓這些對遊戲設計有著憧憬夢想卻還不得其門而入的遊戲愛好份子們誤以為 DirectX 是遊戲程式設計最重要的一環。事實上，如同衣裝之於人類，DirectX 之於 Game Application 的關係僅止於外表的打理而已，至於內涵則是另話，與人的衣裝，Game Application 的 DirectX 同樣是八竿子打不著邊。

DirectX 中，最重要的當屬 DirectDraw，它是屬於不吃不可，不用不行型的，也就是說，若微軟沒有推出 DirectDraw 這套程式庫，堂堂一個功力深厚上可飛天下能遁地的程式設計師也照樣沒輒，無法突破 GDI 及 HAL 這些層層的防護直接控制顯示卡，唯有如此才能得到最高的畫面顯示效能呀。也就是說，我們無法不依賴 DirectDraw 而擁有 DirectDraw 所提供的功能及效率。至於另一個重要的元件，Direct3D，也與顯示卡驅動程式直接私

通，才能獲得快速的 3D 繪圖效能，同樣屬於不吃不可型（除非你偏好 OpenGL）。其它的元件，如 DirectSound，DirectMusic，DirectPlay，DirectInput 及 DirectSetup 等，都大大地方便我們撰寫遊戲程式所花費的功夫，你也可以不使用它們而作到一樣的功能，不過既然別人都為我們做的好好的了，除非有充分的理由（例如自行發展快速而穩固的 TCP/IP 連線函式庫），否則不去使用還還真是自找苦吃:p

另外，雖然 DirectX 是真正構築於 COM (Common Object Model) 上的套件，我想是爲了降低學習門檻及使用的方便性，設計者將較容易讓 COM 初學者困惑之處包裝起來，讓你不必先閉關修煉 COM 三年後才能使用 DirectX，就當作是一般的物件導向程式庫來用即可。所以當你迫不急待想感受 DirectX 的 power 時，可以先暫時不理 COM，用了再說；當你已能靈活運用 DirectX 的好處時，可別忽略在底層辛苦出力的 COM，是它的功勞才讓 DirectX 能以二進位形式突破原始碼種類的限制讓你可以任意程式語言工具輕鬆享用物件導向程式設計的好處，對這大功臣還是不可不熟悉熟悉喲。

嗯，回到原題，提供極欲入門遊戲程式設計的各位一句經驗談，不必花太多的時間金錢及精神在學習鑽研 DirectX 的微末細節，遊戲程式設計關鍵之處莫過於整個遊戲的核心製作及內容設計，相較之下，DirectX 真是比較輕鬆愉快的課題呢。試想：以遊戲程式設計員的角度來瞧，對 DirectX 來說是 "user"，使用 DirectX 的 API；而對遊戲核心及內容來說，是 "designer"，要設計高效率運作良好的核心及豐富迷人的內容來滿足遊戲玩家們，你覺得，學習的重心該放在哪呢？

噢，順便一提，直到現在，我仍認爲對稍有經驗的程式員來說，學習 DirectX 最好的教科書即是 DirectX SDK 裏頭附的文件，厚厚幾百頁，讀一讀再寫一寫，絕大部分 DirectX 的馬步工夫是可以這樣就學得的。

## 開發環境的選擇

對於許多剛接觸 Windows 程式設計的新手而言，要在各家說法紛云、眾多開發工具強伺的環繞之下，選擇一個明智有把握又最適合自己的開發工具的可算是最難的一道入關匣

門了。不啻是新手，就連許多老手也常執著於同一套工具軟體，寧可使用鍾愛的工具埋頭苦幹，無視於身旁更方便好用的解決方法，即使它可以省下開發者三倍的工作時間。開發工具確是這樣重要的一項選擇，選的好，可讓你天天有時間陪女友看看電視逗逗小狗，萬一不幸選到難學難用難寫或者根本不適合該專案的開發工具，就有無盡的漫漫長夜等著你囉。

常跟朋友聊起，面對開發工具及程式語言的選擇，約略可將所有的程式員分為三大類：

## 新手型

對所有的開發工具程式語言甚至開發平臺全然陌生，只是大略聽過一些開發工具的名稱，而且還經常背錯，如 "Virtual C++", "Virtual Basic", "Dephli", "Borland C Builder" 等等。這個族群所佔的比例最高，也往往是在網路論壇上詢問「該學什麼程式語言？」「我想寫遊戲，該用什麼語言？」這類問題，甚至常是引發程式語言及開發工具優劣論辯的導火線，雖然是懵懂無辜的。

## 專家型

所謂專家，即是訓練有素的...呃，技術實力高人一等的...呃，嗯，專家。他們通常精通一種開發工具，獨衷一派程式語言，擅於撰寫特定領域的程式，該開發工具提供的函式庫，framework 滾瓜爛熟。但，卻往往獨衷特定工具及語言，甚至帶點宗教式的狂熱，這類型的玩家常是網路論壇上語言及工具論辯的超級打手。

## 無入而不自得型

他們往往會熟悉至少兩三種以上的開發工具及程式語言，並將火力集中在與語言無關的系統呼叫 (API)。於是，開發 Client/Server Database 專案時，拿 Delphi 來拉拉資料庫

元件; 撰寫遊戲時, 裝起 C++Builder 下載 DGC 元件立刻拼湊出一個遊戲外框; 專案用到 VxD, WDM 或 kernel mode driver 時, 捲起袖子拿出 SDK, DDK 加上 Visual C++, 再買套 VToolsD 或 Driver::Work 來立即動工。無所為無所不為, 不執著於任何開發工具及語言, 自然不會被任何公司的規劃 (如 MS 的 VBA 吃遍天下) 或美好遠景 (如 Inprise 的 Information Network) 等解決方案所羈絆, 而隨波逐流了。

現在有許多電腦玩家常不由自主地對自己所鍾情的作業系統、開發工具、應用軟體甚至遊戲及軟體公司產生幾近宗教崇拜式不明究理的狂熱。在此情況下, 很容易去找到一個貌似敵對的「對手」來反, 以堅定自己的信仰、壯大自己的聲勢, 例如 PC vs MAC、FreeBSD/Linux vs Microsoft Windows、Visual C++ vs C++Builder、Delphi vs VB 等等。

在情況越益嚴重無法遏抑其勢的今日, 只能期待點醒一些狂熱份子, 擁 X 反 Y 並沒有什麼不對, 但是:

一個人若是爲有了宗教信仰而驕傲、自滿, 甚至因此鄙薄無信仰的人, 或動輒排斥與他信仰稍稍不同的人, 便表示他自己還沒有找到信仰, 所以, 他也在他自己鄙薄和排斥之列。

<<疑神>> 楊牧

TANet 一位大哥級的人物也曾語重心長地說: 「科技的出現應該是要爲人來服務的, 你盡可以擁抱 X 痛罵 Y, 不過切記, 知己知彼百戰百勝」。這是看到網路上一堆連保護模式、檔案系統、系統軟體、排程問題、虛擬記憶體都不知何物的網友痛罵 Windows 作業系統, 連 API、DLL、物件導向、application framework 都摸不清脈絡的新手卻大聲疾呼推行或反對某某程式語言或開發工具的怪現狀所發的無奈之語吧! 吾亦有同感。

對於「我想寫遊戲, 該用什麼語言?」這類常見也會永遠存在的問題, 我曾在網路上見過這樣的回答: 「Visual C++ 最適合了」、「VB 好學, 所以 OOXX」等等十分 no-sense, 說是不負責任也不爲過的言論。不想流於空談, 我想還是來談談我自己的看法好了, 就像人類戰士通常拿雙手巨劍, 侏儒通常肩著巨斧, 而魔法師無可選擇地手執魔杖一樣, 選擇一把稱手武器還是得視個人的情況及需求來量身訂作:

## 一.學習動機

遊戲程式設計是程式設計各領域中，狂熱及理想成分比例超高的一群，甚至有許多各種性質的程式設計師，當初都是因為熱衷遊戲設計，而就這樣持續掉入程式設計的漩渦呢。學習動機是毫無疑問地：想要具備撰寫遊戲的程式功力。不過，就依遊戲種類的不同，日後學習的方向與重心也迥異，例如 2D RPG，重點在於畫面設計及故事劇情，外加 DirectDraw 提供的快速捲軸能力；3D 動作遊戲，Direct3D 或 OpenGL 是跑不掉的，對於圖學的理论，演算法及應用面，也最好花心思時間去努力研究學習；再如多人連線棋類遊戲，DirectDraw，Direct3D，DirectInput 我想都用不太上，好好研究提供連線功能的 DirectPlay 或發展一套穩固的 Client/Server Socket Library 才是重點。

## 二.目前基礎

目前基礎是決定工具及語言上手度的最重要因素。許多人在高中時代學了 QB，之後便接著玩 VB；有些大學的大一計概課程教的是 Pascal，接著便可順利進入 Delphi。必須承認的是，Basic 確實不是開發大型程式適當的語言，它的先天不良，例如執行速度慢，不是物件導向語言卻硬加入類物件導向功能（事實上，它只可算是 object-based，而非 object-oriented），甚至使得微軟爲了 Visual Basic 一個語言，將 COM 規格做了些修改以配合之（如 IDispatch interface），即使有微軟如此強而有力的老大哥極力護盤，先天缺陷仍舊無法去除，除了易學外，實在找不出太多該使用 VB 的理由。VB 雖然可以使用 DirectX，但還必須透過其它程式庫的幫忙，因此除了 VB，我很贊成就配合你目前的所學，會 Pascal 就用 Delphi，愛用 C++ 就請用 C++Builder 或者 Visual C++。

若原本是一張白紙，還未學習過任何程式語言呢？那也好，請見下段，視你的個人偏好囉。

### 三.個人偏好

Basic, C/C++, Object Pascal 這三個程式語言，雄霸著整個 Win32 程式設計領域。Basic 易學易用，Pascal 嚴謹明確，C++ 強大複雜，各有各的擁護者及理由。Basic 簡單是因為微軟希望 VB 及 VBA 維持在簡單到任何想依靠電腦來做自動化程序的電腦用戶都可以輕易地上手的程度，因此雖然功能不斷上疊，語言本身維持著 Basic 的所有特性。不過缺乏物向導向的支援及執行速度的緩慢，確是超級無敵讓人沒力的致命傷，因此我會建議所有的初學者，若能有力能夠接受學習其它的語言如 C++/Pascal，轉移陣地為上策。

C++ 的強大無庸致疑，template, exception-handling, RTTI, Standard Template Library 等功能不斷地加入翻新，由於使用者眾，要求必多期望必高，再加上 C++ 本身定位於功能強大範圍廣泛的通用性語言，如江海之納百川，C++ 自然日益複雜。著名的雜誌 C++ Journal 上曾有段話讓我印象頗深，「如果你認為 C++ 還不算太複雜，那麼請你解釋何謂 protected abstract virtual base pure virtual private destructor，何你又會在何時需要它呢？(Tom Cargill, C++ Journal, Fall 1990)」雖然是最流行的 OOP，但除非你有足夠的耐心及精神來全盤掌握它，否則輕易嘗試的後果可能只會得到一臉的挫折。當然囉，十分的複雜也帶來十分的便利及不同的樂趣，我有一位朋友，工作上使用其它語言，但將 C++ 當作興趣來把玩，跟酷企鵝一樣酷呆了。

Pascal，其實應該說是 Object Pascal，為 Borland Delphi 所採行的語言。Pascal 的嚴謹明確是自 Niklaus Wirth 發明它以來一直遵行的宗旨，而之所以可以順利演化為完全的物件導向程式語言 Object Pascal 是由於 Borland 公司對 Pascal 語言的全盤掌握，就像 FreeBSD 的 core team 全盤控制所有 FreeBSD 套件的更新撰寫一般，Pascal 控制權控制在 Inprise 一小撮人手中，雖失去開放性，但保有該有的堅持及清新，也因此我認為它的物向導向支援恰得其所，該支援的全都支援了但也沒有更多。它與 C++ 的優劣是沒有答案，見人見智的，正如同大禮服及小洋裝，好不好看，適不適合，因人而異。

## RAD Tool 無罪論

最近網路上興起一道 "Visual C++ 與 C++Builder 孰優孰劣的討論", 其中可以看到一些頗為中肯的言論:

發信人: Meou@m2.dj.net.tw ( Dadai ), 看板: programming

這兩個東西都蠻好用的。但是現在我摸了幾個月之後，我反而比較喜歡 VC++。VC++ 的使用者介面看來繁瑣，但是真的用心花個幾天把 VC++ 的功能摸熟，VC++ 用起來還蠻順手的；再加上把 VC++ 的 Application Wizard 的來龍去脈搞清楚，把 Class Wizard 的用法弄懂，MFC 一點一點慢慢弄熟之後，VC++ 的功能還蠻強大的，加上 VC++ 的 HTML Help 比 BCB 好上百倍，我現在覺得 VC++ 比較好用。

其實這兩個工具，一個是倚天劍，一個是屠龍刀，放在不會用的人，那一把都不順手。這兩個工具都需要相當好的 C++ 基礎。好的 C++ 基礎對 MFC、OWL、VCL 來說都是基本功夫而已。程式學了一陣子，覺得 MFC 摸熟之後，construct 介面不比 BCB 來的慢，**重點在於介面之下你到底要如何解決問題**。這個問題遠比 Application Framework 及那一套開發工具比較好來的重要多了。

發信人: dyliau@msi.hinet.net ( 四眼的王蟲 ), 看板: programming

VC++ 的 help 系統的確比 Borland 的 Delphi，BCB 等好太多了，Borland 在這一方面一直沒有甚麼進步，有時候我用 Delphi 或 BCB 時還是會執行 VC++，目的就是要用 VC++ 的 help 系統，Borland 實在應該在這一方面大力改進才是。

介面方面 MFC 實在是比不上 VCL，簡單的介面還好，複雜一點的 MFC 就得搞上老半天。

發信人: oesd@email.gcn.net.tw (Dadai) , 看板: programming

VC++/MFC 的 Learning Curve 看起來比較長，但是值得。因為你如果搞懂的話，再配合上一些 SDK 的 knowledge 及合適的 development kit，你幾乎可以在 Windows 下做到任何你想做的事（剩下的只是你怎樣解決你要解決的問題）。我自己覺得 MFC 要用的有點基礎，最起碼你要能把 VC++ Wizard 能做的東西知道如何全部用手工打造出來，不然用 Application Wizard 建出來的東西，你一樣看不懂，更不要講如何去改它了。

**其實像 BCB 這種 RAD Tool 要學得好，我覺得比 MFC 還難，因為在那漂亮介面下的底層機制往往比 MFC 複雜許多。**真的大聲喊 BCB 好簡單的，我只看到兩種人。一種是在 Win32 SDK 裡面打滾多年，幾千條 Win32 API 就算沒用過也都大概摸過，沒摸過也知道大概會叫什麼名字，該往那邊找。問他一個問題，腦袋裡面會自動列出一堆 Win32 API，一條條過濾該如何解決。寫 Windows 程式可能打字到手指頭都長肌腱炎了。BCB 對他們是種解脫，VCL 更是不成問題。

而另一種則是完完全全的初學者，拿 BCB 來學 Windows Programming。最多只能學到 Component 有提供的功能都會用，Component 不會的他也不會，Component 不行的他也不行，Component 的限制就是他的限制。AP 寫到一半，中間要 call Raw WIN32 API，他大概就掛了，要做到現有 component 做不到的功能？那你要不要花錢買 VCL library？同樣是 RAD Tool 的使用者，有的是全然解脫，輕巧駕御 RAD，大部分卻屬於鎮日搜索元件，侷限於別人製作的元件功能內的 RAD enabled-expert，你願當哪種人？

另外，再聽聽 Visual C++ 知名作家侯捷在他的「懷璧其罪 RAD」一文中怎麼說：

RAD 並非罪惡，而是優點。要怎麼用它則是 developer 自己的問題。

侯捷對於 RAD 工具如 Visual Basic 或 Delphi 或 C++Builder 並不擅長，但我知道

Visual Basic 可以呼叫 Windows API，做相對低階的動作；當然，以軟體工程角度來看，VB 是比較弱，因為它不具 OO 特性。至於 Delphi，我有兩位這方面的專家朋友（Wolfgang 和 Xshadow），他們可以使用 Delphi 做任何事情，沒有任何你想像中 RAD「該有」的限制；C++Builder 和 Delphi 系出同門，一樣沒有什麼限制。

以下引一段我在【汗如雨下·雜感·1998 / 11】的文章片段：

● 關於 RAD（Rapid Application Development）

<Delphi 學習筆記> 作者錢達智先生，是我的好友。我們之間對於 RAD 有段討論，或許你想聽。

■ 侯：BBS/News 上時有關於 RAD 的工具見解。我認為你很有資格說些話。不少人對 RAD 有誤解。如果你能點醒大家：

1. RAD 是很好的開發工具
2. 使用 RAD 並不代表不需要底層紮實的基礎，那麼誤解的人就會比較少一點，知道該怎麼做的人就會比較多一點。

■ 錢：過去在 DelphiChat、NEWS Group 以及我的書中，這樣的想法都不只一次宣揚過。

其實，就我接觸過的人，不論是網路或者是學生，RAD 的使用者的確是比較急功近利，也難怪會有這樣的刻板印象。:(

雖然說過，但還是要持續宣揚這種理念，就如大哥說的，誤解的人會少一點，知道該怎麼做的人就會比較多一點。

RAD 真是「匹夫無罪，懷璧其罪」呀。

Visual C++ 及 C++Builder，一個採 MFC 一個用 VCL，一個必須配合 Wizards 撰寫大量程式碼，一個雖然可用滑鼠完成大部分的介面設計，不過程式邏輯及核心還是得撰寫程式碼，此時 RAD 派不上用場。不論如何，他們提到了幾個重點：

1. 一個是倚天劍，一個是屠龍刀，放在不會用的人，那一把都不順手。
2. 重點在於介面之下你到底要如何解決問題。
3. RAD 開發工具要學得好，不比 non-RAD 開發工具簡單，在漂亮介面下的底層機制往往出人意料地複雜。
4. RAD 及 non-RAD 開發工具，擁有相同的「願望達成能力」，功夫底子好的人要起來，樣樣都能實現。

所以，重點在人身上。不管是 RAD 或 non-RAD 開發工具，用得嚇嚇叫的那些人永遠可以做出他們想要的成果；不管是 RAD 或 non-RAD 開發工具，學得不夠透徹的人們永遠也只能抱怨程式語言太難、開發工具太爛，無法享受程式設計的樂趣。

換個角度來看，同樣是 RAD 開發工具的使用者，有的是全然解脫，輕巧駕御開發工具，善用 RAD 的特性來提升程式開發速度及品質；有些卻只能拉拉元件，能力侷限於別人製作的元件功能，因為跨不出開發工具的格局，完全被 RAD 的服務範圍限制綁死。

## Talk

也許有人抱持著相同的懷疑：Delphi 到底能做些什麼？

我必須再一次大聲呼喊：「在 Win32 下，除了驅動程式撰寫<sup>1</sup>以外，只要是其它開發工具辦得到的，Delphi 就辦得到！」

就我所見到的，埋怨開發工具能力不足的人，通常同時在揭露著自身能力的不足。而真正見到開發工具能力不足或設計不良的那些人，不會大落落地成天埋怨責怪，他們總有辦法另找出路來解決問題。

---

<sup>1</sup> 其實，若搭配如 WinDriver 的發展套件，Delphi 也可以拿來撰寫驅動程式。

## 開發工具的差異

以大局觀 Visual C++、C++Builder 及 Delphi 這三套工具，就最重要的差異來列出以下這張表：

表 1-1 / Visual C++、C++Builder 及 Delphi 三套工具的比較

比較項目	Visual C++	C++Builder	Delphi
設計公司	Microsoft	Borland	Borland
前端語言	C++	C++	Object Pascal
Application Framework	MFC	VCL	VCL
介面設計方式	傳統 (Class Wizard 及手工打造)	RAD (拖拉點按)	RAD (拖拉點按)
程式核心	手工打造 (有許多程式庫及類別可供運用)	手工打造 (有許多元件、程式庫及類別可供運用)	手工打造 (有許多元件、程式庫及類別可供運用)
運作原理	呼叫 Windows API	呼叫 Windows API	呼叫 Windows API

其實，不論什麼程式語言，不論什麼開發工具，只要在同一個作業系統內，它們的運作原理都是一樣的：**呼叫作業系統提供的服務 (通常以函式呼叫的方式)**。在 Win32 環境下，我們稱這些系統服務為 Windows API。

## Win32 開發工具的演進

呼叫由 kernel32.dll、user32.dll 及 gdi32.dll 三大模組為首所提供的數千個 Win32 API，原本是 Win32 應用程式開發者唯一可用的方式，這些 API 分成幾種類別，各擅其職，只要利用它們，少有做不到的事，唯一的缺點是，API 多且繁雜，而且如同 RISC CPU 的指令，每一道 API 所做的事情並不多，往往一些必須頻繁撰寫的例行公事，例如建立新視窗，註冊視窗類別，更改按鈕顏色等等動作，還得花上十幾行程式碼來做，麻煩

透了。需求乃創造之本，於是程式庫出現了，緊接著挾著物向導件的優點類別庫也出現了，最負盛名的兩套就是 Microsoft 的 MFC 及 Borland 的 OWL。慢慢地，雖然有類別庫及 wizards, experts 的輔助，仍有人不知足地想要發展能夠更快地開發應用程式的方法，於是就有了 Visual Basic 這類可靠滑鼠完成大部分介面設計工作的 RAD 開發工具，最後才是 Delphi 及 C++Builder。

## RAD 無罪，輕鬆有理

隨著時間的腳步，人們總要適應大環境的變遷及進化，RAD 的確為程式開發員省下不少介面開發的時間，但相對地來說，它也降低了程式設計的門檻，太多的初學者沈溺於 RAD 元件的強大及使用，不知有程式語言之重要，無論底層的系統呼叫。我想這也許是 RAD Tool 開始受到部分人們的質疑之故。

看透非 RAD Tool 及 RAD Tool，抹穿 MFC 及 VCL 這兩套 application framework，它們只是包裝一薄一厚，用法各異罷了。MFC 薄薄的一片，讓你擁有全盤掌握的滿足，相對地，學習曲線既陡峭且高峻，需有足夠的背景知識才能充份融入 MFC，享受它的好處。VCL 包裝地並不徹底，但厚厚地這一層，讓人完全看不到骨子裏的究竟，如同寒流一來，一個身穿五六件襯衫外加夾克兩條的女人打從你眼前走過，天知道究竟是蛇腰豐臀亦或瘦骨嶙離呢。就介面打造來說，VCL 包裝的真是好用方便，不過常有 VCL 力有未逮，包裝不足時，此刻，是 RAD 也好，非 RAD 也好，任何工具幫不上忙，只有瞧自己琢磨 API 的功夫，若沒有三兩下功夫，馬腳隨著 framework 的不足就立即露出了。

這讓我想到幾年前曾發生的「command line 優劣之爭」，有些高手們及 UNIX fans 喜愛命令列，一來速度快，二來有全盤掌握的感覺；而另一派當然是傾向 GUI 囉。傳統確實有傳統獨到之處，否則為什麼在電子音響大行其道的今日，仍有玩家願意傾幾十倍的价格，把玩真空管音響呢（我老爸就是一個:p）？Command line 用得熟，操作速度比 GUI 還來得快上幾倍倒是真的，我覺得這是 command line 需要存在的最大理由。也有人對我說，「command line 較讓人懂得電腦真正的運作原理」，我告訴他，「為什

麼 GUI 就會妨礙到我們去瞭解電腦的運作原理呢？」守舊也得要合理的理由，否則只是戀舊，潛意識裏的觀感其實是怕跨入新的領域而失一向保有去優勢。

所以呢，手工打造的好，還是拖拉點放的方便？手工打造的快，還是拖拉點放的省事？我想答案是很明顯的，寧可在例行公事上能省時間就多省點時間精神，我們還有未來等著去創造呢，焉可鎮日沈迷在手工打造全盤掌握之感覺中呢？我想，RAD Tool 無罪，它只是讓門檻變了低，讓程式設計變得簡單輕鬆，何罪之有！