從瀑布式到反覆式開發—對專案經理具有挑戰性的轉換

Philippe Kruchten

Rational Software 白皮書

TP 173, 5/00

Rational®
the software development company

目錄

簡介

Rational Unified Process (RUP) 對軟體開發生命週期提倡一種反覆式或螺旋式方法，而此方法在許多方面一再證明優於瀑布式方法。但不要輕易相信反覆式生命週期的許多好處得來全不費工夫。反覆式開發不是魔杖，一揮就能解決軟體開發的所有問題或難題。專案的設定、規劃或控制一樣不容易，因爲它們是反覆的。專案經理實際上要面對更具挑戰性的工作，尤其是在第一個反覆式專案中，當然最有可能是在該專案的初期反覆期間，那時候的風險高，初期失敗的可能性也高。在這篇文章中，我從專案經理的觀點來說明反覆式開發的一些挑戰。我也說明在我們的 Rational 諮詢經驗中，或從 Rational 同事的報告和實戰報導中，常見專案經理落入的一些「陷阱」。

反覆式開發

典型軟體開發流程遵循瀑布式生命週期，如下圖所示。如「圖 1」所示，在這種方法中，從需求分析到設計、程式碼和單元測試、子系統測試和系統測試，開發一路呈直線前進，對先前階段的結果回饋不多。

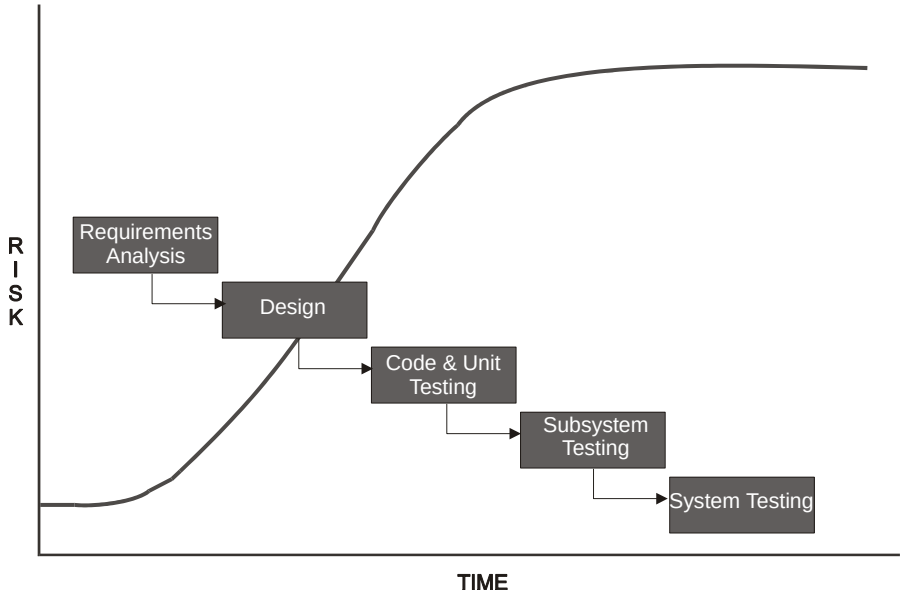

圖 1：瀑布式開發流程

這種方法的基本問題在於它隨著時間增加風險，要從先前階段的錯誤中復原，需付出很高的代價。起始設計在其主要需求方面可能有缺陷，不僅如此，最近發現設計瑕疵很可能導致超限運轉和/或專案取消，而付出昂貴代價。瀑布式方法有可能遮蔽專案的實際風險，等到要補救時爲時已晚。

瀑布式方法的替代方法是反覆式和漸進式流程，如「圖 2」所示。

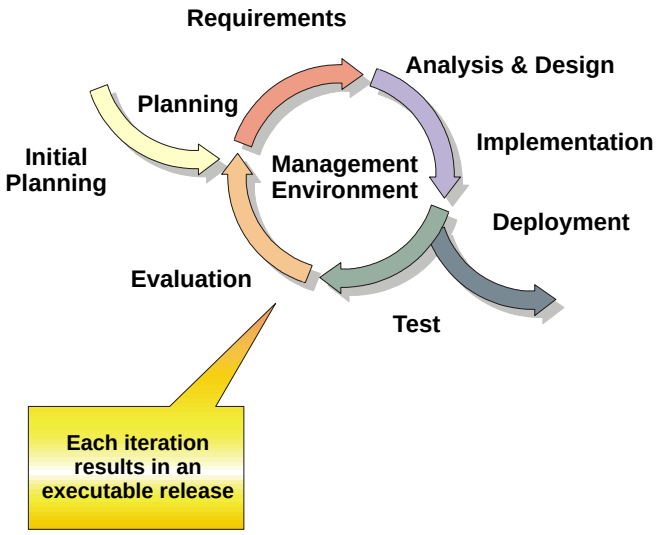

圖 2：反覆式開發方法

在這種方法中，以 Barry Boehm 螺旋式模型（請參閱深入閱讀）爲基礎建立的專案風險識別，會在生命週期初期會強制執行，而能夠即時有效地對風險提出反擊。這種方法是連續探索、創造和實作的其中之一，每一個反覆強迫開發團隊致力於以可預測及可重複的方式來結束專案構件。”

好處：反覆式開發的好處

與傳統瀑布式流程相比，反覆式流程有許多優點。

1. 在生命週期初期若有明顯的嚴重誤解，可對它們作出反應。
2. 它允許並鼓勵使用者提供寶貴意見，以誘導出系統的真正需求。
3. 開發團隊被迫將焦點放在對專案最重要的那些問題上，而團隊成員則受到庇護，使他們遠離專案的實際風險。
4. 連續反覆測試可達到專案狀態的客觀評量。
5. 可早期偵測到需求、設計和實作之間的不一致。
6. 團隊的工作量，尤其是測試團隊，在生命週期內更平均地分散。
7. 這種方法可讓團隊發揮所獲得的經驗，因而持續改進流程。
8. 專案的關係人可獲得專案狀態在生命週期內的具體證明。

減輕風險

反覆式流程可讓您儘早減輕風險，因爲整合通常是發現或處理風險的唯一時機。當您推出初期反覆時，會討論所有流程元件，同時運用專案的許多方面，包括工具、現用軟體和使用者技巧等等。所看到的風險將證明不是風險，而將發現新的未知風險。

如果專案因爲某種原因而一定會失敗，請在花費很多時間、人力和金錢之前，儘快讓它失敗。不會逃避現實太久；相反地，要勇敢面對風險。在建立錯誤產品等其他風險當中，反覆式開發流程幫助儘早減輕的風險有兩種：

- 整合風險

- 架構風險

反覆式流程可產生更健全的架構，因爲您可以透過數個反覆來更正錯誤。當產品越過初始階段，可在初期反覆中偵測到缺點。在還能夠處理效能瓶頸時發現效能瓶頸，而不是等到交付前夕才發現。

整合不是在生命週期結束時「大爆炸」；而是日益增加地整合元素。實際上，我們建議的反覆式方法差不多意味著連續整合。以前那種長時間不確定和痛苦—在專案結束時佔了總工時的 40%—現在分成 6 到 9 個更小的整合，以整合更少元素作爲開始。

容納變更

您可以面對數個變更類別：

- 需求的變更

  反覆式流程可讓您考慮到變更的需求。事實上，需求通常都會改變。需求變更和「需求匍匐前進」一向是困擾專案的主要來源，它會造成延後交付、未趕上排程、客戶不滿意、開發人員受挫。但是讓使用者（或使用者的代表）接觸到產品初期版本，可確保產品在作業上有更好的適應性。

- 策略性變更

  例如，反覆式流程提供管理階層對產品進行策略性變更的途徑—來與現有的產品競爭。您可以決定在初期發行縮減功能的產品來反擊競爭對手的招數，或採用指定技術的另一個廠商。您也可以重組反覆的內容，來減輕需要由供應商修正的整合問題。

- 技術性變更

  在某種程度上，反覆式方法可讓您容納技術性變更。您可以在詳述階段使用它，但應該在建構和轉換期間避免此類變更，因爲它天生會有風險。

邊做邊學

反覆式流程的優點是開發人員可以順便學習，且不同能力和專長可以更充分地運用在整個生命週期當中。例如，測試者儘早開始測試，技術文件作者儘早寫作等等，而在非反覆式開發中，同一批人可能要等一陣子才能開始工作，並不斷提出一個個計劃。培訓需求—或需要其他的（也許是外界的）幫忙—在評量檢閱期間儘早發現。

流程本身也可以順便得到改善和修正。反覆結束時的評量是從產品/排程視景中查看專案的狀態，並分析在組織和在流程中應該做何種變更，以便能夠在下一個反覆中執行得更好。

提高重複使用的機會

反覆式流程可促進專案元素的重複使用，因爲識別局部設計或實作的共同組件比較容易，而不必一開始就識別所有共同性。識別及開發可重複使用的組件很困難。初期反覆中的設計檢閱，可讓設建構師識別未知的、潛在的重複使用，及開發後續反覆中的共用程式碼，並使其完善。在詳述階段的反覆期間會發現一般問題的一般解決方案，並識別套用在系統中的模式和架構機制。

提升整體品質

從反覆式流程中產生的產品，比從慣用循序處理中產生的產品有更好的整體品質。系統會經過多次測試，以改進測試的品質。需求會加以修正，更貼近使用者的實際需求。在交付時，系統會執行更久一點。

難題：非預期的不利和一般設陷

反覆式開發不一定表示工作減少及排程縮短。它的主要優點是要爲輸出結果和排程帶來更多的可預測性。它會產生更高品質的產品，來滿足一般使用者的實際需求，因爲您會有時間來引申出需求以及設計和實作。

反覆式開發實際包含更多規劃，因此可能加重專案經理的負擔：必須先開發整體計劃，再由每一個反覆開發詳細計劃。它也包含連續協商問題、解決方案和計劃之間的取捨。更多架構規劃也會更早進行。構件（計劃、文件、模型和程式碼）必須在每一次修訂中反覆地修改、檢閱和核准。技術性變更或範圍變更將強制連續重新規劃。因此，團隊結構必須在每一個反覆中稍作修改。

設陷：到最後過度詳細規劃

除非要作爲評估排程和資源的廣域封套的練習，否則建構一個端對端的詳細計劃通常很浪費。在第一個反覆結束之前，此計劃會作廢。在備妥架構及掌握需求之前—大約發生在 LCA 里程碑—您不能建置實際計劃。

因此，請在規劃中併入與所規劃之活動、構件或反覆相稱的精準度。短期計劃會更詳細和精密。長期計劃則維持粗略格式。

忍耐不夠專業或消息不靈通的管理階層在嘗試推出「綜合性的整體計劃」時所承受的壓力。教育經理，解說反覆式規劃的概念，並說明嘗試預測遙遠未來的細節乃是浪費人力。有一個類比很有幫助：從紐約到洛杉磯的車程。您計劃了整體路徑，但其實只需要離開紐約及進入此車程第一段的詳細開車指示。不需要規劃車子行經堪薩斯州的確切詳細資料，更遑論要抵達加州，因爲您會發現，行經堪薩斯州的這條道路正在修復中，您需要尋找替代道路才行。

在最前面承諾重做

在瀑布式方法中，有太多重做到最後才進行，這是在最後測試及整合期間尋找嚴重錯誤的令人困擾的意外結果。更糟的是，您會發現「毀損」的大部分原因來自設計上的錯誤，您試圖在實作中建立暫行解決方法來減輕這樣的問題，結果造成更嚴重的毀損。

在反覆式方法中，您直接在最前面承諾會重做，而且一開始有很多要重做：當您在初期架構原型中發現問題時，需要修正它們。同時，爲了建置可執行的原型，必須重建 Stub 和 scaffolding，稍後再由更成熟健全的實作取代之。在健全的反覆式專案中，報廢或重做的百分比應該會迅速減少；隨著架構穩定且困難問題獲得解決，變更應該更少才對。

設陷：專案不聚集

反覆式開發不表示在每一個反覆拆毀一切。報廢和重做必須在反覆之間慢慢縮減，尤其是在 LCA 里程碑建立架構基準線之後。開發人員通常想要利用反覆式開發來錦上添花：推出更好的技術來執行重做等等。專案經理必須保持警惕，不要重做未破損的元素—沒問題或很好的元素。同時，隨著開發團隊規模逐漸壯大，一些人四處移動，新手也跟著進來。他們對於一些做法有自己的想法。同樣地，客戶（或專案中的代表：行銷、產品管理）也想要濫用反覆式開發提供的自由來容納變更，和/或無止盡地變更或新增需求。此作用有時稱爲「需求匍匐前進」。同樣地，專案經理在做取捨和協商優先順序時態度必須堅決。在 LCA 里程碑附近，會建立需求基準線，除非重新協商排程和預算，否則任何變更都有限定成本：有所得，就會有所失。

而且，記住，「好還要更好 (Perfect is the enemy of good)」。（或法文所說的：“Le mieux est l’ ennemi du bien.”）。”

設陷：先出發，以後再決定去向

反覆式開發不表示永遠模糊的開發。您不應該只爲了讓團隊保持忙碌或希望明確目標突然出現才要開始設計及撰寫程式碼。您仍然要定義明確目標，將它們寫下來，並尋求各方面一致；然後修正它們、展開它們及再次獲得一致。反覆式開發有好的一面，即您在開始設計、撰寫程式碼、整合、測試和驗證所有需求之前不需要先陳述所有需求。

設陷：成爲自己成功的犧牲者

在接近專案結尾「消費者位元」變動時，高風險出現。這表示原本不相信有任何東西可交付的使用者，現在相信團隊可能真的成功了。好消息是外界對專案的認知改變：星期一使用者還興高采烈地以爲會交付，到了星期二，他們開始擔心什麼都交不出來。這是壞消息。在第一和第二測試版之間，您會發現人們關心成爲第一版的那些特性讓你忙不過來。突然，這些成了主要問題。專案經理從擔心交付最少可接受的功能，變成擔心每一個最後需求現在都成爲第一次交付的「要項」的情況。幾乎好像這個位元一變動，所有尚未完成的物品都提升到優先順序狀態。現實情況卻是仍有同樣多的事情要做，而完成它們的時間量也沒有改變。當外界認知改變時，優先順序仍然非常重要。”

如果在這個重要時刻，專案經理慌張起來，開始屈服於所有要求，那麼他真的使排程中的專案再度處於危險中。這時候他必須繼續堅持下去，而不要屈服於新的要求。這時候甚至以新換舊也會增加風險。如果失去警覺性，就可能功敗垂成。

先安置軟體

在瀑布式方法中，非常強調「規格」（亦即，問題空間說明），並使它們正確、完整、完善及登出。在反覆式流程中，由您開發的軟體先開始。軟體架構（亦即，解決方案空間說明）需要驅動初期生命週期決策。客戶不購買規格，軟體產品才是關注的主要焦點，故規格和軟體的發展是平行的。此「軟體優先」的焦點對不同團隊有一些影響：例如，測試者可能習慣於接收完整而穩定的規格，在開始測試之前有許多事前的注意事項；然而在反覆式開發中，當規格和需求仍在發展當中，他們就必須馬上開始工作。

設陷：太多焦點放在管理構件上

有些經理說：我是專案經理，因此我應該把焦點放在我能力所及的管理構件集上；它們是萬能之鑰。其實不然！雖然好的管理很重要，但專案經理必須確定在結束時，最後產品是可產生的最佳產品。儘管有最好的管理專案管理並不是用來掩飾您自己的失敗。同樣地，您可以將焦點放在開發最佳規格，因爲過去您受到需求管理不足所苦；但如果相對應的產品有問題叢生、緩慢、不穩定而且脆弱，則這一點用也沒有。

早點命中困難問題

在瀑布式方法中，許多困難問題、風險和實際未知事物被推進規劃流程的右邊，在令人畏懼的系統活動中進行解析。這使得專案的前半部進行非常順暢，其中的問題都在書面上處理，而不涉及許多關係人（測試者等等）、硬體平台、真實的使用者或真實的環境。然後突然間，專案進入整合的地獄，一切像脫繮的野馬，完全不受控制。在反覆式開發中，規劃主要是基於風險和未知事物，因此一開始就非常棘手。有些困難、重要且通常低階的技術問題必須立即處理，而不能推到以後再處理。總之，就像某人曾經告訴我的：在反覆式開發中，您不能長時間說謊（對自己或對外界）。注定失敗的軟體專案，在反覆式方法中應該會更早碰到命運之神。

打個比方，在大學裡有一堂課，教授在學期的前半部花了很多時間講解相當基本的概念，讓學生以爲這門課程很容易，可以在期中考輕鬆得到好成績。但隨著學期接近尾聲，難度卻突然間升高。教授在期末考前不久才開始講解所有具挑戰性的題目。這個時候，最常見的情況就是班上大部分學生都在壓力之下全力以赴，但期末考的表現令人惋惜。聰明的教授卻年復一年，一班接著一班地被這重複的不幸所震驚。更聰明的方法是前置課程，在期中之前處理 60% 的工作，包括一些困難的資料。管理反覆式專案的相互關係是不要一開始就浪費寶貴的時間在解決非問題及完成瑣碎工作上。啓動時技術失敗的最常見原因是：把所有時間花費在執行簡單的工作上。”

設陷：逃避問題

常常聽到這句話：這是一個棘手的問題，需要很多時間思考。將它延後解決，這樣我們就有更多時間去思考它。於是專案開始從事一切簡單的工作，不去關注困難的問題。當需要解決方案的時候，就採取倉促的解決方案和決策，或眼看專案出軌。您要做的剛好相反：立即處理困難的事情。有時候我會說：如果專案因爲某種原因而一定會失敗，請在花費所有時間和金錢之前，儘快讓它失敗。”

設陷：忘記有新的風險

您在初始階段執行風險分析，並將它使用於規劃，但忘記在專案後面逐漸產生的風險。這些風險稍後會回過頭來傷害你。應該每週（如果不是每月）持續評估風險。您所闡述的原始風險清單只是暫訂而已。唯有當團隊開始從事具體開發（軟體優先）時，他們才會發現到許多其他風險。

因爲不同生命週期模型而發生衝突

沒有採用反覆式開發—或甚至不瞭解其本質—的反覆式專案經理，常常看到其環境與其他群組（例如最高管理階層、客戶和承包商）之間發現衝突。他們預期在主要里程碑已完成及凍結構件；他們不想以小型分期付款的方式檢視需求；他們對重做感到震驚；他們不瞭解一些難看的架構原型的目的或價值。他們認爲反覆只是毫無目的的摸索，只和技術打交道，在確定規格之前就開發程式碼，及測試宣傳小冊子的程式碼。

至少，您要把意圖和計劃表達清楚。如果反覆式方法只存在您的腦海中，以及與團隊分享的一些白板上，您以後會有麻煩。

專案經理必須保護團隊不被外界攻擊，以防止外界瓦解團隊或使團隊感到沮喪。他必須扮演緩衝的角色。爲了能夠成爲「操縱桿上堅定的手」，專案經理必須與外部社群之間建立信任和可靠性。因此，可見性和「計劃的追蹤」仍然很重要，尤其所按照的「計劃」在許多人眼中有點不符合常規時。事實上，它的確更重要。

設陷：不同群組依照自己的排程來操作

最好讓所有群組（或團隊或轉包商）根據相同階段和反覆計劃來操作，這樣會更容易。通常專案經理在細部調整每一個別團隊的排程中看到一些時間最佳化，每一個團隊最後有它自己的反覆排程。發生此情形時，所有看得見的好處以後都會不見，而且會強制團隊與較慢的群組同步化。只要行得通，請讓每個人同步。

設陷：在初始階段爲固定價出價

許多專案爲了契約的開發，差不多在初始階段的中期就提早出價。在反覆式開發中，所有單位進行出價的最佳時間點是在 LCA 里程碑（在詳述結束時）。這裡沒有訣竅：它需要關係人進行協商和教育，展現反覆式開發的好處，最後變成二步驟的出價流程。

進度的說明不同

說明進度的傳統增值系統不同，因爲構件不完整或凍結，但會以數種增量重做。如果構件在增值系統中有特定值，且您在建立它的第一次反覆中因它而得到好評，則您對進度的評量會太過於樂觀。如果您只有在二次或三次反覆之後該構件變穩定時才得到好評，則您對進度的測量會變得非常悲觀。因此，在使用這樣的方法監督進度時，構件必須分解成片段；例如，初始文件 (40%)、第一次修訂 (25%)、第二次修訂 (20%)、定稿文件 (15%)。每一個片段必須配置一值。然後您可以使用增值系統，而不必完成每一個元素。

替代方案是以反覆本身爲中心來組織增值，並從評估準則量測該值。然後，會以反覆計劃爲中心來建立「狀態評定」中所報告的中間追蹤點（通常爲每月）。這需要比傳統需求規格、設計規格等等更精細地追蹤構件，因爲您追蹤的是不同使用案例、測試案例等等的完成與否。

誠如 Walker Royce 所言：專案經理應該更重視測量和監督變更：需求、設計、程式碼中的變更，而非計算文字頁數和程式碼行（請參閱下面的「深入閱讀」）。Joe Marasco 補充說：留意變更，也留意大量生產。變更多次而回到相同起點的事物是更深刻問題的症狀。”

在積極面，聰明的專案經理可利用早期執行的具體軟體來取得一些早期可靠性點。它可以使用比已完成及已檢閱的書面作業更有意義的方式來炫耀進度，將幾百個勾選框取消勾選。而且，工程師也喜歡「示範如何運作」勝過「示範應該如何運作」。先示範，再記錄。

決定反覆的數目、持續時間和內容

第一步是什麼？對反覆式開發很陌生的經理通常很難決定反覆的內容。一開始，此規劃是由風險、技術和程式化所驅動，再由建構中系統的功能或特性的關鍵性來驅動。（RUP 提供決定反覆數目和持續時間的準則）。準則也會在生命週期內形成。在建構時，已準備好規劃來完成特定特性或特定子系統；在轉換時，已準備好來修正問題及增加健全性和效能。

設陷：在第一次反覆操之過急

我們上面有談過不要先處理困難的問題。在另一方面，背道而馳也是失敗的原因之一。在第一次或前幾次反覆中處理太多問題及涉及太多，已成爲一種趨勢。這樣無法確認其他因素：需要形成（訓練）團隊，需要學習新技術，需要獲取新工具。通常，許多開發人員對問題領域感到陌生。這通常導致第一次反覆嚴重超限運轉，而使整個反覆式方法遭到懷疑。或者，中止反覆— 在一切都沒有執行時宣告結束—基本上這是在沒有可導引的課程時宣告「勝利」，因而錯失反覆式開發的大部分好處。

若不能肯定或面臨危機時，以某種方式讓它變小一點（這適用於問題、解決方案和團隊）。記住，完整性是生命週期晚期的考量。適當的不完整卜應該是經理初期生命週期的考量。如果第一次反覆包含太多目標，請將它分割成兩個反覆，然後堅決以要先達成的目標作爲優先。”

在專案初期，最好爭取更簡單更保守的目標。注意，我們並沒有說它很容易辦到。在流程初期有完整獲得的結果有助於提高士氣。有許多專案錯過第一個里程碑就無法再回復。大部分錯過很多的專案即使後來力挽狂瀾，仍注定失敗。規劃一下，以確保您不會錯過初期里程碑太多。

設陷：太多反覆

首先，專案不應該將每日或每週建置與反覆混淆。由於在規劃、監督和評定反覆時有固定的額外負荷，因此，不熟悉這種方法的組織不應嘗試在其第一個專案上，以瘋狂的速率反覆。反覆的持續時間也應該考量組織的大小、其地理分散程度和包含的不同組織數目。再看一遍我們的「6 加減 3」的簡略規則。

設陷：重疊反覆

另一個極爲常見的設陷是讓反覆重疊太多。朝現行反覆的倒數第五個開始規劃下一個反覆，同時嘗試大量重疊活動（亦即，在完成現行反覆並從中學習之前，先啓動詳細分析、設計和編碼下一個反覆），在注視甘特圖表時，這樣做看似很吸引人，但會導致問題發生。有些人不表態貫徹和完成他們自己對現行反覆的貢獻；他們解決問題的回應性不夠；或決定只在下一個反覆才將所有意見列入考慮。軟體的部分組件尚無法支援已推進的工作等等。雖然一些人力可以轉向執行與現行反覆無關的工作，但應該儘量維持最少並只准例外情況。通常是因爲組織有些成員的技能太少或組織太過嚴苛而引起此問題： Joe 是一位分析師，這是他唯一可以或想要做的事；他不想參與設計、實作或測試。另一個負面的範例：大型指令和控制項專案有反覆重疊，以致於在某時間點基本上全部都平行執行，這需要管理階層將全體人員在反覆之間分割，但不希望將所獲得的經驗從一個反覆回饋到後面的反覆。

有關一些常見的非生產性反覆模式，請參閱「圖 3」。

Teams not synchronized -> chaos

First iteration too loaded -> panic

Too much overlap -> no feedback loop

圖 3：一些危險的反覆模式

好的專案經理和好的建構師

若要成功，軟體專案同時需要好的專案經理和好的建構師。若無好的架構，最佳管理和反覆式開發不會產生成功的產品。反之，如果專案管理不佳，再好的架構也會不幸失敗。因此，這是平衡問題，焦點完全放在專案管理上，並不會成功。專案經理不能漠視架構：需要同時具備架構專門知識和領域專門知識，才能決定要進入初期反覆的 20%。

設陷：使用同一人作爲 PM 和架構師

使用同一人作爲專案經理*和*架構師只對小專案（5-10 人）有效。爲了做最大的努力，而讓同一人同時扮演專案經理和建構師的角色，其結果通常是既無法妥善管理專案，又無法有完善的建構。首先，角色需要不同技能集。其次，角色本身或角色中的角色不止是一份全職的工作。因此，專案經理和建構師必須每天協調、彼此溝通和妥協。角色類似電影導演和電影製片。兩者都致力於一個共同目標，但所負責的工作層面完全不同。當同一人扮演兩個角色時，專案很少成功。

結論

在這個階段，您可能感到受挫：事前有這麼多問題，這麼多設陷。如果規劃和執行反覆式開發這麼困難，爲什麼要這麼費事？高興一點；有訣竅和技巧可以有系統地處理所有這些問題，而在達成可靠的高品質軟體產品方面，其收益遠大於不便之處。一些關鍵主題：主動攻擊風險，否則它們會攻擊你。（引用自 Tom Gilb 的書，列在*參考書和深入閱讀*的標題之下)。軟體優先。確認報廢和重做。選擇一起工作的專案經理和架構師。利用反覆式開發的好處。

瀑布式模型對經理很容易，對工程團隊比較困難。反覆式開發與軟體工程師的工作方式較爲一致，但管理複雜性較高。假設大部分團隊的工程師對經理的比例是 5 比 1（或更高），則這種犧牲是值得的。

雖然第一次執行時，反覆式開發比傳統方法更難，但它有真實的長期收益。一旦您得知執行的竅門之後，會發現自己變成能力更強的經理，也發現管理更大更複雜的專案越來越得心應手。一旦讓整個團隊都以反覆方式瞭解和思考之後，此方法會比傳統方法做更好的調整。

附註：John Smith、Dean Leffingwell、Joe Marasco 和 Walker Royce 分享他們在反覆式專案管理上的經驗，協助我撰寫本文。這篇文章有一部分併入我同事 Gerhard Versteegen 關於軟體開發的新書的第 6 章（請參閱下面的「參考書和深入閱讀」）。

關於作者

Philippe Kruchten 在 1987 年加入 Rational Software，目前是 Rational Fellow，總部設在 Vancouver, B.C。他之前擔任 Process Business Unit 的主任和總經理，領導過 Rational Unified Process 的開發。除了將焦點放在軟體架構和設計上之外，他也熱心於軟體工程練習和開發流程。他擁有機械工程的學位並擁有法文機構的電腦科學的博士學位。

參考書和深入閱讀

1. *Rational Unified Process 2000*, Rational Software, Cupertino, Ca., 2000.
2. Barry W. Boehm,鼈 piral Model of Software Development and Enhancement, *Computer*, May 1988, IEEE, pp.61-72.
3. Tom Gilb, *Principles of Software Engineering Management*, Addison-Wesley, 1988.
4. Philippe Kruchten, *The Rational Unified Process—An Introduction*, Addison Wesley Longman, 1999.
5. Walker Royce, *Software Project Management—A Unified Approach*, Addison Wesley Longman, 1999.
6. Gerhard Versteegen, *Projektmanagement mit dem Rational Unified Process*, Springer-Velag, Berlin, 2000.

Rational
the software development company

兩個總公司：

Rational Software
18880 Homestead Road
Cupertino, CA 95014
電話：(408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
電話：(781) 676-2400

免付費專線：(800) 728-1212
電子郵件：info@rational.com
網址：www.rational.com
國際辦事處：www.rational.com/worldwide