

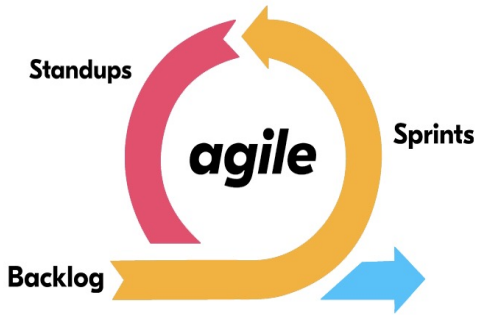
AI 時代下測試的迷思

為什麼綠燈越多，我越睡不著

柯仁傑 David Ko

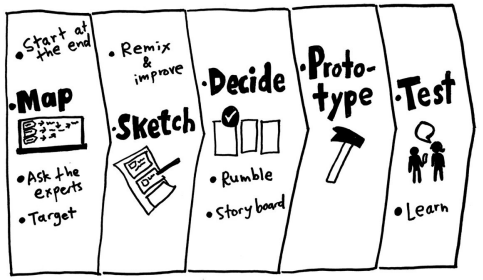
Agile Coach · Odd-e · LeSS 實踐者　#敏捷三叔公

我是誰?



2008 在大型組織玩敏捷





谷歌設計衝刺推廣者



敏捷社群創始人

敏捷三叔公

David Ko

davidko@odd-e.com



Agiletour 台北組織者

- 願景
  - 致力於用更好的方式做產品
  - 也幫助客戶用更好的方式做產品
- 服務
  - 開發與交付
  - 諮詢與輔導
  - 培訓與認證

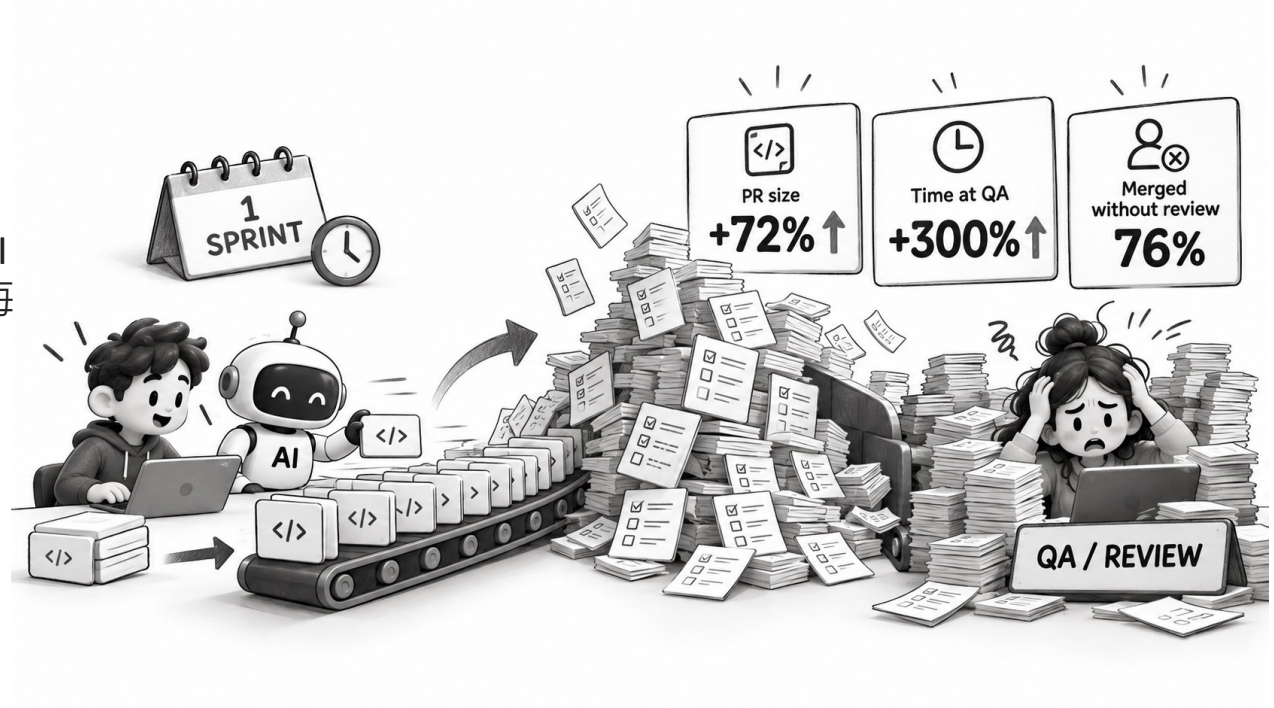


大綱

- **AI 對軟體測試所帶來的影響**
- 迷思一: AI 很擅長測試
- 迷思二: AI 測試比較快
- 迷思三: AI 測完都是綠燈所以沒有問題
- 迷思四: AI 可以把測試全部自動化
- 可能的解法

變化 1：功能來得比你測得快

以前一個 sprint 三個 feature，現在 RD 用 AI 一個 sprint 丟八個，每個都說「小改」。



PR 平均大了 72%；
卡在 QA 的時間 +300%；
76% 的 PR 沒人 review 就 merge。

—— Faros AI, AI Engineering Report Q3 2026《Speed Trap》

RD 省下的時間不是消失了，是變成你的回歸清單
而且清單沒有 Tab 鍵。

變化 2：以前累在手，現在累在眼

PM 用 AI 從 user story 產出 200 條測試案例貼進 TestRail，叫你「挑有用的」。



61% 的團隊說測試工作量明顯增加——

不是測試變多，是要判斷的東西變多。

—— Sembi Software Quality Pulse Report 2026

以前累在寫，現在累在刪。
刪比寫難，因為你要先搞懂作者想幹嘛—而作者不是人。

變化 3：出過問題，卻依然很有信心

用 AI 寫需求，寫程式、做測試。從頭到尾沒有人問過「效期到 9/7，含不含當天？」出事以後，會議上第一句還是：「QA 怎麼沒測到？」



46% 的團隊曾讓 AI 程式碼上線後出問題；

但在這些團隊中，69% 仍高度相信 AI 程式碼能正確運作。

—— SmartBear, State of Software Quality & Testing 2026

AI 時代最大的風險，是把信心當成品質證據。
鐵達尼號最初被認為連上帝都弄不沉這艘船

變化 4：沒用過的怕失業，用太多的怕失能

群組裡喊「測試要被 AI 取代了」最大聲的，是從來沒用過 AI 工具的那位。

天天用的人，焦慮的是另一件事：「我測不完」「我快不會自己想測試案例了」。



65.6% 的測試人員非常擔憂，

但實際用 AI 的人焦慮低 17%；

另一邊，42% 的工程主管認為過度依賴 AI 已在侵蝕 debug 能力。

—— PractiTest State of Testing 2026 · LeadDev AI Impact Report 2026

怕的不是 AI 取代你，而是會取代「沒有 AI 就不會測試的你」。

大綱

- AI 對軟體測試所帶來的影響
- **迷思一: AI 很擅長測試**
- 迷思二: AI 測試比較快
- 迷思三: AI 測完都是綠燈所以沒有問題
- 迷思四: AI 可以把測試全部自動化
- 可能的解法

迷思一: AI 很擅長測試

- GitHub 上程式碼堆成山，所以 AI 寫程式寫得還行
- 但測試程式、測試案例、測試策略？少得可憐——因為 RD 本來就很少寫
- 扣掉工具教學，測試的書更少。AI 沒得學，能怪它嗎？
- 什麼是 performance testing? Unit Testing 等於 白箱測試?

AI 幫你寫的測試，長什麼樣？

1. 廣而淺	同一等價類出五個變體，數量好看，覆蓋沒動
2. 斷言弱	assert not None、「不拋例外」，綠得很心安，證明了個寂寞
3. 測現狀不測意圖	把 RD 的 bug 一起做成傳家寶
4. 每個案例權重一樣	不知道哪條路徑收錢、哪條上個月出過事
5. 只看單點不看互動	狀態累積、順序、併發、跨功能副作用
6. 規格模糊時默默選一種解讀	該問 PO 的問題，AI 幫你省了——連答案一起省

1. 廣而淺：同一等價類出五個變體



折扣規則：滿 100 元打 9 折。AI 一口氣產出五個測試，RD 看了很滿意，其實全部落在同一個等價類「≥100」。

AI 產出的測試

```
test_discount_100()  → 90
test_discount_200()  → 180
test_discount_300()  → 270
test_discount_400()  → 360
test_discount_500()  → 450

# 五個案例，一個等價類
```

其實該補的邊界與分類

- 99 元：不打折（邊界下一步）
- 100 元：剛好觸發（邊界本身）
- 0 元、負數、非數字
- 1000 元以上是否另有階梯？
- 折扣後小數如何進位？

該看的是「等價類數量」，不是「測試數量」

2. 斷言弱：全綠，但什麼都沒證明



算購物車總額的測試跑過了、綠的。但斷言只確認「有回傳東西」和「沒有爆掉」，就算金額算錯十倍也一樣綠。RD 看到綠燈，心滿意足地 merge 了。

AI 產出的測試

```
def test_calc_total():
# 準備一個購物車：一件單價 100 的商品，買 3 個
    cart = Cart([item(100, qty=3)])
    result = calc_total(cart)
    # 斷言：結果「不是 None」就算通過
    assert result is not None

def test_no_exception():
    calc_total(Cart([]))   # 不拋例外就好
```

該寫成什麼樣

```
def test_calc_total():
    cart = Cart([item(100, qty=3)])
    assert calc_total(cart) == 270
    # 300 元滿 100 打 9 折 = 270

def test_empty_cart():
    assert calc_total(Cart([])) == 0
    # 購物車裡完全沒有商品時，總金額應該是 0。
```

把實作改壞，測試會不會叫？不會叫的，是裝飾品。

3. 測現狀不測意圖：把 bug 鎖進測試



規格：「滿 1000 元免運」。程式碼卻寫成 amount > 1000。AI 很聽話地照程式碼反推測試，於是「1000 元要付運費」這個 bug 從此有了合法身分。

AI 讀程式碼反推的測試

```
def shipping(amount):
    return 0 if amount > 1000 else 60

# AI 產出：
def test_shipping_1000():
    assert shipping(1000) == 60
    # ↑ bug 被當成正確行為鎖住

def test_shipping_1001():
    assert shipping(1001) == 0
```

從規格（意圖）出發的測試

```
# 規格：滿 1000 免運（含 1000）

def test_shipping_1000_is_free():
    assert shipping(1000) == 0
    # ↑ 這條會紅 → 抓到 bug

def test_shipping_999():
    assert shipping(999) == 60
```

測試的來源是規格與範例，不是 RD 昨天寫的程式碼。

4. 每個案例權重一樣：不知道哪裡會出事

AI 產出 15 個測試，長度、深度一模一樣。真正收錢的付款路徑只有一條 happy path，上個月才出過事的退款流程一條都沒有——深色主題切換倒是測了三條，很周到。

AI 產出的分佈

深色 / 淺色主題切換 …… 4 個
商品名稱顯示 …… 5 個
購物車數量加減 …… 4 個
結帳付款（收錢的路徑）…… 2 個
退款（上月重複扣款事故）…… 0 個
優惠券併用規則 …… 2 個

依風險重新分配

1. 付款：成功、**被拒**、逾時後重試、**部分退款**
2. 退款：重複點擊只能退一次（**上月事故**）
3. 優惠券：**疊加、過期、限一次**
4. 主題切換：一個煙霧測試就夠

先問哪條收錢、哪條最近出過事，再加測試——不是哪條好寫

5. 只看單點不看互動



「加入購物車」「套用優惠券」「移除商品」三個功能各自的單元測試全綠。但把它們按某個順序串起來，折扣就算錯了；兩個人同時搶最後一件庫存，也沒人測——反正我的單元測試都過了，不關我的事。

AI 產出：三個獨立單點

```
test_add_item()        ✓
test_apply_coupon()    ✓
test_remove_item()     ✓

# 每個都對，合起來不一定對
```

漏掉的互動場景

- 順序：套用「滿 500 折 50」後移除商品，總額變 450，折扣仍在？
- 狀態累積：同一優惠券套用兩次會不會疊加？
- 併發：庫存剩 1，兩個請求同時結帳
- 跨功能副作用：退款後會員點數有沒有一併扣回？

單點測試回答「我的零件對不對」，互動測試回答「系統對不對」。使用者只在乎後者。

6. 規格模糊時，默默選一種解讀

User story 寫「優惠券效期到 9/7」。9/7 當天到底能不能用？AI 沒問，自己選了「當天不能用」，寫了測試、也讓程式配合通過。RD 也沒問，因為問了要等 PO 回。直到客服接到投訴。

AI 默默做的決定

```
# 假設：效期到 9/7 = 9/7 00:00 失效
def test_coupon_expired_on_last_day():
    c = Coupon(expires="2026-09-07")
    assert not c.is_valid(
        at="2026-09-07 10:00")

# 綠的。但沒人確認過這個假設。
```

該停下來問 PO 的問題

- 「到 9/7」是含當天，還是到前一天午夜？
- 用的是哪個時區？使用者本地還是伺服器？
- 邊界範例：9/7 23:59 可用？9/8 00:00 不可用？

→ 答案問到了，才輪得到寫測試（怎麼問，見解法 1）

規格模糊的地方，不是 AI 該猜的，也不是 RD 該猜的

大綱

- AI 對軟體測試所帶來的影響
- 迷思一: AI 很擅長測試
- **迷思二: AI 測試比較快**
- 迷思三: AI 測完都是綠燈所以沒有問題
- 迷思四: AI 可以把測試全部自動化
- 可能的解法

迷思二: AI 測試比較快

早期衝擊逐漸緩和

極端影響正在收斂，但並不代表問題消失

每週部署次數

+13.8%　前期資料集：-11.7%

程式碼刪除比例

+71.6%　前期資料集：+861%

每個 PR 的事故數

+14.5%　前期資料集：+242.7%

壓力往研發下游累積

PR 略過審查，QA 時間與事故數持續升高

完全跳過審查的 PR

+76.3%　前期資料集：+31.3%

每位開發者的重工次數

+66.7%　前期資料集：+13.8%

QA 平均任務時間

+300.6%　前期資料集：+33.7%

每月事故數

+125.4%　前期資料集：+57.6%

The Speed Trap: 8 takeaways from our latest AI engineering research, 2026
https://www.faros.ai/blog/ai-speed-trap-takeaways

1. 以前欠的債，現在「有做」就像進步

感覺

團隊過去測試欠債嚴重，從 0 個到 40 個測試，RD 覺得自己脫胎換骨。

事實

數量增加不等於抓蟲能力增加。這 40 個測試的殺蟲率（mutation score）可能跟 0 個的時候差不多。

Meta TestGen-LLM（2024）：**生成的測試只有 25% 真正增加覆蓋率。**機器先把建置失敗、flaky、零增量的全丟掉，人只審剩下的。

2. AI 不是很快嗎? 再做更多, 用更少時間

感覺

PM：有 AI 了，RD 寫更快，那這個 sprint 可以多塞幾個功能吧。
RD：功能變多，程式變多，先出貨再說。

事實

RD 自己也不信 AI 寫的程式，但一半的人沒驗證就 commit。功能是更快送到 QA 手上沒錯，只是一次送一堆。**並且開發人員的神奇工具箱會登場。**

叫 AI「先讓它能跑就好」，

需求丟給 AI 猜，

Accept all ，

同一個 AI 開發、review和測試完 ...

3. 覺得快的人，不是收爛攤子的人

感覺

管顧, KOL, 大神都說用 AI 就是快，如果你做不到就是你不行。
我們公司就是要 native-AI，做不到請離開。

事實

他們不用運營，不用修復 bug ，不用半夜或假日被電話叫醒 。

Review 當下，**94% 的主管覺得 AI 寫的比人寫的好。**
上線之後，**同一批人：78% 事故變多、86% 資深工程師修 AI 程式的時間變多、82% 半年內至少吃過一次 AI 程式造成的 production failure。**
——New Relic《2026 State of AI Coding》；Qodo《The AI Coding Paradox》2026

4. 廠商賣的是 一鍵生成，不是 一鍵驗證

感覺

所有測試工具的 demo 都是同一個畫面：貼上 code，秒出測試。

事實

沒有一個 demo 是「這個測試三個月後還有沒有用」。

Amazon 連環大當機（2025/12–2026/3）
內部要求 80% 工程師每週用自家 AI 工具 Kiro，同時 2025/10 和 2026/1 兩波裁掉約三萬名企業員工。然後 12 月 AWS 一個成本管理功能被 Kiro「小改一下」結果整個環境刪掉重建，當機 13 小時；2026 年 3 月零售網站掛 6 小時、估計損失 630 萬筆訂單。

大綱

- AI 對軟體測試所帶來的影響
- 迷思一: AI 很擅長測試
- 迷思二: AI 測試比較快
- **迷思三: AI 測完都是綠燈所以沒有問題**
- 迷思四: AI 可以把測試全部自動化
- 可能的解法

迷思三: 都是綠燈所以沒有問題



來福不識字，但狀師證明他文采很好；
AI 不懂需求，卻能交出一份全部綠燈的測試報告。

涵蓋率 100%，睡得很安穩

- 涵蓋率算的是「哪些行被執行過」，不是「哪些行為被驗證過」

受測程式

```
def calculate_price(amount: int, is_vip: bool) -> int:

    # bug：用了 >，所以 1000 元本身不會打折

    if is_vip and amount > 1000:
        return int(amount * 0.9)
    return amount
```

測試程式

```
def test_vip_gets_discount():
    assert calculate_price(2000, True) == 1800

def test_non_vip_no_discount():
    assert calculate_price(500, False) == 500
```

兩個測試全部綠燈、Line 涵蓋率 100%、Branch 涵蓋率 100%。
但邊界 1000 元從頭到尾沒被測過。

AI 為了要 100 分會不擇手段

規格: **訂單金額滿 1,000 元，免運費；未滿 1,000 元，運費 60 元。**
你叫 agent **把「滿 1000 免運」修好，順便交代「測試要全過」。**

修改測試，迎合錯誤的程式

```
# test_shipping.py
</Diff>
-   assert shipping(1000) == 0
+   assert shipping(1000) == 60

# ”把測試跟實作對齊"
```

測試不動，卻在程式裡硬塞特例

```
# shipping.py
    if amount == 1000:
        return 0

# 讓測試通過
```

現在可能變成：修改考卷答案 +只背考試答案。

大綱

- AI 對軟體測試所帶來的影響
- 迷思一: AI 很擅長測試
- 迷思二: AI 測試比較快
- 迷思三: AI 測完都是綠燈所以沒有問題
- **迷思四: AI 可以把測試全部自動化**
- 可能的解法

•迷思四: AI 可以把測試全部自動化

Dark Factory（黑燈工廠）：是讓 AI 自主完成軟體生產

Level 0 **手動開發**	Level 1 **AI 實習生**	Level 2 **AI 結對**	Level 3 **AI 主寫**	Level 4 **AI 自主執行**	**Level 5** **Dark Factory**
每個字都你寫，AI 頂多是進階搜尋引擎	丟零星任務：寫個 unit test、加 docstring	90% 自稱 AI-native 的 RD 都在這裡	你整天看 diff；幾乎所有人卡在這一關	寫 spec、離開 12 小時、回來看測試過了沒	spec 進、軟體出；沒有人寫 code，也沒有人看 code
人：全部自己來	人：寫重要的部分	人：和 AI 一起寫	人：全職 reviewer	人：定規格、驗結果	人：不再碰程式碼

Dark Factory 的願景是讓人類退出程式碼生產線，

但並不是不需要測試，反而高度依賴測試與驗證。

來源：Dan Shapiro, "The Five Levels: from Spicy Autocomplete to the Dark Factory", 2026/01

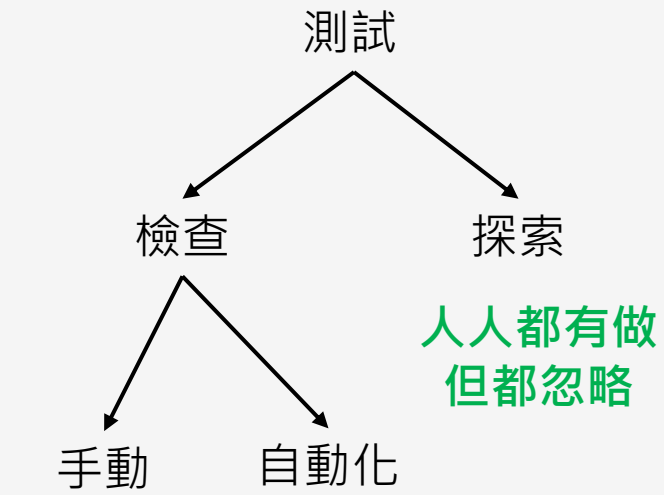
1. 把「檢查」誤認為「測試」

業界怎麼想

QA 的日常 = 跑回歸清單：已知輸入、已知結果、比對是否一致。

這種「檢查」確實 100% 可自動化，於是被外推到整個測試。

現況



人人都有做
但都忽略

傳統測試人員　開發人員

盲點

探索、質疑需求、判斷「這樣對嗎」—這些才是測試的核心，工具沒碰，很多 RD 也沒碰。

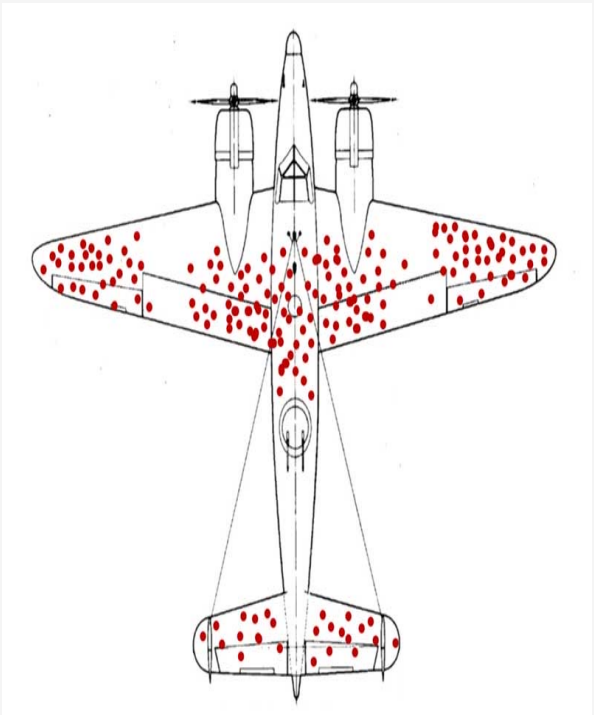
2.「大廠也沒有 QA」倖存者偏差

業界怎麼想

Google、Facebook、Netflix「沒有 QA 部門」的故事流傳極廣。

所以我們現在靠 AI 來做自動化測試就好 。

二次大戰



盲點

這些公司用 dogfooding、canary release、feature flag、線上監控取代 QA，靠大量真人回饋。

跟風者只學了裁 QA 那一半；Yahoo 的自動化還是前 QA 團隊建的，這段沒人學。

3. 工具商賣藥：測試會自己好

業界怎麼想

利用 UI 自動化，你不需要會寫程式，只要錄製操作過程，之後便可以重新播放測試。

因此 ，工讀生或是不會寫程式的 QA 都可以做測試自動化。

Self-healing

Testim、mabl：「AI 自動修復 locator，不用再維護測試」。

Playwright codegen：錄一次就能回放。

盲點

self-healing 修的是「找不到元素」，不是「功能做錯了」。這不是修 bug，是滅口。

就算 UI 可以修 ，但是 non-UI 的部分呢?

4. 以為 AI 寫測試不用成本

業界怎麼想

一句 prompt，300 個測試，覆蓋率 90%。

Token 費用比一個 QA 一天的薪水還少。

老闆：測試終於免費了。

盲點

測試的成本是：寫出來、看得懂、改得動、跑得完。

AI 只把第一項壓到零，後面三項一個都沒省，反而放大。

案例

Meta 自己的 TestGen-LLM，在 Instagram 上實測：

AI 生 4 個測試，丟掉 3 個，只有 1 個真的有用。

這還是過了編譯、執行、flaky、覆蓋率四道關卡之後的成績。

大綱

- AI 對軟體測試所帶來的影響
- 迷思一: AI 很擅長測試
- 迷思二: AI 測試比較快
- 迷思三: 都是綠燈所以沒有問題
- 迷思四: AI 可以把測試全部自動化
- **可能的解法**

解法總覽

迷思	解法
一、AI 很擅長測試	A. 實例化需求先行
二、AI 測試比較快	B. AI 產出過同一道門檻
三、都是綠燈所以沒有問題	C. 驗證測試會不會紅
四、測試可以全部自動化	D. 探索性測試

迷思一：AI 很擅長測試

解法方向：**AI 不知道什麼是「對」，所以「對」要在它動手前先定好**

前面看到的問題

- 廣而淺：同一等價類生五個變體
- 斷言弱：全綠但什麼都沒證明
- 測現狀不測意圖：把 bug 鎖進測試
- 規格模糊時，默默選一種解讀

對應解法

A

實例化需求先行

先用具體範例定義「對」，再讓 AI 生成

解法 A｜實例化需求先行

對照迷思一：AI 很擅長測試 → AI 不知道什麼是「對」，只有人知道

1

開工前：Example Mapping

- PO · 開發 · QA 一起

2

範例先定案

- 具體輸入 → 預期輸出

3

AI 照範例生成

- 程式和測試都從範例

4

每個斷言可回溯

- 斷言從範例來

這裡講的是「先有範例」這件事，不是工具。

迷思二：AI 測試比較快

解法方向：**不是「有做、做完」，而是「做對、做好」**

前面看到的問題

- 以前沒做的事，現在「有做」就像進步
- 為了做更多更快 ，犧牲做的品質
- 感覺快的人，不是承擔成本的人
- 廠商賣的是一鍵生成，不是一鍵驗證

對應解法

B

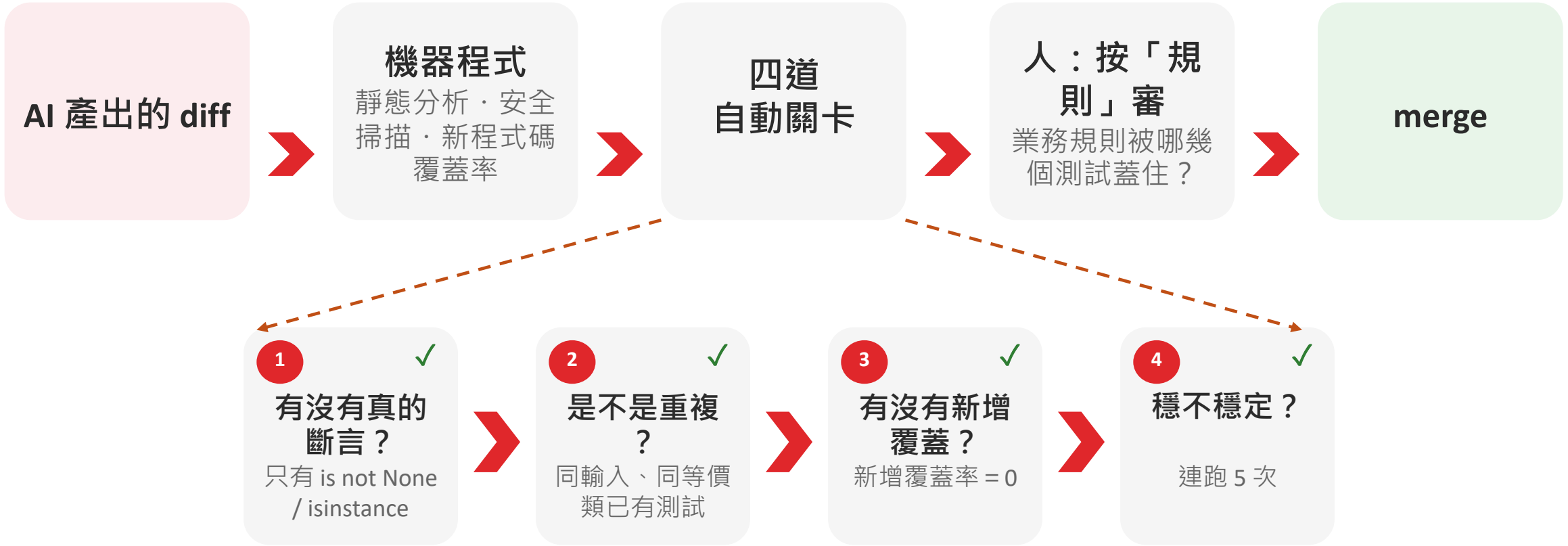
AI 產出過同一道門檻

機器先擋掉四種垃圾，人只審剩下的

解法 B｜AI 產出過同一道門檻

對照迷思二：AI 測試比較快 → 快在少假警報；AI 寫的和人寫的過一樣的門檻

不因為「AI 寫的」放寬，也不因為「資深 RD 寫的」放寬



Meta：4 個 AI 測試，只留不到 1 個

TestGen-LLM（FSE 2024）：LLM 生的測試先過四關，全過才給人看（Instagram Reels / Stories 實測）

LLM 生出的測試	① 編得過？	② 跑得過？	③ 連跑 5 次都綠？	④ 覆蓋率有增加？
100%	75%	–	57%	25%
	builds	passes	殺 flaky = 你的第 ④ 關	新增覆蓋 = 0 就丟 = 你的第 ③ 關

25% x 73% = 18.25%

再經過人檢視

過了門檻的建議，工程師接受進 production

迷思三：都是綠燈所以沒有問題

解法方向：綠燈只證明「測試和程式一致」，還可能是 AI 改到一致；
要另外驗證「測試會不會紅」

前面看到的問題

- 綠燈只證明「有跑過」
- 涵蓋率 100%：邊界從沒被測過
- AI 把測試改到一致：reward hacking (獎勵作弊)

對應解法

C **驗證測試會不會紅**
沒紅過的測試不算測試

解法 C1｜驗證測試會不會紅

對照迷思三：都是綠燈所以沒有問題 → **全綠不代表測試有在驗證**

手動版：零成本，今天就能做（人寫的、AI 寫的都一樣）

1. **新測試**
   - merge 之前
2. **故意改壞實作**
   - 把 > 改 >=、
   - 0.9 改 1.0、
   - 刪一行
3. **紅了嗎？**
   - 紅 → 還原 → 綠 → merge
4. **還是綠**
   - 從頭到尾沒在驗證，退回

解法 C2｜驗證測試會不會紅

對照迷思三：都是綠燈所以沒有問題 → **全綠不代表測試有在驗證**

agent 交的 PR 多一道；先靠人看，再讓工具抓漏

1
先看 diff

2
AI 有改原本的測試嗎?

- 小心 AI 沒修好 Bug，反而把測試改成綠燈。

3
對 diff 跑 mutation testing

- PIT / mutmut 往改動處注入 bug
- 測試抓不到就是漏洞；
- 只跑 diff，幾分鐘

4
把測試有效性納入 CI 門檻

- Mutation Score 低於門檻，不Merge。
- 每次 PR 只檢查修改範圍。
- 定期全量執行

Uncle Bob：我也這樣做

Prompt 寫十頁沒用（lost in the middle）；他把規則搬進「跑完才放行」的工具

1

Specifier 規格官

- **人寫的需求** → Gherkin 驗收條件 + QA 程序

2

Coder 寫手

- 寫單元測試 + 實作
- 只做一件事，context 小、規則才記得住

3

Cleaner 清潔工

- 跑 **CRAP score**（coverage × cyclomatic complexity）
- 收拾 coder 留下的爛攤子

4

Hardener 鐵面判官

- **對原始碼跑 mutation testing**
- 活下來的 mutant 一律補測試殺掉
- AI 很難作弊繞過的約束

5

QA agent 驗收員

- QA 文件 → 可執行腳本
- 另有 dependency 檢查員守模組邊界

迷思四：測試可以全部自動化

解法方向：**自動化做 checking，人做 testing**

前面看到的問題

- 把「檢查」誤認為「測試」
- 大廠光環與倖存者偏差
- 工具商賣自動修復 bug
- 以為用 AI 寫測試不用成本

對應解法

D **探索性測試**
固定時間、有 charter 的人工探索

解法 D｜探索性測試

對照迷思四：測試可以全部自動化 → **自動化做 checking，人做 testing**

每個 sprint

1. **排 2–3 個 session**
2. **Mob testing**
   聚焦 AI 最容易出錯處：邊界、狀態、整合
3. **debrief**
   發現什麼、下次測哪、記進 charter

發版前

1. **一到兩小時 bug bash**
   開發、PO、QA 一起探索
2. **抓蟲比賽**
   輸人不輸陣
3. **回饋到 charter**
   原來客戶是這樣使用產品

Program testing can be used to show the presence of bugs, but never to show their absence!

程式測試只能證明錯誤的存在，卻永遠無法證明錯誤不存在 ！

~ Edsger W. Dijkstra



Challenge First. Collaborate Next.”
先質疑，再共舞。



Q&A

