台灣軟體產業的失落十年

By Victor Lin

台灣軟體產業的失落十年

Victor Lin

This book is for sale at http://leanpub.com/the-lost-ten-years-of-taiwan-software-industry

This version was published on 2014-12-02

Contents

試閱樣本

此書為「台灣軟體產業的失落十年」試閱樣本，欲閱讀完整內容，歡迎前往購買[1]，如需要 PDF 與 MOBI 格式者，可前往 Leanpub 購買[2]。

[1] https://readmoo.com/book/210010206000101
[2] http://lost-ten-years.victorlin.me

人才篇

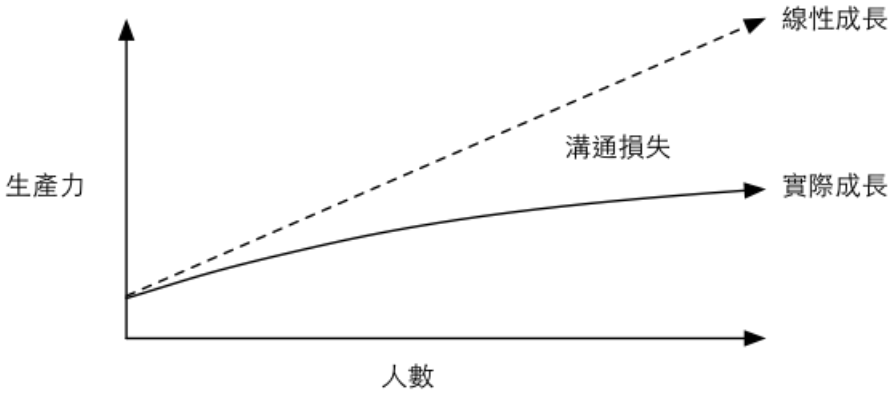
人的素質決定一切

軟體開發是一個很特別的過程，集成衆人的智慧進行協作，並非像是工廠生產產品那樣，輸入原料，經過一定製程生產產品，許多人有一個錯誤的觀點是將其它產業的經驗帶入軟體的開發，認為軟體只要投入足夠的資金，購買廠房設備，僱用工人，就能夠進行軟體的生產，然而事實上程式碼皆由人所撰寫，因此軟體開發的品質成功與否完全是基於人的素質之上，並非有好的設備和足夠的資金就能夠保證成功，這是和其它產業有非常大的差別，除此之外還有一個很重要的差異是，軟體的生產力和品質通常難以被量化，不像是許多工業或硬體的產品可以輕易計算出每個月的產量、硬體的良率以及各項數值，軟體開發的產出比較像是找了人來給他一個題目寫一篇文章，究竟這篇文章生產力高與低，如果只單純用字數來判斷，那麼只要加很多贅字在文章裡，生產力乍看之下很高，但其實不然，文章品質的好與壞，在經過許多人的閱讀可以給出一個大略的評價，然而很難有一個公平客觀的標準，以軟體開發的角度來看也是一樣，程式碼的行數並不能完全代表生產力的高低，軟體的品質好與壞除了依賴有經驗的工程師閱讀也難有簡單的方式做出評斷。

當軟體生產的品質好壞由人的素質決定，連判斷軟體的品質和生產力都需要有經驗的工程師達成，這就引出了一個簡單的事實，對於軟體開發的來說最重要資產，不是設備、不是廠房，而是在於人，忽視人才的軟體公司註定要失敗，不幸的是在台灣許多企業無法理解這點，以傳統產業的經驗來開發軟體，將人才視為用完即丟的工具而非公司最重要的資產，是很常見的失敗原因。

輕視溝通成本

對於軟體開發的一個常見錯誤認知即是以為只要增加團隊的人數，生產力就可以隨之增加，就像是擴增生產線一樣簡單，事實上軟體的本質是思想與方法，團隊裡的工程師要在一起開發一套軟體需要將不同的想法統一整合在一起，需要付出溝通的成本才有辦法達成，軟體工程的經典書籍人月神話曾用收割小麥來比擬可以透過增加人力增加生產力的工作，收割小麥不需要太多的溝通，因此只要增加人力即可加速工作的完成，但對於軟體開發來說，越多人其實表示需要越多的溝通，當一個團隊有兩個人時溝通的組合就只有一條，當有三個人就有三條溝通的組合，四個人增加到六條，隨著人數的增加，團隊成員之間的溝通連線是以指數成長的，這也意謂著人數越多溝通的成本也越重，舉個例子，如果一個團隊裡十個工程師每個人自行開發的生產力定為十，兩個人一起開發可能生產力只剩十八，三個人一起開發可能只有二十四，如果十個人一起開發說不定生產力總合只剩五十。



團隊溝通造成的生產力損失

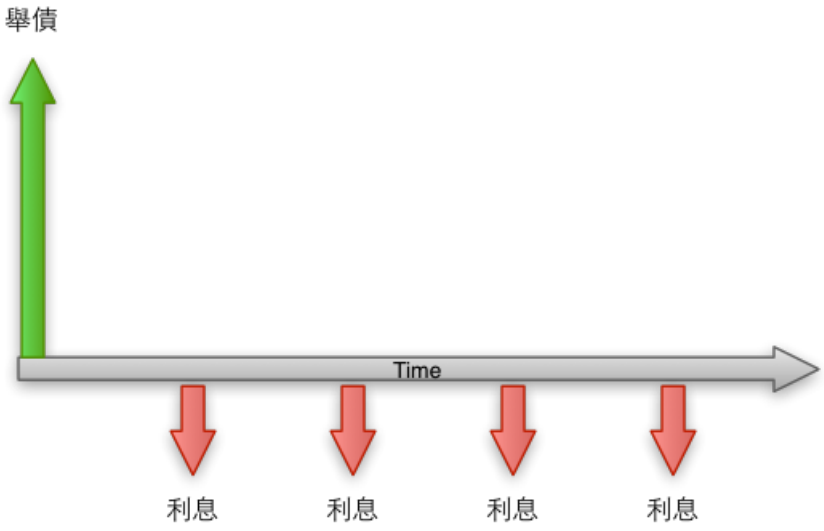
輕忽溝通的成本是常見的軟體開發錯誤，很容易就將工廠生產線的經驗套用在軟體開發上，例如可能找來許多工程師以期待可以加速開發，將寶貴的開發時間浪費在無意義的會議上，如果不能正視並改善溝通成本問題，生產力很容易就被消耗殆盡，最後團隊都只是在空轉。

工程篇

技術債

大多數人對於債務都多少有個概念，當你買了房子會有房貸，買了車子會有車貸，不管你多富有，或多或少都生活在債務之中，對於軟體開發來說，事實上也有所謂的技術債，技術債和現實的債務有點類似，有趣的是，並非所有軟體工程師對於技術債都有正確的概念，特別在台灣，技術債往往是被忽略的，甚至有很多人都不知道技術債的存在。

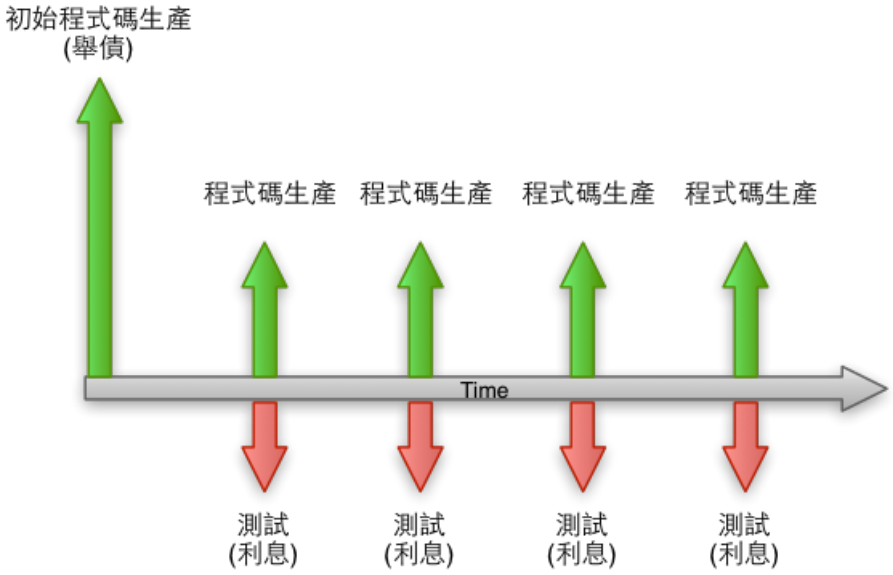
我們在此以財務上的現金流量表來解釋技術債，以下圖為例，如果你向銀行借款，約定每個月給付利息給銀行，在一定期限後還清債務，那麼它的現金流量表看起來如下圖所示



銀行債務現金流量圖

綠色（上方）的箭頭代表了收入，紅色（下方）代表了支出，在你舉債之後，立刻有了一筆收入，接著你每個月必需支付給銀行利息，技術債就是類似財務上的債務。

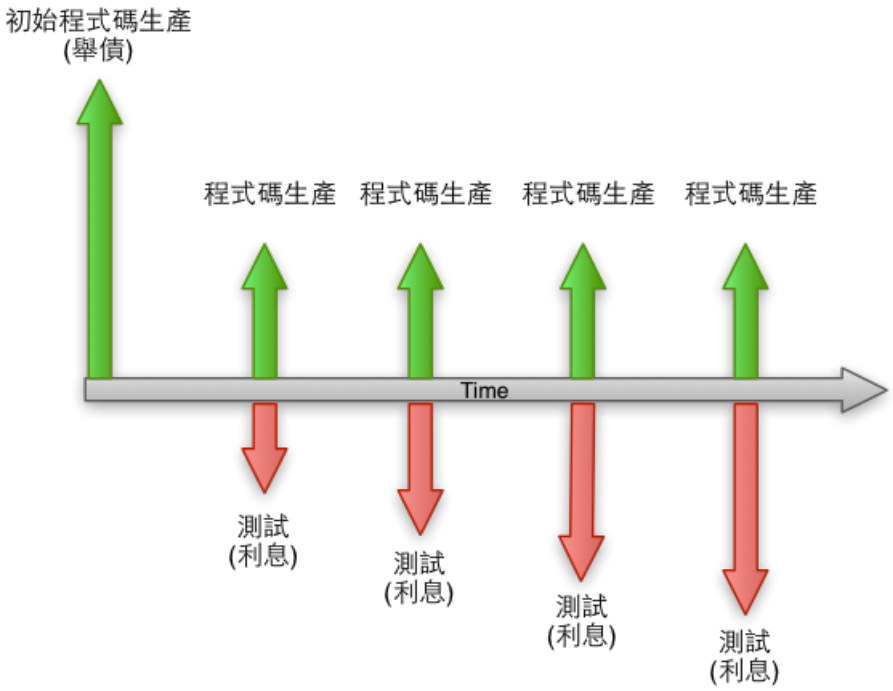
技術債不像是財務上的債務需要特別去舉債，很多是技術債在軟體開發的過程就內建了，舉個例子，當你開發一個只有單一功能的系統，每次你修改或改進程式時，你都得測試過這個功能一次，確保它可以正確運作，因此每次的修改，你都付出了額外的時間進行測試，這便是技術債所產生的利息，它是軟體開發與生俱來你需要付出的成本，如下圖所示



技術債現金流量圖

綠色（上方）的箭頭代表生產，紅色（下方）的箭頭代表損失的生產或是開發額外的時間花費，隨著你持續改進這個程式，雖然每次都得付出測試的心力，但是整體而言程式的品質隨著時間提升

事情看起來還在控制之中，然而當越來越多的功能被增加到這系統之中，每次修改程式都得重新測試過這些功能，確保沒有因為這次修改而弄壞某個功能，隨著功能增加，每次你修改要付出的利息都越來越多，如下圖所示

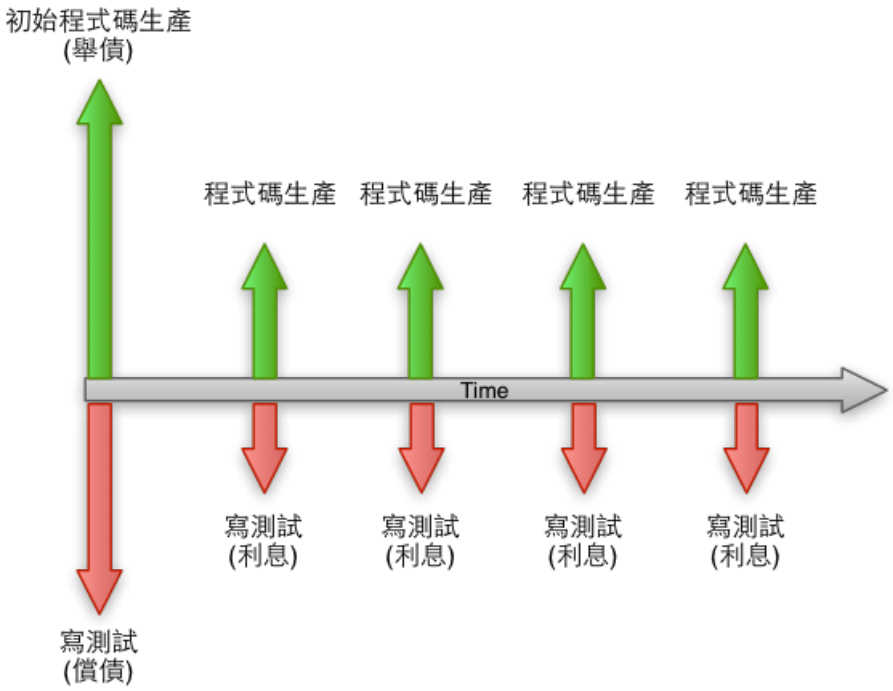


技術債利息隨時間增長現金流量圖

直到有一天，你測試所花費的心力遠大於花在程式的生產上，你的軟體開發進度就會從此陷入泥沼，你發現你不管怎麼修改，都會產生新的問題，永遠都在原地踏步，簡單的來說你的支出遠大於收入，當然不可能有進度，在台灣很多人的做法是忽略修改後的測試，其結果就是可能產品發佈到使用者手上時才發現出了問題。

要解決因為測試的成本花費所引起的技術債，其實只要引入自動化測試，而自動化測試，簡單的來說其實就是將針對每項功能或元件應該有的正確行為，以預先寫好的程式進行自動的測試來替代昂貴又重覆的人力花費，當然撰寫自動化測試程式碼需要額外的心力，我們可以把這視為償債，更詳細的細節我們會在後面的自動化測試章節介紹。

有了自動化測試，技術債的現金流量表看起來會像這樣子



技術債受自動化測試控制的現金流量圖

一開始花費了額外的時間建立了自動化的測試，隨著功能的增加，也同時增加相對應的自動化測試案例，這樣一來，隨著系統規模的成長，花費在測試上的心力受到了控制，軟體的品質也有了一定的保障。

缺少自動化測試只是技術債的一種體現，事實上技術債有很多種不同的來源：

- 不良的程式碼風格
- 不良的設計
- 缺少自動化測試
- 缺少文件
- 程式碼缺少適當的註解
- 未使用版本控制系統
- 骯髒的寫法

在此未必所有的技術債來源都詳盡的被列出，但他們都有一個共同的特點，就是在開發的過程中你必需為這些因素持續付出額外的心力，我們都可以視為技術債，例如不良的程式碼風格，每次工程師本身或是團隊裡的其它人要讀懂那段程式片段時，都得花額外的心力才有辦法讀懂，這類持續性的利息支出正是技術債的最大特徵。

雖然債務聽起來好像不太受歡迎，然而事實上妥善運用債務不管是在軟體開發上或是公司的經營上其實都很有幫助，投資者在評估一間公司績效時不會因為一間公司完全沒有債務而認為這是一件好事，反而會被視為沒有效率，適當的舉債對於公司的擴張是有一定的幫

助，同樣的技術債對於軟體開發來說，並非完全是邪惡的存在，有經驗的工程師懂得在對的情況下，有計劃的容許技術債的存在，來加速軟體的開發，舉個例子，當你的新創公司的新產品，是否真的能在市場上存活還是一個未知數時，如果花太多時間在一開始嘗清債務反而是浪費了寶貴的開發時間，這種情況下其實可以容許技術債，甚至刻意舉一些債來加速開發，用最快的速度把產品做出來，但仍保有一定的品質，當產品經過市場的考驗之後再來償債，風險低了很多，因為如果這個產品無法存活，那麼有多少債務其實都沒有任何意義。

很多時候技術債不必償還，像是產品死亡或是交付專案給業主，有許多軟體外包合約就是很好的例子，因為外包的團隊很清楚知道，當這個專案交付給業主之後，對於團隊而言就再也沒機會維護這樣的專案，因此他們在開發這專案時並不會在意欠下多少債務，正因為不用負責，產生了道德風險，特別是對於軟體品質不要求、花費也無關痛癢的政府單位容易出現這樣的情況，很常見到政府單位的網站被入侵，也就是因為軟體品質低劣造成的，如果原有的系統需要增加功能或是修正問題，當遇到原有的合作廠商倒閉或是由另一家廠商得標時，接手的團隊就得面對這樣債務堆積如山的專案，通常有兩種選擇，一種是打掉重建，另一種是在原有的系統上再疊上更多的債務，只要能夠成功交付專案，負責的不是開發團隊，就很少會有人在乎。

對於技術債的無知或無視在台灣是很常見的錯誤，因為大多數人對於軟體開發的認知不足，在上面的論述中，其實不難發現，軟體開發有一個特性，開發最初的成本可能只佔兩成，而維護的成本會佔八成，並且隨著系統存活的時間越久而越來越多，如果當技術債台高築時，維護的成本更是雪上加霜，因此軟體開發不該被技術債所控制，而是該主動去瞭解並控制技術債。

程式碼風格

程式碼的風格和易讀性往往也是台灣的軟體產業所忽視的，乍看之下好像只是程式碼容不容易被讀懂，但實際上對於軟體的開發影響很大，因為一個事實是

> 程式碼讀比寫多次

因為當你撰寫程式碼，你就寫這麼一次，但是你每次接著修改，都會不斷的去讀先前所寫的程式碼，因為沒有人可以記住所有程式碼，更別說那段程式如果不是你寫的，對於團隊來說，當團隊的成員越多，一段程式碼需要被閱讀的次數就越多，假如因為一個成員故意或非故意寫出很難讓人理解的程式碼時，在一個五個人的團隊每個都讀一次這段程式都額外浪費十分鐘的話，那麼整個團隊就白白浪費了五十分鐘在這段難懂的程式上，隨著時間推進，這樣的浪費在技術債被還清以前都不會消失，日積月累也是相當可觀的成本浪費。

解決的方法其實不難，現在有許多工具可以自動幫你檢查程式碼是否符合特定風格，如果有成員寫出不合格的程式時，那些程式碼應該被團隊拒絕，不過比起程式碼風格的自動檢查，更重要的是團隊成員對於程式風格重要性的認知和實行，有許多項目是自動檢測無法檢查的，例如變數的命名是否有意義，工具就只能檢查規則，無法得知該變數名稱是否能

正確表達它在片段裡的用途，因此除了自動檢測，許多團隊還會增加一個複閱的流程，每有新的程式碼被寫出來，要被接受前都得經過其它成員的檢閱。

值得注意的是，程式碼的風格最好從專案的一開始就開始維持，因為沒有多少人願意去清理前人所留下來的技術債務，如果已經欠了一屁股債，後面接手的人往往會選擇忽視債務，如果真的不得已，也只能先停下腳步來把債務償清再繼續往下走。

開源篇

加法思考與減法思考

台灣是一個資源有限的島，上面擠了兩千三百餘萬人，在這樣的環境底下，減法思考是很常見的，從小到大多少會有長輩向你灌輸這類的思想

> 顧好你自己就好，別浪費時間教別人又多一個競爭對手

在學校裡一切都只是為了成績和名次，因此不是你死就是我活，在軟體開發也很常見到這類的思維，有些人會告訴你

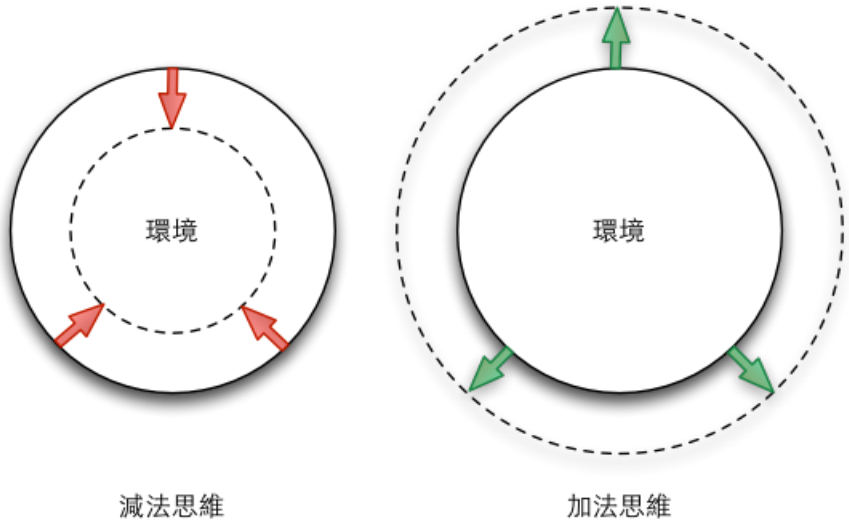
> 你要藏一手，不然別人都學走，公司就不要你了

也有人會這麼說

> 我的程式碼只有我能維護，如此一來公司就不能沒有我

諸如此類都是很典形的減法思維，在資源有限的環境下，透過各種手段去排除其它人的競爭，來確保自己的利益，確實在短期內看來或許有成效，但長期看來環境會因為有太多人抱持著這樣的思想越來越惡劣，很遺憾的，台灣目前的現況有不少人是這樣的心態，才會走到今日的地步，這些人會不停的抱怨各種環境的改變，把責任和過錯都推給他人，但是卻不願意承認是自己造成這樣的環境。

俗話說「人不為己，天誅地滅」，確實人是自私的生物，但相對於減法思維的人，加法思維的人，除了在考量自己利益同時，也會思考對整體環境有所改善的選擇，在一個班級上雖然花了時間教同學，確實花了自己的時間又讓你的競爭者實力增加，或許因此輸了排名，但是在互通有無的情況下，大家整體的能力是提升的，失去了一點短期的利益卻換來整體和長遠的提昇，同樣的，在一間軟體公司裡，當員工互相交流，整體的能力提升，對公司來說是正向的提升，確實有些不懂軟體開發的公司將人才當做用完即丟的工具，但如果因為這樣公司開除你那也只是認識到這是一間不懂軟體開發的公司，必定沒有什麼前景，也不值得久留，盡力將程式碼寫得人人都可以輕易維護，雖然多花了一點心力，但是對團隊來說是整體的生產力提升。



減法思維與加法思維

以更廣的視野來看，很多時候人們可以改變的環境不止是一個班級、一間公司，甚至可以是一個國家的整個產業，一個產業能否跟上世界潮流的關鍵，抱有加法思維的人數佔整體比例就可以看出端倪，從分享的風氣其實就可以看出個大概，不幸的是在台灣分享的風氣並不盛行，許多人還是抱持著一但分享就會增加競爭對手的心態，即使是在團隊裡也是如此，如果說一個人的所學是獨步武林的絕學，藏私也是理所當然的，但是當一個人所學不多，已經落後其它高手數十年功力，在這小小的地方有著惡劣的環境還在想著透過藏私勝過他人也未免太過可悲。

有趣的是，比較兩岸的軟體產業環境，對岸的工程師是樂於分享的，除了工程師人數基數的巨大差距以外，在那裡分享技術心得是很常見的，並且很常看見他們在文章裡寫這麼一句話：

> 拋磚引玉

在他們的思維裡，他們希望透過自己分享這樣一點點心得，引來更多人分享更多經驗，讓整體的環境有所提升，這也是為什麼中國大陸的軟體開發能力在短短十年內超越台灣的一個重要原因之一，雖然慢了別人好幾拍，但是台灣的分享風氣已有好轉的跡象，越來越多人願意寫部落格分享自己的心得，越來越多人參與開放源碼的活動，但還是都以學生和個人為主，企業的參與還是少數，因此離先進國家還是有一大段距離要走。

之所以提到加法思維，是因為開放源碼就是加法思維的最好體現，不幸的是在台灣大部份人對於開放源碼都有嚴重的錯誤觀念，作者就有曾聽過不少人以不屑的口吻表示：

> 那做不起來啦

這些人可能都抱持著減法的思維，自然無法理解開放源碼的動機，運作的方式，甚至他們不知道開放源碼早已經創造數百億產值，現在每個人的智慧型手機裡都有基於開放源碼的

軟體在裡面運行，不管是作業系統還是函式庫，也有無數的公司的商業模式是基於開放源碼在進行的，這些對於開放源碼的錯誤認知都是需要被修正，否則台灣的軟體產業永遠都會處於閉門造車的困境中無法脫離。

開放源碼的加法思維

開放源碼究竟是怎麼一回事？其實簡單來說，就是將軟體的程式碼透過一份授權，在一定特定的條件下每個人都可以無償利用、修改、散佈程式碼，許多人將軟體的程式碼視為一間軟體公司的命脈，開放程式碼這件事情聽起來很瘋狂，為什麼會有人想要平白免費讓人使用辛辛苦苦寫出來的智慧結晶？這些人其實並沒有發瘋，因為他們以加法的思維看到，透過開發源碼可以改善整個環境，或著只是某種商業的手段，端看開發者是抱持著什麼樣的心態，對大部份人來說，他們開放源碼的動機不外乎：

- 自我能力的證明
- 避免其它人重覆開發類似的程式，增進全人類的生產力
- 透過開放源碼獲利

對於一個開發者來說，證明自我實力的最好方法之一莫過於開放源碼，對一般的公司而言，一個工程師再怎麼優秀，他所寫的程式最終都是屬於公司的，而其它人都看不到這個工程師寫了什麼、他寫得如何，開放程式碼不一樣的是任何人都可以看見、利用這些程式，因此一個人所生產的程式好壞都清楚可見，如果有工程師參與或甚至主導一個流行的大型的開放源碼專案，這不言自明地證明了這些人的能力，因此他們一般都是產業裡的搶手人才，為了贏得工程師社群裡的尊重與基於對自己聲望的累積，許多工程師都願意花心力在於開放源碼的專案上，他們在為自己著想的過程中也改善了整體的環境。

軟體開發中其實有很多會重覆使用到的部份，就好比打造一台車子會用到四個輪子一樣，一間汽車工廠未必有生產輪子的能力，他們可能會向上游廠商訂製或是購買現成的輪胎來組裝，以減低生產的成本或風險，同樣的在軟體開發的世界也是如此，這也是為何會有：

> 不要重覆打造輪子

這句名言，如果每個軟體工程師都為同樣一個輪子多花了一個工作天，那麼全世界假設有一百萬個工程師都重造了一個輪子，那整體浪費掉的生產力就是一百萬個工作天，正因為同樣功能的程式，在許多地方都派得上用場，具有加法思維的工程師在考慮到除了滿足自己的需求以外，將這些會重覆利用到的部份以開放源碼的形式分享讓大家使用，如此一來就省掉了全世界工程師們將會浪費掉的時間，或許讀者會質疑，這麼做可能會讓以同性質軟體賣不出去，事實上確實是如此，但是因為這樣，軟體產業的巨輪往前推進了一大步，除此之外也創造了新的商業機會，舉例來說，MySQL 做為開放源碼的資料庫，打敗了其它類似的收費資料庫軟體，但是它讓原本無法負擔昂貴資料庫授權費用的開發者可以省下這筆開支，基於資料庫系統的開發進而創造更多的價值，同時也因為有無數人使用這套資料庫系統，它也創造了技術諮詢服務的需求和其它各種不同的商業模式。

簡而言之，開放源碼是一種基於利人利己的個人職涯或是商業的經營策略，並非只是許多人所認為的吃飽沒事做或是慈善事業，軟體產業之所以能夠在短短的數十年內改變世界原有產業的樣貌，正是因為聰明的工程師懂得使用加法的思維，在考慮到自身好處以外還能兼顧到整體環境的改進，他就像是一股巨浪，推著整個產業前進，把不願意乘浪抱持古老思想的人遠遠甩在後頭，對岸早就駕馭開源，甚至做出貢獻，淘寶就有釋出一系列開源專案[3]，台灣軟體產業如果想跟上世界腳步，必然得乘上這波巨浪。

[3] http://code.taobao.org

經營篇

沒有創投

對一間新的軟體或網路公司來說，從創立到開始獲利需要不少的資本投入，在初期最沉重的就是人事成本，想要留住好的工程師自然不可能以台灣常見的 22K 等級薪水打發，除了人事費用，辦公室、水、電和其它行政費用，也會隨著團隊的規模成長，因此一間新創公司在初期往往會需要募資，管道有很多種，從自己的積蓄、親朋好友到風險投資公司，其中最關鍵的可能要算是風險投資公司，然而不幸的是在台灣所謂的風險投資公司或是創業投資公司，所做的事情大多與歐美的風投公司不一樣，在台灣一般的這類的投資公司都會要求你的公司要有辦法獲利他們才肯借錢給你，說穿了這類業務其實就只是貸款，而矛盾的地方在於，如果已經能夠自己自足，有穩定成長的獲利，在不尋求積極擴張的情況下，需要這些公司的資金做什麼呢? 除此之外，一間新創的網路公司要成功，並非只要有資金就可以成功，要知道網路創業所做的事情許多都是從前沒有任何人做過的事情，並非只是另外的一間工廠重覆著以往成功的模式用更便宜的成本生產更好的品質這種形式的創新，成敗與否在早期都是難以預測的，新創公司在摸索階段除了資金也很需要來自創投身為夥伴關係的各種資源幫助，不管是意見方面的諮詢、人脈的幫助、產業的合作、幫忙尋找下一輪的投資、尋找人才等等，在台灣未曾有過真正的創投與網路公司成功案例，未曾嘗過如矽谷那樣獲利百倍、千倍的投資甜頭、又沒有相關的經驗，在這方面理所當然也只能提供貸款的服務，除了環境先天上的不足，政府法規的落後和對新創網路產業的無知，更使得這個地方成為一片荒漠 .

因為先天環境上的限制，如果在台灣要創立新的網路公司，除非能有其它的管道找到足夠資金，否則應該盡量在早期能以類似「精實創業」的方式自給自足，在台灣要先做出成功的產品再來找資金，相較於矽谷或中國大陸是非常困難的，很容易在半路上就會資金用罄，創投投資的其實是一個團隊而不是一間公司，先建立能自給自足的堅强團隊是首要目標，許多團隊在初期甚至靠接案子維生，雖然失去了一些專注在產品上的時間，但是在這樣的環境下也只能如此。

體驗篇

軟體是服務業

軟體產業雖然頂著高科技的光環，但是比起硬體產業，它其實更像是服務業，軟體和網路服務的目的最主要都是服務人群，不同的是，一間飯店、餐廳能服務的最多不過幾百人，但是軟體服務能夠由少數幾個工程師來服務來自全世界各地數千萬甚至上億人，這也是為什麼軟體工程師在歐美國家能有這麼高的待遇，在台灣軟體常常只被視為是解決問題的工具，其產生的結果只有可行和不可行，這其實是台灣的軟體產業失敗的眾多原因之一，一個軟體服務的好壞其實是有很大的差異，並非只有二元的結果，對於消費者導向的服務而言，最重要的莫過於使用者體驗，使用一項軟體服務就像是去一家餐廳吃飯一樣，並非只有吃飽和吃不飽兩種感覺，一家餐廳從門口、招牌、店名開始就讓客人開始體驗，體驗的好壞從菜色的好壞、服務的品質到各種細節都會影響到客人最終的評價，軟體其實也是一樣的道理，一個手機 App 從商店的陳列給的第一個印象、標價、描述，到下載、學習和開始使用，不管是使用者介面，和軟體的穩定性、功能性，每一樣都讓使用者體驗，很多時候賣給使用者的並不是一個工具，而是一個體驗，如果兩個軟體服務都能滿足需求，使用者自然是選擇體驗較好的那個，最重要的反而不是功能强不强大等等無關緊要的問題。

服務業在台灣發展得很好，不幸的是，在台灣軟體設計出來的體驗往往都是很糟糕的，在使用的過程中都可以感受到設計者對於使用者的不在乎與不尊重，舉例來說從一個網站使用者的註冊開始就可以感受到濃烈的敵意，在台灣設計者往往為了市場調查等等需求，喜歡在註冊時要求使用者輸入一大堆無關緊要的資料，看過最誇張的是使用者必需輸入血型才能完成註冊，這就好比進如一家餐廳前還得經過身家調查一樣離譜，除此之外也很常見到許多電子商務網站的設計停留在數年前從未有任何改進，甚至有些搜尋功能形同虛設，同樣的商品標題站內搜尋找不到反而要到 Google 才找得到的誇張情況，許多網站在使用者人數過多時服務會停擺也都已經是家常便飯，數年過去同樣的問題一再發生也未見任何改進，這些都顯示出業者對於使用者的體驗毫不在乎，就好像方圓百里內就此一間客棧似的。

在台灣軟體之所以不重視體驗，最主要是因為現有的產業幾乎都是以業務為導向，不管是電子商務，軟體開發承包商，對於這些服務來說很多都是在低度競爭或壟斷的環境下所生存，外包的專案完成後，不管好用與否，都與成敗無關，自然沒有人會在意使用者的體驗，在桃花源的太平時代確實無所謂，然而當全球化的競爭越來越激烈，國外的市場都開發成熟時，這些人進軍時很可能就是不求改進者的末日。