# 序論

## 專案經理如何跨越橋樑

我是史黛西亞・鮑德瑞克。我想訴說一則與改變有關的極爲私人故事，希望即使在非常不舒服與提心吊膽的時候，依舊能協助你認清傾聽自己及學習如何成長的重要性。

自 1993 年以來我就一直擔任專案經理，2003 年成爲敏捷專案經理。我也是 PMP，正式受過已有數千位取得合格專案管理師（Project Management Professionals）證照者所使用詞彙的訓練。當開始管理專案時，我相當自豪對於規劃專案、學習如何將資料輸入專案管理工具中、舉行狀態會議、與承包商和第三者供應來源協商資源和材料、減輕專案中的風險、及當然還有在控制範疇這些方面的能力。我能夠在睡覺時進行前推（forward-）推算與倒推推算（backward-pass calculation）。

專案管理非常適合我。我在三年級時，就與我的兩位姊妹分配工作，每週排定輪流做雜務的時程。我甚至設計流程，僅依據一種拉出與批次（pull-and-batch）系統清空洗碗機，而減少洗碗裝載次數（這種系統只有需要時才取出餐具、直到要重新放入洗碗機之前在水槽中收集所有骯髒餐具、並一起同時將餐具重新放入），但我父親不支持這種新方法。對於自認具有控制癖好的我，非常適合專案管理。

我與 Scrum 這種軟體開發敏捷方法的衝突始於 2003 年。我強烈反對未由任何正式主管團體所發起的這種無影響力新方法論（我當時大致上的想法）。當肯恩・施瓦伯（Ken Schwaber）前來訓練與教導我們的經理與軟體開發人員團隊時，我的生活被搞得亂七八糟。做爲忠誠的 PMP，或者也可能由於在軟體開發方面的經驗仍有所不足，最開始我對肯恩關於自行管理團隊與反覆式開發的教學有點懷疑。當我在兩天 ScrumMaster 訓練中，忽而投入忽而渾渾噩噩時，最引起我注意的一行是：「你沒有權力。」就某種意義來說，肯恩的意思是指產品負責人（product owner）與成果交付團隊本質上是協同合作的。如同宗教真言（mantra），我一再重複這句話，看看是否能習慣它。我持續想：「你怎麼可能在沒有權力的狀況下管理專案或人？專案成功的必要條件，不是你必須費勁進行專案，並要求人們超時與在週末工作（但承諾要給他們免費披薩吃）嗎？當專案團隊人數日益增多，但進度實際變得更慢時，這不是意味著你能更容易鞭策他們而迫使他們屈服嗎？」（我是開玩笑的啦！）

當老闆未出席 ScrumMaster 訓練時，我自動被推出，成爲（剛成爲前）老闆的替代者。恭喜我吧，我成爲三個專案團隊新任命的 ScrumMaster。

哇！所以現在我必須帶領人。我以前從未帶領過人。我當然要管理他們、收集他們的工作狀態、並提問他們對於那些工作，他們還需要多少時間才能完成。而且當然我會懷疑他們估計的時間。（每個人都知道，開發人員是可怕的評估者！）我有時候甚至提供有關某些技術工作容易或困難的有助益意見，並確定這會令開發人員很高興。

當然，當時不瞭解的是，從一開始我就真的沒有權力。你可以瞭解到，我一直管理一群知識工作者，他們是玩弄數字長大、撰寫複雜程式碼、並有創造力地迅速創造出根本上僅由 1s 與 0s 組成產品的人。我真正相信，直到學習如何帶領人之前，這些知識工作者只是容忍我。我從未真正管理過他們。他們決定藉由填寫時間進度表讓我高興，而來管理我。當我要求進行專案計畫的測試階段時，他們再度迎合我。對於事情應該如何運作，他們肯定知道得比我還多。我的生活由不可能完成的專案計畫所支配（請參考圖 I-1）。整整好幾個月，

我大量加班工作，在辦公室待到很晚，以求甘特圖（Gantt chart）完美，而且心中知道，隔天（若不是正好下一分鐘的話）這張圖就會過時。我經常被要求爲高階主管「製作一張儀表板」，也就是自知會反映錯誤正面真相的一份報告。現在再回頭看，我都懷疑自己是如何在「經理」這個頭銜之下生存下來的。

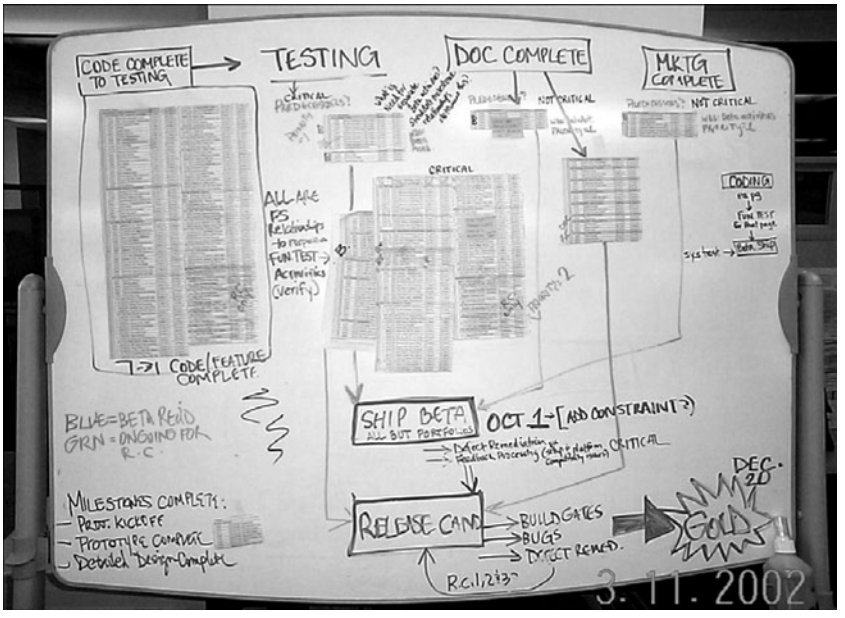


圖 I-1 作者所提供的「不可能完成的專案計畫」

我最先的想法是去追蹤專案狀態。如何知道「我們的進度到哪裡」？如何知道實質上已獲得多少價值？（當時我的執行長已撰寫關於實獲值管理的一本書）如何管理範疇？（我已替部門產生一份範疇變更管理程序，並且花好幾星期使圖表完美）最令人驚慌的是，因爲我們再也無法告訴他們，何時能交給他們想要的每件東西，我好奇客戶會認爲我們是多麼的瘋狂。而且沒有價值的 Scrum 的追蹤機制是甚麼？一張燃盡圖（burndown chart）？那張圖有可能告訴我們甚麼事？燃盡圖不可能告訴我們是否按進度進行！我想要完成百分比狀態報告！而且容我不客氣地說，與高階主管召開的前幾場會議徹底失敗。我知道很多時候離開時是臉紅的。

所有這些問題都因我即將經歷的個人掙扎而激起：「哇！若團隊能自行管理，它們將不再需要我。既然我們打算使用 Scrum，這個組織之內就沒有我的容身之地。」在這個新領域之內，我不知道該如何做事。要超越「自我」而專注於團隊，對我來說是非常真實的掙扎。歸屬權（ownership）問題，也同樣困擾著我。我常態性地在沒有更新產品工作清單（product backlog）下，極力掙扎處理這項管理性工作。對我來說，這代表範疇，而且沒有讓這件事在自己的管轄之下，令人覺得非常惶恐。我覺得沒有權力，好像什麼人都不是一樣。

大約在第三次全力衝刺（sprint），我開始明白道理。一旦團隊開始交付可見到與碰觸到的真正價值時，「燈泡」持續點亮著。一度大聲吼叫的產品負責人，現在成爲投入又充滿活力的產品負責人，他們實際與團隊合作，談論使用者體驗，並協助開發人員交付有價值的產品遞增。觀察共處一處的（collocated）團隊成員，經常聽到他們的笑聲、看到他們緊密工作在一起、並享受他們的個人生活，再度令我感動。這種感動是任何完美計算的專案甘特圖或巢狀工作分解結構（nested work breakdown structure），所不能辦得到的。讓團隊在持續的步調下工作，並將心力放在真正要緊的事情上，也就是爲他們任職的公司開發軟體，同時能在其餘時間享受他們的個人生活（畢竟，這不就是讓我們所有人都保持頭腦清楚的基礎嗎？），我開始瞭解這樣做所代表的意義。連同一位「瞭解情況」的副總一起禁止部門加班後，可維持步調的敏捷原則真正提高士氣並提升工作生活品質。我甚至有時間在某天晚上拜訪其中一位開發人員的家、與他太太見面，並得知更多真正的印度食物。那是一次很棒的個人經驗，而且現在我熱愛牛奶酥綿糖（soan papdi）這種超棒的印度甜點。

將有關敏捷開發價值的個人「燈泡」熄掉後，我開始瞭解到，做爲敏捷專案經理如何提供價值。首先，我撒手不管工作清單，而且它也鬆開對我的束縛。如此一來，我將控制權交給產品負責人，讓他排定清單項目的優先順序。這讓我有更多時間集中心力建立團隊。我與其中一個團隊一起搬到共處的空間，並處理讓預算合理化，使其他團隊能共處（這點成功辦到！）。我設計要給所有

專案團隊，稱爲 Daily Collaborator 的一份業務通訊，內容包括照片、故事，及關於專案的有趣事實。透過瞭解高階主管的需要，並要求團隊協助我決定如何展現專案資料，我學會如何向高階主管報告，這點是不小的功績。我確定利害關係人會涉入產品審查，有時候他們很難撥出時間。在「反覆」的審查過程中，我讓來自訓練與支援單位的人參與，並且在測試實驗室人工測試部分系統時獲得他們的支持。我協助建立產品工作清單會議，取代傳統的變更控制會議。當顧客建置我們產品早期的發表時，我與他們合作蒐集回饋資訊，並瞭解我們如何提升他們的體驗。而在團隊房間、每日站立會議（standup meeting）、檢討，與一般的團隊互動時，當安靜的時候，我總是觀察、觀察、再觀察。這些觀察協助我判定，接下來有哪些障礙要處理。我整理詳細的筆記，並在發現需要注意的變更或組織阻礙時，會將任務加到自己的阻礙工作清單中。我同時扮演變色龍與孔雀的角色，保持與環境融合爲一的能力，而在需要變更環境時挺身而出展現我的羽毛。

實施 Scrum 九個月後，我們慶祝一次非常成功的產品發表，那是讓所有人都感到驕傲的時刻。我們每個人都歷經其他人未曾經歷過的一趟個人旅程與改造。產品發表時，我們的 T 恤上寫者：「用心開發，自豪交付。」（Develop with Heart; Deliver with Pride.）。那個 85 人的部門，永遠都是我對於真正執行 Scrum 軟體開發時最喜愛回憶的一個組織。

讓我魂都嚇掉的前三個 Scrum 團隊，在我心中永遠才是教導我「放手」這種艱難教訓的人。

我的職業生涯最棒的一天，是走進我的 Scrum 團隊某一場每日彙報時，團隊成員微笑看著我說：「史黛西亞，我們不需要妳待在這裡。妳可利用這段時間從事其他事情，或協助另一個團隊。專案在我們掌控之中」。你知道的，專案的確在他們的掌控之中。我走開時幾乎掉下眼淚，但眼淚不是爲了我自己或任何「損失」，而是來自能夠放手並知道一切狀況都還好，所感受到的快樂，及協助個人變得賦有權力，所感受到的滿足。

爲跨過敏捷之橋，我必須瞭解將其他人擺在我前面的意義。對我而言那不是一件自然而然的事。由於在艱困中長大，並缺乏自我意識，我在專案經理頭銜中產生強烈認同感。我必須學習如何助人，並傾聽表面下的問題。最重要一點，我必須學習做事的人對事情最瞭解，並也能想出從 A 點到 Z 點最好的辦法。他們真正需要我做的是去清除路障。他們已經知道這一點，而 Scrum 協助我瞭解這件事。

由於米雪兒馬上就搞懂 Scrum，而我卻花了一段時間。當建立個人通往敏捷的橋樑時，我們是各自來自非常不同的地方。米雪兒的橋短又筆直，而我的橋是一座搖晃的吊橋、成 45 度角、充滿著狂風暴雨。我們都認同，在我們的職業生涯中感到最快樂的事是在協助（幾百個）團隊轉移到敏捷方法。在後面各章，我們樂於告訴你一些概念，如何將你已經知道，與管理專案有關的觀念，轉化成你自己的敏捷典範。我們將更深入挖掘你應該期待的東西、如何成功做轉換，及爲跨越通往敏捷之橋所需要採取的步驟。

# 第一部份

# 敏捷綜述

# 1

# 何謂「敏捷」？

在努力求生過程中，能成功針對環境自我調適者，才能打敗競爭者贏得最後勝利。

—查爾斯・達爾文（Charles Darwin）

達爾文搭小獵犬號（HMS Beagle）著名的航行，耗時四年九個月又五天，於 1836 年結束。我們都熟悉這段故事：「達爾文啓航後，在南美洲與世界上其他地區發現獨特的動植物物種，經過多年努力後，以他的進化論（Theory of Evolution）達到最高潮。」[1] 達爾文開始他的冒險時，他家人十分失望，認爲這會延遲他所期待，成爲神職人員的任用資格。航行歸來後，知道維多利亞時代的人堅定信賴神職人員，他隱匿自己的新興理論二十年不公開，而持續研究藤壺獲得充足資料，以便有一天證明並徹底支持他的論點。他知道他的思想與信念（或懷疑！）將會引起騷動。

如同老生常談所流傳的，故事的尾聲則回歸歷史。達爾文的理論影響如此深遠又富有創造力，這些理論依舊在今天全世界的學派系統與醫學研究中激起辯論。他的發現威脅到當時的宗教規範，也在他亡故超過 125 年後，刺激著今天的信仰系統。

達爾文的航行，就像專案經理走過橋樑通往敏捷之路，這是信念的跳躍、冒險進入未知領域、發現的過程，及有時不可預測的結果。達爾文與新的敏捷

專案經理，都想知道甚麼東西構成他們的新知識，及在目前社會背景或企業組織之下，如何融合那項知識。

即使從事軟體開發的敏捷方法，自 1960 年代便已出現，卻直到最近敏捷運動才得到較多推動，現七家公司當中有一家正在導入或探索敏捷方法。[2] 我們生活在軟體業界推行一項重要運動的時代。這項運動促使很多專案經理去探究「敏捷」所代表的意義。他們的發現是，就像達爾文的發現，敏捷方法論不同於軟體開發的傳統規範（norm），也就是經常代表他們的企業組織社會結構的規範。對於閱讀本書的一些人，敏捷方法論就像是個全新的「島嶼」（island），然而對其他人，這些方法可能僅代表與你所熟悉的東西相搭配的途徑。

前往敏捷宣言（Agile Manifesto）網頁查看，就會發現數千名支持這種過程者的姓名。隨機抽樣那個網頁上的評語，會產生像「敏捷宣言是重新打造軟體開發技藝的重要第一步」[3] 這類引述。而且從數千個評語張貼明顯看出，這項支持是全球性的：「Me siento feliz porque nuevamente me siento emocionado por los avances en mi profesion.」（I feel happy because again I feel touched by the advances in my profesion [sic].）[4]

儘管敏捷軟體開發獲得很多支持，也有一部分的批評者相信，它只是一時的流行，實際上並行不通。堅持計畫導向與硬性規定方法，是交付軟體更適當方法的人，也架起網站與部落格。我們的經驗顯示，全世界都存在成功的敏捷專案與敏捷團隊。我們主張，當管理階層支持團隊，而且團隊能與客戶互動時，他們會比運用連續與硬性規定方法開發軟體的團隊更加成功。

我們瞭解，第一次拜訪敏捷團隊房間，又沒有敏捷經驗的傳統專案經理，必定覺得有點像達爾文發現加拉巴哥群島（Galapagos Islands）上的獨特生物與生命時的感覺，因爲景觀有很大不同。敏捷團隊交出日益增多的產品發表，這容許組織在解決問題時能有所回應與做前瞻性規劃。對個人而言，團隊是安全處所，藉此信任、協同合作與辛勤工作成爲團隊成立時的價值觀。在團隊內部，人們也彼此交談。

本章讓你對敏捷方法的起源、價值觀與原則有紮實瞭解，讓你能瞭解另一座「島嶼」看起來像甚麼。

## 敏捷的起源是甚麼？

敏捷是描述發表軟體的一套原則與實務。今天我們所知道的敏捷，實際上已出現一段相當長的時間。

審視敏捷的起源，將我們帶回到幾十年前。美國國防部（Department of Defense, DoD）與太空總署（NASA）自 1950 年代以來，就已經使用反覆與漸增式開發（iterative and incremental development, IID）[5]。IID 是一種系統建立方法，其特色是連續執行幾次反覆，每次反覆都含有新特色發表。因爲可依據可行的軟體做決策，IID 是有彈性的。

1960 年代，湯馬斯・吉爾伯（Thomas Gilb）概念化亦稱爲 Evo 的演進式（Evolutionary）專案管理，並於 1976 年正式介紹這種觀念。Evo 建議一至兩週反覆，每次反覆都把焦點放在產品交付。如同 IID，Evo「強調，爲獲得早期商業價值，及指導與逐步形成未來交付成果的回饋，在早期就要交付部分解決方案進入生產。」[6] Evo 也具體化敏捷軟體開發的遞增（incremental）、反覆與回饋成分。

甚至溫斯頓・羅伊斯（Winston Royce）博士最先介紹「瀑布式」（waterfall）方法觀念的 1970 年論文「管理大型軟體系統開發」（Managing the Development of Large Software Systems），也討論到反覆式軟體開發的價值。的確羅伊斯談到，瀑布模型「既冒險又帶來失敗」，因爲這個模型將測試留在最後，主要設計缺失到那時才可能發現，並且需要大量重工（rework）。「實際上開發過程回歸原點，而我們也能預期在時程及/或成本上，最多會有 100% 超支。」[7] 隨後他的論文九頁中有七頁專注於基本瀑布模型的改善，包括反覆式開發與客戶全程參與。閱讀這篇論文時，你會感到奇怪，我們怎麼會遺漏他最後的建言，而以實作他的基本模型替代。諷刺的是，並非羅伊斯的觀念本身，而是對羅伊斯的觀念採取有限度接納的產業界，創造出軟體開發的瀑布式方法。

1986 年時，竹內弘高（Takeuchi）與野中裕次郎（Nonaka）發表的一本白皮書「The New New Product Development Game」（新型態新產品開發競賽）暗示，「產品開發的遊戲規則在變化中。很多公司已發現，要在今日競爭

性市場中勝出，光做到接受高品質、低成本與差異化的基本觀念還不夠，而且還要做到速度與彈性。」[8] 同時，竹內與野中也討論，專注又自行組織成團隊的「英式橄欖球方法」（rugby approach）。這種團隊成員就像實際英式橄欖球並列爭球團隊，共同合作贏得橄欖球控制權並在球場上將球往前移動，所有成員一起合作交付產品。竹內與野中發現，在一種「像接力賽跑般」的連續系統中，「重要問題傾向於發生在一個小組將專案交給下一個小組的交接點。藉著維持跨越階段的連續性，英式橄欖球方法剔除這種問題。」竹內與野中介紹另一項敏捷原則，也就是以專注、跨功能性部門、自行組織的團隊，減少功能性部門人員小組（塔狀穀倉）所產生的瓶頸。這份白皮書是 Scrum 方法的基礎。你可在附錄 A「敏捷方法論」中閱讀到關於 Scrum 的更多知識，及其他敏捷方法。

此外，如同豐田生產方式（Toyota Production System, TPS）所提倡的，精實產品開發（Lean Product Development）的概念，甚至提供更多基礎給敏捷軟體開發方法系列。精實透過持續改進消除浪費、第一次就使品質達到要求、並只生產客戶所要求的產品。因爲敏捷團隊藉由聚焦於有分等級產品工作清單中，具最高價值的項目，而只設計客戶目前所需要的特色，敏捷團隊所做的事非常類似精實產品開發。在《Lean Software Development: An Agile Toolkit》[9] 這本書中，將精實觀念應用於軟體產品開發時，瑪麗與湯姆．帕本迪克（Mary and Tom Poppendieck）使這些精實觀念成爲不朽。

談到這些年來敏捷軟體開發的進步時，對照 1920 年代以來，源自製造業企業文化，而且依舊存在的管理風格，一些敏捷實踐者（agilist）說它是軟體業的一項運動。在當時是創新，菲德列克．泰勒（Frederick Taylor）提出將管理人員（白領）與一般工人（藍領）分開。遵循泰勒「科學管理」方法的製造商，將工作細分成可管理的區塊，並指派經理人管理需負責的工人。然後經理人針對完成工作的最佳方法計時，並分發給工人以這些時間與動作研究爲依據的「說明卡」。工人的誘因很簡單，只要按照經理人分發的說明卡工作，就會得到報酬。此處明確的訊息是，經理人比工人更瞭解工作。

第一次世界大戰結束時，泰勒的管理理論贏得支持，並且在當時經證明適用於員工。然而，經過多年後，工作者變得更專業，也接受更多教育。近年來全球製造職位的減少（從 1995 年至 2002 年減少 11% [10]），已導致教育與職

業選擇的轉移。事實上，一份 2005 年美國勞動力市場報告陳述：「預料成長最快的職業，十項中有三項與電腦有關。」[11] 這群數量日益增多的「知識工作者」（彼得・杜拉克 1960 年代所創造的術語）代表新類型的員工，他們利用知識與腦力而不用雙手，做爲賺取所得的方法。

在知識工作者的新時代中，舊有的組織典範不再適用，而開始轉向敏捷原則與實務。因爲就像志願者，這些工作者可帶著他們的「生產方法」（也就是他們的知識）離開，現在必須將工作者當作「志願者」看待。同樣也像志願者，知識工作者不想到處被下命令。相反地，他們想要投入、參與，而且想要知道要往何處去，以及他們的工作對其他人將造成甚麼影響。這意指命令工人的舊方法，必須以知識分享與說服的新方法取代。如同杜拉克所說：「一個人不是從 [我想要甚麼？] 這個問題開始，而是從 [另一方想做甚麼？他的價值觀是甚麼？目標是甚麼？認爲結果是甚麼？]...這些問題開始...出發點可能必須爲了績效而加以管理...出發點可能是結果的定義。」[12] 因此在敏捷軟體開發中，我們聚焦於提供價值給客戶，並在做這件事時不斷提升生產力。

1990 年代，我們見到各種組織中，涉及全面性的一大堆敏捷方法，從伊索公司（Easel Corporation）的 Scrum、克萊斯勒公司（Chrysler Corporation）的極致軟體製程（Extreme Programming）、水晶方法（Crystal Methods）到 IBM 的 Rational 統一流程（Rational Unified Process）。此外，動態系統開發方法（Dynamic Systems Development Method, DSDM）在歐洲被加入組合中。顯然更輕量級的方法，在全球軟體開發景象中湧現出來。90 年代中期到晚期，顯示使用這些方法設計產品的顯著增加，並在 2001 年隨著「敏捷宣言」的撰寫而達到高潮。

## 何謂敏捷宣言？

2001 年時，一群 17 位「輕量級」方法學家，在猶他州雪鳥（Snowbird）渡假村見面，討論他們的軟體交付方法。這群人由來自極致軟體製程、Scrum、DSDM、適應性軟體開發（Adaptive Software Development），及「對文件驅

動重量級軟體開發流程替代方案的需求表示贊同」[13] 的其他人所組成。吉姆・海史密斯（Jim Highsmith）說，大多數敏捷原則歸結成「含糊不清的東西」。他也說：「對敏捷方法論感興趣的迅速加溫，及有時對它的猛烈批評，與價值觀及文化的含糊不清東西有關。」[14] 對前述「含糊不清的東西」，及他們設計軟體的方法進行諸多討論後，敏捷宣言（Agile Manifesto）寫成：[15]

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and Interactions over Processes and Tools
- Working Software over Comprehensive Documentation
- Customer Collaboration over Contract Negotiation
- Responding to Change over Following a Plan

That is, while there is value in the items on the right, we value the items on the left more.

| | | |
|---|---|---|
| Kent Beck | James Grenning | Robert C. Martin |
| Mike Beedle | Jim Highsmith | Steve Mellor |
| Arie van Bennekum | Andrew Hunt | Ken Schwaber |
| Alistair Cockburn | Ron Jeffries | Jeff Sutherland |
| Ward Cunningham | Jon Kern | Dave Thomas |
| Martin Fowler | Brian Marick | |

儘管四項價值陳述會在以下幾節討論，我們覺得強調宣言的最後一句陳述是重要的。撰稿人並未宣稱，敏捷團隊不撰寫文件（舉例來說，儘管這種神話圍繞在敏捷團隊週遭）。更確實來說，敏捷團隊質疑它們傳統上所產生之文件的價值，並削減那些文件，只撰寫有用又有價值的文件，例如測試或高階設計，或最終使用者文件交付成果（deliverable），例如線上輔助說明或手冊。考慮

宣言最後一句陳述時，記得強調將左邊項目當成價值陳述，並視需要利用右邊項目，同時也記得項目何時增加價值，這點是重要的。現在讓我們更深入探討「敏捷宣言」。

## 個人與互動勝過流程與工具

宣言的第一個價值觀，強調建立軟體系統時，個人與互動的重要性。所有敏捷方法把重心放在賦與權力的自行管理團隊，這些自主團隊不需要管理人員每日介入。反而若管理人員保護團隊免於外在干擾，並聚焦於移除設計產品過程中的障礙，團隊就會變得非常有成效又有生產力。[16] 此外，人們普遍接受複雜系統不可預測 [17]，而且使用依據經驗的流程控制來管理這些系統最好。[18] 因此，管理人員允許自行管理的敏捷團隊依據經驗建立系統。

爲完成工作，敏捷團隊會被授予權力做必要決策。竹內與野中將這點稱爲「自我超越」，意指團隊應該處於「永無止盡探索以找出極限」的狀態。[19] 經授權的團隊接受管理人員給予指導方針，然後從那些指令制定自己的目標。他們以整個團隊的形式，設計出獨特解決方案，找出避開問題的辦法。

敏捷團隊由具備不同技能的人所組成，團隊裡會有創造產品加值所必要的每個人。這意思是敏捷團隊由開發人員、測試人員、資料庫專家、撰寫員、經營分析人員、使用者介面專家，及其他具備技能的專業人士所組成。藉由每日一起工作，達成每次「反覆」所訂定的目標，團隊裡的人開始替產品產生共同方向。構想部分重疊。領導者出現。敏捷團隊的成員必要時能彼此互相介入對方的工作。他們不是經由一系列交接，而是由整個團隊建立系統。通常我們將那些交接與連續流程相關聯。他們應用所學到的知識，而且共同的知識會增長，結果促使設計、品質與生產力提升。

人類已嘗試創造工具取代面對面溝通。從協同合作的入口網站到線上社群到虛擬白板，技術空間有許多產品協助我們進行更好的溝通。不幸地，對很多

人而言，這些工具被視爲是倒數第二的溝通工具，不過，很多團隊已得知，運用工具練習腹語與流程牽線木偶（process marionette），在產品開發過程中產生驚人的浪費。藉由聚焦於個人與互動，「敏捷宣言」迫使團隊重新思考最佳溝通方法，並瞭解團隊成員親自協同合作的力量，以解決共同問題。而且當團隊成員沒有面對面時（這個世界的確是個全球市場），工具可協助支援（而非取代）那些交談。絕不要低估簡單的一通電話所帶來的價值。

## 可行的軟體勝過詳盡的文件

敏捷專案重視可行的軟體（working software），這點與傳統分階段專案所重視的極爲不同。傳統上，我們會以功能里程碑（也就是指分析完成、文件完成、程式碼完成等等）的完成百分比測量專案進度。不過，在敏捷專案中，可行的軟體是專案狀態的最終量化。每個人不在狀態會議中報告「我已 90% 完成」，敏捷團隊在每次反覆（iteration）結束時，要提供實際可行的產品做爲狀態報告，這項報告稱爲「產品檢討」（product review）。審查可行的軟體，使我們能適當回應專案真正狀態。每樣東西都看得到，而且可依據已存在的產品做決策，而不是記載於文件的產品表述。

「敏捷宣言」的這句價值陳述有雙重目的。這句陳述背後的第二種觀念是，很多團隊認爲有些文件是浪費的。事實上，有些開發人員直接告訴我們，他們不需要「規格」。以商業術語來說，一開始花很多時間在規格中記錄每項設計細節，可能是浪費時間。大多數敏捷團隊說，設計會隨著系統的建立而變更，因而造成文件過時，因此，開始撰寫程式碼與接收到客戶回饋時，爲何要花時間將構想記載成文件？發現超過「60% 的軟體功能極少用到，或甚至從未使用」[20]，著名的史丹迪希公司（Standish Group）提到詳盡的文件所帶來的浪費。這種結果是如何造成的？理由在於傳統專案中，當範疇最初就定義好，爲讓專案「按照時程並維持在預算內」，甚且當最初定義的特色需要變更，或因

客戶所處的環境有變動，需要將這些特色去除掉時，範疇都會受到保護。因此，爲何我們要開發客戶用不到的東西？事實上，甚且我們懷疑爲何要撰寫它？

一位開發人員非常簡潔地陳述這個觀點：

> 若有人交給我一份 40 頁文件，並告訴我進行編程，我不知道要先開始做甚麼事？事實上，我可能會先處理架構上的東西，或可能從事有趣又令我興奮的事情。對客戶而言，那可能不會是最有價值的特色。所以現在費時撰寫文件，又因我認為應該最先開始編寫的程式（而這可能不是正確做法），所造成的浪費，讓情況更惡化。

除浪費的因素之外，因爲面對面溝通更簡單、更快，又更可信賴，敏捷團隊喜愛面對面溝通甚於文件。

對照前面的說法，在團隊所處的組織背景下，有團隊認爲有價值的各種形式文件存在。例如，沒有最終使用者文件，組織可能就活不下去。事實上，爲使特色在每次反覆結束時可被接受，這份文件的重要性已提高到成爲每項特色必須通過的標準。另一個團隊決定記載每次「反覆」所做的決策，因爲日前有五位有價值的團隊成員，因敵意團隊挖角而離開。還有另一個組織必須遵從政府法規，所以必須與內部審計人員一起找出，能通過稽核的「適當」文件水準。縱使重點著重在可行的軟體，所有文件都不錯。若心中有懷疑，那就去問團隊「這份文件有價值嗎？撰寫這份文件對我們更好嗎？我們負有法律義務嗎？若是如此，能找出甚麼是我們能做的最低限度嗎？」

## 客戶協同合作勝於合約協商

以傳統意義來看，合約協商意指我們找出並定義客戶想要的每樣東西，然後制定詳細說明付款與日期規範的合約。這已導致很多固定價格/固定範疇的情況。我們所學到的是，這未必是軟體開發專案的最佳方法。太多團隊發現自己處於「死亡行軍」（death-march）[21] 的情況，每週要工作 80 小時以符合合約

所提出的截止期限，這份合約最初是由團隊以外的某個人所估計與同意。同樣糟糕的是，客戶發現自己對不具意義的工作做出承諾，而所有這些工作都在合約的名義之下訂定。

另一方面，敏捷的「客戶協同合作」暗示，客戶變成開發過程的一部分。爲開發正確系統，客戶回饋不可缺少。敏捷團隊重視客戶（或客戶代表）所做出的貢獻，並且學習讓客戶做經營決策。對客戶而言，爲做出適當決策，客戶倚重團隊所提供的重要技術資訊。有時候，客戶直到親眼看見，才知道他們想要甚麼。

近來我們見到大量把重點放在更適合敏捷專案的合約。瑪麗．帕本迪克談到依據豐田汽車與鋼模製造商的合約簽訂範例，在合約中事先訂定雙方的成本責任。他們的合約依據目標成本原則擬定，一旦達到合約目標成本，豐田與製造商共同分攤額外的成本。爲將變更成本降至最低，這對雙方都是誘因，但還是承認與允許變更成本發生。另一種類型的合約稱爲「分階段合約」（staged contract），這種合約允許在合約中預先約定查核點，以提供客戶合約進行當中的能見度與「繼續/不繼續」決策。這些並不是新的合約類型，但是它們獲得更高能見度，成爲軟體開發團隊更可行的替代方案。

若客戶提供回饋，並在專案過程一路引導團隊，團隊就能建立滿足客戶需求的系統。撰寫好規格，並將規格丟到籬笆外，不僅沒有效果，而且也讓很多團隊陷於超時工作進行系統修正，以符合客戶需求的境地。也許客戶在開發過程中若有一直參與，團隊將不會陷入這種處境當中。我們已輔導過最後陷於這種情境的很多團隊，並且使用敏捷方法做爲改變這種情況的方法。

## 回應變更勝過遵循計畫

當組織與客戶都清楚瞭解專案狀態時，回應變更就容易得多。聚焦於可行軟體的遞增特色，並與客戶協同合作，允許開發團隊更快速回應變更。

在計畫驅動的環境中，所有要求從一開始就規定好、分解至任務層級並加以估計。從成本與日期這些非常細微的任務由底層往上層計算。最終的時程變成專案基準，並用來衡量專案績效。因此，在設定目標去限制或消除成本超支或時程滑動的計畫驅動專案中，停留在任務層級並控制範疇蔓延（scope creep）變得非常重要。

「敏捷宣言」沒有說計畫不重要。事實上，敏捷團隊非常有紀律，而且專注於規劃與修訂那些計畫。因爲團隊成員涉入計畫催生，他們會非常投入。敏捷團隊使用由上而下規劃的更加滾動式（rolling wave）方法，我們在本書後面會更深入探討這種方法。

## 甚麼是引導團隊的敏捷原則？

「敏捷宣言」的撰寫，有連同指導團隊的一組總共 12 項原則。我們將每項原則列出，並加上一段簡短描述：

- **我們首先要做的，是儘早持續並交付有價值的軟體使客戶滿意。**

  這項原則強調交付價值給客戶的焦點。到最後，驅策敏捷團隊的是這項客戶滿意哲學。

- **即使到開發後期，也歡迎變更需求。敏捷過程利用變更爲客戶創造競爭優勢。**

  在計畫為基礎專案中，維持範疇意指不允許經常性範疇變更。藉由容許變更，我們可確定開發出的產品，能協助帶給客戶價值。藉著儘早且持續交付，同時也允許變更，我們有能力維持彈性與「敏捷」，以因應變動中的情況。

- **從幾週到幾個月，交付時間越短越好，經常性交付可行軟體。**

  這項原則具體指定將時間箱（timebox）當成持續交付給客戶的方法。更短的時間範圍意指沒有得到回饋之前，團隊不要花太長時間設計產品。這種方法可減輕因為有長時間開發週期，最後使客戶感到驚訝所引起的風險。在敏捷軟體開發過程中，因為交付非常頻繁，因此極少造成驚訝。

- **整個專案期間，經營管理人員與開發人員必須每天一起合作。**

  經營管理人員（business people）或利害關係人，是產品商業需求的代表人。正如客戶，對於系統應該如何運作，他們有某些構想與看法。另外，正如客戶，直到見到系統之前，他們未必知道那些需求是甚麼。因此，當經營管理人員與開發人員協同合作時，透過分享來龍去脈、構想與解答，他們保證更能符合他們的商業利益。

- **以受到激勵的個人爲主軸建構專案。提供他們所需要的環境與支持，並信任他們能將工作完成。**

  在敏捷專案中，我們說團隊是在時間箱之內自行管理。也就是說，每次反覆期間建構產品時，團隊應該能擁有隨時可供利用的所有必要資源，以及管理階層對工作能完成的信任。

- **在開發團隊內部傳遞資訊，及將資訊傳遞給開發團隊，最有效率與最有效果的方法是面對面交談。**

  儘管似乎是常識，這項陳述涉及以談話取代一些文件。例如，取代撰寫詳細設計規格，團隊成員一起討論與探討軟體應如何建構的構想。這不是說敏捷團隊不提供文件，他們的確會提供。只是說文件不是主要溝通工具。我們知道很多團隊分散在幾千英哩遠，有時候似乎唯一的溝通方法是透過大量文件，不過很多團隊已成功建置即時訊息傳遞軟體、視訊會議、維基百科（wiki）、現代化工程環境，及使用其他技術，來支持有效的協同合作。

- **可行的軟體是主要的進展量測方法。**

  檢視系統目前狀態，是瞭解專案狀態最好的辦法。什麼東西已交付？交付的產品如何運作？我們如何能確定它的運作方式符合我們的期待？藉著在建構過程看看產品，經營管理人員與顧客能問這些問題，也能藉由看到剛完成的產品得到答案！

- **敏捷過程提倡可持續的開發。贊助人、開發人員與使用者，應該能無限期維持固定進度。**

  對忙於專案的每個人，在可維持的步調下工作，是保持生活品質的重要因素。若過度工作的工程師費力地想集中心力完成工作，不僅生活品質，而且產品品質也可能變糟。

- **持續注意技術優越性與好設計能提高敏捷性。**

  任何人的技藝當中，專業精神是最重要的。撰寫簡單好設計軟體的技術專業人員，能快速又有效回應變更。這種彈性容許組織變敏捷。

- **「簡單」這種完成最多未完成工作量的藝術是必要的。**

  「簡單」是敏捷的反鍍金（anti-gold-plating）機制。敏捷團隊建構客戶想要的東西，除此之外沒有其他。這項非常簡單的原則，保證團隊不會建構客戶不想要或用不到的功能。一旦建構特色，就必須在系統存在期間維護，或必須做投資移除它。無論何種方式，這是應該避免的浪費。

- **最好的架構、需求與設計出自自行組織的團隊。**

  團隊的集體智慧，大幅勝過個人智慧。當團隊成員一起集思廣益時，他們會協同合作打造出可能的最好系統。這個敏捷原則回應杜拉克關於知識工作者的學說。

- **團隊應定期省思如何變得更有效，然後依此協調與調整團隊行爲。**

  儘管產品可在每次反覆結束時檢查與改造，這項原則討論過程協調一致的需要。為決定哪些流程運作良好、哪些運作得不好，敏捷團隊需要做回顧。團隊成員一起合作共同解決過程中的任何挑戰，並隨著時間聚焦於改善。

對某些人來說，敏捷代表專案管理的人本方法。對其他人來說，敏捷描述精實開發，與過程中浪費的移除。還有對另外其他人來說，在充滿擾亂與變動的快步調世界中，敏捷描述一種工作方法。在你的「島嶼」上，敏捷將對你有何意義？

## 摘要

從本章中你可學到以下各項重點：

- 敏捷專案管理企圖有別於傳統專案管理。
- 敏捷開發自 1960 年代以來便已存在。「敏捷宣言」誕生於 2001 年，是現有「輕量級」方法論指導原則的正式調和。
- 敏捷提出新重點：
  - 個人與互動勝過流程與工具
  - 可行的軟體勝過詳盡的文件
  - 客戶協同合作勝過合約協商
  - 回應變更勝過遵循計畫

「也就是說，儘管右邊的項目有價值，我們更重視左邊的項目。」

# 末註


**1.** Charles R. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. (London: John Murray, 1859) 1–6.

**2.** Carey Schwaber. “Corporate IT Leads the Second Wave of Agile Adoption.” Trends, a Forrester Research Inc. report. November 30, 2005, 2.

**3.** Kent Beck, et al. “Independent Signatories of the Manifesto for Agile Software Development.” http://www.agilemanifesto.org/sign/display.cgi.

**4.** Kent Beck, et al. Translation: “I feel happy because again I feel touched by the advances in my profesion [sic].”

**5.** Craig Larman and Victor R. Basili. “Iterative & Incremental Development: A Brief History.” Computer, June 2003.

**6.** Craig Larman. *Agile and Iterative Development: A Manager’s Guide*. (Boston: Addison-Wesley, 2004), 225.

**7.** Winston W. Royce. “Managing the Development of Large Software Systems: Concepts and Techniques.” (paper presented at the Western Electronic Show and Convention [WesCon]), Los Angeles, CA, August 25–28, 1970), 329.

**8.** Hirotaka Takeuchi and Ikujiro Nonaka. “The New New Product Development Game.” *Harvard Business Review*, January–February 1986 (reprint 86116), page 1.

**9.** Mary Poppendieck and Tom Poppendieck. *Lean Software Development: An Agile Toolkit*. (Upper Saddle River, NJ: Addison-Wesley, 2003), xxiv–xxv.

**10.** Tim Kane, et al. “Ten Myths About Jobs and Outsourcing.” The Heritage Foundation, http://www.heritage.org/Research/TradeandForeignAid/ wm467.cfm.

**11.** Bureau of Labor Statistics. “Charting the US Labor Market in 2005.” U.S. Department of Labor, http://www.bls.gov/cps/labor2005/home.htm.

**12.** Peter F. Drucker. *Age of Discontinuity: Guidelines to Our Changing Society*. (New York: Harper & Row, 1968), 264.

**13.** Kent Beck, et al. http://www.agilemanifesto.org/.

**14.** Ibid.

**15.** Ibid.

**16.** Takeuchi. “New New Product Development Game.”

**17.** Ralph D. Stacey. *Strategic Management and Organisational Dynamics, Fifth Edition*. (New York: Prentice Hall/Financial Times, 2007).

**18.** Babatunde A. Ogunnaike and W. Harmon Ray. *Process Dynamics, Modeling and Control*. (New York: Oxford University Press, 1994).

**19.** Takeuchi and Nonaka, 4.

**20.** Jim Johnson, “The Cost of Big Requirements Up Front (BRUF).” Keynote presentation at the annual XP (eXtreme Programming) Conference. Alghero, Sardinia, May 2002.

**21.** Ed Yourdon. *Death March*. (Upper Saddle River, NJ: Prentice Hall, 2003), xv–xviii.

**22.** Michele Sliger refers to this instead as the “Oh, crap!” moment.