本書的目標讀者

本書的目標對象包括以下類型的讀者。

在組織內負責推動敏捷開發的人員

- 與之前的作法相比，覺得只有名稱改變，沒有突破
- 看不到提升產品價值或縮短交付時間等實際成果
- 沒有發現阻礙敏捷開發的問題

團隊開發經驗尚淺的新手工程師

- 參與業務開發的時間還不長
- 不瞭解實踐衍生的背景與使用目的

負責開發現場或團隊的技術主管、資深工程師

- 不清楚有哪些實踐方法
- 不知道如何選擇、導入適合狀況的實踐
- 雖然正在進行實踐，卻無法證明作法是否適當

本書的閱讀方法

本書的第 1 章將說明與敏捷實踐有關的基本概念。第 2 ～ 4 章將介紹技術實踐及其應用。接著第 5 ～ 6 章要講解牽涉到其他團隊及利害關係人的廣泛實踐。先從頭開始大致看過一遍，就能掌握在開發現場導入實踐的流程。如果目前已經發生令你困擾的問題，也可以直接從相關章節開始閱讀。希望這本書對各位讀者而言，能成為導入敏捷開發技術實踐並擴大應用的指南。

以下將詳細介紹每一章的內容。

第 1 章 推動敏捷開發的實踐

這一章將介紹敏捷開發的目標與實踐的關係，以及有助於理解實踐的思考方法。

第 2 章 可以應用在「實作」的實踐

這一章將介紹實作方針、分支策略、程式碼審查、建立測試階段所需的規則，以及讓團隊成員之間密切溝通的技術實踐。

第 3 章 可以應用在「CI/CD」的實踐

這一章將介紹在整個開發流程中持續整合與自動化以維持、改善產品品質的方法。

第 4 章 可以應用在「運作」的實踐

這一章將介紹讓系統穩定運作並持續進行敏捷開發的相關技術實踐。

第 5 章 可以應用在「達成共識」的實踐

這一章將介紹讓開發團隊內外人員取得共識的實踐，以及在開發過程中重新審視計畫的實踐。

第 6 章 可以應用在「團隊合作」的實踐

這一章將介紹適合交付客戶價值的團隊組成方法、讓團隊之間順利溝通的實踐、納入利害關係人並取得共識的實踐。

結語

最後將介紹一些網站與資料來源，你可以從中找到本書沒有介紹的實踐方法。

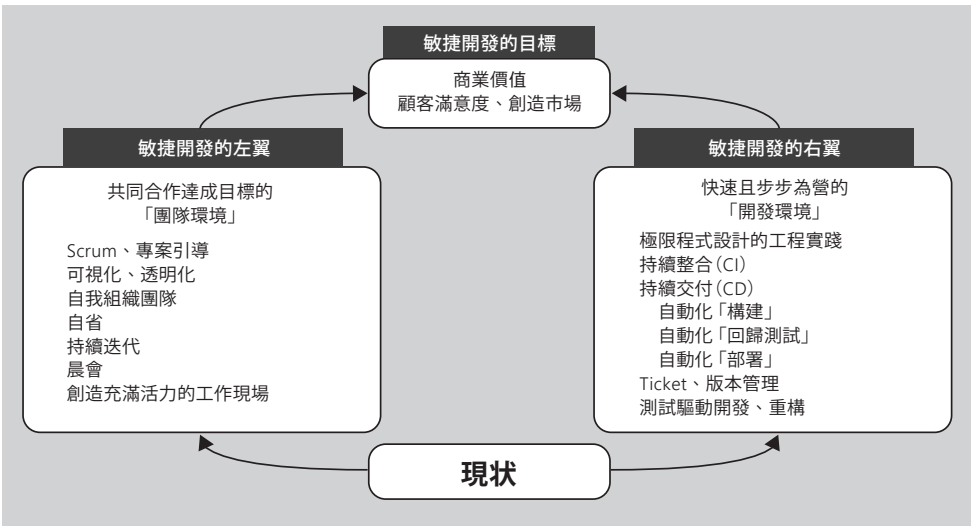
進行實踐

敏捷的「左翼」與「右翼」

簡而言之，敏捷開發是**「踏出一小步，根據經驗中學到的知識，不斷改進的開發方法」**。我們很難在開發初期就完美規劃出產品，必須實際向使用者提供價值，再從中學習，隨時調整產品方向。

那麼，如何實現這種開發呢？要進行敏捷開發，不僅需要管理團隊，還得改善開發流程與工具。這裡將廣為人知的機制與實踐整理成下圖，說明「敏捷開發的『左翼』與『右翼』」1-1（圖 1-1）。在這張圖中，歸納了實現敏捷開發目標的兩種手法，包括**與「改善團隊環境」有關的左翼（涉及流程與團隊管理的實踐）**，以及**與「技術／工具」有關的右翼（技術實踐）**。這張圖不是要表達「成員或團隊必須選擇左邊或右邊其中一條道路」，而是**團隊成員要視狀況同時兼顧兩者以達到敏捷開發的目標**。在此分類方法中，技術實踐是邁向敏捷開發目標的關鍵之一。如果想在不降低生產力的狀態下持續開發，增加功能，向使用者傳遞價值，絕對不能缺少技術實踐。

圖 1-1 敏捷開發的「左翼」與「右翼」



透過技術實踐讓文化紮根

上面的圖 1-1 列舉了幾種技術實踐。光是本書介紹的技術實踐就包括以下幾種，其實世上還有更多技術實踐存在。

- **持續整合**
- **測試驅動開發**
- **持續交付**
- **重構**
- **版本管理**

對開發人員來說，導入技術實踐不僅可以改善產品與開發流程，也能獲得新的想法與知識，是一個有趣且有意義的工作，還可以輕易想像採用技術實踐後的狀態，讓工作變得容易處理。

不過，過程中也可能遇到與現有開發流程衝突而難以導入，或即使導入也無法發揮技術實踐效果的情況。只增加開發規則，反而可能降低實際產生商業價值的速度，造成負面影響。技術實踐有其產生背景，**並非所有情況都適用，也不見得有效**。有時也可能出現以下把導入技術實踐變成目的的情況。

- 使用專案管理工具，不論多小的任務都想管理
- 想為所有原始碼準備測試碼

想避免上述情況，就得重新審視按照敏捷開發目標導入技術實踐的方法。導入技術實踐直到產生效果的流程應如下所示：

1. **思考本章介紹的基本想法**
2. **意識到導入技術實踐的目的**
3. **即使感到不安或懷疑也要認真實驗**

實作方針

確保在實作之前，先讓團隊成員瞭解目前已知的狀況及花點時間溝通就能明白的事情。

將使用者故事拆解成任務

團隊會從優先順序較高的使用者故事開始著手開發。此時，並不是直接由（任務）負責人員管理使用者故事，而是拆解成幾個小時到半天，最長一天就可以完成的小任務。如果團隊成員在進行各個任務時，可以瞭解該做什麼，就能順利進行開發（圖 2-1）。

圖 2-1 使用者故事與拆解任務

×

ID	使用者故事	（任務）負責人員
1	修改商品清單頁面的樣式問題	欣守
2	在電子報顯示橫幅廣告	優生
3	在管理畫面增加確認對話視窗	老手
4	刪除已經結束的活動程式碼	行世

○

	ToDo	Doing（WIP 上限 2）	Review（WIP 上限 2）	Done
1. 修改商品清單頁面的樣式問題	測試	修改設計	調查樣式跑掉的原因	
2. 在電子報顯示橫幅廣告	傳送測試 修改電子郵件範本 上傳橫幅廣告的圖片	設計橫幅廣告的點擊記錄	建置橫幅廣告的顯示邏輯	
3. 在管理畫面增加確認對話視窗	測試 修改確認對話視窗的設計 建置對話視窗的顯示邏輯			
4. 刪除已經結束的活動程式碼	測試 重構 刪除活動的程式碼			

分支策略

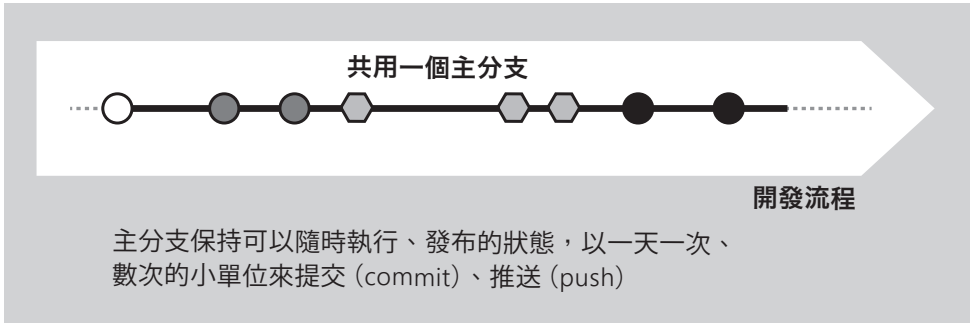
如果是網路應用程式這種不需要傳送，卻要頻繁發布的情況，git-flow 的發布步驟就比較繁瑣。頻繁發布可以降低一次發現大量嚴重錯誤的風險，即使有錯誤，也能及早修正並重新發布。GitHub Flow 的詳細說明請參考「GitHub Flow」2-9。

累積頻繁的小規模提交來進行開發

主幹開發

「主幹開發（Trunk Based Development）」2-10 是不建立分支，直接在唯一的主分支上累積頻繁的小規模提交來進行開發的分支策略（圖 2-13）。

圖 2-13 主幹開發



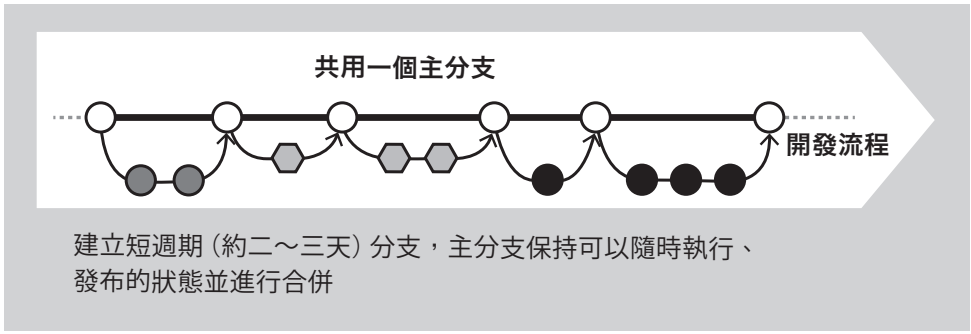
主幹開發（Trunk Based Development）的重點如下：

- **不建立特性分支**
- **以一天一次到多次的小單位進行提交、推送**
- **尚未完成的功能可以設定成「隱藏顯示」或「禁止執行」，在預設狀態下不執行該功能**
- **主分支隨時保持通過測試、可執行、可發布的狀態**
- **如果測試失敗或現有功能無法執行就立刻修復**

將原始碼的修正分成小單位，頻繁合併到主分支，主分支就能保持最新進度。對最新版本的主分支持續進行自動化測試，一旦出現問題，立即回饋給開發人員，保持隨時可執行、可發布的狀態。

儘管如此，隨著開發人數增加，如果每個人都直接向主分支推送，就容易出現設計、命名規則不一致或主分支無法運作的情況。因此，還有一個方法是**建立短週期（約二～三天）分支，透過拉取請求進行程式碼審查，再合併到主分支**，當作主幹開發的應用（圖 2-14）。這張圖與 GitHub Flow 類似，但是在主幹開發中，分支的週期短，功能尚未完成時，就開始合併。與直接在主分支累積提交相比，大部分的團隊比較能接受這種方法。

圖 2-14 大型團隊的主幹開發



建立短週期（約二～三天）分支，主分支保持可以隨時執行、發布的狀態並進行合併

如果採取的分支策略是讓特性分支長期存續，每次進行重大功能開發或發布時，通常都需要「讓運作穩定的閉鎖期間」。一旦在特性分支上的開發時間拉長，偏離主分支時，原始碼就容易發生衝突，處理成本也會增加。此時，如果試圖整理現有方式並進行大規模修改，發生衝突的機率會更高。因此，即使在特性分支中發現設計或實作上有瑕疵的原始碼，大多無法徹底修正，或者為了避免衝突而不得不進行不完美的修正。這樣會增加維護性差的原始碼，使得程式碼庫（Code Base）膨脹到無法維護，形成龐大的技術負債。採用主幹開發的優點恰好與此相反，其優點如下：

- **縮短合併所需的時間**
  修正的差異變小，比較容易進行程式碼審查
  頻繁進行小規模整合，原始碼比較不易發生衝突

- **發生問題時，比較容易進行調查**
  頻繁自動測試主分支可以及早發現問題
  最近的修正很可能是引發問題的原因，因此能縮小調查範圍

- **驗證組合變簡單**
  可以將操作驗證的對象集中到主分支上
  跨多個服務的操作驗證也可以在主分支之間進行

儘管主幹開發看起來盡是優點，但是除了要準備自動化測試外，還必須在保持運作狀態下，小規模整合開發中的功能。

Q&A 不曉得與目前的分支策略有何差別

使用拉取請求在一個主分支上進行開發的團隊，可以算是採用了主幹開發的分支策略嗎？我們現在處理的使用者故事很複雜，不曉得能不能把拉取請求分成更小……。

把使用者故事對應的修正拆解成多個小修正，並隨時合併到主分支，就算是主幹開發。分支或拉取請求的生存週期最長不要超過二～三天。

Q&A 何時進行程式碼審查

如果是主幹開發，什麼時候要進行程式碼審查？目前我們的團隊規定在合併拉取請求之前進行程式碼審查。

如果和圖 2-14「大型團隊的主幹開發」介紹的一樣，使用短週期分支與拉取請求的話，可以在傳送拉取請求時進行程式碼審查。另一種方法是，採用 2.5 小節介紹的結對程式設計或群體程式設計，邊進行程式碼審查邊開發。