時或幾分鐘內快速部署到生產環境。對這些組織來說，部署作業終於變成低風險的例行工作。這些組織可以針對商業創意進行實驗，找出哪些點子能為客戶和整個企業組織創造最大商業價值，然後進一步開發，快速且安全地部署到生產中。

表 1　更快速、更低廉、低風險的軟體交付發展趨勢

	1970 ～ 1980 年代	1990 年代	2000 年代至今
世代	大型電腦	客戶端／服務端	商品化與雲端
代表性科技	COBOL、DB2 on MVS 等	C++、Oracle、Solaris 等	Java、MySQL、Red Hat、Ruby on Rails、PHP 等
週期	1 至 5 年	3 至 12 個月	2 至 12 週
成本	數百萬至數億萬美元	數十萬至數千萬美元	數萬至數百萬美元
風險層級	整個公司	生產線或某部門	產品功能
失敗成本	公司破產、出售、大量裁員	虧損、更換首席資訊長	微不可計

（來源：2013 年 11 月，Adrian Cockcroft 於美國加州 FlowCon 的演講內容：Velocity and Volume (or Speed Wins)）

現在採用 DevOps 實踐原則的企業組織，通常每天都會進行數百甚至數千次的部署變動。在需要儘速推入市場和不斷實驗才能取得競爭優勢的時代，無法複製這些成果的企業組織，註定會在市場競爭中輸給更加靈活的對手，而且可能完全一厥不振，就像過去那些不曾採用精實開發原則的製造業者一樣。

不論身處哪一個產業，我們吸引消費者並傳遞價值的方式都必須仰賴科技價值流程。簡言之，就像奇異公司首席執行長 Jeffrey Immelt 曾說：「任何產業、任何公司，如果沒有讓軟體成為業務核心的一部分，那麼他們將要自食後果。」或者，如微軟技術院士 Jeffery Snover 所言：「過去的經濟年代，企業靠著移動原子創造價值。現在的公司改以位元（bit）創造商業價值。」

另一方面，DevOps 告訴我們，當具備正確架構、正確的技術實踐和正確的文化規範時，小型開發團隊能夠快速、安全、獨立地進行開發、整合、測試和部署變更到生產環境中。

正如前任 Google 工程總監 Randy Shoup 觀察到，採用 DevOps 的大型組織「擁有數千名開發人員，但組織架構和實踐方法讓小型團隊仍然具有難以置信的高度生產力，就像剛剛起步的新創企業一樣。」

在《*2015 State of DevOps Report*》中，不僅檢驗「每日部署量」，也一併將「每位開發人員的每日部署量」列為觀察要點。我們假設：隨著團隊規模擴大，高效能組織有能力將部署數量規模化。

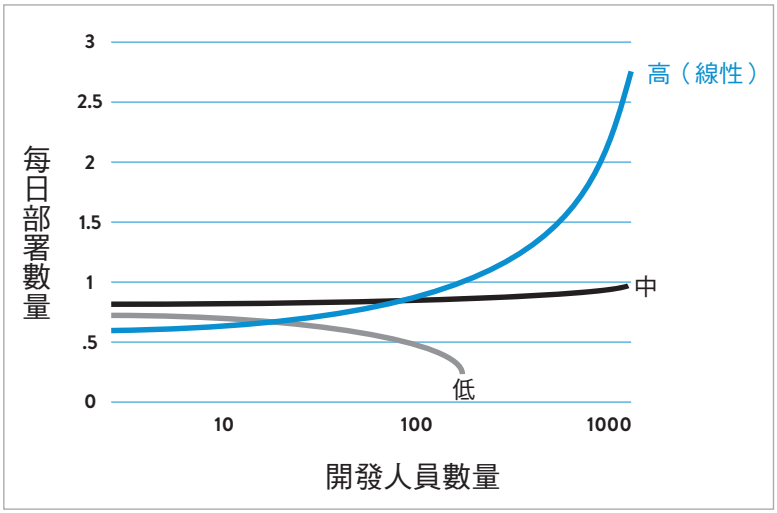


圖 1　每日部署數量 vs. 開發人員數量 [8]

（來源：Puppet Labs，2015 State of DevOps Report）

事實上，這個假設可以從資料中得到驗證。圖 1 顯示，在低效能組織中，每位開發人員的每日部署量隨著團隊規模擴大而下降，在中效能組織中變化不大，而對於高效能組織來說，數據則以線性成長。

8　只有每日至少進行一次部署的組織才會計入。

1 敏捷、持續交付與「三步工作法」

本章將會介紹「精實生產」的基礎理論，以及可以衍生目前所有 DevOps 行為的「三步工作法」原則。

我們將聚焦討論理論與原則，從製造業、高可靠度組織、高度信任的管理模型，以及其他方面習得的數十年經驗教訓所衍生出的 DevOps 實踐。DevOps 所蘊生的具體原則和模式，以及在科技價值流當中的實際應用，將在本書其餘章節深入探討。

製造業的價值流

精實生產的基本內容之一是「價值流（value stream）」的概念。首先，我們先在製造業的情境中定義它，接著再討論如何應用於 DevOps 和科技價值流（technology value stream）。

Karen Martin 和 Mike Osterling 在著作《*Value Stream Mapping*》（價值流程圖）將價值流定義為「一個組織根據客戶需求所執行的一系列有序的交付活動」，或是「為客戶設計、生產和提供產品或服務所需從事的一系列活動，包括資訊和原物料的雙重價值流。」

在製造業的生產流程中，隨處可見價值流：始於接收到客戶訂單，將原物料發送到工廠。為了縮短和預測價值流內的前置時間，需要努力建立一套流暢的暢流程，包括減少批次規模、降低「在製品」（Work In Process，WIP）數量、避免重工（Rework）等，同時還需要確保不會將殘次品傳遞到下游工作中心，並且持續從全局目標來優化整個工作系統。

科技價值流

在製造業中成就實體產品快速加工流程的原則和模式，同樣可以應用到科技業（及所有知識工作）中。在 DevOps 中，我們通常將科技價值流定義為「以科技轉化商業構想，向客戶交付價值所需要的流程」。

一開始啟動整個流程的是既定業務目標、概念、創意和構想，接著由開發部門接收工作，並將該任務新增到工作清單當作整個價值流的開始。

開發團隊接收工作之後，運用敏捷或迭代的開發流程，將創意點子轉化為使用者故事（user story）及功能性說明，然後設計程式，再將程式碼嵌入版本控制庫中，接下來每一次變更都被整合到軟體系統並進行測試。

應用程式或服務只有在生產環境中按預期正常運行，為客戶提供服務，所有的工作才產生價值。所以我們不但要快速交付，同時還要保證部署工作不會產生混亂和破壞，比如中斷客戶服務，導致效能下降或者資訊安全不合規等問題。

聚焦在部署前置時間

「部署前置時間（deployment lead time）」是價值流的一個子集，也是本書的重點所在。價值流開始於工程師[2]（包括開發、QA、IT 營運和資訊安全人員）向版本控制系統提交一個變更，結束於這個變更在生產環境中成功運行，為客戶提供價值，並產生有效回饋和監控資訊。

第一階段的工作主要包括設計和開發，它和「精實產品開發（Lean Product Development）」有很多相似之處：具有高度變化性和高度不確定性，不僅需要創意，某些工作還可能無法重來，導致無法給出確切的處理時間。第二階段工作主要包括測試和營運，類似於「精實生產（Lean Manufacturing）」。比起上一階段，此時需要創造性和專業技能，力求可預期性和自動化以滿足業務目標，將變化的可能性降到最低（比如，短而可預期的前置時間，接近零缺失）。

2 從此處開始，「工程師」一詞指在價值流當中的任何工作者，不僅限於開發人員。

我們並不提倡在設計／開發階段，串接式完成了一大批工作後，才轉入測試／營運階段（比如大批量的瀑布式開發流程或是長生命週期的功能分枝）。恰恰相反，我們的目標是讓測試／營運和設計／開發同步發生，從而產生更快的價值流和更高的品質。只有當工作任務以小批多次傳遞，並將注重品質這件事內建到價值流的每個部分時，這種同步模式才有機會實現。[3]

定義前置時間 vs. 處理時間

在精實社群中，「前置時間」與「處理時間」（有時候也稱為「接觸時間」或者「任務時間」）[4] 是評斷價值流效能的兩個常用指標。

前置時間在工單一建立就開始計時，在工作完成時結束；處理時間則從實際開始處理這個工作時才開始計時，它不包含這個工作在佇列中排隊等待的時間（見圖 2）。

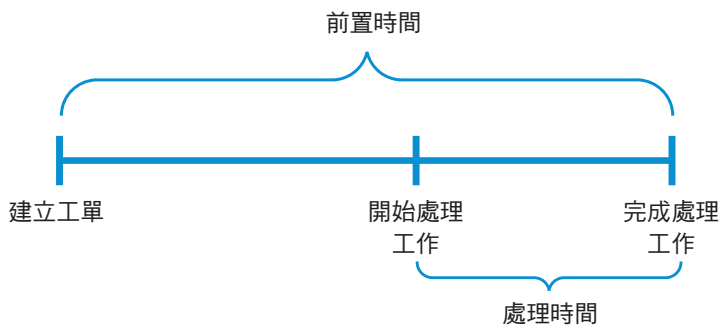


圖 2　部署工作的前置時間和處理時間

3　事實上，使用「以測試驅動開發」技法，甚至可以在編寫第一行程式碼之前首先進行測試。

4　Karen Martin 和 Mike Osterling 曾說：「為了避免混淆，我們選擇不使用『循環時間』這個詞，因為它還有其他的同義詞——處理時間、輸出速率或輸出頻率等。」同理，本書中傾向採用「處理時間」一詞。

因為前置時間才是客戶真正體驗到的時間，所以我們把重點放在縮短前置時間而不是處理時間上。不過，處理時間與前置時間的比例是十分重要的效率指標。想要實現快速流程並縮短前置時間，就必須縮短工作在佇列中的等待時間。

普遍情形：部署前置時間耗時數月

通常，部署前置時間動輒好幾個月。在龐大、複雜的企業組織中情況更是屢見不鮮，採用密切耦合（tightly coupled）的單體應用、少有整合測試環境、測試和生產環境的前置時間很長、嚴重依賴手動測試，或者需要各種審核批示流程等等。這種情形的價值流看起來如圖 3。

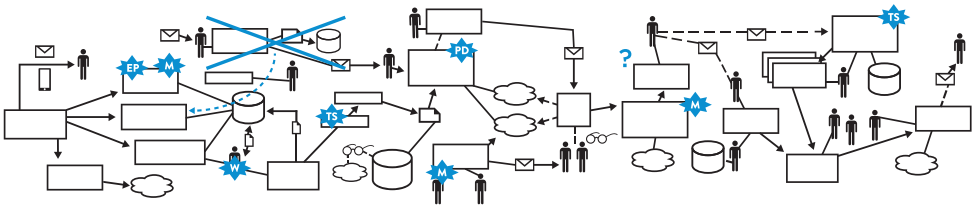


圖 3　某個部署前置時間為三個月的科技價值流

（來源：2015 年 Damon Edwards “DevOps Kaizen”）

一旦拉長部署前置時間，價值流的每個階段幾乎都需要英雄人物捨身搶救。有極大可能在專案即將結束前、當我們將開發團隊的變更合併後，才發現整個系統根本無法正常運作，有時甚至會出現連程式碼都無法成功編譯和測試的情況。每一個問題可能都需要幾天甚至幾週的時間來定位錯誤和修復問題，因此導致非常糟糕的用戶體驗。

DevOps 目標：部署前置時間只需要數分鐘

在 DevOps 的理想情況下，開發人員能快速且持續地獲得工作回饋，快速且獨立地開發、整合和驗證程式碼，並將程式碼部署到生產環境中（自己部署或交由他人部署）。

我們可以藉由以下方式達成理想目標：向版本控制系統持續提交小批量的程式碼變更，針對程式碼進行自動化測試和探索測試，然後再將它部署到生產環境中。如此一來，我們就能對程式碼變更是否能在生產環境成功運行保持高度信心，同時還能快速發現並修復可能出現的問題。

為了更便於實現上述目標，還需要模組化、高內聚、低耦合的方式優化架構，賦予小型團隊高度自治。即使不幸遭遇失敗，也能保持在可控範圍內，不至於對全局產生影響。

採用上述方式，可以有效將前置時間大幅縮短到分鐘級，即便在最壞的情況下，也不會超過幾小時。此時的價值流程圖如圖 4 所示。

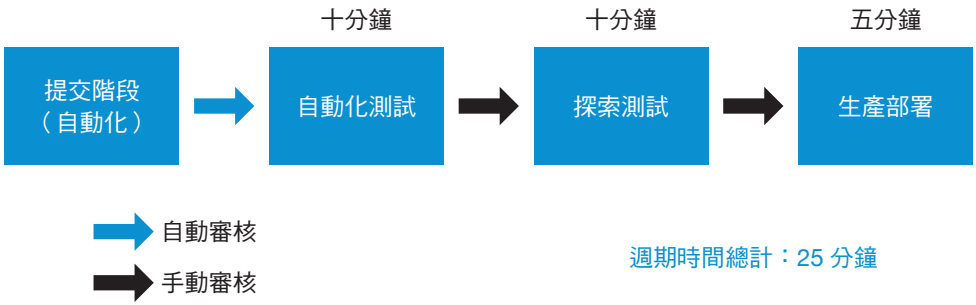


圖 4　前置時間以分鐘計算的科技價值流

採用上述方式，可以有效將前置時間大幅縮短到分鐘級，即便在最壞的情況下，也不會超過幾小時。此時的價值流程圖如圖 4 所示。

重工指標：%C/A

除了前置時間和處理時間外，科技價值流中的第三個關鍵指標是完成時間和精確所花費時間之百分比（% C/A）。這項指標可以呈現價值流中每個步驟的輸出品質。Karen Martin 和 Mike Osterling 認為「要獲取 % C/A，可以詢問下游端有百分之多少的時間接收到『真正可用』的工作，讓他們可以專心工作，不必修復錯誤資訊、補充資訊，或者釐清那些本該明確清楚的資訊。」

三步工作法：DevOps 的基礎原則

《**鳳凰專案**》將「三步工作法」視為基礎原則，並依此衍生出 DevOps 行為和模式（圖 5）。

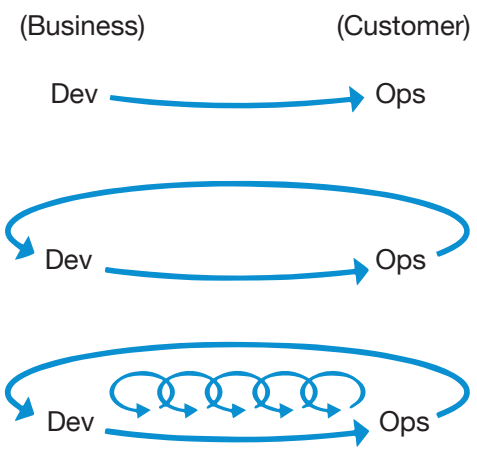


圖 5　三步工作法

（來源：Gene Kim 在 IT Revolution Press 部落格發表的「三步工作法：DevOps 的基礎原則」，http://itrevolution.com/the-three-ways-principles-underpinning-devops/）

第一步，暢流快速地從左向右流動，從開發平順過渡到營運，最後交付給消費者。為了最大程度地優化暢流，必須將工作以視覺化呈現，減少每批工作量的規模和等待間隔，避免將缺失傳遞至下游工作中心，注重每一環節的工作品質，並持續地優化全局目標。

透過加快科技價值流的流動速度，縮短前置時間以滿足內部或外部客戶需求，特別是縮減程式碼部署到生產環境的所需時間，可以有效提高工作品質和生產量，使企業具備強大競爭力。

相關實踐包括持續建構、整合、測試和部署，依照需求搭建環境、限制在製品（WIP）數量、建立能夠安全實施變更的系統與組織等。

因此當我們著手改善棕地系統時，不但要致力降低複雜性，提高可靠性和穩定性，同時必須將系統變得更快、更安全、更容易進行變更。即使只是為綠地型的互動式系統增加新功能，也常常會給所依賴的棕地型記錄式系統造成可靠性問題。讓下游系統能夠進行更安全的變更，可以幫助整個組織更快速、更安全地達成目標。

從最樂於創新的團隊開始

在每一個組織中，不同的團隊或個人，對於創新的態度各異。傑佛瑞 • 墨爾在《**跨越鴻溝**》一書中以曲線圖呈現了這種現象。所謂的跨越鴻溝，是指克服重重困難，找到比**創新者**和**早期採用者**（見圖 9）之後的更大群體。

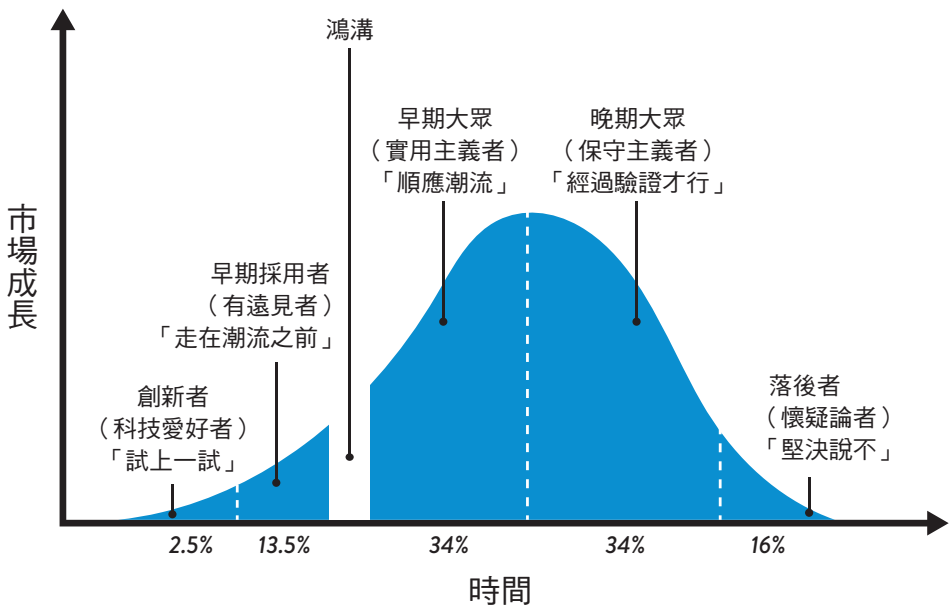
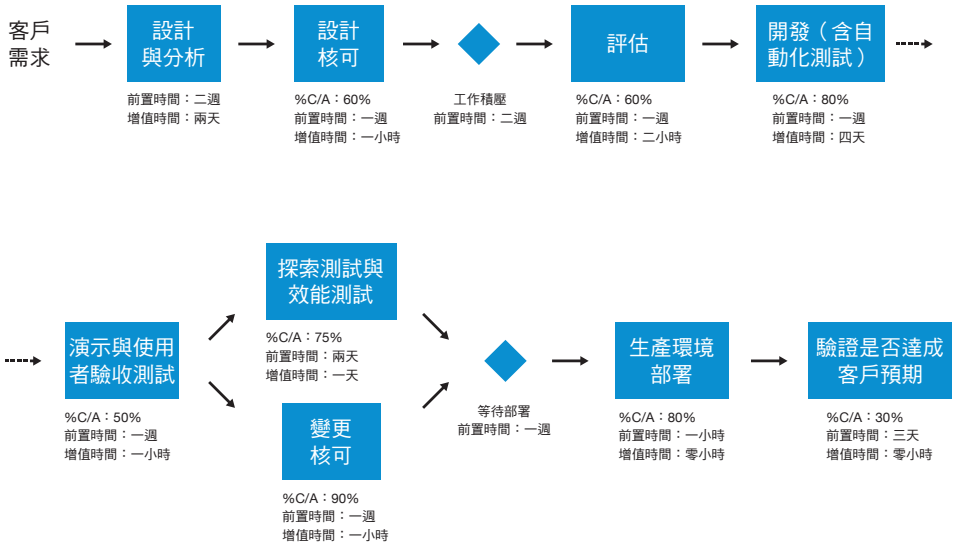


圖 9　技術採用生命週期

（來源：《跨越鴻溝》。）



數值匯總：
總前置時間：十週
總增值時間：七天半
綜合 %C/A：8.6%

圖 10 價值流程圖範例

（來源：《精實企業》）

領導者協助確定改善目標並指導團隊集思廣益，思考可能假設和相應對策。透過實驗測試各種假設，然後分析結果來判斷假設是否正確。團隊透過不斷重複及迭代以上做法，將新獲得的學習經驗應用於日後實驗中。

組織專門的轉型團隊

DevOps 轉型所面臨的一個先天挑戰是，這項計畫不可避免地會與目前業務產生衝突，這也是成功企業自然面臨的矛盾。任何一個成功運作多年（幾年、幾十年甚至幾百年）的組織早已建立符合該組織的實踐與運作機制，例如產品開發、訂單管理和供應鏈營運等。

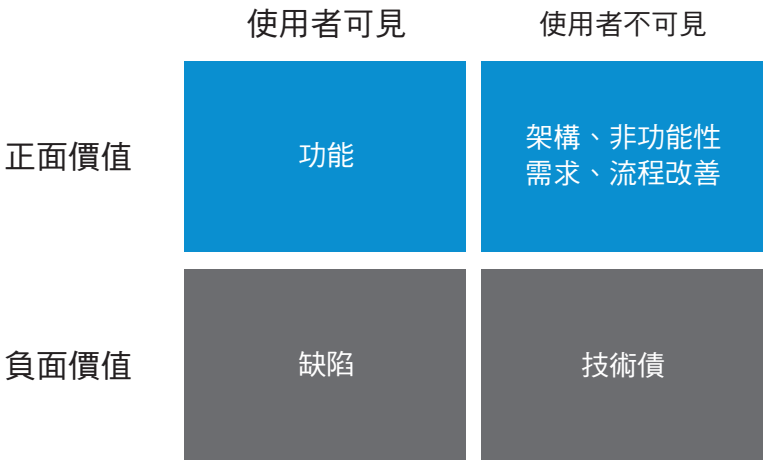


圖 11　將 20% 的時間用來創造使用者不可見的正面價值

（來源：Podcast 節目 Software Engineering Daily 於 2015 年 11 月 17 日播出的〈D. Sculley 的機器學習和技術債〉單元。）

eBay 在九零年代末期有過一次死裡逃生的經歷。之後，曾任 eBay 產品設計資深副總的 Marty Cagan 出版了關於產品設計和管理的重要著作《**啟示錄：打造用戶喜愛的產品**》。他總結了下列內容：

> 產品負責人和工程師之間的協作像是這樣：產品負責人將 20% 的團隊資源分配給工程相關活動，比如用來重寫或重新建構程式碼資料庫中有問題的部分，或者工程師認為有必要改善的部分，以免哪一天他們突然說：「我們必須停下來手邊進度，重寫所有程式碼。」如果情況非常糟糕，那可能需要投入 30% 或更高比例的資源。然而，如果發現團隊認為他們不需要 20% 的資源就能做這些事情，我反而會感到非常擔心。

Marty Cagan 指出，如果組織連這「20% 的稅」都不願意支付，那麼技術債將會持續惡化，最終消耗組織內所有可用資源。終有一天，服務變得脆弱不堪，功能交付將停滯不前，所有工程師都在解決可靠性問題或尋求臨時方案。

投入這 20% 的時間與資源，可以幫助開發人員和營運人員為日常工作中所遇到的問題找出長久對策，並保證技術債不會妨礙快速、安全地開發和營運工作。緩解員工的技術債壓力，對於降低工作倦怠程度也有效益。

Case Study

2011 年 Linkedin 的「反轉行動」

LinkedIn 的「反轉行動」（Operation InVersion）是一個相當有趣的案例，它證明了為何要將償還技術債作為日常工作的一部分。2011 年，LinkedIn 成功 IPO 上市。但半年過去了，公司依然在部署作業上面臨百般問題，苦苦掙扎。於是，他們啟動「反轉行動」，在兩個月內，停止所有功能開發，並對運算環境、部署和架構進行全面優化。

LinkedIn 成立於 2003 年，旨在幫助使用者「建立個人社交網路以獲得更佳就業機會」。網站上線的第一週就獲得了兩千七百位使用者。一年後，會員數超過了一百萬，並呈指數型增長。截至 2015 年 11 月，LinkedIn 已經擁有超過 3.5 億位使用者，每秒產生數萬次使用者請求，後台系統每秒要處理數百萬次查詢。

起初，LinkedIn 服務主要運行在自己開發的 Leo 應用上。這是一個單體 Java 應用，透過 servlet 服務每一頁面，並使用 JDBC 與後台 Oracle 資料庫進行連線。然而，早年為了跟上不斷激增的流量，團隊將兩個關鍵服務從 Leo 中切割出來：一個是在記憶體中處理會員連線關係的查詢，另一個則是基於該查詢的使用者搜索功能。

截至 2010 年，大多數新開發的功能都部署為新的服務，已經有近百個服務獨立運行於 Leo 之外。但是 Leo 本身還是只能每兩週部署一次。

LinkedIn 的資深工程經理 Josh Clemm 說，Leo 在 2010 年面臨重大挑戰。儘管使用了增加記憶體和 CPU 的垂直擴容方式，「但在生產環境中，Leo 還是經常崩潰，很難進行故障排除和復原，發佈新功能也非常