在小型專案上使用 Rational Unified Process：

依 eXtreme Programming 擴充

Gary Pollice

Rational Software 白皮書

TP 183, 3/01

Rational®
the software development company

目錄

摘要

Rational Unified Process® 或 RUP® 產品是一個完整的軟體開發流程架構，有附帶數個立即可用的實例。從 RUP 衍生的流程不盡相同，有輕量型—以簡短產品週期處理小專案的需求—也有處理大型分散式專案小組更廣泛的需求的更複雜流程。所有類型和大小的專案已成功地使用過 RUP。本白皮書將說明如何在輕量型到小型專案中套用 RUP。我們會說明如何在完整專案更廣泛的環境定義內有效套用 eXtreme Programming (XP) 技術。

簡介

簡短報導

有一天早上，經理過來問我，能不能花幾個星期的時間來設定一個簡單的資訊系統，供公司已經開始的一項新活動使用。由於我已經對目前專案感到厭煩，希望有新的運作來刺激一下，因此，我馬上答應接受這個機會—沒有我目前工作的大型組織的繁文褥節和程序的牽累，我可以快速推動及開發出很棒的新解決方案。

一切順利展開。前 6 個月，我大部分是獨立作業—時間久，樂趣多。我的生產力令人刮目相看，而且交出了職業生涯中最好的成績單。開發週期很快，我通常每幾週就會製作出系統的一些主要新零件。與使用者的互動簡單而直接—我們全都是緊密結合的團隊的一員，可以省掉一堆形式和文件。在設計上也沒有什麼形式；程式碼就是設計，設計就是程式碼。一切是那麼美好！

但好景不常。隨著系統的成長，要做的工作更多。隨著問題的改變，現有的程式碼必須跟著發展，對於我們必須完成的工作，我們修改了一些看法。我雇用幾個人來幫忙開發。我們像同一個單位在工作，常常配對來處理問題。這樣可加強溝通，也可以省掉形式。

一年過去。

我們繼續增加人手。團隊成長成三個人，然後五個人，然後七個人。每次新人一來，都會有長時間的學習曲線，而且因爲缺乏經驗，也更難瞭解和說明整個系統，即使只是一個概要也不容易。我們開始拍攝白板圖，以更正式的方式來顯示系統的整體結構和主要概念及介面。

我們仍舊使用測試作爲主要工具，來確認系統執行它必須執行的工作。由於許多新人都是「使用者」，因此我們發現在專案初期有作用的非正式需求和個人關係已經不夠。要想出我們打算建置什麼，需要更久的時間。因此，我們保留討論的書面記錄，這樣就不必不斷回想已經決定的事。我們也發現說明需求和使用實務有助於在系統上訓練新的使用者。

隨著系統的大小和複雜性增加，會發生一些非預期的情況—系統的架構需要特別注意。在早期，架構主要存在於我的腦海裡，後來會存在於幾張草稿上或活動掛圖上。然而，專案的人越多，越難控制架構。因爲並不是每一個人都有和我一樣的歷史背景，所以他們看不出架構某一項特定變更的含意。我們必須在系統上以更精確的詞彙定義架構限制。可能影響架構的變更需要團隊的共識和我的核准。我們好不容易才發現這一點，而且經過一些困難的學習經驗之後，我們不得不承認架構的重要性。

這是真實報導。它只說明此專案一些痛苦的經驗。這些經驗只有在某一點上才是獨特的：我們當中有幾個人是從頭到尾參與，歷經了好幾年。專案的人通常來來去去，看不見他們的動作對下游帶來的影響。

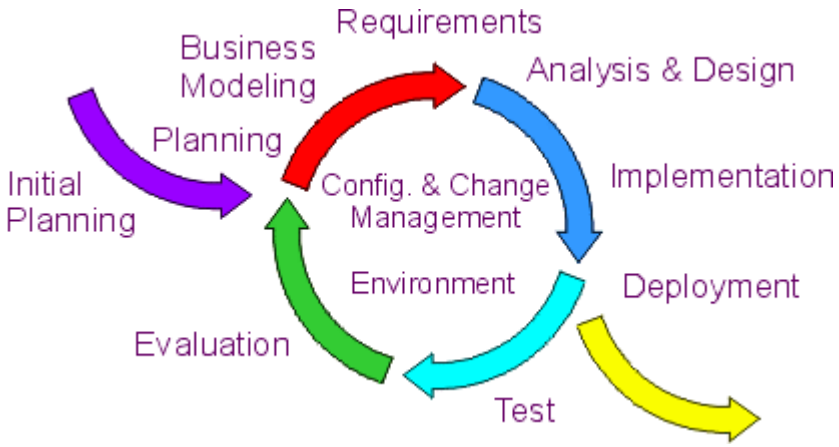
此專案受到一些流程的幫助。太多流程會擋路，但缺少流程則會帶來新的風險。和投資在高風險股票的人一樣，只看到高報酬率，使用太少流程的團體忽略其專案環境的主要風險，這些人都是「抱最大的希望，但不作最壞的打算」。

概觀

本白皮書討論如何在剛才所描述的專案中套用流程。我們的焦點在於取得流程的「適當層次」。瞭解開發團隊所面臨的挑戰和它操作的商業環境，才可衍生流程形式的適當層次。瞭解這些挑戰之後，我們就只提供足夠流程來減輕風險。不管是輕量型或其他類型，並沒有「一體適用」的流程。在下列各節，我們將探索一個概念：流程的適當層次就是風險的作用。

我們的焦點在於如何使用兩個受歡迎的方法來取得流程的適當層次：Rational Unified Process 或 RUP 和 eXtreme Programming (XP)。我們會示範如何爲小專案調整 RUP，及它如何處理 XP 未考慮到的許多區域。兩者的組合則提供專案小組所需的指引，來減輕風險及達成其交付軟體產品的目標。

RUP 是 Rational Software 所開發的流程架構。它是以六個業界認同的最佳作法爲基礎的反覆式開發方法（請參閱 RUP 附錄）。在這段時間，以 RUP 爲基礎的專案會經歷四個階段：初始階段、詳述、建構和轉換。每一個階段包含一或多個反覆。在每一個反覆中，您將不同的精力花在每一個規範中，例如需求、分析和設計、測試等等。RUP 的主要動因是*風險減輕*。數千個 Rational 客戶和夥伴在數千個專案中使用之後，RUP 已獲得修正。下圖說明典型反覆的流程：



典型反覆流程

作爲風險如何形成流程的範例，或許我們可以自問，是否應該建立商業模型。不瞭解企業可能導致我們建置錯誤系統，如果有如此重大的風險，我們應該執行一些商業模型。我們需要正規的建模人力嗎？這要視讀者而定—如果小團隊將非正式地使用此結果，我們只要提出一些非正式的注意事項。如果組織中的其他人要使用此結果或檢視此結果，我們可能必須投入額外的人力，而且更加重視呈現方式的正確性和易懂性。

您可以自訂 RUP 來符合幾乎任何專案的需求。如果立即可用的流程或導覽圖都不適合您的特殊需求，您可以輕鬆製作自己的導覽圖。導覽圖說明專案如何規劃使用此流程，及代表該專案的特定流程實例。這表示 RUP 可以根據需要而成爲輕量型或重量型，本白皮書對此有所說明。

XP 是小專案的輕量型程式碼中心流程（請參閱 XP 附錄）。它是 Kent Beck 的創作，並在 1997 年使軟體業界注意到 Chrysler Corporation 的 C3 薪資專案。和 RUP 一樣，它也是以反覆爲基礎，這些反覆收錄了數個作法，例如袖珍版、簡單設計、測試和連續整合。XP 推銷數個對適當專案和情況有效的技術；然而，仍有一些隱藏的假設、活動和角色。

RUP 和 XP 來自不同原理。RUP 是流程元件、方法和技術的架構，您可以將它套用至任何特定軟體專案；我們希望使用者能夠將 RUP 特殊化。在另一方面，XP 是一個有更多限制的流程，需要新增項目才能適合完整的開發專案。這些差異說明整體軟體開發社群的觀念：大型系統的人視 RUP 爲問題的解答；小型系統的社群視 XP 爲問題的解決

方案。根據我們的經驗，大部分軟體專案介於這兩者之間—嘗試達到適合其狀況的流程的適當層次。此範圍的任何一端對他們來說都不夠。

當您結合 RUP 的廣度和一些 XP 技術時，可達到訴諸於專案所有成員之適當數量的流程，及處理所有主要專案風險。對於在信任度相當高的環境中工作的小型專案小組而言（其使用者是團隊整體的一部分），XP 可以運作得非常好。當團隊變得更加分散，程式碼庫也成長，或架構沒有完善定義時，您會有其他的需求。對於具有「契約式」使用者互動的專案，您需要的不止是 XP。RUP 是一個架構，您可以根據需要，用更健全的技術集來延伸 XP。

本白皮書的其餘內容是描述以 RUP 四個階段爲基礎的一個小流程。在每一個階段中，我們識別產生的活動和構件。[1] 雖然 RUP 和 XP 識別不同角色和責任，但我們並未在這裡處理這些差異。對於組織或專案，實際專案成員必須與流程中的適當角色相關聯。

專案開始 — 初始階段

初始階段對新的開發工作很重要，您必須先在這個階段處理重要的商業風險和需求風險，然後才能進行專案。對於聚焦於現有系統的增強功能的專案，其初始階段較短，但仍然成爲焦點，以確保專案值得進行而且行得通。

在初始階段期間，您建立企業案例來建置軟體。*願景*是初始階段期間產生的主要構件。它是系統的高階說明。它告訴大家系統是什麼，並指出誰可以使用系統，爲何使用，必須有哪些特性，以及有哪些限制存在。*願景*可能很短，也許只是一兩個段落。*願景*通常包含軟體必須提供給客戶的重要特性。

下列範例顯示針對 Rational 外部網站的再造工程之專案而寫的小*願景*。

> *透過動態個人化網站呈現，提供訪客自助式、支援和已設定目標的內容，來加強與客戶的關係，將 Rational 定位爲電子開發（工具、服務和最佳作法）的全球領導者。新的流程和賦予能力的技術，能夠讓內容提供者透過簡化自動化解決方案，加速內容的發行和品管。*

在 RUP 裡指定的四個重要初始階段活動爲：

- **使專案的範圍公式化** — 如果我們要製作系統，需要知道它的什麼，以及它如何滿足關係人。在此活動中，我們擷取環境定義和最重要的需求明細，來衍生產品的驗收準則。
- **規劃及準備企業案例** — 有了*願景*作爲指引之後，我們定義風險減輕策略，開發起始專案計劃，識別已知的成本、排程和利潤交易。
- **合成候選的架構** — 如果考慮的系統一點也不新奇，而且有一個眾所周知的架構，您可以跳過這個步驟。當我們知道客戶需求之後，就會分配時間來探索可能的候選架構。新技術爲軟體問題的新的和改良的解決方案帶來可能性。在流程中早點花時間評估購買與建置的交易，及選取技術，還有開發起始原型，可減少專案的一些主要風險。
- **準備專案環境** — 任何專案都需要專案環境。無論您是使用一些 XP 技術（例如雙人程式設計）或是要判斷實體資源、軟體工具和程序所需要的更傳統的技術，團隊都會遵守。

對小專案執行初始階段，不會花費很多「處理時間」。您通常可以在幾天以內完成初始階段。下列各節說明此階段預期的構件，*願景*不包含在內。

核准的企業案例

關係人有機會同意，從企業觀點來看專案是否值得進行。RUP 和 XP 同意最好盡早發現專案不值得進行，而不要在注定失敗的專案上浪費寶貴資源。XP，如 *Planning Extreme Programming*[2] 對於專案如何實現及包含哪些角色模糊不清

[1] XP 定義三個階段：探索、確定和操縱。這些並沒有好好對映到 RUP 階段，因此，我們選擇使用四個 RUP 階段來描述此流程。

[2]所描述，這本書是目前有 eXtreme Programming 所發行的三本書的其中一本。

（在現有的商業或系統的環境定義中，它看起來最清楚），但在其詳述階段，XP 可以處理同等的 RUP 初始階段構件。

不論您像 XP 一樣非正式考量商業問題，或使企業案例成爲頭等專案構件，像 RUP 一樣，您都需要考慮它們。

風險清單

您需要在整個專案內維護風險清單。這可以是一份簡單的風險清單，內有規劃好的減輕策略。風險有優先順序。與專案相關聯的任何人都可以查看有哪些風險，以及在任何時間如何處理它們。Kent Beck 說明 XP 所處理的一組風險，以及如何處理它們，但未提供處理風險的一般方法。[1]

初步的專案計劃

資源評估、範圍和階段計劃已併入此計劃中。在任何專案上，這些評估會不斷改變，您必須監督它們。

專案驗收計劃

您有沒有正式計劃，視專案類型而定。您必須決定客戶要如何評估專案成功與否。在 XP 專案上，需要由客戶建立驗收測試。在更通用的流程中，客戶可能沒有真的建構測試，但驗收準則必須由客戶直接驅動，或透過直接與客戶互動的另一個角色來驅動，例如系統分析師。可能還有其他驗收準則，例如一般使用者文件和說明的製作，XP 並未涵蓋這些。

起始詳述反覆的計劃

在以 RUP 爲基礎的專案中，您會在上一個反覆結束時詳細規劃每一個反覆。在反覆結束時，您會依據反覆開始時指出的準則來評估進度。XP 提供一些好的技術來監督和測量反覆的成功與否。測量值很簡單，您可以輕易將它們併入反覆計劃和評估準則中。

詳述

詳述階段的目標是要建立系統架構的基準線，以便在建構階段提供設計和實作成果的主體的穩定基礎。此架構是基於考量最重要的需求（對系統架構有重大影響的那些需求）和對風險的評量而推斷出來的。架構的穩定性是透過一或多個架構原型來評估。

在 RUP，設計活動焦點在於系統架構的概念，對於軟體密集系統，則焦點爲軟體架構的概念。使用元件架構是在 RUP 具體化的軟體開發的六個最佳作法的其中之一，它建議花時間開發及維護架構。對此工作所花的時間減輕與脆弱而沒有彈性的系統相關聯的風險。

XP 以「隱喻」來取代架構的概念。隱喻擷取架構的一部分，架構的剩餘部分則發展成爲程式碼開發的自然結果。XP 假設架構會從產生最簡單設計中浮現，並繼續重構程式碼。

在 RUP，架構不只是隱喻。在詳述期間，您會建構可執行的架構，從中可減少與符合非功能需求相關聯的風險，例如效能、可靠性和健全性。在閱讀 XP 文件時，可推斷 RUP 在詳述階段所做的一些要求（尤其是過度集中在 XP 所謂的基礎架構上）是白費工夫。XP 說，在需要之前花費在建置基礎架構上的工夫，會造成過度複雜的解決方案，所產生的結果對客戶毫無價值。在 RUP，架構和基礎架構不一樣。

在 RUP 與 XP 之間，架構的方法非常不一樣。RUP 建議您注意架構，以避免隨時間而增加的範圍、其他的專案大小和新技術的新增等相關聯的風險。XP 採用現有的架構，或假設該架構簡單易懂，可在您撰寫程式碼時逐步形成。XP 建議不要爲明天設計，但要爲今天實作。我們相信，如果您的設計越簡單，明天就會自行負責。RUP 請您評定

[1]在 *eXtreme Programming eXplained*，Kent Beck 說明 8 個風險及 XP 如何處理這些風險（第 3-5 頁）。請讀者查看它們，並判斷它們是否足夠。我們相信還有許多其他風險，一般處理風險策略是任何流程的必要部分。

這種提議的風險。如果系統或其中一部分未來必須改寫，XP 表示與現在規劃可能性相比，它還是比較好的選擇，而且通常比較不昂貴。對於某些系統而言是沒錯，而使用 RUP 的話，在詳述階段期間的風險考量會讓您得到此結論。RUP 不主張所有系統都是如此，而且根據經驗，對於越大越複雜、前所未見的系統，它會變成一項災難。

太過注意未來的可能性（也許絕對不會發生）會浪費時間，對未來有適度的關注需要審慎一點。有多少家公司可以承受不斷改寫、甚至重構程式碼呢？

在任何專案上，您應該在詳述期間至少執行這三個活動：

- **定義、檢驗和建立架構基準線** — 使用風險清單來開發在初始階段候選的架構。我們對於確保想像的軟體合用感到興趣。如果選擇的技術並不稀奇，或系統並不複雜，則這項作業不會花太多時間。如果您要新增至現有的系統，但現有的架構並不需要變更，則不需要這項作業。當真正的架構風險出現時，您不想要讓架構聽其自然。

  在此活動的過程中，您可以執行一些元件選取，並做出購買/建置/重複使用的決定。如果這需要很多工夫，您可以將它劃分成個別活動。

- **修正願景** — 在初始階段期間，您會開發願景。當您在判斷專案的可行性，且關係人有時間檢視及評論系統時，願景文件和需求可能會變更。它的修訂和需求通常發生在詳述期間。等到詳述結束時，您已經完全瞭解驅動架構和規劃決策的最重要使用案例。關係人需要同意，在現行架構的環境定義中，如果您執行現行計劃來開發完整系統，則現行願景有可能發生。在後續反覆期間，變更量應該減少，但您想要配置一些時間在每一個反覆的需求管理上。

- **建立建構階段的反覆計劃及建立基準線** —請立即填寫計劃的詳細資料。在每一個建構反覆結束時，您已重閱計劃並依需要調整。調整發生的原因通常是不正確評估人力、商業環境變更或需求變更。使用案例、實務和技術人力優先順序化，然後將它們指派給反覆。在每一個反覆結束時，您計劃擁有一個有效的產品來提供價值給關係人。

您可以在詳述期間執行其他活動。我們建議建立測試環境並開始開發測試。雖然詳細的程式碼可能不存在，但您仍然可以設計及實作整合測試。程式設計師應該準備開發單元測試，並且知道如何使用爲專案選取的測試工具。XP 建議在程式碼之前撰寫測試。尤其當您要新增至現有的程式碼主體時，這真是個好點子。不過，您選擇執行測試，建立一般測試體系的時間是在詳述階段。

RUP 所描述的詳述階段包含 XP 的探索和確定階段的元素。XP 處理技術風險（例如新奇和複雜）的方法是 鼈 pike 解決方案，亦即花一些時間嘗試，然後再評估人力。此技術在許多案例中很有效，如果有單一使用案例或報導中沒有包含的更大風險，您需要申請多一點人力，以確保系統成功及精確的人力評估。”

您通常在詳述期間更新初始階段的構件，例如需求和風險清單。在詳述期間可能出現的構件如下：

- **軟體架構文件 (SAD)** — SAD 是複合構件，它提供整個專案的技術資訊的單一來源。在詳述結束時，它可能包含在架構上重要的使用案例的詳細說明，及主要機制和設計元素的識別。當專案加強現有的系統時，您可以使用先前的 SAD，或決定沒有該文件比較保險。無論如何，您應該執行通往此文件的思考流程。

- **建構的反覆計劃** — 您在詳述期間規劃建構反覆的數目。每一個反覆有特定的使用案例、實務和已指派給它的其他工作項目。此資訊會在反覆計劃中擷取及建立基準線。請檢視及核准計劃，這是詳述結束準則的一部分。

在非常小而簡短的專案上，您可以合併詳述反覆與初始階段和建構。基本活動仍然會執行，但反覆規劃和檢視的資源會減少。

起始使用案例模型

雖然這聽起來很正式且令人生畏，但它是非常直接明確的。使用案例對應於客戶在 XP 所寫的「報導」。差別是使用案例是參與者所起始的完整動作集，參與者是指在系統外提供可見價值的人或事物。使用案例可包含數個 XP 報

導。爲了定義專案的範圍，RUP 建議在初始階段識別使用案例和參與者。從使用者觀點將焦點放在完整動作集，有助於將系統分割成提供價值的組件。這有助於決定適當的實作特性，讓我們在每一個反覆結束時有東西可以交付給客戶（可能早期初始階段和詳述反覆除外）。

RUP 和 XP 都可以幫助我們確保不會已完成完整系統的 80%，卻沒有完成任何可交付項。我們一直希望能夠發行系統，以提供一些價值給客戶。

此時的使用案例模型是以少數或完全沒有詳細資料來識別使用案例和參與者。它可以是簡單的文字或是以手或繪圖工具所繪製的 UML（統一建模語言）圖。此模型幫助我們確保已包括關係人需要的適當特性，而且沒有忘記任何東西，並讓我們輕易看到整個系統。使用案例基於數個因素優先順序化，例如風險、對客戶的重要性和技術困難度。

初始階段沒有一個構件需要太正式或太大。請根據您的需要，讓它們簡單或正式。XP 包含規劃和系統驗收的指引，而 RUP 則在專案初期新增一些東西。這個小小的新增因爲處理更完整的風險集而產生很大的效益。

建構

建構的目標是要完成系統的開發。在某種程度來說，建構階段是一種製造過程，您可以強調管理資源和控制作業，使成本、排程和品質最佳化。在這方面，管理心態經歷初始階段和詳述期間的智慧財產開發，到建構和轉換期間可部署產品的開發這樣的轉換。

XP 全神貫注於建構。建構階段是您製作程式碼之處。XP 階段是作爲規劃用途，但 XP 的焦點在於建置程式碼。

每一個建構反覆有三個主要活動：

- **管理資源和控制流程** — 每個人都需要知道誰會執行什麼，以及什麼時候執行。您必須確定工作量不超出您的能力範圍，且工作有依照排程進行。
- **開發和測試元件** — 您建置必要的元件來滿足使用案例、實務和反覆的其他功能。您以單元測試和整合測試來測試它們。
- **評定反覆** — 當反覆完成時，您需要判斷是否滿足反覆的目標。如果沒有，您需要重新優先順序化及管理範圍，以符合您的交付日期。

不同類型的系統需要不同技術。RUP 提供軟體工程師不同準則，並協助建置適當的元件。需求是以使用案例和增補（非功能）需求的形式詳細說明，讓工程師能夠執行其工作。RUP 中的數個活動提供設計、實作和測試不同種類之元件的指引。有經驗的軟體工程師並不需要詳細查看這些活動。經驗不夠的工程師則發現對最佳作法的幫助很大。每一個團隊成員可以依需要，或深或淺地投入流程中。然而，他們全都查看單一流程知識庫。

在 XP，報導會驅動實作。在 *Extreme Programming Installed* 這本書中，Jeffries 以及其他人都說，報導是與程式設計師「對話的承諾」。[1] 持續而有效的溝通是件好事。雖然總有一些細節需要澄清，如果報導不夠詳細，讓程式設計師無法執行大部分工作，這樣就是還沒有準備好。使用案例必須夠詳細，程式設計師才能實作使用案例。在許多案例中，程式設計師幫助撰寫使用案例的技術詳細資料。Jeffries 和其他人也說，對話將記錄並附加到報導中。RUP 也建議如此，但使用案例規格的形式除外，它可以依需要作爲非正式的。對話結果的擷取和管理是您必須處理的作業。

XP forté 是建構。有一些適合大部分團隊的智慧金礦和指引。最值得注意的 XP 作法如下：

- **測試** — 程式設計師持續撰寫測試來配合其程式碼。測試反映報導。XP 極力主張您先撰寫測試，這是很好的作法，因爲它強迫您深入瞭解報導，及在必要時詢問更多問題。不論您是在程式碼之前或之後撰寫，請一定要寫！將它們新增至測試套組中，並確定每次程式碼變更時都會執行它們。

[1]此說明爲 Allistair Cockburn 所寫。

- **重構** — 不斷重組系統，但不變更其行爲，以使它更加簡單或增加彈性。您需要判斷這是否是您團隊的理想作法。對某人而言很簡單的事，對別人而言可能很複雜。舉例來說，專案中有兩個非常聰明的工程師每晚花時間重寫對方的程式碼，因爲他們都認爲它太過複雜。他們不斷在隔天破壞團隊其他人的建置。測試雖然有幫助，但如果他們沒有陷入這場程式碼戰爭，對團隊會更有利。
- **雙人程式設計** — XP 聲稱雙人程式設計可在更少的時間產生更好的程式碼。這種情況有憑有據。[1] 如果您實作此作法，則要考慮許多人爲和環境因素。程式設計師願意嘗試這麼做嗎？實體環境有空間給兩位程式設計師在單一工作站有效率地一起工作嗎？您如何對待在遠距離工作或在其他地點的程式設計師？
- **連續整合** — 整合和建置系統通常一天不止一次。這是確保程式碼結構性完整的好辦法，而且可以透過整合測試進行持續的品質監督。
- **群體擁有權** — 任何人都有許可權可隨時變更任何程式碼。一組好的單元測試將減少此作法的風險，XP 相信這個事實。讓每個人熟悉每件事的好處是有停損點的——萬行程式碼、兩萬行，一定不超過五萬行？
- **簡單設計** — 就像重構一樣，系統的設計也會不斷修改，來消除複雜性。同樣地，您需要決定修改幅度多大，以免讓它無法運作。如果您在詳述期間花時間設計架構，我們相信簡單設計會出現，且更快變穩定。
- **程式碼撰寫標準** — 這一向是很好的作法。不管是什麼標準，只要您擁有它們，而且大家都同意使用它們就好。

RUP 和 XP 都同意您必須管理（或操縱）反覆。測量值可以提供很好的規劃資訊，因爲它們可幫助您選擇什麼才是對團隊最好的。有三件事要測量：時間、大小和問題。您可以藉此取得一切有趣的統計資料。XP 提供簡單測量值，用來判斷進度和預測成就。這些測量值把已完成的報導數、通過的測試數和這些統計資料內的趨勢集中。XP 有充足的理由來使用最少量的測量值，因爲看得越多不一定會提高專案成功的機會。RUP 對於您可以測量及如何測量提供準則，並提供測量範例。無論如何，測量值必須簡單、客觀、容易收集、容易解譯，而且不容易誤解。

在建構反覆期間會出現哪些構件？視反覆是早期或晚期建構反覆而定，您可以建立下列其中之一：

- **元件** — 元件代表一個軟體程式碼組件（原始檔、二進位或執行檔），或包含資訊的檔案，例如開機檔或 Readme 檔。元件也可以是其他元件的聚集，例如由數個執行檔組成的應用程式。
- **培訓資料** — 根據使用案例，如果系統有充足的使用者介面層面，請盡早製作使用者手冊和其他培訓資料的初稿。
- **部署計劃** — 客戶需要系統。「部署計劃」說明安裝、測試和有效地將產品轉換成使用者群組所需的作業集。對於以 Web 爲中心的系統，我們發現「部署計劃」的重要性越來越高。
- **轉換階段反覆計劃** — 當您接近爲使用者部署軟體的時間，您需要完成及檢視「轉換階段反覆計劃」。

全部都是程式碼的問題嗎？

除了 RUP 和 XP 之間的架構方法的差異之外，還有別的。其中之一是您溝通設計的方式。XP 指出，程式碼就是設計，設計就是程式碼。程式碼和本身總是一致的，這點無庸置疑。我們相信花點工夫擷取及溝通設計（非程式碼）是值得的。這個小故事足以闡明這一點。

一位工程師有兩種軟體專案的經驗，他將設計放在程式碼中，而且程式碼是唯一可以找到設計資訊的地方。這兩個專案都與編譯器有關：一個是要改進及維護 Ada 編譯器的最佳化程式，一個是要將編譯器的前端系統移轉到新平台上，並鏈結第三方程式碼產生器。

編譯器技術雖然複雜，但眾所周知。在兩個專案上，工程師想要設計的概觀和編譯器（或最佳化程式）的實作。在每一個情況下，他都會收到一疊程式碼列表，有幾英寸那麼厚，而且有人告訴他要「參考這裡」。他可能爲了一些

[1] *Strengthening the Case for Pair Programming*, IEEE Software, July/August, 2000.

建構完善的圖和一些支持的文字而付出一切。但最佳化程式專案從來不會完成。編譯器專案真的可以順利完成，而且有很棒的程式碼品質，因爲有伴隨程式碼開發了一個很長的測試集。工程師花了幾天的時間置身於除錯程式的程式碼中，嘗試瞭解它做了什麼。從個人的煎熬和團隊付出的代價來看，這樣做並不值得。我們並沒有像 XP 建議的 40 小時之後停止的選項，我們以很多的大膽作爲來完成此工作。

主要問題在於光有程式碼—不論寫得多好—無法得知所解決的問題是什麼，它只傳達問題的解決方案。有些需求的文件在原始使用者和開發人員已離開很久之後才煞費苦心地解釋原始目標。爲了維護系統，您通常需要知道原始專案小組的想法。有些高階設計文件是類似的 – 通常程式碼的摘要層次太低，以致於無法確實表達整個系統的目的。這在物件導向系統中更是如此，光看類別很難或根本無法追蹤其執行緒的執行。設計文件指引您以後遇到問題時該參考什麼—以後*總是*會有問題。

這個故事的寓意是指花時間擷取和維護設計文件確實有幫助。它減少誤解的風險，並可加速開發。XP 方法是花幾分鐘時間概述設計或使用 CRC 卡。[1] 團隊並不維護這些，但會去使用程式碼。有一個隱含的假設：這些作業太過於簡單，所以我們都知道該如何進行。即使我們知道，下一個出現的人不一定這麼幸運。RUP 建議您花多一點時間擷取和維護這些設計構件。

轉換

轉換的焦點是要確保軟體可供一般使用者使用。轉換階段包含測試產品以準備發行，並依據使用者意見做小幅的調整。在生命週期的這個時刻，使用者意見需要聚焦在細部調整產品、配置、安裝和可用性問題上。

提早且頻繁地發行，是一個好辦法。但是，我們所謂的發行是指什麼呢？XP 對此交待不清，而且沒有處理要發行商業軟體所需的製造問題。您或許可以在內部專案上走捷徑避開一些問題；但即使如此，您還是需要文件、培訓等等。支援和變更管理是什麼？期待駐點客戶也會控制這些，是否不切實際？Bruce Conrad 在 InfoWorld 中對 XP 的評論指出，[2] 客戶不想得到一直在變更的軟體。您必須權衡客戶快速獲得變更資訊的優點與變更和可能帶來的不穩定的缺點。

當您決定要發行時，需要提供給一般使用者的不只是程式碼而已。「轉換」中的活動和構件指引您完成軟體開發流程的這個部分。活動集中在爲客戶提供可用的產品。主要轉換活動如下：

- **將一般使用者支援資料定案** — 此活動很簡單，只是核對清單上的項目而已，但您需要確定組織已準備好支援客戶。
- **在客戶環境中測試可交付的產品** —如果您可以就地模擬客戶環境，就模擬吧。否則，請到客戶那裡安裝軟體，以確定它可以運作。您不想要尷尬地對客戶說：「但產品在我們的系統上是可以運作的」。
- **根據客戶意見細部調整產品** — 可能的話，規劃一或多個上市前測試期，將軟體交付給限定數量的客戶。如果您這麼做，需要管理上市前測試期，並在「結束階段」考量客戶的意見。
- **交付最終產品給一般使用者** — 根據軟體產品和發行的類型，有許多詳細資料是關於要處理的套裝、製造和其他製作問題。您很少只將軟體放在目錄中，然後寄出郵件，讓客戶群知道軟體垂手可得。

和其他大部分階段一樣，您流程的形式和複雜度也不同。然而，如果您未注意部署詳細資料，就會使幾個月以來認真的開發工作無效，結果產生的產品在目標市場上一敗塗地。

您可以在轉換階段製作數個構件。如果產品以後會有新的版本（又有幾個產品不是這樣呢？），您要開始確認下一版的特性和問題修正。任何專案的重要構件如下：

- **部署計劃** — 將您在建構階段啓動的部署計劃定案，並使用它作爲交付客戶的導覽圖。

[1] CRC（類別、責任和合作）卡是由 Kent Beck 和 Ward Cunningham 開發出來的，教導實踐者物件導向設計的原則。

[2] http://www.infoworld.com/articles/mt/xml/00/07/24/000724mtextreme.xml

- **版本注意事項** — 很少有軟體產品沒有最新指示給一般使用者。請提供它，並在注意事項中使用有用而一致的格式。
- **培訓資料和文件** — 這些資料採用的形式很廣泛。您要線上提供一切嗎？您有教學指導嗎？產品說明完整而有用嗎？不要假設客戶與你心靈相通。幫助客戶成功，您才算成功。

摘要

建置軟體不只是撰寫程式碼而已。軟體開發流程必須聚焦於爲客戶帶來高品質產品所需的一切活動。完整的流程不一定繁忙。我們已示範如何聚焦於專案的主要活動和構件，來擁有一個小而完整的流程。請執行活動或製作構件（只要它們有助於減輕專案的風險）。使用專案小組和組織需要的流程和形式，數量不拘。

RUP 和 XP 不一定互斥。藉由合併這兩種方法的技術，您可以達成一個流程來幫助您以比現在更快的速度交付品質更好的軟體。Robert Martin 說明一個流程，它叫作 *dX 流程*，他聲稱這個流程與 RUP 相容。[1] 它是從 RUP 架構建置的流程實例。

好的軟體流程包括業界認同的最佳作法。最佳作法是在真實的軟體開發組織中經過一段時間測試的那些作法。XP 方法成爲目前關注的焦點。它以程式碼爲中心，提出具有最少流程額外負荷和最大生產力的承諾。XP 有許多技術可保證在適當情況下加以考量和採用。

XP 聚焦於報導、測試和程式碼—它討論一些規劃，但對計劃的擷取著墨不多。XP 暗示您可以製作其他東西，例如「利用幾張卡來進行 CRC 設計，或概述一些 UML」，或「不要製作沒有在使用的文件或其他構件」，但都只是順便處理而已。RUP 請您考慮只製作當您闡述和更新開發計劃時有用且必要的東西，並識別這些東西是什麼。

RUP 是處理完整軟體開發生命週期的流程。它聚焦於數千個專案所開發的最佳作法。我們鼓勵研究和發明可產生最佳作法的新技術。當新的最佳作法出現時，我們期待將它們併入 RUP 中。

附錄 A：Rational Unified Process

Rational Unified Process 或 RUP 爲軟體開發提供有規範的方法。它是*流程產品*，由 Rational Software 開發及維護。針對不同類型的軟體專案，它附帶數個立即可用的導覽圖。RUP 也提供資訊協助您使用其他 Rational 工具進行軟體開發，但若要有效應用到組織內，並不需要 Rational 工具；可以與其他供應商產品整合。

RUP 提供軟體專案一切指引。它不需要您執行任何特定活動或產生任何特定構件。它有提供資訊和準則，讓您決定何者適合自己的組織。如果立即可用的導覽圖都不適合您的專案或組織，則它也提供準則來幫助您調整流程。

RUP 強調在現代軟體開發中採用特定*最佳作法*，是在開發新軟體時減輕內在風險的方法。這些最佳作法如下：

1. 反覆式開發
2. 管理需求
3. 使用以元件爲基礎的架構
4. 視覺化模型
5. 連續驗證品質
6. 控制變更

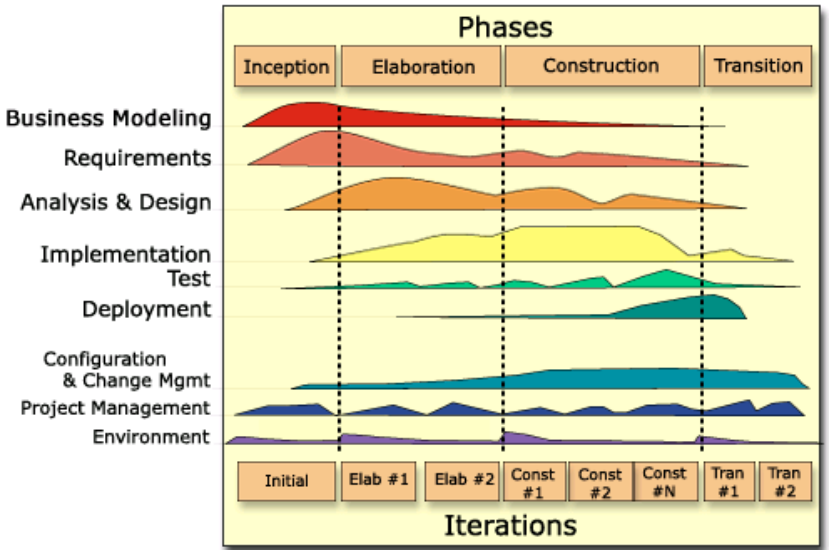
這些最佳作法編入下列的 Rational Unified Process 定義：

- *角色* — 執行的作業集和擁有的構件。

[1] http://www.objectmentor.com/publications/RUPvsXP.pdf。這是 Martin、Booch 和 Newkirk 合著的一本尚未發行的書當中的一章。

- *規範* — 軟體工程工作的焦點區域，例如需求、分析和設計、實作和測試。
- *作業* — 產生和評估構件的方式之定義。
- *構件* — 使用的工作產品，在作業執行時產生或修改

RUP 是一種反覆式流程，可識別任何軟體開發專案的四個階段。在這段時間，專案會經歷初始階段、詳述、建構和轉換階段。每一個階段包含可讓您產生可執行但或許不完整的系統的一或多個反覆（除了初始階段之外）。在每一個反覆期間，您以不同的明細層次來執行數個規範的活動。下列是 RUP 的總覽圖。



RUP 總覽圖

*Rational Unified Process 簡介第二版*是 RUP 的最佳概觀。您可以在 Rational Software 網站上找到進一步資訊和 RUP 的評估：www.rational.com。

附錄 B：eXtreme Programming

eXtreme Programming (XP) 是軟體開發規範，由 Kent Beck 在 1996 年開發出來。它以這四個價值爲基礎：溝通、簡單化、意見和勇氣。它強調客戶和開發團隊成員之間的持續溝通，並在進行開發時有駐點客戶隨伺在側。駐點客戶決定建置的內容及順序。您可以不斷重構程式碼及產生最小非程式碼構件集，來實現簡單化。許多簡短發行和持續單元測試都是意見機制。勇氣是指做對的事，即使一般人並不這麼做。它表示要對您可以做和不能做的事誠實以對。

有十二個 XP 作法支援這四個價值。它們是：

- **規劃階段** — 透過優先順序化的報導和技術評估的組合，來決定下一版的特性。
- **袖珍版** — 經常以迷你增量版的形式向客戶發行軟體。
- **隱喻** — 隱喻是對系統如何運作的簡單報導或說明。
- **簡單設計** — 保持簡單程式碼來保持簡單設計。不斷尋找程式碼的複雜處，並立刻將它移除。
- **測試** — 客戶撰寫測試來測試報導。程式設計師撰寫測試來測試可能中斷程式碼的任何東西。請在撰寫程式碼之前撰寫測試。

- **重構** — 這是從程式碼中移除重複和複雜處的簡化技術。
- **雙人程式設計** — 兩個程式設計師組成的團隊，他們在同一部電腦開發所有程式碼。一人撰寫程式碼或*驅動程式*，另一人同時檢視程式碼的正確性和易懂性。
- **群體擁有權** — 每個人都擁有所有程式碼。這表示每個人都有能力隨時變更任何程式碼。
- **持續整合** — 每當實作作業完成時就建置和整合系統，一天數次。
- **四十小時一週** — 如果程式設計師太累，工作時就無法達到最高效率。加班時間不得超過連續兩週。
- **駐點客戶** — 由一位真正的客戶全天候在開發環境中工作，幫助定義系統、撰寫測試和回答問題。
- **程式碼撰寫標準** — 程式設計師採用一致的程式碼撰寫標準。

目前有三本描述 XP 的書可用：

1. 說明 eXtreme Programming
2. 安裝 Extreme 程式設計
3. 規劃 Extreme 程式設計

有關 XP 的進一步資訊，有幾個網站可參考。

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