前言

軟體架構指標用於衡量軟體專案的可維護性和架構品質，並在開發過程初期提供有關架構或技術債務危險累積的警告。在本書中，10 位主要的實際實踐者（Christian Ciceri、David Farley、Neal Ford、Andrew Harmel-Law、Michael Keeling、Carola Lilienthal、João Rosa、Alexander von Zitzewitz、Rene Weiss 和 Eoin Woods）介紹了每個軟體架構師都應該了解的關鍵軟體架構指標。這 10 位架構師都發表過著名的軟體架構文章和書籍，也定期地參加國際活動，並舉辦實踐研討會。

雖然我們一直在力求理論和實踐的平衡；然而，這本包含了寶貴經驗和案例研究的書不是關於理論的；它是關於實踐和實作的、是關於已經嘗試過和已經奏效的。我們不只關注改善架構的品質，而且專注在以考慮到你自己的情況和所涉及權衡的方式，將客觀的指標與業務成果聯繫在一起。

我們進行了一項調查，發現對軟體架構指標資源有強烈的需求，但可用的資源卻很少。我們希望這本書能夠發揮作用，並幫助你設定正確的 KPI 並準確而有洞察力地測量結果。

我們感謝全球軟體架構峰會，是它讓我們重新聚集在一起，並讓我們有了一起編寫軟體架構指標書籍的想法。本書的所有章節與案例研究就如同作者本身般的不一樣：我們堅持要使用來自不同行業和挑戰的範例，以便每位讀者都能找到解決方案或靈感。

你將學到什麼？

讀完本書，你將了解如何：

- 衡量你的軟體架構在實現目標上有多好
- 引導你的架構朝向可測試性和可部署性邁進
- 確定軟體架構工作的優先順序
- 從可觀察性建立可預測性

- 確定軟體專案的關鍵 KPI
- 建立並自動化指標資訊看板
- 分析和衡量專案或過程的成就
- 建立目標驅動的軟體架構

誰應該讀這本書

本書由軟體架構師編寫，也是為軟體架構師而寫。如果你渴望探索成功的案例研究，並更加了解決策和測量有效性相關的內容，那無論你是在軟體開發公司內部工作還是作為獨立顧問，本書都適合你。

10 位作者都是經驗豐富的從業者，他們分享了他們的建議和智慧，提出了不同的觀點和想法。當你在不同的專案上工作時，你可能會發現某些章節與你的工作比其他章節更相關。你可能會經常使用這本書，或者你可能會使用它設定 KPI 一次，然後稍後再用它來指導和激勵新的團隊成員。

擁有正確的軟體架構指標和工具，可以使架構檢查更快、成本更低。它可以讓你在軟體專案的整個生命週期中執行檢查，從一開始就進行。指標還可以幫助你在每個衝刺中評估你的軟體架構，以確保它不會逐漸走向無法維護的情況。它們更可以幫助你比較架構，以挑選最適合你專案要求的架構。

本書編排慣例

本書中使用了以下編排慣例：

斜體字（*Italic*）

表示新的術語、URL、電子郵寄地址、檔案名稱和檔案副檔名。中文用楷體表示。

定寬字（`Constant width`）

用於程式列表，以及在段落中引用程式的元素，像是變數或函數名稱、資料庫、資料類型、環境變數、敘述和關鍵字等。

第一章

發揮 4 個關鍵指標

Andrew Harmel-Law

認為 Nicole Forsgren 博士、Jez Humble、和 Gene Kim 的開創性著作《*Accelerate*》（IT Revolution, 2018）是關於如何轉變你軟體交付性能的先決條件與最後測量的想法，是可以理解的，而這一切都是透過 4 個簡單且強而有力的關鍵指標測量。

我本身的轉換工作也是基於他們書中的許多建議進行，我當然對其中所有的內容都沒有異議。但是，我認為與其消除對更多細節的需求，不如進一步的討論和分析這本書，以便能夠分享經驗並聚集一群想要改善架構的實踐者。我希望本章對於這樣的討論能有所貢獻。

我看到，當以本章後面所描述的方式使用時，這 4 個關鍵指標——部署頻率、變更前置時間、變更失效率和恢復服務時間——會導致學習更有成效，並讓團隊理解對高品質、寬鬆耦合、可交付、可測試、可觀察和可維護架構的需求。有效的部署，這 4 個關鍵指標可以讓作為架構師的你放鬆對舵柄的控制。你可以使用這 4 個關鍵指標來與團隊成員溝通，並激發出想要超越自己改善整個軟體架構的願望，而不是只會發號施令和控制。你可以逐漸地邁向更為可測試的、連貫和內聚的、模組化的、容錯和雲端原生的、可執行和可觀察的架構。

在後續章節中，我將展示如何建立和執行你的 4 個關鍵指標，以及（更重要的是）你和你的軟體團隊能如何最好地使用這些指標來關注持續改善的工作並追蹤改善進度。我的重點是將 4 個關鍵指標心智模型的實際面向視覺化，尋求所需要的三個原始資料點，然後計算並顯示這 4 個指標。但是別擔心：我也將討論在生產中執行的架構好處。

定義和檢測設備

> 範式是系統的源泉。從它們那裡，從關於現實本質的共同社會協議中，產生系統目標和資訊流、回饋、庫存、流動以及其他與系統有關的一切。
>
> —Donella Meadows,《*Thinking in Systems: A Primer*》[1]

《*Accelerate*》基礎的心智模型導致了 4 個關鍵指標。我從這裡開始是因為當你閱讀本章時，要記住這個心智模型是必不可少的。在最簡單的形式下，這模型是一個活動的管道（或「流程」），每當開發人員將他的程式碼更改推送到版本控制時開始，而當這些更改被吸收到團隊正在處理的執行系統，即交付給使用者的執行服務時結束。這個心智模型顯示在圖 1-1。

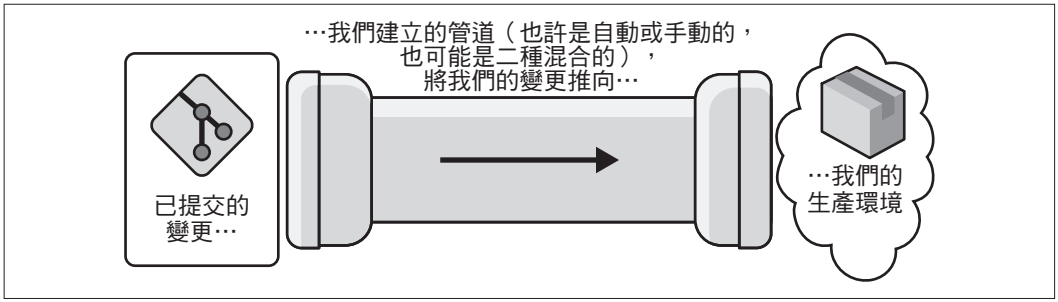

圖 1-1　4 個關鍵指標背後的基本心智模型

為了清楚起見，讓我們將這個模型中 4 個關鍵指標的測量具體化表示：

部署頻率

> 隨著時間的推移，從管道末端移出的單個變更的數量。這些變更可能是由「部署單元」：程式碼、配置或兩者的組合所組成，例如，包括有新的功能或錯誤修復。

變更的前置時間

> 開發人員完成的程式碼 / 配置變更，通過管道並從管道另一端移出所需要的時間。

綜合起來說，第一對是測量**開發的吞吐量**。這不應該與包括編寫程式碼時間的**精實週期時間**或**前置時間**混淆，這個測量甚至有時候在產品經理第一次提出新功能的想法時就開始計時。

1　Donella Meadows 所著的《*Thinking in Systems: A Primer*》，Diana Wright 編輯（Chelsea Green Publishing, 2008），第 162 頁。

變更失效率

在我們執行的服務中，從管道移出的變更導致失效的比例（「失效」的具體定義稍後會說明；現在，只需要將失效看成是阻止服務的使用者完成工作的事情）。

恢復服務的時間

在服務出現失效後，需要多長的時間才能意識到有失效，並對使用者提供恢復服務的修復[2]。

綜合起來說，第二對提供了**服務穩定性**的指示。

這 4 個關鍵指標的力量在於它們的組合。如果你改善了開發吞吐量的一個要素，但在過程中卻降低了服務的穩定性，那麼你的改善是處於不平衡的方式，將無法實現長期持續的利益。最根本的一點是，你要密切關注所有 4 個關鍵指標。實現可預測長期價值的轉變是那些可以**全面產生**積極影響的轉變。

現在我們已經清楚了指標的來源，我們可以透過將通用心智模型映射到實際交付的過程使事情更為複雜化。我將在下一節展示如何執行這種「心智重構」。

重構你的心智模型

針對你的情況定義每個指標相當重要。正如你很可能已經猜到的那樣，前兩個指標是以 CI 管道中發生的事情為基礎，而第二對指標則需要追蹤服務的中斷和恢復。

在執行這種心智重構時，應該仔細考慮它的範圍。你是否查看整個組織中所有軟體的所有變更？或者只是考慮你所工作程式中的那些？是包括了基礎架構的變更還是只觀察軟體和服務的變更？所有這些可能性都很好，但請記住：**你考慮的範圍對於這 *4* 個指標中的每一個都必須相同**。如果你在前置時間和部署頻率中包含基礎架構的變更，那也應該包括由基礎架構變更所引起的中斷。

第一個選擇是管道

你應該考慮哪些管道？你需要的是那些在目標範圍內監聽原始碼存儲庫中的程式碼和配置變更，執行各種不同的操作（例如編譯、自動化測試和封包等），並將結果部署到生產環境中。你不會想要進行包含像是資料庫備份之類事情的 CI 實作工作。

2 這不一定是程式碼修復。在這裡我們也考慮服務的恢復，因此像自動失效轉移這樣的東西可以完全地阻止時間流逝。

如果你只有一個由一個端到端管道提供服務的程式碼存儲庫（例如，存儲在 monorepo 的整體式架構，並在一組活動中直接部署到生產裡），那麼你在這裡的工作就很容易。這種模型如圖 1-2 所示。

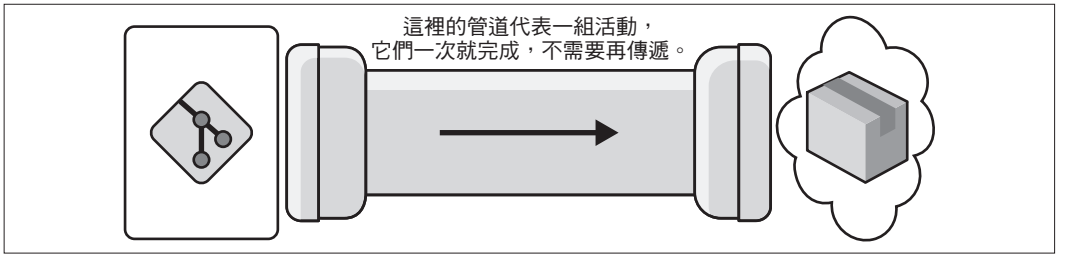

圖 1-2　你將發現這是最簡單的原始碼控制 / 管道 / 部署模型

不幸的是，雖然這與我們基本的心智模型完全相同，但很少在現實中看到這種情況。我們很可能必須對心智模型進行更廣泛的重構，以達到可以代表你情況的模型。

下一個最容易衡量也是我們第一個重要的心智重構，是這些端到端管道的集合，每個工件或存儲庫一個（例如，每個微服務一個），每個管道都做自己的工作，並且再次在生產中結束（圖 1-3）。例如，如果你使用 Azure DevOps，建構這些就很簡單 [3]。

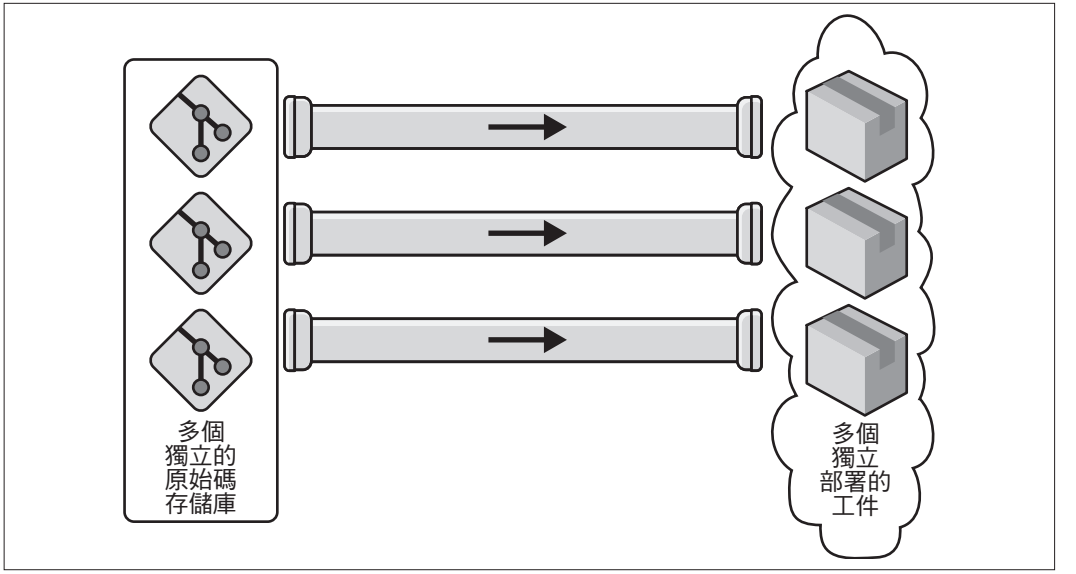

圖 1-3　「多個端到端管道模型」非常適合微服務

3　事實上，這是微軟希望你採用的模型。

前兩種管道形狀很可能與你的情況**相似**，但我猜測你的模型圖版本會稍微複雜一些，需要再進行一次重構才能拆分成一系列的子管道（圖 1-4）。讓我們考慮一個有三個這種子管道的範例，這個範例適合於端到端的將變更交付給生產。

也許第一個子管道會監聽對存儲庫的推送，並進行編譯、封包、以及單元和組件測試，然後發布到二進制的工件存儲庫。也許接下來是第二個獨立的子管道，它將這個新發布的工件部署到一或多個環境進行測試。可能還會有由像是 CAB 流程[4]所觸發的第三個子管道，最後將變更部署到生產中。

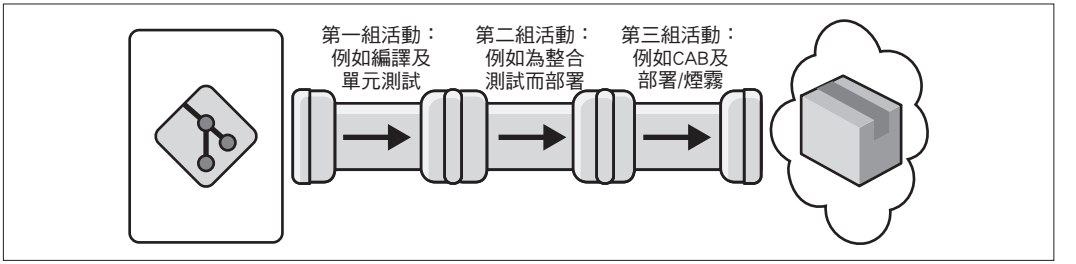

圖 1-4　我經常遇到的「由多個子管道組成的管道」模型

希望你已經確定了自己的情況。如果還沒有，還有第四種主要的管道模型，我們最後的心智重構步驟將提供我們：如圖 1-5 所示的多階段扇入管道。在這裡，我們通常會為第一階段尋找單獨的子管道，每個存儲庫一個，然後「扇入」到將變更帶到生產其餘部份的共享子管道或一組子管道。

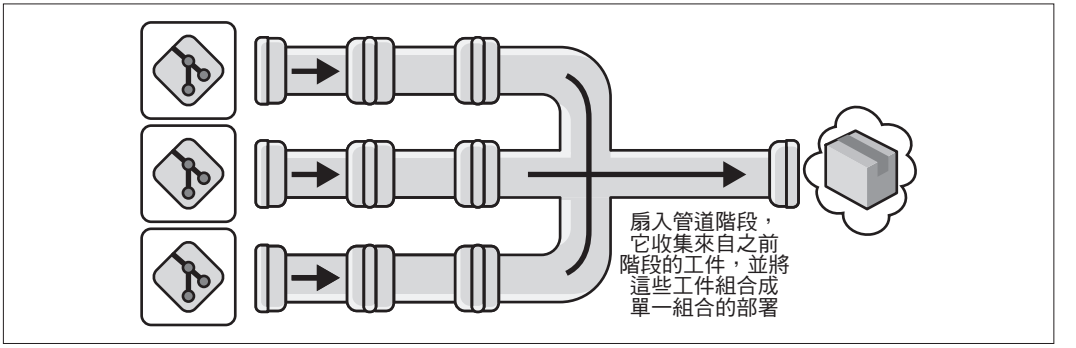

圖 1-5　多階段「扇入管道」模型

4　CAB 表示「變更諮詢委員會」。最著名的例子是定期開會核准由 Gene Kim、Kevin Behr、和 George Spafford 合著的經典著作《*The Phoenix Project* 》（*https://oreil.ly/b5609*）（IT Revolution Press, 2018）中的程式碼和配置小組。

定位你的檢測點

除了有 4 個指標以外，我們還有 4 個檢測點。現在無論你採用哪種形式，讓我們在心智模型上定位這些檢測點。到目前為止，我們一直專注在管道上，因為它們通常會提供這些檢測點中的兩個：提交時間戳記和部署時間戳記。第三個和第四個檢測點則來自於當檢測到服務降級以及標記為「已解決」時所建立的時間戳記。我們現在可以詳細地討論每一個檢測點。

提交時間戳記

當你考慮團隊的工作實踐時，這裡不可避免地會出現一些微妙的情況。他們是依據功能分支嗎？他們是在做拉取的請求嗎？他們是否混合了不同的實踐？理想情況下（就如同《*Accelerate*》作者所建議的），只要任何開發人員的變更集被認為已經完成並被提交，就已經開始計時了，無論它是在哪裡。如果團隊正在這樣做，請注意：在分支上保留變更不僅會延長回饋的週期，也會增加工作的開銷和基礎架構要求（我將在下一節介紹這些內容）。

由於這種複雜性，有些人選擇使用從合併到主管道的觸發作為代理觸發點或提交時間戳記。我了解這聽起來像是在面對次優的實踐時承認失敗[5]，但是如果你選擇代理觸發，我知道你會有罪惡感（因為你知道你沒有遵循標準的最佳實踐）。無論我們是否包括額外的等待時間，即使你讓自己休息一下、並在程式碼進入主管道時開始你的初期取樣，這些指標都會帶來許多其他的好處。如果這些代理確實成為重要的交付次優化的來源時，《*Accelerate*》會為你提供一些建議（例如基於主幹的開發和結對程式設計）[6]，這些建議會透過使變更進入主管道的時間作為你再啟動計時器的時間，以影響你的提交時間戳記。到那時，你將開始看到指標的好處，並希望改善對它們的擷取。

部署時間戳記

隨著提交時間的結束，你會很高興聽到時鐘的「停止」要簡單得多：它是管道完成最終部署到生產的時間。這難道不是讓那些在事後進行手動煙霧測試的人休息一下嗎？確實如此，但我還是要把它留給你的良知，如果你真的想包括這個最後的活動，你總是可以在管道的末尾放置一個手動檢測點，一旦 QA（或檢查部署的人）對部署成功感到滿意，就會按下這檢測點。

---

5 特別是如果你有長期存在的分支或永無止境的拉取請求，但我打賭你無論如何都知道這些，而且它們也不難獨立出來量化。

6 有關基於主幹開發你想知道的所有資訊，請參考這個文件（*https://oreil.ly/L5cs0*）。有關結對程式設計的原始定義，請參考極限程式設計（*https://oreil.ly/pGAfY*）。

多階段和扇入管道引起的複雜性

鑑於這兩個資料來源，你可以計算出我們從管道中需要哪些資訊，那就是**運行的總時間**：從時鐘啟動到時鐘停止之間所經過的時間。如果你有我們之前討論過較為簡單的管道場景，就是那些沒有扇入的場景，那這就相對的容易了。那些擁有一個或多個端到端管道的執行總時間最容易做到這一點[7]。

如果你運氣不好並且有多個子管道（就如同圖 1-4 所看到的），那麼就需要對包括在「啟始」時間戳記和「部署」到生產中的變更集執行額外的資料收集。得到了這些資料，你可以進行一些處理以計算每個單獨變更的總運行時間。

如果你運行一個扇入的設計（如圖 1-5 所示），這種處理可能會涉及的更多。為什麼？就如你將在圖 1-6 中看到的，你將需要一種方法獲知變更部署編號 264 的來源（Repo A、Repo B 或 Repo C），以便得到變更的「啟始」時間戳記。如果你的部署聚集了許多變更，那麼你將需要單獨追蹤每個變更以獲得它的「啟始」時間。

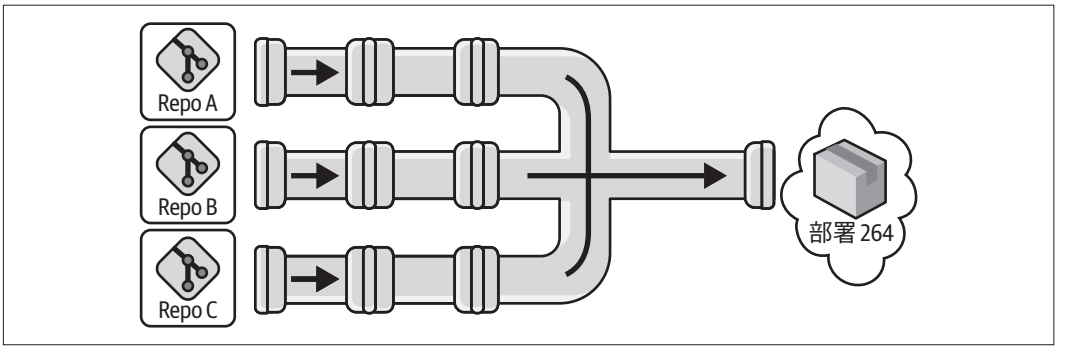

圖 1-6　在「扇入管道」模型的變體中定位資料收集點

顯然，在所有情況下，無論你的管道有多複雜，你都只想計算向使用者部署服務更新的建構。應確保你只測量這些[8]。

7　如果這讓你認為有多個獨立的管道（每個工件一個）是一個好主意，那恭喜你：你已經提醒自己微服務的關鍵原則之一——獨立部署性。如果這讓你喜歡整體式架構，那麼請記住微服務會帶來的其他好處，其中一些我們將在本章結尾處介紹。

8　有時候會出現這樣的問題：「基礎架構建構怎麼樣？」我已經看到那些包括在「4 個關鍵指標」的計算中，但如果它們不包括在內，我也不會感到懊惱。至於由時間而不是變更所觸發的管道如何呢？不要計算他們。它們不會造成部署，因為沒有任何的變更。

在我們繼續之前，關於從管道獲取資料還有最後一點要說明，那就是要計算哪個管道的運行？同樣地，《*Accelerate*》在這一點上並未明確的說明，但是你關心的只有成功的運行。如果一個建構開始但在編譯的步驟失敗了，這將使你的前置時間人為的偏向正值，因為你只是在混合中添加了一個非常快速的建構。如果你想玩遊戲（這 4 個關鍵指標的最大好處是它們不能玩遊戲，至少據我所知是不能），那麼你只需要提交許多你知道會崩潰的建構，最好是非常快的崩潰。

監測服務失效

雖然在管道周圍進行準確地量測相對簡單，但原始資訊的第三個和最後一個來源更容易解釋。

困難是出現在「生產失效」的定義上。如果出現了一個失效但是沒人發現，那這失效真的存在嗎？每當我使用這 4 個關鍵指標時，我對這問題的回答都是否定的。我將「生產失效」定義為任何使服務的顧客不能、甚至不願意繼續完成他們試圖執行工作的事情。外觀上的缺陷不能算是服務失效，但一個「工作系統」如果緩慢到使非典型的使用者流失，顯然是在經歷著服務失效。這裡面有一個判斷的成分，但沒關係：選擇一個讓你感到舒適的定義，並在你堅持的時候對自己誠實。

你現在需要記錄服務失效，這是我們的第三個和最後一個檢測點，一般是透過「變更失效」單據。開立這張單據提供你另一個時鐘的啟始時間資料點；關閉它會給你對應的結束時間。這個啟始和結束時間，加上單據的數量，是你需要的所有剩餘資料點。當服務恢復時應該關閉單據。這可能與解決中失效的根本原因不一致；但沒關係，我們談論的是服務穩定性。轉返以便你在線上並為客戶提供服務是可以接受的[9]。

但是如果你不在生產中呢？首先，你還沒有嘗試開始移向持續部署嗎？你真的應該這樣做。但其次，這個選項並不是所有人都能使用。它是次優的，但你仍然可以在這些情況下使用這 4 個關鍵指標。為了這樣做，你需要定義你的「最高環境」：最接近所有團隊交付生產的共享環境。它可能被稱為 SIT（用於系統整合）、前產品或分段傳遞。關鍵是當你接受你的變更時，你相信不需要進行更多的工作以在最後一步驟將變更帶到生產。

鑒於所有這些考慮，你需要像對待生產一樣對待這個「最高環境」。將測試人員和合作的團隊視為你的「使用者」。他們可以定義服務失效，像對待真正的失效一樣認真對待測試失效。將這個環境假裝是生產環境並不完美，但這總比沒有好。

9 同樣地，有人會指出，單據開出的時間與服務首次出現失效的時間不同。確實是如此；也許你會想將你的監測與這些單據的建立聯繫在一起來解決這個問題。如果你有辦法，那恭喜你：你可能正處於 4 個關鍵指標所採用的「微調」端。大多數人，至少在他們開始的時候，都只能夢想這種準確性，因此，考慮到這一點，如果你是從手動單據開始，那這樣就足夠了。

擷取和計算

> 系統建模者說，我們透過建構系統的模型來改變範式，這會將我們置身於系統之外並迫使我們看到整體。
>
> —Donella Meadows,《*Thinking in Systems*》[10]

現在你有了定義，可以開始擷取和計算了。雖然可以將這個擷取過程自動化，但用手動執行也是完全可以接受的[11]。事實上，每次我推出 4 個關鍵指標的時候，我都是從這裡開始，而且經常不僅僅是為了我們的最初基線。你將會明白為什麼可以用手動擷取和計算。

擷取指標可以是一項簡單或複雜的工作，這取決於管道的性質。無論如何，4 個關鍵指標將使用來自 4 個檢測點的 4 組資料來計算：成功的變更部署計數、每個變更運行管道的總時間、變更失效單據的計數、以及開立變更失效單據的時間長度。只有這些擷取的資料集還不足以獲得你的指標；你仍然需要計算，所以讓我們依次看看其中的每一項：

部署頻率

這是一個頻率，而不是計數，因此你需要有在**給定的時間週期內**成功部署的總數（我發現一天就很好了）。如果你有多個管道，無論你是否有扇入，你都需要將所有管道的部署次數加總起來。

有了這些資料，以及每天的記錄和匯總（記住要包括沒有部署日子的「零」總和），很容易就得到你的第一個標題指標。使用最新的每日數據或過去 24 小時的數據（根據我的經驗），會受到太多的波動影響。最好是顯示較長時間週期內的平均值，像是過去 31 天的平均值。

變更的前置時間

這是觸發啟始的任何單一變更的經過時間。這可能會有波動，因此不要只報告最近部署中的最新資料。如果你有多個（包括扇入）管道，這種波動會更大，因為有些建構會因為阻塞而執行得比其他建構快很多。你會想要一些更穩定的東西來反映整體情況，而不是最新的異常值。因此，我通常會測量每個單獨的前置時間，並且計算一天內這些的平均值。要報告的數值是過去 31 天內所有前置時間測量值的平均[12]。

---

10 Meadows，第 163 頁。

11 確保你對自己是誠實的：收集你應該做的所有建構，不要挑選。也試著讓你的數值盡可能準確，如果你是用猜的，也應估計你的準確程度如何。

12 記住！你不能在不引入問題的情況下得到一個平均值，所以最好是避免它。我們每天做總計，因此我們在我們指標的後面可以有一個很好、很漂亮的圖表，我們稍後會介紹它。

變更失效率

這是已解決的變更失效單據的比例，特別是導致失效的部署數量占同一週期部署總數的比例。例如，如果你一天內進行了 36 次部署，並且在同一天解決了 2 次的變更失效，這意味著你當天的變更失效率為 2/36，即 5.55555556%。

要獲得你報告的指標，查看同一時間週期內的這個比率：前 31 天。這意味著你將過去 31 天內恢復的失效數量加總，然後除以同一週期的部署總數。

你會注意到這裡有一個出於信念而大膽的舉動。我們假設失效是不同的，而且單一的失效是由單一部署所引起的。為什麼？因為根據我的經驗，很難將失效與單一建構聯繫起來，而且在絕大多數的事件中，這兩個假設都是成立的，至少是足夠讓它們值得損失些保真度。如果你能在這方面變得更聰明，那恭喜你！

眼尖的人還會注意到，我們只談到**已經解決**的失效。為什麼我們不包括仍然存在的失效？這是因為我們希望所有 4 個指標的一致性，也因為恢復服務的時間只能考慮已經解決的失效 [13]。如果我們不能為一件事計算尚未解決的失效，我們就不想為另一件事計算它們。但是不要擔心：我們仍然有尚未解決失效的資料，而且我們不會隱藏這一點，就如你將在後續章節中看到的那樣。

恢復服務的時間

這是變更失效單據從建立到關閉所需的時間。《*Accelerate*》的作者稱這是**恢復服務的平均時間**，雖然在早期的《*State of DevOps*》報告中，它只稱為**恢復的時間**，而在 Google *Four Keys* 專案的 *METRICS.md*（*https://oreil.ly/VlSgn*）文件中稱為**恢復的中位時間**。我則同時使用了**平均值**和**中位數**；平均值對異常值很敏感，有時候這正是你在學習時想要看到的。

平均值和中位數都很容易可以透過變更失效單據的解決時間資料輕鬆計算得到。無論哪種方式，你都要在資料的範圍內選擇你的輸入。我通常會使用過去的 120 天。取出所有在這個週期內所有失效的解決時間，計算它們的平均值，並針對這個指標提出報告。

這可能會是另一個出於信念而大膽的舉動：當你手動提出變更失效時，可能會透過將單據開立時間延遲到立即發現點之後而扭曲了這些數值。老實說，即使人們有最好的意圖，也會發生扭曲。然而，你仍然可以得到足夠好的資料，以保持對事情的關注並推動改善。

13 遺憾的是，因為未解決的失效沒有「已經解決」的時間戳記。

無論你如何擷取用於這些計算的資料，都要確保這一切都是公開進行的。首先，應鼓勵開發團隊閱讀這 4 個關鍵指標，你的努力應該沒有什麼秘密。

第二，提供所有原始的資料和計算結果，以及計算出的重要數據；這在以後會變得很重要。

第三，確保你具體應用於每個指標的定義，以及你如何處理這些**與資料本身一起**提供的定義。這種透明度將加深理解並提高參與度[14]。

注意這個存取問題（存取資料、計算和視覺化），因為如果你的 4 個關鍵指標沒有與大家共享，那麼你就錯過了它們最大的優勢。

展示與理解

> *[那麼] 你如何改變範式呢？ ... 你不斷地在指出舊範式中的異常和失效。你持續地用語言和行為強調並保證新的範式。*
>
> —Donella Meadows,《*Thinking in Systems*》[15]

每當我部署這 4 個關鍵指標，我通常都會從一個**最小可行的資訊看板**（MVD）開始[16]，這是 wiki 頁面的一個重要名稱，包含了以下內容：

- 4 個關鍵指標中每一個的目前計算值
- 每個指標的定義，以及我們計算它們所採用的時間週期
- 資料的歷史值

我還標記了資料來源，因此每個人都可以參與它們。

目標對象

指標，就像所有統計數據一樣，描述了一個故事，而故事會有聽眾。誰是這 4 個關鍵指標的目標對象呢？主要是交付軟體的團隊，也就是希望看到指標得到改善，而會實際進行變更的人。

---

14 如果你弄錯了，它甚至可能會給你一些關於你計算的錯誤報告——我最好的一些圍繞在 4 個關鍵指標的學習是透過這種方式獲得的。

15 Meadows，第 163 頁。

16 為 Matthew Skelton 和 Manuel Pais「最小可行平台」的想法表示支持，這給了他們靈感。

因此，你要確保無論你選擇以何種方式展示事物，它都必須放在這些個人和團體**主要**容易存取的地方。「容易」這點很重要。需要非常容易地看到這些指標，並深入研究它們且通常會挖掘出更多那些對他們擁有服務特定的資料點。

這 4 個關鍵指標還有其他的對象，但這些都屬於次要的。次要的對象可能是高級管理人員或執行管理人員。他們可以看到這些指標，但這些指標需要被包起來而且是唯讀的。如果執行管理人員想要知道更多的細節，那麼他們會到團隊那裡弄清楚，而這正是你希望發生的。

理想情況下，一旦你的 MVD 啟動，你就可以開始自動化的收集和計算工作。在我寫這篇文章的時候，已經有多種不同選擇。也許你最終會使用 Google 的 Four Keys（*https://oreil.ly/BPRaw*）、Thoughtworks 的 Metrik（*https://oreil.ly/1EDTb*）或像是 Azure DevOps（*https://oreil.ly/vMSBR*）等各種平台的擴展。雖然我確信這些都適合我們的目的，但沒用過其中的任何一個，我將分享我手動捲包經驗的好處，希望它能幫助你評估你是否想使用某些成品、或投入時間和精力自己打造一些產品。

視覺化

一項自己打造的努力造就了我曾經用過功能最齊全的資訊看板。它是用微軟的 PowerBI 建構的（因為客戶端都使用 Azure DevOps）。在經過對一系列與日期和時間的角力之後，我們擷取了原始資料，進行了計算，並著手建立圖表和其他視覺顯示的元素。

部署頻率

對於這些資料，我們選擇了長條圖（圖 1-7），以日期為 x 軸，部署次數為 y 軸。每個長條形表示當天的總數，我們將關鍵的統計數據提取到匯總中。

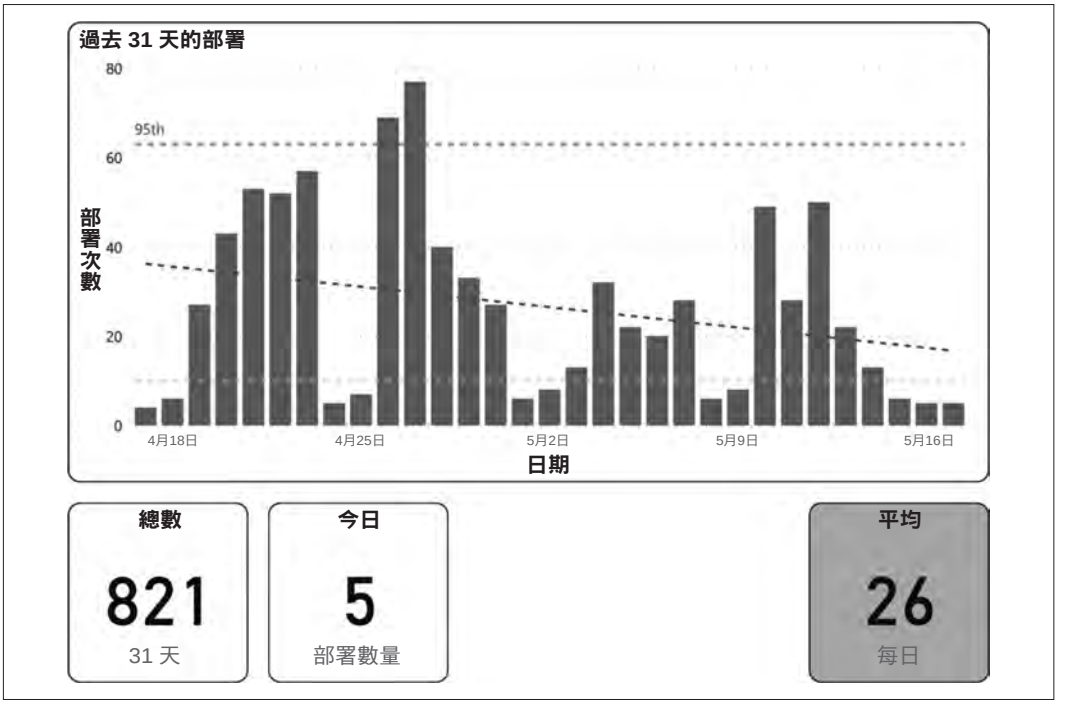

圖 1-7　部署頻率；右下角的方框表示「DORA Elite」（DevOps 研究和評估）軟體交付性能的等級

每天的平均部署次數顯示了部署頻率的關鍵指標，我們在關鍵指標中用綠色強調，以表示 Accelerate Elite Low 軟體交付性能量表上的「Elite」[17]。為了提高透明度，我們也顯示了當天的部署次數和所繪製週期（31 天）的總部署數。最後，我們將平均值、第 95 位數和整體資料趨勢用虛線繪製在圖表上。

變更前置時間

圖 1-8 的長條圖顯示了我們變更資料的前置時間，同樣的 x 軸為日期，現在長條在 y 軸的值為給定日期前置時間的*平均值*。

17 參考《*Accelerate*》書中圖 2-2 和 2-3，以及最近的 DORA《*State of DevOps*》報告中更多的最新表格（*https://oreil.ly/IetZp*）。

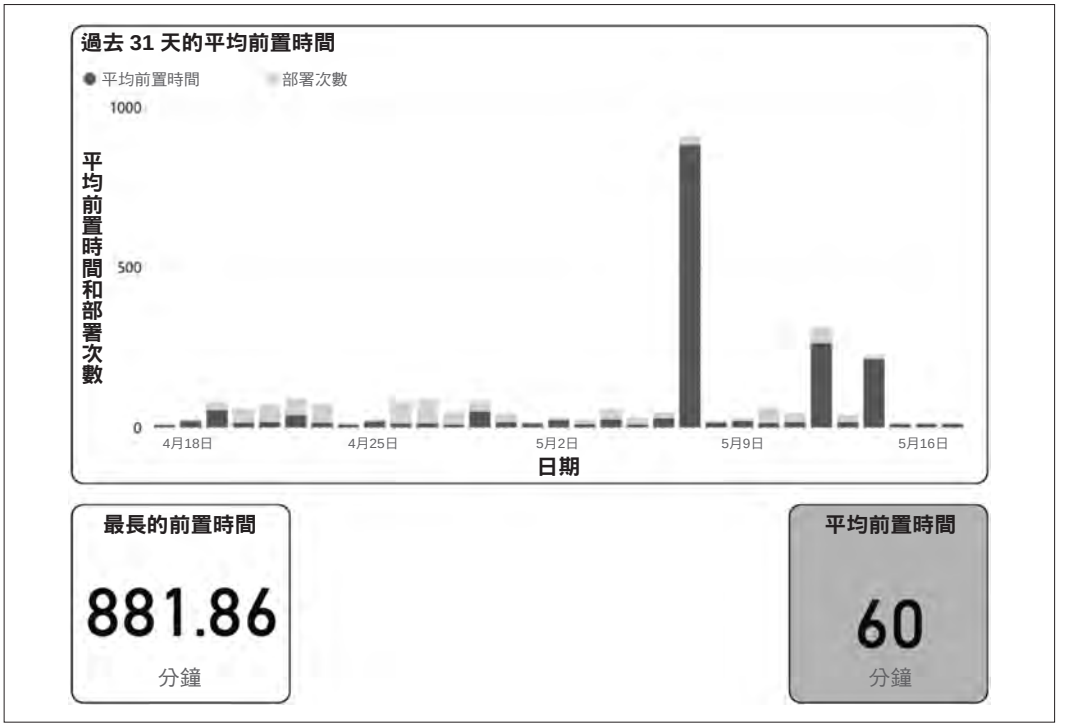

圖 1-8　變更的前置時間；右下角的方框表示在 Accelerate 評量表上軟體交付性能的「DORA High」等級

和以前一樣，我們突顯了圖上的關鍵指標，這裡是所顯示週期前置時間的平均值，並在我們的關鍵指標中突顯以表示 Elite-Low 性能量表上的「Low」。我們還發現突顯我們最長的個別前置時間很有用（參考左下角的方框）[18]。

我們意識到一直在問：「那天我們做了很多部署嗎？」我們沒有添加更多的趨勢線，而是將部署數量以陰影方式（淺灰色）和前置時間繪製在一起。

變更失效率

圖 1-9 顯示變更失效率的另一個長條圖，但這個指標呈現的方式完全不同。從 y 軸可以看出，在給定的 24 小時內，我們通常會有零個或一個失效 [19]。因此，當我們遇到問題時非常清楚。

18 我們有一個受到阻礙的建構，這不難發現。我們也使用了「average」而不是「mean」這個平均，來使它更平易近人。

19 有時候我們會看到多次失效，但這是非常的不尋常。

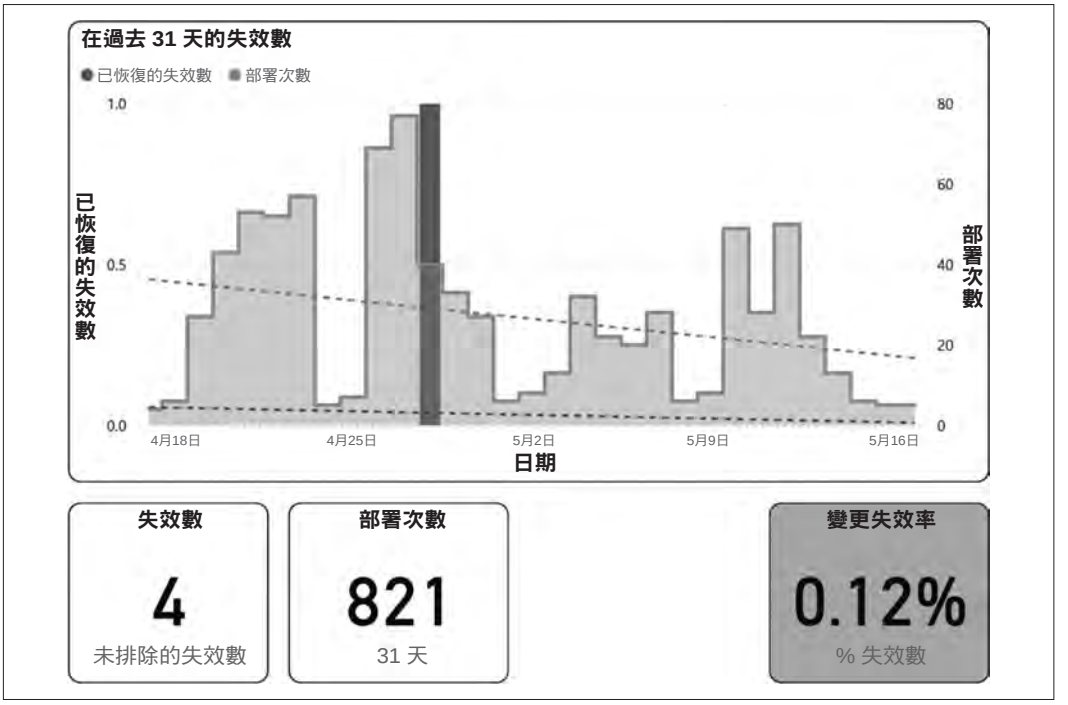

圖 1-9　變更失效率；右下角的方框表示在 Accelerate 評量表上軟體交付性能的「DORA Elite」等級

在此之上的一切都是背景。將部署數量一起繪出讓我們可以快速回答這個問題：「這可能是由於當天有大量部署活動而造成的嗎？」

最後，像往常一樣，在底部可以看到我們的關鍵指標：失效次數佔這個週期內部署總次數的百分比。伴隨這一點的是其他一些重要的統計數據：尚未排除的失效數量和顯示時間週期內的部署總次數。

恢復服務的時間

最後一個指標，恢復服務時間的呈現是我們花了最多時間來適應的指標——但一旦我們了解並穩定了我們的部署頻率和前置時間，這個指標就成為我們主要的焦點[20]。同樣地，我們又有一個時間序列長條圖（圖 1-10），但現在繪製的時間尺度比其他時間尺度更長（120 天，這是為了得到更好的背景），以便我們可以比較我們是如何針對一個應該具有較少資料點的指標上進行改善。同樣地，我們也同時繪製了變更的前置時間，以提供一些背景訊息。

20 根據經驗，我願意打賭這也將成為你的焦點。

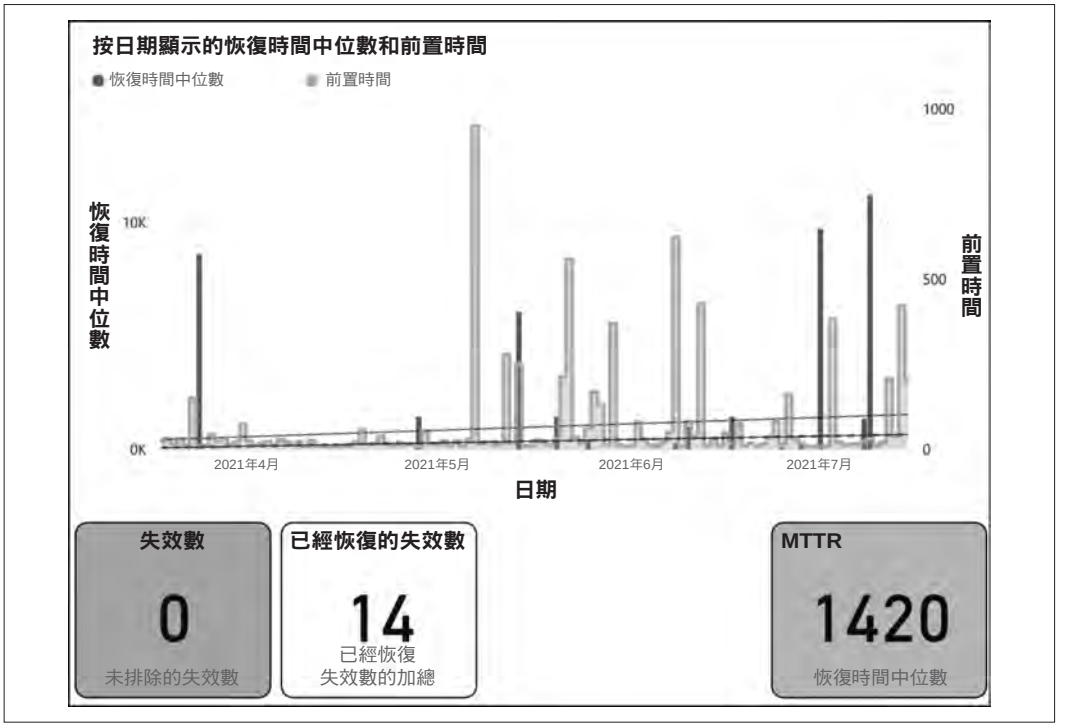

圖 1-10　恢復服務的時間；左下角的方框表示零個未排除的失效——不是 DORA 指標，但很重要應該知道——右下角的方框表示軟體交付性能的「DORA High」等級

最後，像往常一樣，在右下角你可以看到我們的關鍵指標：在給定的週期內恢復失效所有恢復時間的中位數。伴隨這個的是其他關鍵統計數據：在顯示的時間週期內未排除的失效數量和恢復的失效總數。

首頁

我們還沒有完成。我們的 PowerBI 報告還有一個「4 個關鍵指標」首頁，其中包括來自每個單獨統計頁面的關鍵指標數值，以及部署頻率和前置時間的圖表。它的目的是讓看的人即時、快速、準確地了解統計數據。隨著我們關注點的改變，我們也可能會推廣其他圖表。

就如同我所建議的，我們現在可以釋放這 4 個關鍵指標的真正力量。讓團隊接觸並確保他們理解這些指標，和支撐它們的模型和系統，是你能獲得它們真正利益最重要的事。這讓他們能夠討論、理解、擁有和改善你所交付的軟體。

討論與理解

> 在範式變更的過程中，沒有什麼實體的、昂貴的甚至是緩慢的。在單一個體上，它可以在一毫秒內發生，所需要的只是頭腦中一下靈光、頓時恍然大悟、以及一種新的觀察方式。
>
> —Donella Meadows,《*Thinking in Systems*》[21]

我們如何得到這些視覺化、額外的細節和特定的時間週期？我們進行了迭代，並依據需要進行了補充和改善。

每週我們都會集體討論即將到來的高峰期和架構決策記錄（ADRs）[22]，並查看 4 個關鍵指標。早期，討論是關於每個指標的含義。隨後幾週的討論則集中在為什麼這些數值會出現在那個位置（例如，數值是否太大或太小，是否缺少資料等），然後如何改善它們。緩慢但肯定地，團隊成員習慣了 4 個關鍵指標的心智模型。讓團隊即時自助服務他們的資料，並只查看來自管道中的資料（PowerBI 資訊看板使這兩件事變得容易），這非常有幫助。增加趨勢線也是如此，我們很快就能夠看到比預設 31 天更長的時間尺度。

我對這些集中、開明和跨職能討論的價值感到驚訝。作為一名架構師，這些問題和議題以前只能由我自己發現、理解、分析和補救。現在，可以由這些團隊發起並推動解決方案。

所有權和改善

每當團隊開始獲得所有權時，我都一次又一次地見證了以下情況。首先是最簡單的要求，也就是使流程和工作方式現代化的要求：「我們可以改變發布的節奏嗎？」接下來，團隊開始更關心品質：「讓我們把測試拉到左邊」和「讓我們增加更多的自動化。」[23] 然後是改變團隊構成的要求：「我們能夠移向跨職能（或串流對齊）的團隊嗎？」

總是會有需要學習的權衡、失效和經驗教訓，但**變更會自己推動**。當你更關注和更理解**端到端**視圖的好處時，你會發現自己正在修改和調整自己的關注點和解決方案。

21 Meadows，第 163 頁。

22 ADR 一詞最初是由 Michael Nygard 想出來的。

23 這讓 QA 和運營部門非常高興。我經常看到 QA 像我作為架構師一樣的使用這 4 個關鍵指標來推動變更。

所有這些變更很快的都在一個地方結束：它們揭示了架構問題。也許這些問題在白板的設計中就已經存在。也許在白板上的設計很好，但最後在生產中的實現卻不好。無論是哪種情況，你都有需要解決的問題。其中一些事情包括耦合不像你想像的那麼鬆散；領域邊界不像最初出現時那麼清晰；框架阻礙了團隊而不是幫助團隊；模組和基礎架構可能不像你希望的那樣容易測試；或微服務在實際流量下運行時無法觀察。這些都是作為負責任架構師的你通常必須處理的問題。

結論

現在你面臨一個選擇。你可以繼續孤軍奮鬥，將手保持在舵柄上，盡你所能的獨自指揮這艘架構之船。或者你可以在你的團隊中善用這種成熟技術。你可以將手從舵柄上移開，也許一開始是逐漸的，並用這 4 個關鍵指標開啟的對話和動力慢慢朝向你的共同目標邁進：更可測試、解耦合、容錯、雲端原生、可運行和可觀察的架構。

這就是將 4 個關鍵指標置於最有價值的架構指標的原因。我希望你能和你的夥伴一起用它們交付出你所見過最好的架構。