可維護性

重要，但時常被忽略

「可維護性」對韌性為什麼重要?

韌性 = 系統在**變動**的環境下
持續提供美好服務的能力

「可維護性」對韌性為什麼重要?

韌性 = 可用 x 易用 x 安心用

任何一項為 0，則系統價值為 0

「可維護性」對韌性為什麼重要?

保持韌性 = 可用 x 易用 x 安心用 x **好維護**

任何一項為 0，則系統**長期**價值為 0

「可維護性」對韌性為什麼重要?

維運成本佔系統總成本的極高比例 (50% ~ 80%)

因此，任何時候有多餘的資源時，

應優先考慮優化「維運」，再考慮優化「開發」

巡航要怎麼提升「可維護性」？

Level 1 巡航目標：掌握現況

需求	1. 清楚呈現系統主要角色和關鍵使用路徑 2. 用簡單明確的方式記錄需求 3. 依重要性和價值排序、分類未來需求
人員	1. 確保總是能快速找到「最熟悉需求的當責人員」及「最熟悉系統的當責人員」
外部依賴	1. 妥善管理外部依賴，如對外部依賴進行版本控制 2. 了解並遵守外部依賴的授權許可
資料	1. 資料可讀性：能清楚識別每個資料欄位的目的、特性、留存時間 2. 資料儲存結構變更管理：檢視資料表結構的異動版本記錄 3. 存取紀錄：對重要資料的讀取及變更，具備存取紀錄 4. 資料落地：資料流、儲存方法與位置、時效、生命週期、銷毀方法與驗證紀錄
系統	1. 系統文件：具備易於持續更新的系統文件，包括系統架構圖、API 文件等 2. 系統變更紀錄：具備系統變更紀錄，載明目的、上線時間
程式碼	1. 程式碼授權：擁有完整的程式原始碼與授權 2. 程式碼版本管理：以版本管理系統進行程式異動管理

Level 1 巡航目標：傳達架構原則

Do Less

解決問題、適合團隊、簡單直接

Level 2 巡航目標：規劃未來

需求	可追朔性、變更管理
人員	知識共享、技能發展、求才
外部依賴	
資料	
系統	變更管理、邊界、自動化、複雜度
程式碼	可讀性、可重用性、可測試性、...

架構為權衡結果，無標準答案

- 用第三方服務比較簡單?
- 應先考量未來可能的變更?
- 要彈性、要通用、要抽象化?
- Single Page Application 互動性佳？

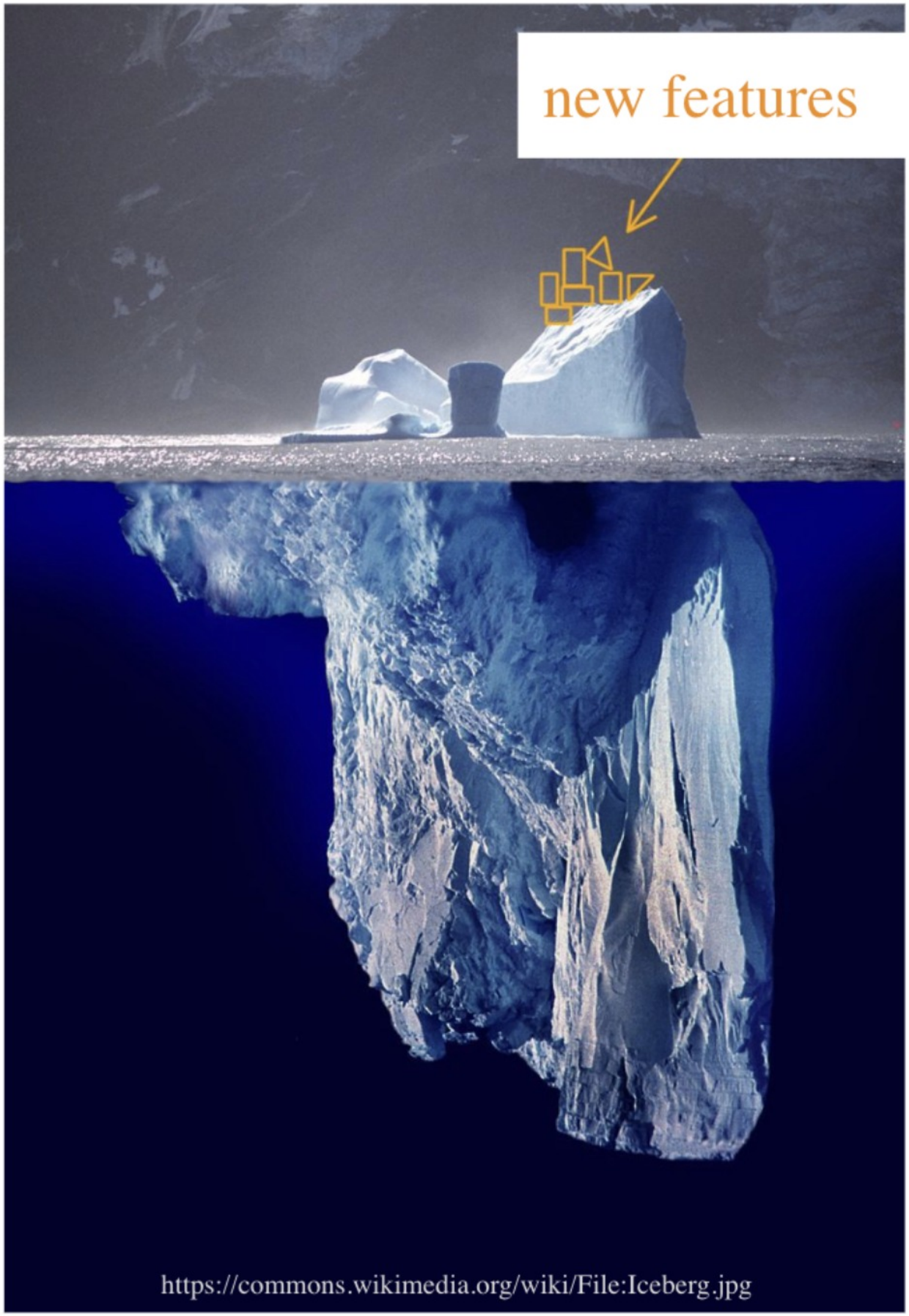
錯誤中累積出的原則

Do Less

解決問題、適合團隊、簡單直接

解決問題。首先，保持覺識

- 系統目的是什麼？
- 現在遇到什麼阻礙？



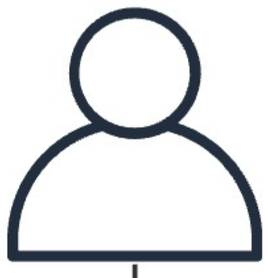
比如技術債的覺識

有效維運能量
（使用者有感的）

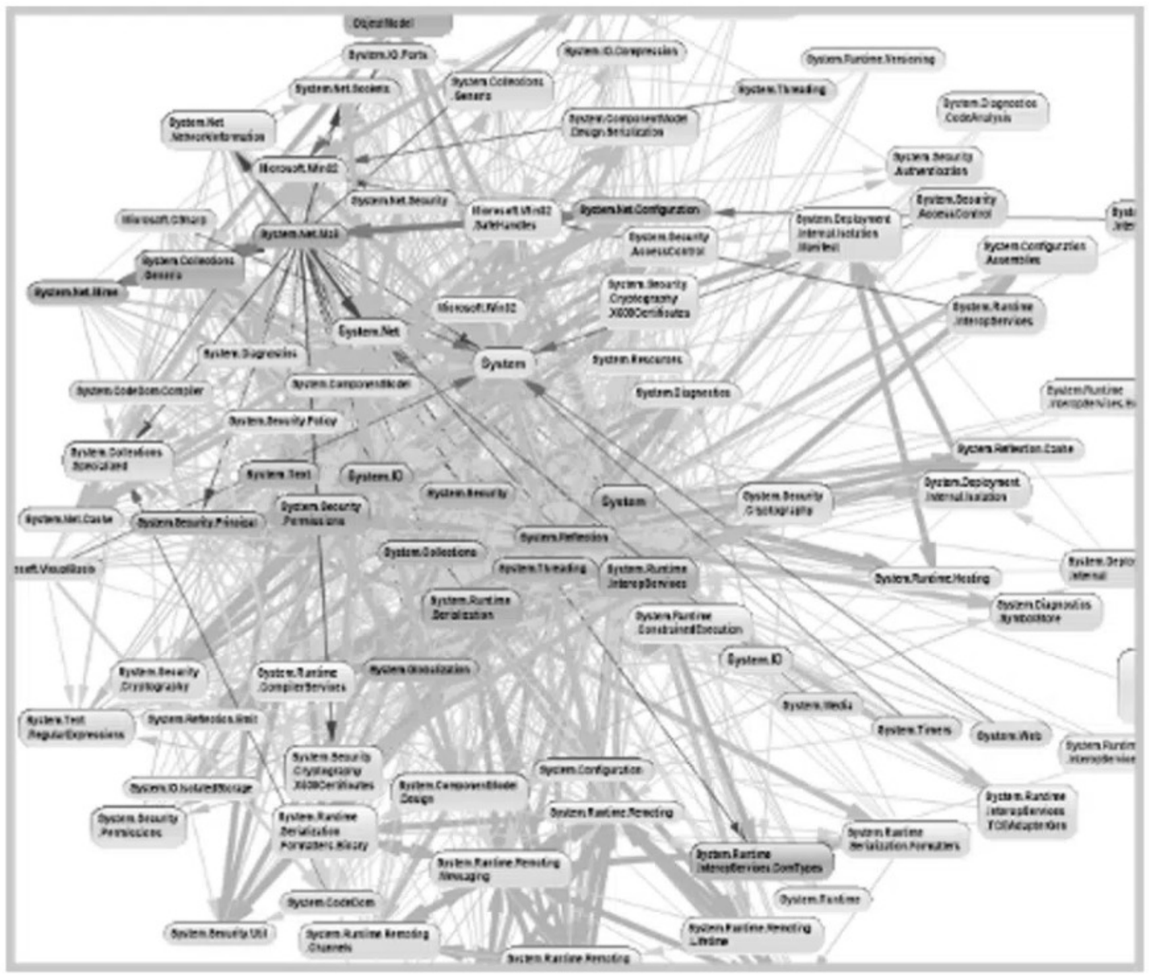
負擔

當疊加新功能時，冰山會下沉更多
我們維運整個冰山
「有效維運」僅佔一小部分

複雜度的覺識



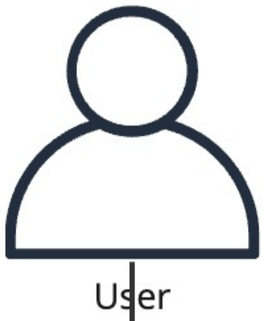
Anti-pattern: 大泥球



source: https://codeopinion.com/long-live-the-monolith-monolithic-architecture-big-ball-of-mud/

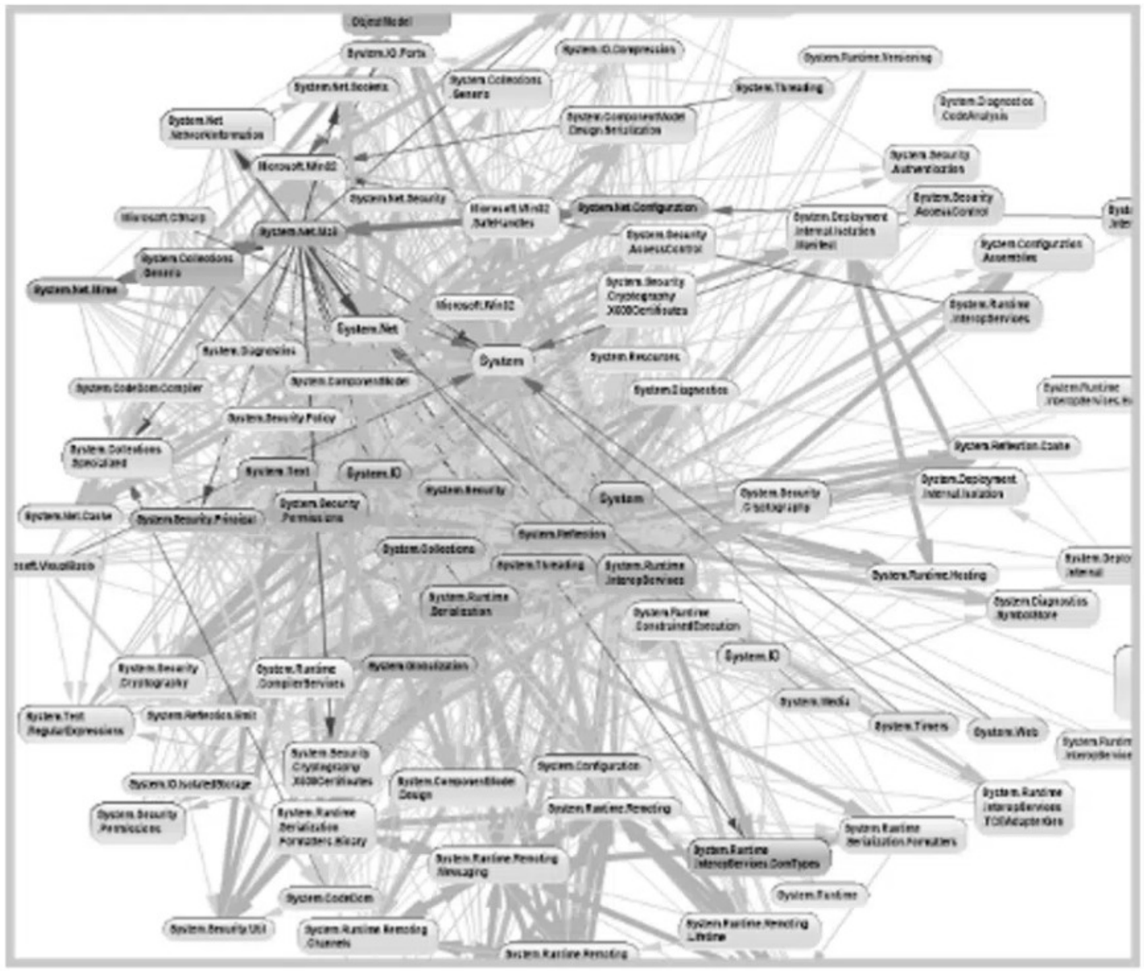
程式改不動了，無法發展新功能。
五個工程師。
重寫？ 5 年
重構？N 年

該怎麼做？



User

工程師甲：
可以逐一抽出子系統重寫，
我先做一個！



source: https://codeopinion.com/long-live-the-monolith-monolithic-architecture-big-ball-of-mud/

問卷子系統

API

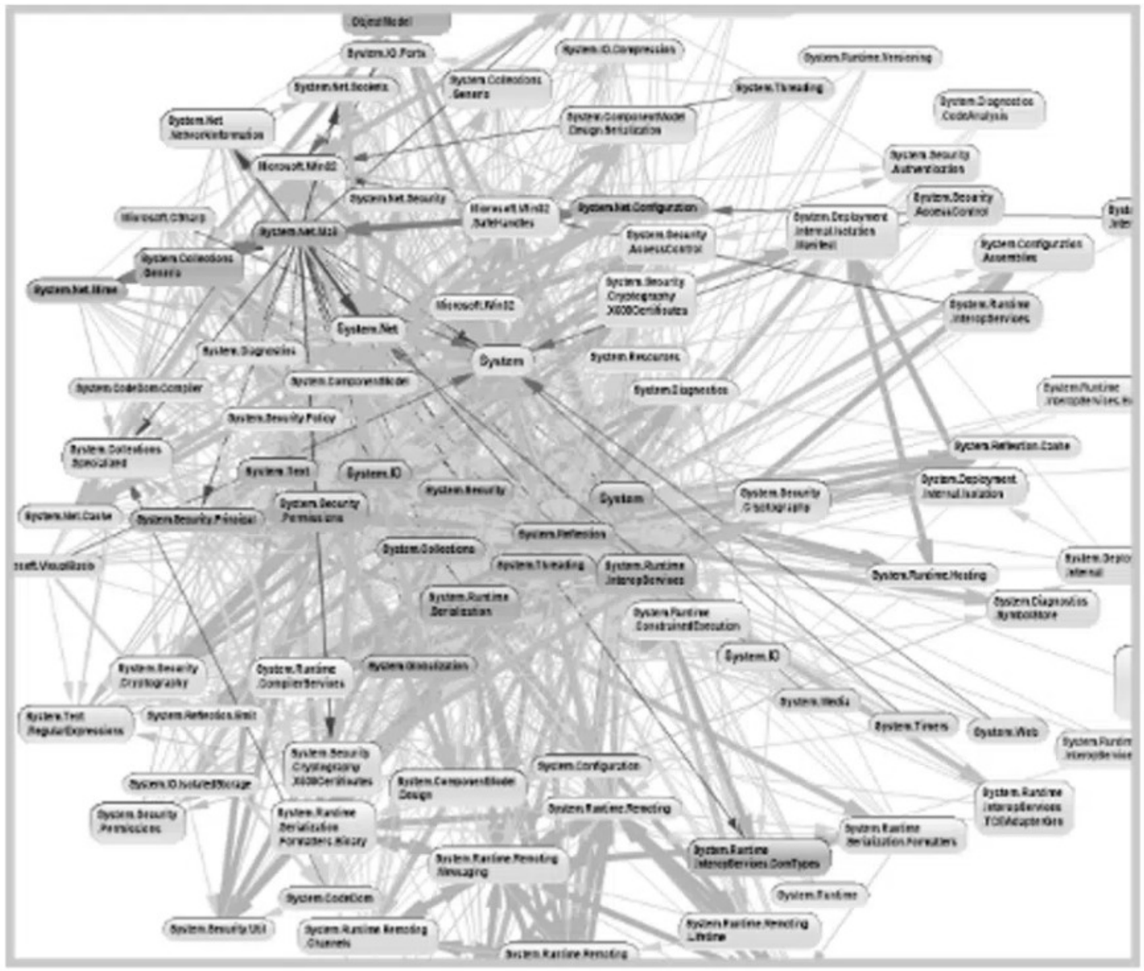
不到一年就完成了
程式很乾淨
新的子系統
新的部署流程
新的 API

dev flow

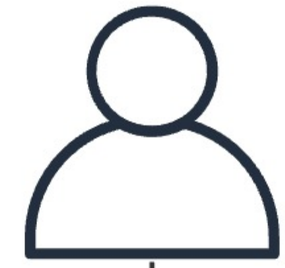
new dev flow

果斷放棄沉沒成本

兩倍以上的維運複雜度
＋ 兩年的開發停滯
＋ > 500 萬的人力資源浪費



source: https://codeopinion.com/long-live-the-monolith-monolithic-architecture-big-ball-of-mud/



User

教 統

dev flow

new flow

不停推演以符合原則

好架構，好流程 =

解決問題、適合團隊、簡單直接

當引入新技術、進行新規劃時，
先了解它解決了什麼問題？有沒有更簡單的方式解決？它適合團隊嗎？

架構需解決問題

滿足使用需求為根本

架構需**適合團隊**

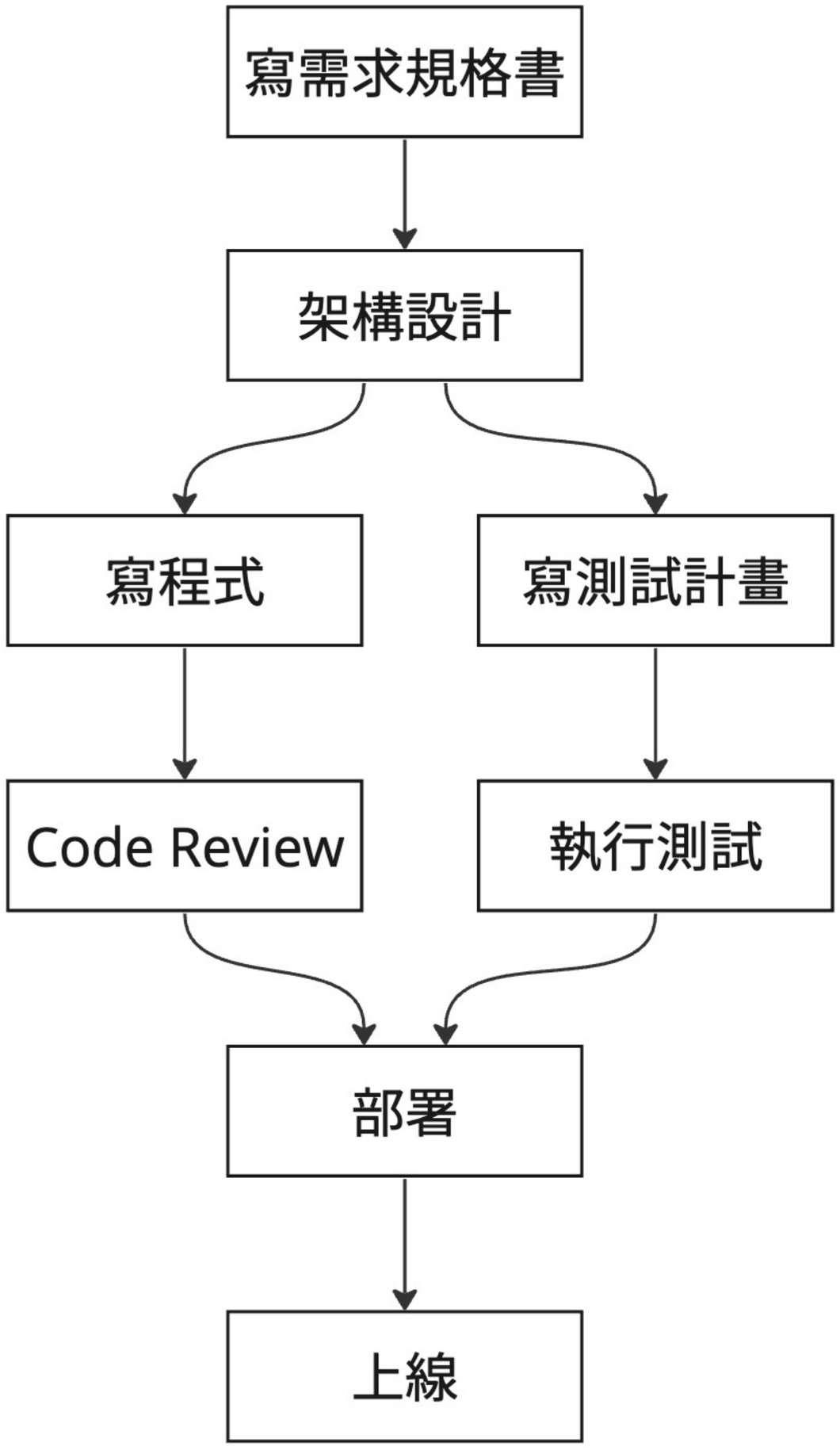
只滿足個人榮譽為大忌

架構需**簡單直接**

- 手裡有錘子，看什麼都是釘子
- 學了 design pattern 就到處用好用滿？
- 學了會計帳，就把金流系統都用會計帳做？
- Blockchain 很潮、AI 是趨勢？
- 試算表很難用，做成系統才好？

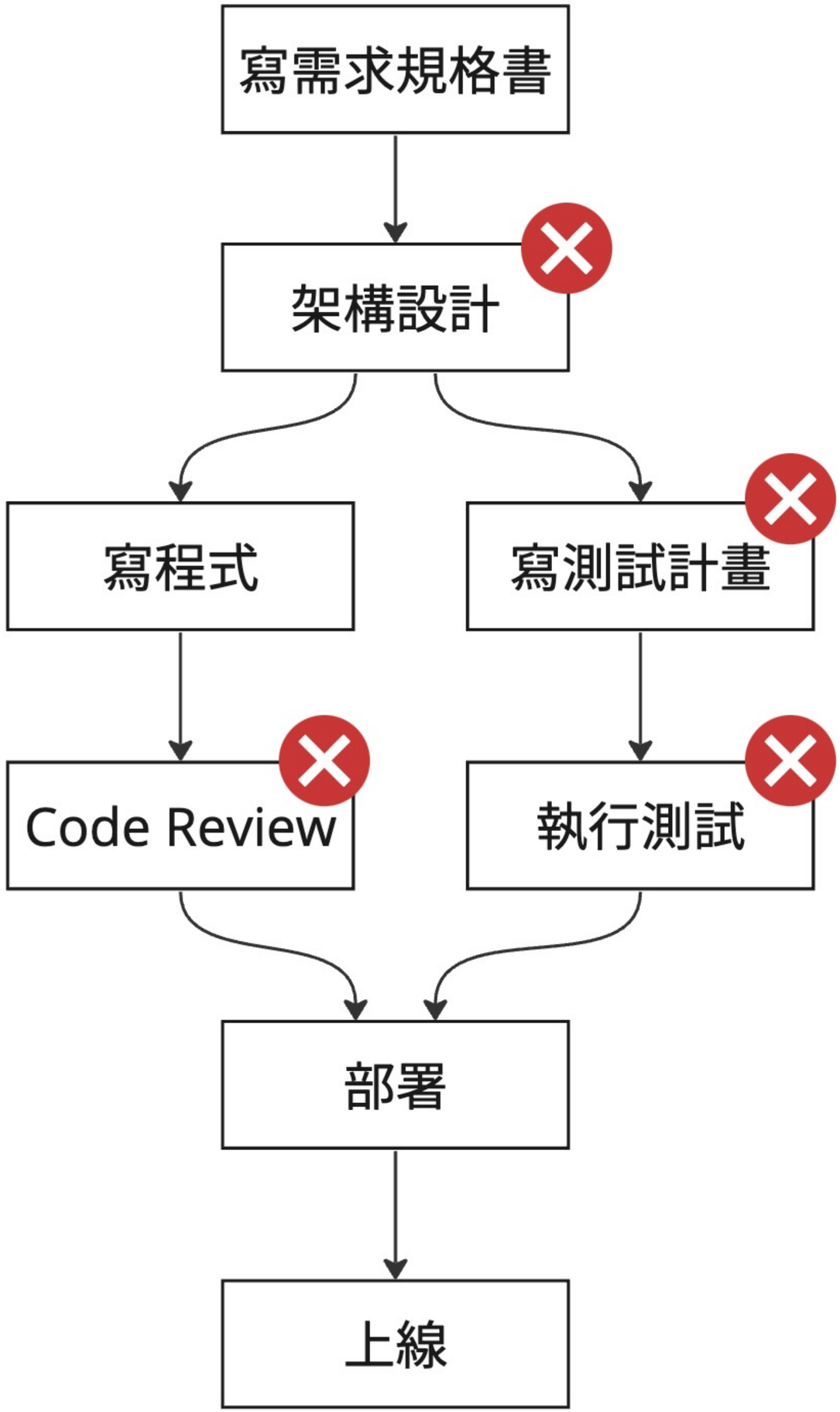
流程也要 Do Less

任何**刻意為之**卻**非強制**的流程，
成本高且難以長久

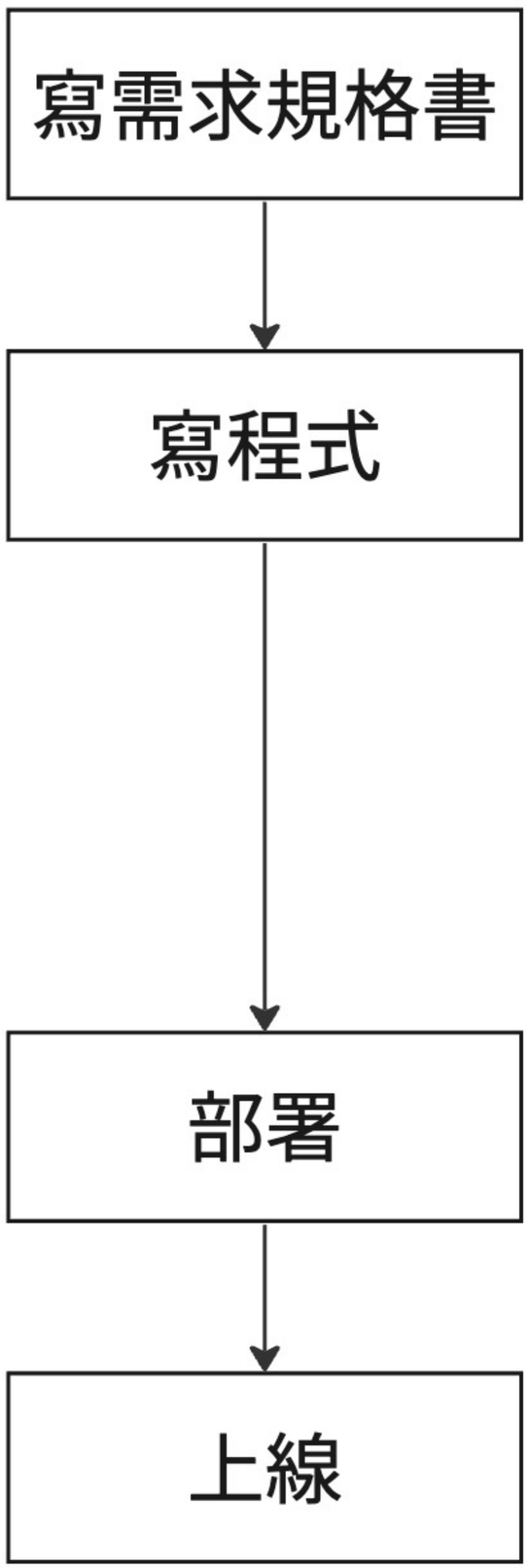


一個經典的開發流程為例

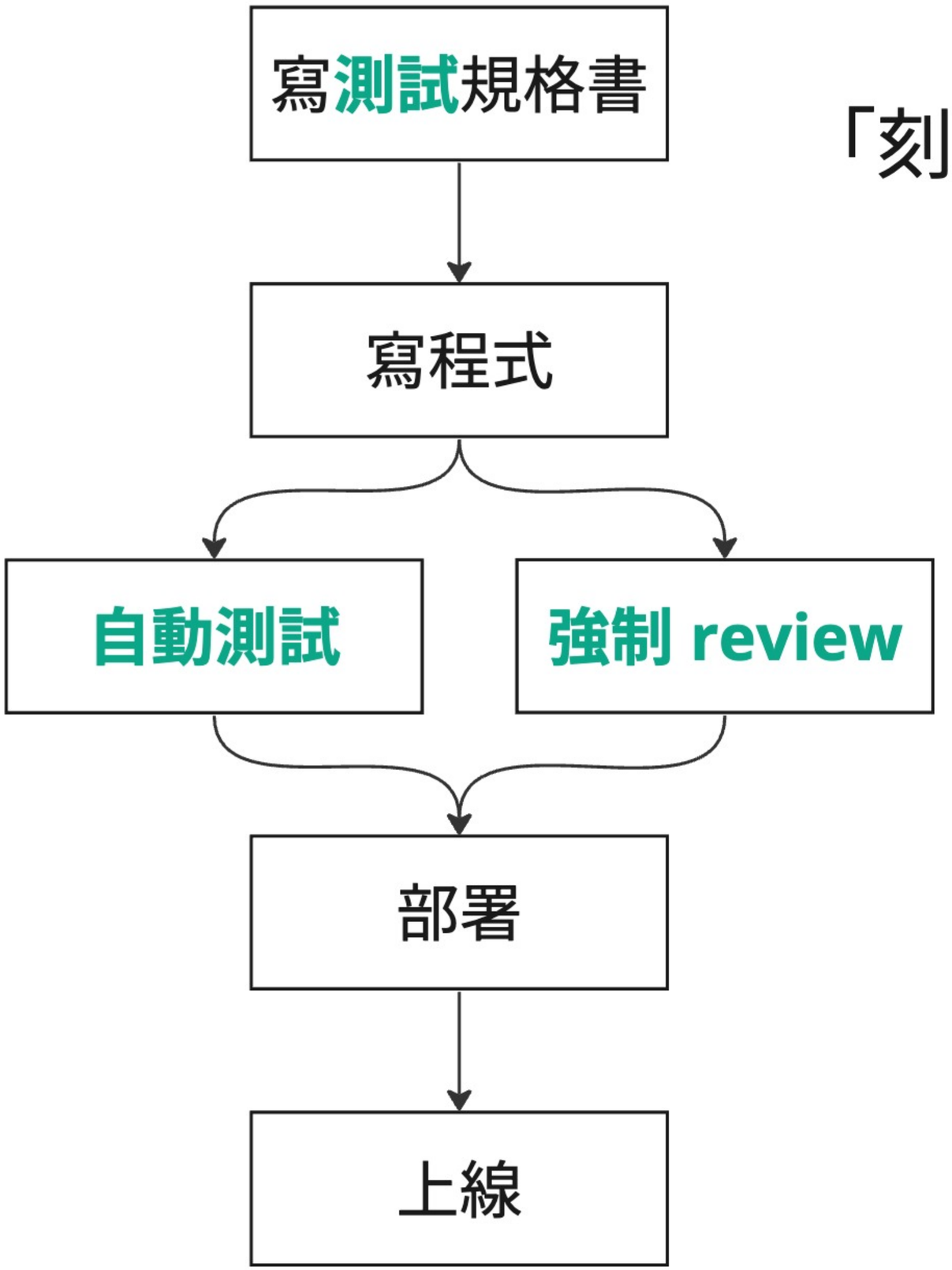
什麼將被犧牲？



人力短缺、時程趕時...



人力短缺、時程趕時...



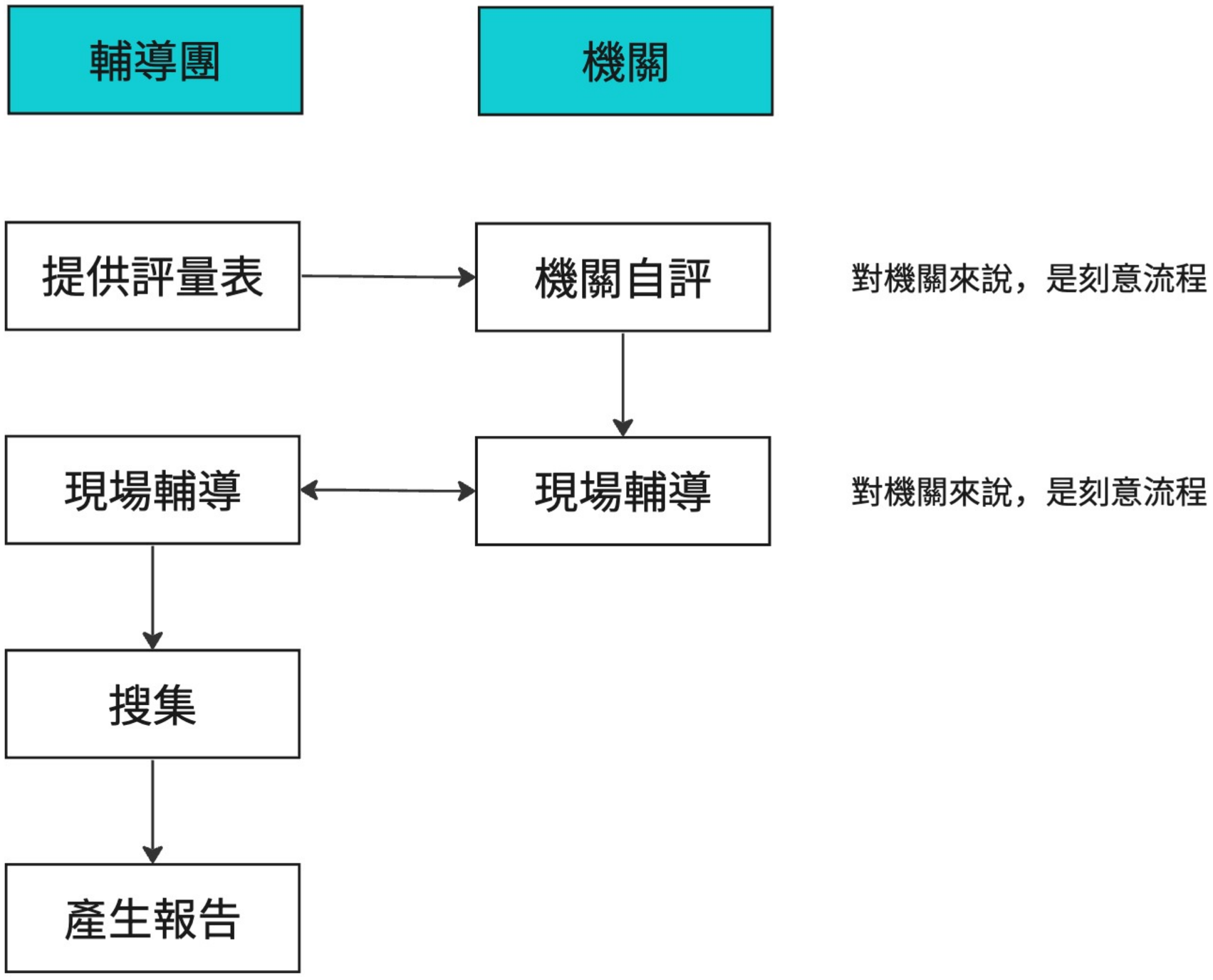
「刻意流程」轉為「必要流程」

自動化

強制性

巡航，然後呢

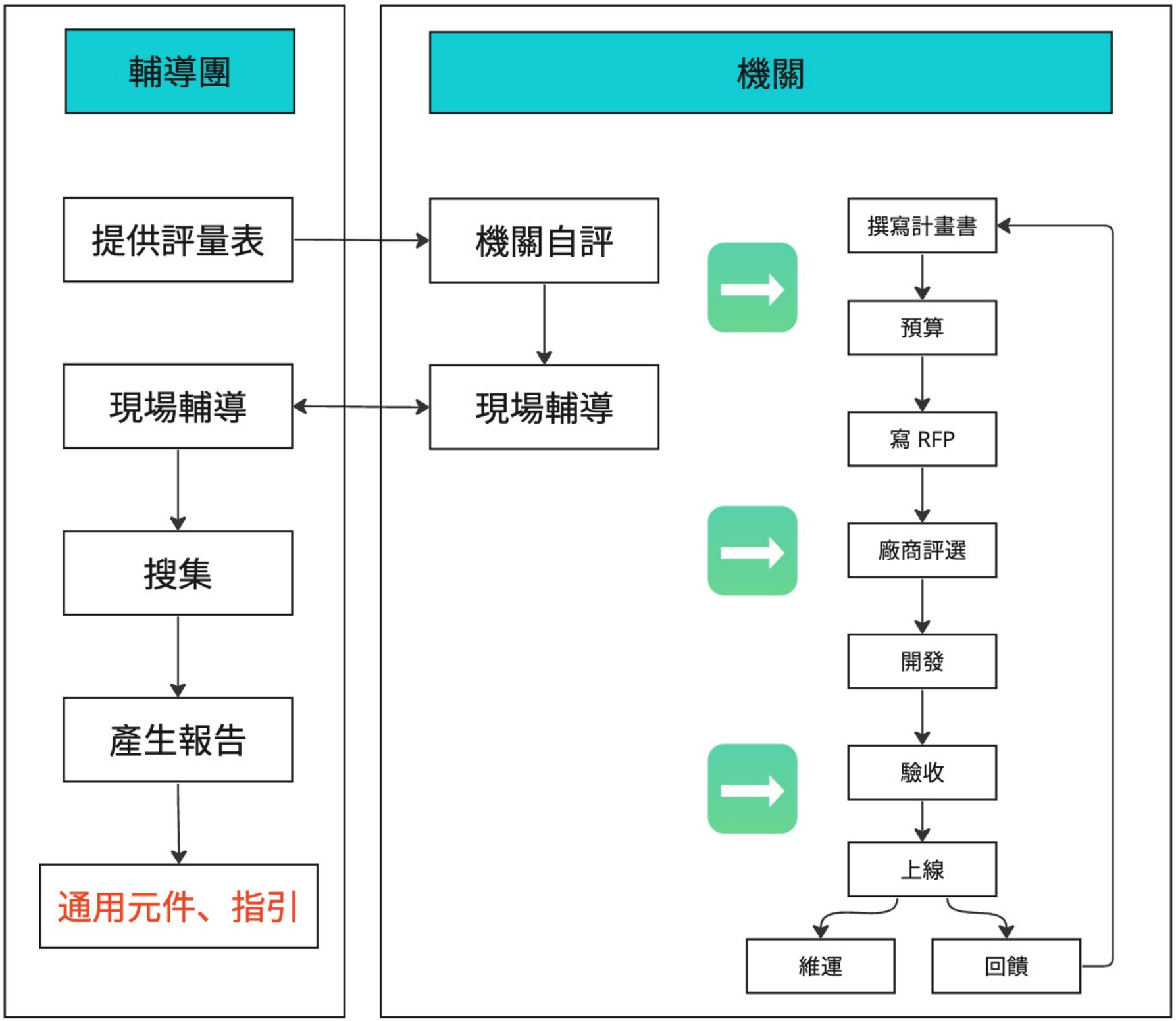
任何**刻意為之**卻**非強制**的流程，成本高且難以長久



巡航，然後呢

如何「溶」入機關的「必要流程」

任何**刻意為之**卻**非強制**的流程，成本高且難以長久



規模化

產生自改善循環，**讓機關自己動起來**

自動檢測、評分、榮譽榜、Status Page、SBOM 上傳、lighthouse

小步快行

最小可行性成果，讓今年不白走

系統有大、有小。資源可能充足、不足。

如果要讓每個系統至少前進一小步，增進韌性。

這個最小可行性成果是什麼呢？