前言

非常感謝您選擇了《**語意軟體設計**》，我們竭誠地歡迎您。

本書介紹了一種新的軟體設計方法，為如何建構軟體提出一種新的思路。它主要著重於大型專案，尤其是對於新開發的軟體專案或大型老舊系統的現代化專案特別有用。

如果一個軟體專案不符合預算或時間表，或者無法以可用的方式交付所承諾的功能，那麼該專案就被算是失敗的。軟體專案失敗的比率相當驚人，無庸置疑也有據可查。在過去的 20 年裡，這種情況不但沒有改善，而且變得更糟。我們必須做點不同的事情來讓我們的軟體設計更為成功，但是該做些什麼？

我在這裡的假設是，你正在製作要出售給客戶的商務應用軟體和服務，或者是在公司內部的 IT 部門工作。本書與導彈制導系統、電話、或硬體無關，儘管它可以應用於物件導向和函數程式設計，但是對這兩個領域的爭論也不感興趣，當然也不想討論某些流行的框架或其他任何框架。為了清晰起見，我在這裡所用的「語意」一詞可以追溯到我的哲學訓練，因此也涉及到**符號**的問題。這裡的「語意」主要指的是**符號學**（*semiology*），與伯納斯·李（Tim BernersLee）所受人尊崇的語意網概念無關。

本書主要受眾是技術總監（CTO）、資訊長（CIO）、工程副總裁、各行各業的架構師（無論是企業、應用程式、解決方案還是其他機構）、軟體開發經理、和希望成為架構師的資深開發人員。任何技術人員，包括測試人員、分析師和高階主管，都可以從本書中受益。

但書中幾乎沒有什麼寶貴的程式碼，如果有的話也只是為了被管理者、領導者、對知識感興趣的主管、和任何正在進行軟體專案的人所理解和接受而編寫的，而這並不是件很**容易**的事。

碁峰 www.gotop.com.tw

本書中的觀點有時可能令人震驚，也可能惹惱一些人，甚至會激怒其他人。這些想法將以新穎的形式出現，在某些情況下甚至會顯得陌生和奇怪；在其他情況下（例如在介紹「設計思維」時），這些想法將作為借鑒並加以重新定義。總之，這是我多年來將各種不同來源拼湊起來所訂做出來的方法，這些想法大多來自於我在研究所的哲學研究。本書代表了經過考驗的概念、流程、做法、範本、和落實方式，我把它們統稱為「語意設計」。

這種軟體設計方法已經被證明是可行的。過去 20 年來，我有幸在大型跨國公司擔任 CTO、CIO、首席架構師等工作，並且設計和領導了許多大型、關鍵任務軟體專案的創建，多次贏得創新獎項，更重要的是，創造了成功的軟體。這裡所介紹的概念在某種意義上構成了我如何處理和執行軟體設計的一覽表。我採用這個方法已經超過十年了，帶領了花費 $100 萬美元、$1,000 萬美元、$3,500 萬美元、和 $5,000 萬美元的軟體設計專案。儘管這可能看起來與傳統的軟體設計思維方式大相徑庭，但它不是猜測或理論：即使並不是那麼顯而易見，但是它一再被證明是行之有效的。

我們被迫使用我們所承襲的語言，我們知道自己的名字，只是因為有人告訴我們自己的名字是什麼。出於一些顯而易見的原因，在本書中，我有時會將「架構設計師」簡稱為「架構師」，這表示我不得不為了清晰起見或是歷史的目的而使用某些詞彙來進行溝通，但當下的前後文中，並非我想要表達的本意。

本書的第一部分介紹了這種方法的哲學框架，重點在於說明我們要解決什麼問題以及為什麼。這一部分是概念性的，並提供了理論依據。

本書的第二部分是無情的實用主義，它提供了一系列文件範本和可重複的做法，你可以立即在日常工作中採用此方法的元素。

第三部分概述了管理和治理軟體產品組合的方法，以幫忙控制整體系統中失序的現象。本書的結尾是一份宣言，簡要總結了囊括這種方法的一系列原則和做法。

總的來說，這本書代表了一個綜合的理論框架及其實踐手段。然而，它並不是封閉的，而是打算作為一個能夠加以闡述和改進的起點。

這本書是我出自愛好而寫的，真心希望您喜歡它，並發現將其應用於自己的工作中很管用。

此外，我想邀請您對這些想法作出貢獻並加以改進，能收到您的來信是我的榮幸，來信請寄到 *eben@aletheastudio.com* 或 *AletheaStudio.com*。

碁峰 www.gotop.com.tw

第一部分

初探：設計的哲學

萬物擁有一切的一部分。

—阿那克薩哥拉斯（Anaxagoras）

在這一部分，我們將探討設計本身的意義，我們將以全新的視角審視軟體設計這項工作是如何形成的，並對業界的一些觀點提出質疑。我們將架構重新想像為創造概念的工作，並瞭解如何與團隊一起表達這些概念以創作出能產生預期結果的軟體設計。

碁峰 www.gotop.com.tw

第一章

軟體架構的起源

我們大多數人都被知識論所支配，而這些知識論往往是錯誤的。

—葛列格里·貝特森（Gregory Bateson）

這本書的目的是要幫助你設計出好的系統，並且幫助你在實務中將設計實現出來。這本書非常實用，而為了讓你把工作做得更好，我們必須從理論和歷史的角度出發。本章的目的是向你介紹一種以軟體架構師的角色來思考的新方法，這將影響到本書其餘的部分以及你著手推動專案的方式。

軟體概念的起源

我們塑造了建築物，而後建築物則塑造了我們

—溫斯頓·邱吉爾（Winston Churchill）

畫面淡入：

內景：1968 年 10 月某日，位於德國加爾米施的一個會議廳

場景：北約軟體工程研討會。

50 位來自各國的電腦教授和工匠聚在一起，以決定軟體行業的現狀。在研討會名稱中故意選用「軟體工程」一詞以表達其「挑釁」的意圖，因為當時的軟體製造商並不被認為是在從事一項科學研究的工作，稱自己為「工程師」必將破壞現有既定的認知。

碁峰 www.gotop.com.tw

麥克羅伊（MCILROY）

毫無疑問，我們在與硬體行業的對抗中明顯處於劣勢，因為他們是實業家，而我們只是小農場中的佃農。
（停頓）
作為一個產業，軟體的創造是落後的。

科倫斯（KOLENCE）

完全同意。在對程式設計過程有更全面的瞭解之前，程式設計管理在成本和進度效率方面的壞口碑將持續下去。

雖然這些對話是發生在 1968 年，並且記錄在當時的會議記錄中（*http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF*），但如果換成是今天的話，也很少人會認為它們不合時宜。

在該次會議上所獲得的共識是軟體的製造必須遵循**工業化**的流程。

這似乎很自然，因為他們主要擔心的問題之一是，當把軟體從硬體中分離出來之後，很難獨自定義為一個領域。當時，研討會上最具**煽動性**、最**危言聳聽**的話題是：「軟體是否應該與硬體分開定價」這一極富爭議性的問題，而這個主題則延燒了為期四天的會議。

這只是在說明，僅僅在 50 年前，軟體甚至不知道自己可以獨立存在於硬體之外的領域。這個領域中非常聰明、有成就的專業人士甚至不確定軟體是否為具有任何獨立價值的「東西」，讓我們先花點時間來理解一下其中的含義。

軟體誕生於硬體之母，幾十年來，這兩者（確實）融合在一起，很難將其視為獨立的事物。其中一個原因是，當時的軟體「被視為沒有經濟價值」，因為它只是硬體必備的附屬品，純粹為了滿足令人神往的慾望而存在。

然而，今天你花 100 美元就能買到一台桌上型電腦，它的功能比 1968 年世界上任何一台電腦都要強大。（在當時北約會議期間，一台 16 位元的電腦，以今天的物價計算大約要花費 6 萬美元。）

而且，硬體是在**工廠生產線**上以清楚、可重複的過程生產，絕對可生產出幾十個、幾千個、幾百萬個相同的實體物件。

硬體是一種商品。

碁峰 www.gotop.com.tw

商品是可以與同類型商品互相交換的東西，相同的商務電子郵件或者製作文字處理檔案可以在 50 家不同製造商的筆記型電腦上輸入。

而商業界人士除了他們各自有不同的「秘密配方」之外，所追求的不外乎就是商品的效率。可口可樂（*HTTP://bit.ly/2mlnZOY*）在全球有將近 1,000 家的工廠都在進行重複製造，每天以同樣的方式將可樂放入瓶子、罐頭、和袋子中裝貨和載運數千次。這是一項經過嚴格審查、嚴格評估的業務：一種內部商品。可樂以相同的方式在工廠中以相同的方式裝在相同的瓶子裡，每天數百萬次。然而，只有少數人知道這種飲料本身的秘密**配方**。可樂每天被複製數百萬次，然後用相同的工藝裝瓶，然而，一但將這個配方當作商品來販售，將會使可口可樂倒店。

在軟體發展的嬰兒期，軟體人員沒有意識到代表可重複的、製造業流程式的商品和更神秘的、創新的、**一次性**配方的製作工藝之間的區別。

可樂就是配方，其生產線就是這個工廠。而軟體就是配方，其生產線則發生在執行時的流覽器中，而不是在程式設計師的小隔間中。

我們的概念起源於硬體和工廠生產線，並借鑒了建築學結構。這些概念的起源混淆了我們，並以非最佳的、非必要的方式支配和限制了我們的思維，而這也是專案業績如此慘澹的主要原因。

軟體中所使用的「架構師」（architect）一詞直到 1990 年代初期才開始流行，1968 年在德國召開的北約軟體工程研討會上，彼得·諾爾（Peter Naur）提出了軟體從業者可以向建築師學習的第一個建議：

> **軟體設計師與建築師和土木工程師的處境類似**，尤其是那些與大型異構建築（如城鎮和工廠）設計有關的人，**因此我們很自然應該向這些課題尋求如何解決設計問題的想法**。我要以克裡斯多夫·亞歷山大的《形式綜合註釋》（哈佛大學出版社，1964 年）作為這種思想淵源的一個例子。

這一點，以及我們這個領域的資深規則制定者在 1968 年的會議上所發表的其他聲明，是我們認為應該如何思考軟體設計定位的先驅。諾爾的說法有一個明顯的問題：這完全是錯誤的，而且也沒有事實根據。

說我們處於「與建築師相似的處境」，無論在邏輯上或事實上，與說我們處於類似於哲學教授、作家、飛行員、官員、橄欖球運動員、兔子或小馬等的處境一樣，都沒有任何關係。類推論證總是錯誤的，更何況這裡甚至沒有提出任何論證。然而，這個想法於此站穩了腳跟，參與者回到他們在世界各地的故土，幾十年來從事寫作、教學和輔導，

碁峰 www.gotop.com.tw

塑造了我們整個領域。現在，它縈繞在我們心頭，悄無聲息地影響著我們如何開展工作、如何思考工作、如何「知道」自己在做什麼，甚至可能人為地塑造和限制了我們的思維。

起源

很明顯，1968 年北約研討會的與會者都是非常聰明、有成就的人，他們在尋找一種方式來談論一個幾乎還不存在、正在形成並準備將自己公諸於世的領域。這是一項艱巨的任務，我對他們表示崇高的敬意，他們創造了像 ALGOL60 這樣的程式語言，贏得了圖靈獎，並創造了符號。他們使我們的未來成為可能，對此我心懷感激和敬畏。在這裡之所以提到該研討會只是為了要瞭解我們的起源，希望藉此能改善我們的未來，而我們都站在巨人的肩膀上。

幾年後，「四人幫」（*http://bit.ly/2mp16ua*）在 1994 年所寫的《**設計樣式**》（*Design Patterns*）一書中明確地引用加州大學伯克萊分校建築學教授克里斯多夫·亞歷山大（Christopher Alexander）的作品作為靈感來源，克里斯多夫·亞歷山大所著的《**樣式語言**》（*A Pattern Language*）被公認是關於城鎮、公共空間、建築和住宅設計的權威著作。**設計樣式**一書是一本關鍵性的作品，它推動了軟體設計領域的發展，並支持了**軟體設計師即架構師**或「類似」架構師這一新興概念，我們應該汲取這一領域的概念並藉此萌生自己的觀點、方法和新的點子。

出席這次北約研討會的還有當今著名的荷蘭系統科學家艾茲赫爾·迪克斯特拉（Edsger Dijkstra）（*http://bit.ly/2lW5UXM*），他是現代計算技術領域最重要的思想家之一。Dijkstra 參與了這些對話，幾年後，他在德州大學奧斯汀分校資訊科學系擔任系主任期間，表示強烈反對軟體機械化，駁斥了在資訊科學中使用「軟體工程」一詞的正當性，並認為這就好比將手術稱作「刀子科學」一樣。相反地，他總結道：「資訊科學的核心挑戰是一個概念性的問題；也就是說，**我們可以構想出哪些（抽象的）機制**，而不會迷失在自己所製造的複雜性中。」

在同一次研討會上，第一次有人提出軟體需要「電腦工程師」的建議，但這對許多參與者來說是一個令人尷尬的概念，因為工程師做的是「正經八百」工作，具有一定的紀律和已知的功能，而軟體從業人員相較之下則是一群烏合之眾。

「軟體屬於思想的世界，就像音樂和數學一樣，應該得到相應的對待。」這個很有意思，但我們先就此打住。

碁峰 www.gotop.com.tw

* * *

跳切：

內景：某日於波蘭華沙的總統辦公室

場景：波蘭共和國總統更新了稅法。

在波蘭，軟體開發人員被歸類為創意藝術家，並因此可以享受高達 50% 的政府稅收減免（參見德勤報告（*http://bit.ly/2ko2zAa*）），以下是在波蘭被歸類為創意藝術家的職業：

- 建築設計
- 室內和景觀
- 城市規劃
- 電腦軟體
- 小說和詩歌
- 繪畫和雕塑
- 音樂、指揮、唱歌、演奏樂器和編舞
- 小提琴製作
- 民間藝術和新聞
- 表演、導演、服裝設計、舞臺設計
- 舞蹈和馬戲團雜技

以上每一項都明確地列在成文的法律中。在波蘭政府看來，軟體開發與詩歌、指揮、舞蹈和民間藝術屬於同一專業範疇。

而波蘭是世界上主要的軟體生產國之一。

跳切：現今，這裡。

也許在結構概念的歷史上發生了一些事情，或可稱之為一個事件，導致了一連串的決裂。

這種決裂不會在一個單獨的爆炸性時刻、一個令人感到舒適的位置、和一個適當而具有戲劇性的時刻出現。

碁峰 www.gotop.com.tw

隨著時間的推移，它會在思想和表達的海洋潮汐中出現，穿越宇宙，時而退潮，時而湧動，帶著憤怒，也帶著慵懶，直到痕跡的緩慢流淌和不同思維模式產生了結合，之後，一些東西才發生了變化。最終，這些痕跡硬化成了溝壑，也固定了思想，從而固定了表達和實現。

這種分類所闡明的是，語言的浪潮是一種塑造我們的思想、對話、理解、方法、手段、倫理、模式和設計實踐的方言。我們給事物命名，然後它們反過來塑造了我們。它們限制了我們的思維模式，從而影響了我們的工作。

因此，一個領域內結構的概念，如同我們在科技領域中所稱的「架構」，必先以語言的物件呈現。

我們的語言是由符號和隱喻的相互作用所構成，隱喻是一種詩意的手段，藉由隱喻來稱呼某些事物，並不是為了透過強調、突顯、或抵消某些屬性來揭露關於該物件更深層或隱藏的真相，莎士比亞的《皆大歡喜》（*As You Like It*）中的名言「世界是一座舞臺，所有男男女女都只是演員而已」即是一句著名的隱喻。

我們如此頻繁地隨意使用隱喻，以至於有時甚至忘了它們是隱喻。當這種情況發生時，則隱喻本身從此「消亡」，並成為名稱本身，抽離其最初並置的意義，從而使得該詞組具有了深度，我們稱這些為「消亡隱喻」。常見的消亡隱喻包括椅子的「腳」，或者當我們「墜入」愛河，或者當我們說時間「快沒了」。我們在日常談話中說這些話時，就像沙漏裡的沙子一樣，並不會以為自己是在作隱喻的詩人，我們並沒有注意到這是個隱喻，也不打算這麼做，而當下所指的只是該物件而已。

在技術領域裡，「架構」一詞是個不必要的隱喻，受到這個詞的拖累，我們的注意力被引向我們工作的某些方面。

建構是個消亡隱喻：我們把這個隱喻錯當成了事實。

幾十年來，關於把「架構師」一詞應用在科技術領域一直存在相當激烈的爭論，其中包含了硬體架構、應用程式架構、資訊架構等等。如果我們連自己所指的架構是什麼都搞不清楚，那麼我們還能說架構是一個消亡隱喻嗎？我們在使用這個術語時，並沒有完全理解它的含義；架構師的流程是什麼？以及他們所製作出來的文件具有什麼樣的價值？「架構師」一詞其實是源自希臘語中「建築大師」的意思。

這又有什麼區別嗎？

碁峰 www.gotop.com.tw

複製與創造力

> 不是偉大的雕塑家或畫家不能成為建築師。如果他不是雕塑家或畫家，那麼他只能是個建築工人。
>
> —約翰·羅斯金（John Ruskin）《**真與美**》（*True and Beautiful*）

在流程中將角色劃分為不同的職責是一種在企業中建立產能非常有用且非常流行的方法。這樣的劃分讓流程中每一刻的價值，對整體的每一份貢獻，更加直接和清晰。這種「勞動分工」的工作方式，使得每個步驟都具有可觀察和可測量的附加價值。

反過來說，這又為我們提供了根據 SMART 原則（*https://en.wikipedia.org/wiki/SMART_criteria*）來陳述這些目標，從而獎勵、懲罰、升遷、和解雇那些無法達到客觀衡量標準的人。這至少在某種程度上要歸功於亨利·福特（Henry Ford），他在 100 多年前設計了自己的汽車製造廠。他的具體目標是讓他生產的汽車便宜到可以賣給製造這些汽車的低工資工人，以確保他在消耗了原料之後所無法保有的純利，至少他所支付的有償勞動力能夠以收入的形式歸還給他。

不過這種建立產能的方式，只有當所製造出來的產品定義明確，而且可以讓你製造許多（幾十、幾千或幾百萬）個相同產品的副本時，才能發揮最大的效用。

在**精實六標準差**（*Lean Six Sigma*）的管理方法中，流程經過不斷的精粹化，直到失敗率降低到平均值的六個標準差以內為止，這樣的生產流程每一百萬個產品只允許出現 3.4 個品質瑕疵的機會。我們試圖定義我們的領域，找到合適的名稱，以便編纂和制定可重複的流程，來提高我們作為工人的幸福感（令人覬覦的「角色明確性」），並提高產品的品質。

但有人一定會問，我們的名字對我們有什麼用？

流程的存在是為了**建立拷貝**，但我們是否曾經建立過軟體本身的拷貝？事實上，**我們從未這樣做過**。當然，建立軟體拷貝的目的是為了分發：我們過去常常把網路流覽器的拷貝燒錄到光碟上，然後透過郵件發送出去，而現在我們則是透過互聯網分發軟體的拷貝。然而，這只是一個便於發佈的過程，與最初建立該軟體應用程式的行為幾乎沒有關係。

碁峰 www.gotop.com.tw

流程就算不是為了重複做同樣的事情而存在，至少也是為了重複同**類型**的事。在軟體開發方法論中通常會把要完成的工作進行分類，而軟體開發部門將我們在建立軟體產品或系統時所經歷的過程進進行劃分並建立（通常是模糊的）概念。因此，為了生產某種類型的軟體，我們定義了參與某一部分流程的角色，

這些角色有可能會被正式地描述、傳達，並隨之被執行。

大型組織為了讓預期的季度計畫結果能夠被衡量，因此制定出我們特有的流程，但是這卻阻礙了科技領域中的競爭和**創新**這兩個前提，導致難以採用最好的方法來完成任務。為了在市場上競爭和取勝，我們必須創新，製造出新穎而引人注目的產品。因此，我們刻板的明確目標是不再生產以前已經生產過的東西。然而，嵌入式的語言卻促使我們去選擇那些可能不是最有效的流程並扮演伴隨而來的角色。

這樣的發明意味著相當大的不確定性，這與福特主義支持者對於可重複和可測量流程的熱愛產生了矛盾。而軟體本身的創造在地球上只存在了幾十年，為了提高成功的機會，我們來看看其他已經完全上軌道的領域是如何運作的。我們已經接受了「工程師」和「架構師」這些從建築領域借用來的術語，為我們自己的流程提供了明確的規格導向視角。我們創造了工作崗位來封裝他們的職責，但是透過軟體的鏡頭，在過去的幾十年裡，我們雇傭了大批有這樣頭銜的人，並對他們寄予厚望。

近來科技領域把目光轉向了一種更為古老的研究模式，這種模式因其精確性和可重複性而備受推崇：即科學本身。我們現在有了資料「科學家」這個稱謂，儘管「電腦科學家」可能是歷史最悠久的名詞，但是除了從事研究的教授以外，沒有人從事過一個叫做「電腦科學家」的工作，因為他們的領域往往完全停留在理論領域。

軟體設計並不是科學。

我們的流程也不應該假裝是我們所沒有，也不想要的工廠模式。

這樣錯誤的分類無形中削弱了我們的工作效率。

軟體架構失敗的原因

正如我在本章前面所提到的，軟體專案的高失敗率令人錯愕：

- IBM 在 2008 年的報告中（*https://ibm.co/2kQHGxP*） 指出 60% 的 IT 專案會失敗，ZDNet 在 2009 年則報導了（*https://zd.net/2m28U4B*）軟體專案的失敗率超過 68%。

碁峰 www.gotop.com.tw

- 到了 2018 年，根據《資訊時代》（*http://bit.ly/2mncfeX*）的報導，這個數字已經惡化，71% 的軟體專案被認為是失敗的。
- 勤業眾信（Deloitte）形容我們的失敗率是「駭人聽聞」的，它警告（*http://bit.ly/2lVXrDQ*），75% 的企業資源規劃項目是失敗的，而行銷顧問公司 Smart Insights 更指出（*http://bit.ly/2mgqn9E*），84% 的數位轉型專案是失敗的。2017 年的《科技共和》（Tech Republic）報導稱（*https://tek.io/2XbQgZ0*），大數據專案的失敗率高達 85%。
- 根據麥肯錫（McKinsey）（*https://mck.co/2kQnVq0*）的資料顯示，有 17% 的 IT 專案執行情況非常糟糕，以至於威脅到公司的生存。

像這樣的數字把我們的成功率排在比氣象學家和算命先生還差的位置。

整體而言，所有的專案都做得不夠好，一想到到世界上有那麼多的軟體在運行，對於我們的客戶、我們從業人員、以及那些依賴於我們的人來說，這個狀況不禁令人擔憂。

在過去的 20 年裡，這種情況不但沒有好轉，反而變得更糟。

麥肯錫對 5,600 家公司進行的一項研究發現（*https://mck.co/2knAHvZ*）：

> 平均而言，大型 IT 專案的經費超出預算的 45%，超過時間的預算為 7%，而交付的價值卻比預期的少了 56%。軟體專案運行成本超支和進度落後的風險最高。

當然，有些專案是按時並且是在預算之內完成的，但這些專案很可能是麥肯錫研究報告中所提到的「IT」專案，包括資料中心遷移、平移（lift-and-shift）專案、災難復原站點的部署等，這些都是複雜的（不過大多只是很「大」的）專案。當然，我也參與過幾個這樣的專案，它們不同於試圖構思出一個新的軟體系統，而這個系統才是設計的目標。

最明顯的區別是，「IT」專案所要處理的是具體的有形器材和物品，例如要移動的伺服器以及要安裝的電纜。無論是當你完成的時候或是還剩下多少比率的電纜等著安裝，都是清晰可見的。也就是說，許多後台辦公室類型的 CIO 專案並不比搬家或在倉庫裝卸包裹更有創造性：只需列出正確的清單，並在適當的時機勾選即可。

執行起來風險最大、損失最慘重的是 CTO 的軟體專案，這是因為它們需要最具有創意的概念性工作，以抽象化的方式表達出真實的世界，這牽涉到符號、語言、事物的意義、以及事物之間的關係。當你在設計這些專案時，其實是根據你所知的認識論（epistemology）在陳述一個哲學觀點。

碁峰 www.gotop.com.tw

你是在創造一個環境，在這個環境中，你可以把符號放在一起讓它們變得有意義，形成對現實世界的一種表像。

這比 IT 專案要困難得**太多**了。

當我們甚至沒有意識到，我們正在做的事情是在陳述一個哲學觀點時，情況就變得更糟了，因為我們所處的是一個語意空間，而不是實體的建築空間，因此無法認出軟體專案是什麼類型的專案。

麥肯錫的研究將 IT 專案劃分為好像它們都是一樣的，因為它們都是「跟電腦有關的東西」（我推測）。如果麥肯錫考慮得更週到，並將這些 IT 專案與設施管理歸為一類，結果將會大不相同。軟體產品的創建是一個**完全不同的問題**，而不是「IT」的一部分，就像你不會把在租來的會議室中進行的設計會議，當成是辦公空間設施公司的一部分是一樣的道理。

但創造性的工作不一定總是失敗的，大量的電影、表演、戲劇作品、音樂演奏和唱片都是按時而且按照預算製作完成的。不同的是，我們已經事先認識到這些都是創造性的工作，並**以此來管理它們**。我們以為自己正在從事「刀子科學」、「資訊科學」或「建築學」，其實不然。我們所從事的是**語意學**（*semantics*）：創造一個由複雜的符號概念所組成的結構，其含意、價值、和存在都是純粹的邏輯和語言。

前提是假定從執行發起人到專案團隊的每個人都公平合理地瞭解到專案的需求、時間、範圍、預算、和截止期限，並提供他們的意見。事實上我們都知道他們並沒有這麼做，即使他們做到了，充其量也只是在猜測，因為他們所做的事情以前從來沒有人做過，它可能是無止境的，因為世界在變化，而世界是命題的無限種組合，你想在哪裡畫出這條底線？所謂的「失敗」到底要怎麼界定？

因為軟體本質上是語意的，所以不是軟體開發人員好像就不太相信軟體的**存在**，這些人包括對沖基金經理、高階主管、和那些有 MBA 學位才能從事的工作，他們習慣於在試算表上把資料搬來搬去，偶爾也會發表一些勵志的演講，不過他們的工作卻不需要**製造出任何東西**。

軟體專案經常因為缺乏良好的管理而失敗。

團隊從一開始就知道專案不可能按時交付，他們想要取悅別人，他們擔心管理階層會輕易相信其他人的謊言，宣稱專案可以在交付日期兌現。

碁峰 www.gotop.com.tw

身為技術部門負責人份內的工作，就是想辦法停止這種思維模式，並與管理階層進行健康、艱難的對話，以便預先設定想要達成目標。即使目前具體是什麼還不清楚，也可確保他們在六個月內能看到**一部分**軟體。當聰明的、有戰略眼光的、支持你的主管們瞭解到這是一筆交易，並且對你所推展的業務有信心，那麼軟體專案就會成功。當貪婪、無知的高層主管所擔心的是失去一筆交易或自己被解雇，而規定了一個不可能完成的最後期限和極大的應交付範圍時，你必須勇敢的拒絕，因為這正是造成波音 737 Max（*https://nyti.ms/2koeYun*）軟體失效的部分原因。

麥肯錫的研究繼續闡述了發現這些問題的原因：

- 目標不明確
- 缺乏業務重點
- 需求改變
- 技術複雜程度
- 團隊無法協調
- 技能不足
- 不切實際的時間表
- 等到出問題才被動改變規劃

這些就是軟體專案失敗的原因。

如果我們能解決其中一半的問題，就能大大提高軟體專案的成功率。事實上，當我們專注於語意關係，專注於我們所設計的概念，並且把重點轉移到將軟體所代表的世界觀理論化時，我們的系統就會變得更好。

在這些原因中，前**五個**可以透過聚焦於**概念**來解決：軟體的概念、軟體的用途、以及清晰和真實的表示所代表的世界；其餘的三個都只是老式的管理不善。

碁峰 www.gotop.com.tw

失敗的影響

因此，也許現在我們可以說，我們想要達成的目標、身為技術專家的處境、以及如何概念化和處理我們的工作之間存在著分歧。這種不一致和對立的情況並非個案，而且已經顯現在沿著瓷器表面出現的裂縫中。

但是認定一個軟體專案失敗的標準到底是什麼？儘管度量標準各不相同，但一般來說，這些指標反映了預算和擬議的時程規劃是否過度超支，以及最終所產生的軟體是否能達到預期的功能。當然，並沒有純粹「失敗」和純粹「成功」的專案，但無法符合這三個標準意味著沒有達到令人滿意的期望和承諾。

而且，即使專案已經完成（不管是否失敗），如果一些軟體已經因為它而被發佈，最終所得到的軟體也不會總是達標。《**科技共和**》援引的一項研究顯示，僅 2017 年一年，軟體失效就「影響了 36 億人，造成 1.7 萬億美元的經濟損失，累計停機時間長達 268 年」。

更糟糕的是，其中一些後果更為嚴重。蓋洛普（Gallup）的一項研究（*http://bit.ly/2moAk56*）顯示，FBI 的虛擬案例檔案軟體應用程式「花費了美國納稅人 1 億美元，卻給 FBI 留下了一個過時的系統，並危害到 FBI 在反恐方面所做的努力。」；2011 年《**哈佛商業評論**》（*Harvard Business Review*, HBR）（*http://bit.ly/2kooQ0R*）的一篇文章指出，IT 專案的失敗讓美國每年在經濟上付出了高達 1,500 億美元的代價。

《**哈佛商業評論**》的這篇文章還講述了 2008 年 Levi's 公司一個 IT 專案的故事。當時的計畫是使用 SAP（一家在科技領域處於領先地位的知名的供應商）和 Deloitte（在其領域內享有盛譽的領先者）來執行，這是一個典型的專案，有知名廠商名聲的加持，卻沒有任何創新，估計花費 500 萬美元。它很快變成了一個 2 億美元的龐大噩夢，致使該公司不得不承受 1.93 億美元的收益損失，也逼得 CIO 被迫辭職下台。

當然，與前美國總統歐巴馬在 2013 年所說「徹頭徹尾災難」的政府健康照護（HealthCare.gov）專案相比（*https:// en.wikipedia.org/wiki/HealthCare.gov*），這筆錢可說是微不足道。該專案的原始成本預算為 9,300 萬美元，很快就爆炸式的成長了 18 倍，成為 17 億美元，而該網站的設計非常糟糕，預期應能同時處理 250,000 個使用者登入的流量，結果實際的負載卻只能夠容許 1,100 個使用者同時登入。

如果硬要從所有這些歷史中找出一個根本的原因都將過於武斷：因為造成軟體失敗的原因包括了領導、管理、流程設計、專案管理、變更管理、需求收集、需求表達、規格說明、理解、估計、設計、測試、傾聽、勇氣，以及原始程式碼的撰寫等。

碁峰 www.gotop.com.tw

在所有這些日益嚴重的失敗中，架構和敏捷設計的英雄們在哪裡？

軟體這個行業在方法、工具和實踐方面的集體努力並沒有改善我們的情況：事實上，它只會變得更糟。我們基本上只是進行了交流，而不是改善。我們軟體行業的人喜歡吹捧失敗的重要性，這也是毫無價值的，失敗本身當然是可怕的，但是我們想要的可不只是這個東西。

當人們這樣說的時候，他們的意思應該是把重點擺在**學習**和嘗試一些新的事情來解決那些導致失敗的觀點，有時（通常）失敗會伴隨著真正新的事物，重複一個已知的公式很容易，但從長遠的眼光來考量，我們必須鼓勵嘗試不同的方法。

在這種情況下，失敗的重要性不在於慶祝失敗，而需要強調的是，我們做得還不夠好，雖然沒有簡單的解決辦法，但我們可以做得更好。正如弗萊德·布魯克斯（Fred Brooks）在他 1975 年出版的優秀著作《**人月神話：軟體專案管理之道**》的後續文章中所說的，靈丹妙藥並不存在。

但是有一個辦法。

這得從一個問題開始：如果不借用殘缺的隱喻和語言來削弱我們的工作，而是剝離這些痕跡，重新思考工作的本質，我們的工作將會是什麼樣子？

現在只剩下**概念**還沒有討論到，這是語意軟體設計的核心，下一章將闡述概念的意義，因為這些概念適用於我們提出的方法，也就是你在設計有效的軟體時所應扮演的角色。

碁峰 www.gotop.com.tw