Hello World Dev Conference 2026

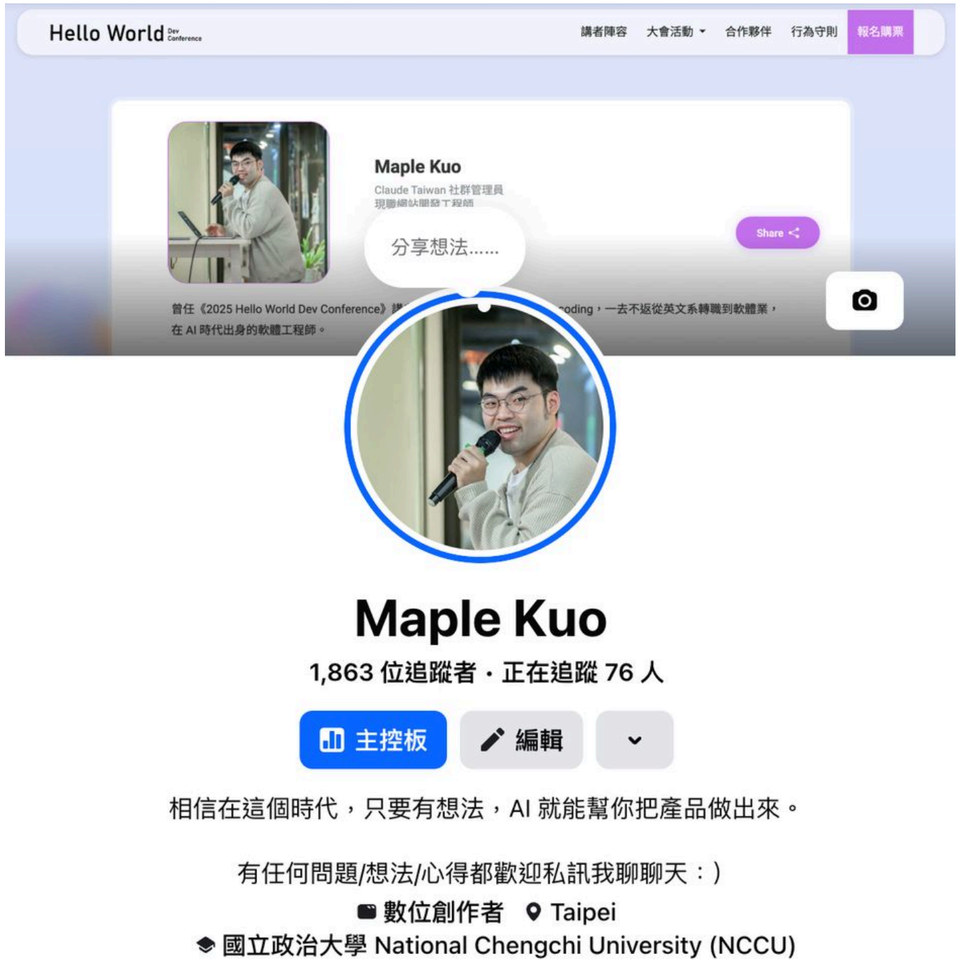
初級工程師
如何 AI 賦能的翻寫一個
跟自己年紀差不多大的框架



Maple Kuo

2026/10/15

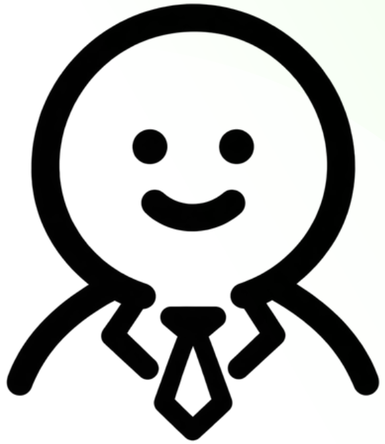
經歷



- 國立政治大學 英國語文學系
- 2025 Hello World Dev Conference 講者
- 衛生所「來學 AI 吧！數位轉型的第一步」講者
- Generative AI 年會小聚 講者
- 台灣人工智慧學校小聚 講者
- Claude Taiwan 社團管理員
- Zeabur “Ship It” Hackathon 第三名
- 一年從 0 開始和 AI 自學轉職（現職）軟體工程師

棘手的題目

老闆



我打算給你一個棘手的題目
看你能用 AI 做到什麼程度！

差不多大的框架是多大

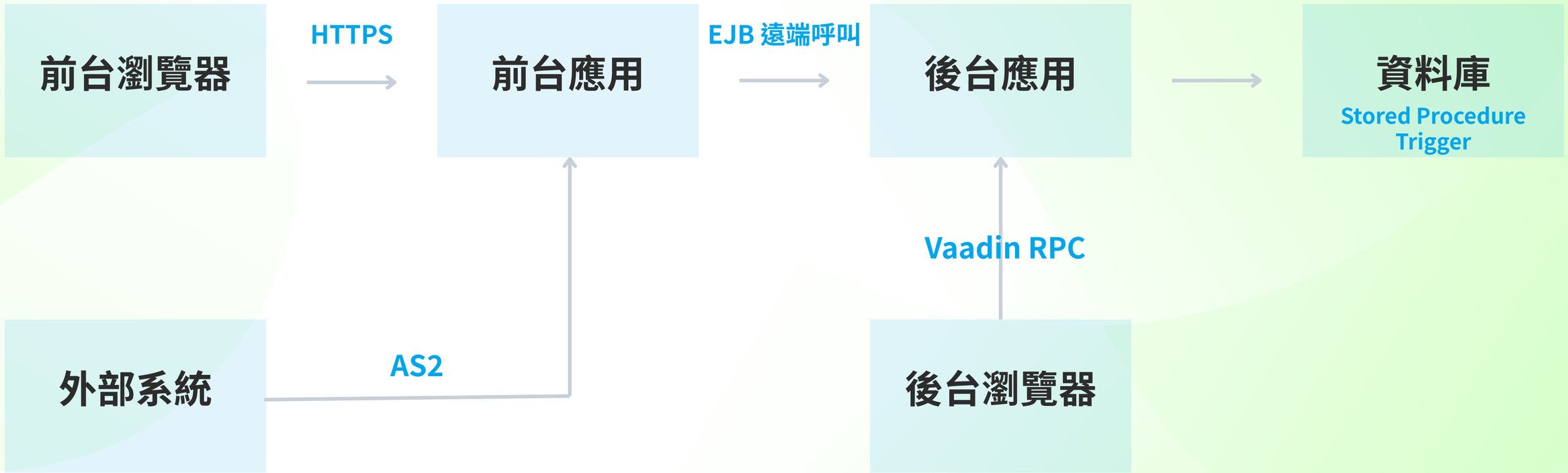


Java EE (Vaadin / EJB)

差不多大的框架是多大


新增
編輯
匯入
匯出
系統管理
刪除
搜尋列
搜尋：
確定
條件式查詢:
1 - 500 of 532 [0 selected]
系統代碼
分組依據
None
選取模式
All Pages
代碼
名稱
建立時間
狀態
操作
01
05
05
93
90
0239
9006
1
2
下一頁
AI 調整後示意畫面

差不多大的框架是多大


前台瀏覽器
HTTPS
前台應用
EJB 遠端呼叫
後台應用
資料庫
Stored Procedure
Trigger
Vaadin RPC
外部系統
AS2
後台瀏覽器

棘手的題目



我

英文系畢業

棘手的題目



我

英文系畢業

第一份工作

棘手的題目



我

英文系畢業

第一份工作

全跟 AI 學

棘手的題目



我



Java EE (Vaadin / EJB)

棘手的題目

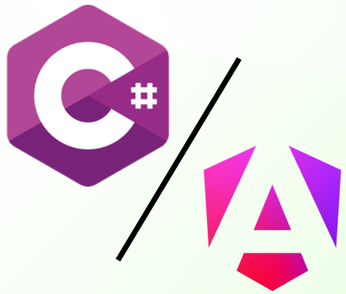


我



Java EE (Vaadin / EJB)





ASP.NET Core

Angular

核心困難



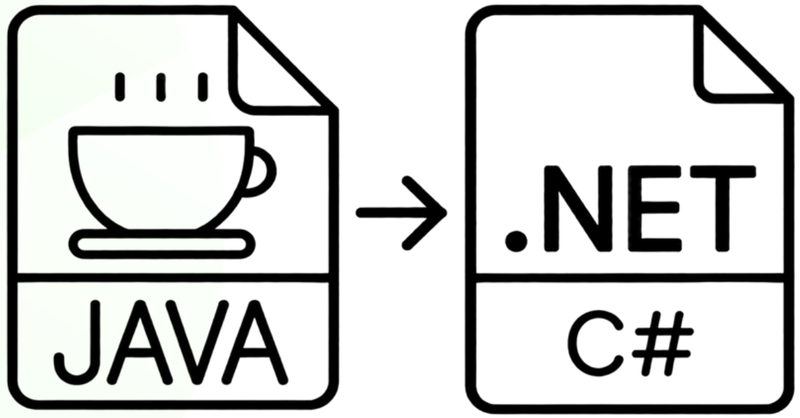
技術老舊



部分源碼

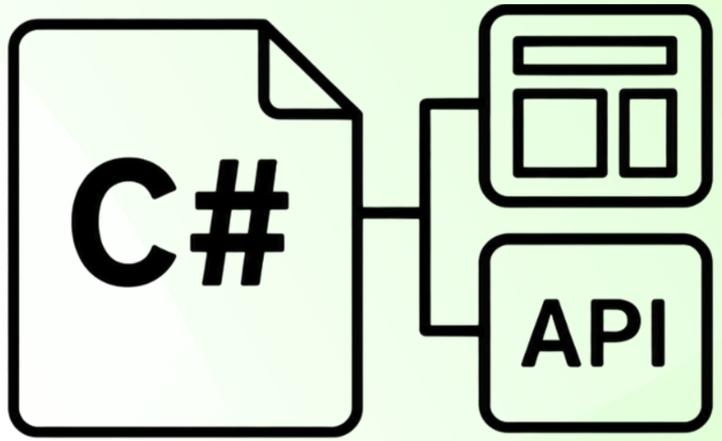
分兩階段進行

第一階段



完全還原行為

第二階段（進行中）



重新設計與擴充

第一階段

- 把黑盒系統變成有原始碼，並初步理解系統

第一階段

- 把黑盒系統變成有原始碼，並初步理解系統
- 不動資料庫，唯一要求是寫出來的行為要與原系統一致

第一階段



我要怎麼知道行為與原系統一致?

第一階段



跟寫新程式碼一樣寫測試！

同樣的操作 DB 狀態要一致

我要怎麼知道行為與原系統一致?

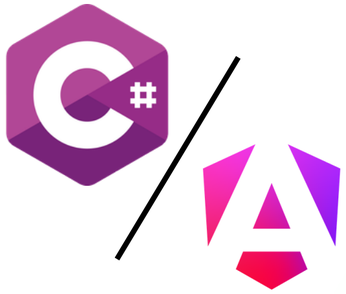
搭建測試環境

舊系統





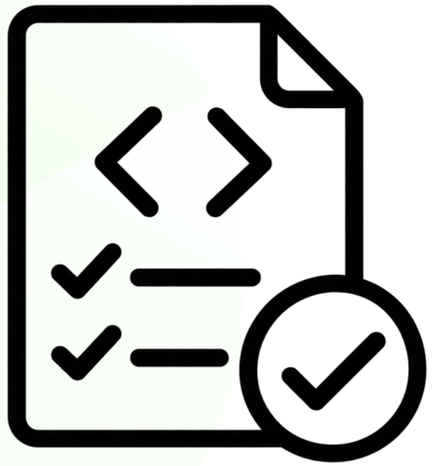
新系統



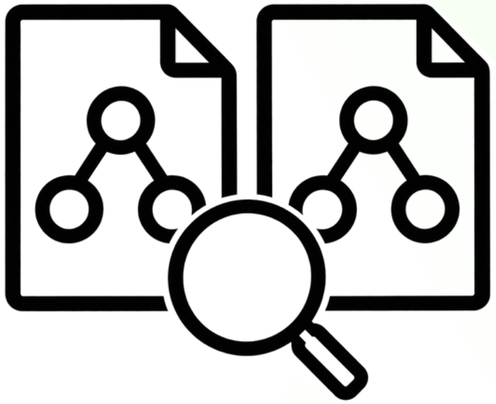


各自獨立的測試 DB

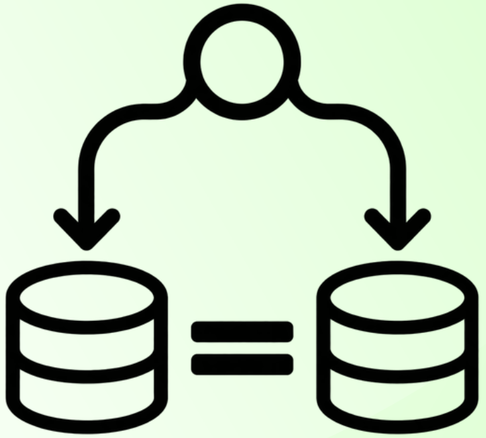
三層驗證



對規格

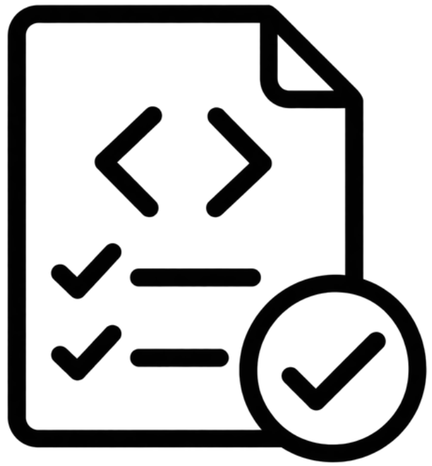


對邏輯

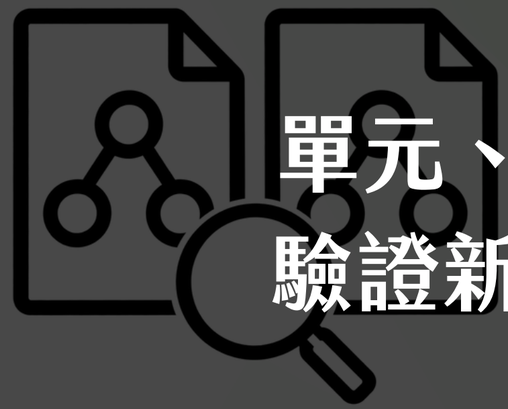


對結果

三層驗證

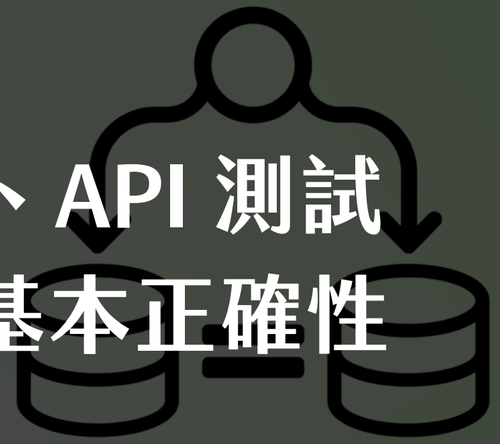


對規格



對邏輯

單元、整合、API 測試
驗證新系統基本正確性



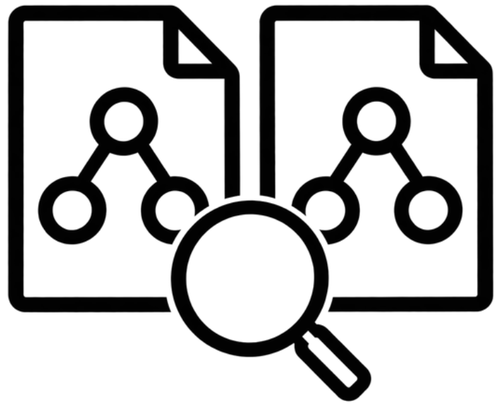
對結果

三層驗證

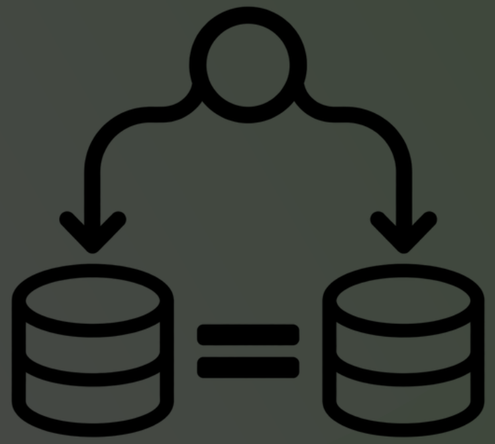
讓兩家 AI 讀取新舊程式碼
撰寫兩邊實作邏輯，對比差異



對規格



對邏輯



對結果

三層驗證



同一筆資料、同一個請求
送進新、舊系統
比回傳和資料表的變化

對規格

對邏輯



對結果

進行中的第二階段也有碰到一些困難...

- 重新設計時，AI 被第一階段的程式碼影響

進行中的第二階段也有碰到一些困難...

- 重新設計時，AI 被第一階段的程式碼影響
- 資料不乾淨，不知道舊系統是否還寫入（歷史遺留?）

如何 AI 賦能的克服困難

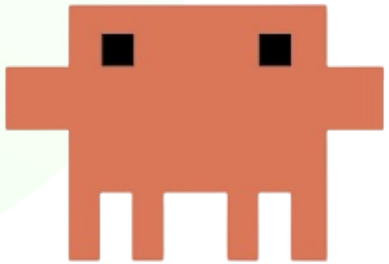


/pm



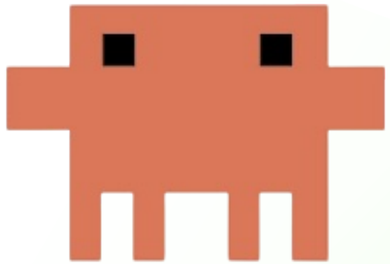
/arch

subagent



Implementer

subagent



Test-reviewer

subagent



Reviewer

如何 AI 賦能的克服困難



負責釐清需求，力求這個功能背後的「為什麼」，
並設定好 User Journey，驗收標準
產出 PM 文件作為後續架構設計和審查的參考

/arch

subagent

Implementer

subagent

Test-reviewer

subagent

Reviewer

如何 AI 賦能的克服困難

提問策略

收到功能描述後，依下列維度評估哪些尚不清楚，主動提問補齊：

維度	要挖掘的內容
Problem	解決什麼問題？為何現在要做？不做會怎樣？
Users	誰會用？不同角色各自的痛點和動機？
Behavior	每個角色的主要使用流程？從哪裡進來？替代路徑？
States	功能裡的東西會經歷哪些狀態？誰能觸發轉移？每個狀態下能做什麼、不能做什麼？ 兩個人同時對同一筆操作時以誰為準？有畫面的話，沒資料、出錯、沒權限時使用者看到什麼？
Design Intent	關鍵設計決策背後的「為什麼」？有什麼不能妥協的原則？
Boundaries	什麼不該發生？錯誤情境？Out of scope？
Integration	外部系統、auth、現有功能依賴？規模（使用者數、資料量、尖峰流量）？預期壽命（一次性活動還是長期維護）？

規模和預期壽命是使用者知道、codebase 看不出來的事實，/arch 要靠它們決定架構該多輕或多重，所以在這裡問。

如何 AI 賦能的克服困難



/pm



/arch

負責開發藍圖，探索盤點當前慣例、可重用的內容
並根據 PM 文件設計測試寫法及內容，產出架構文件

subagent



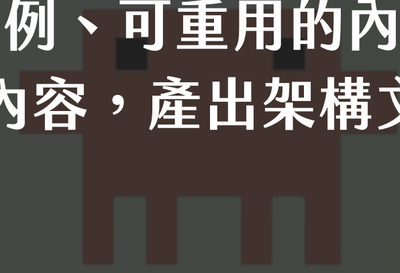
Implementer

subagent



Test-reviewer

subagent



Reviewer

如何 AI 賦能的克服困難

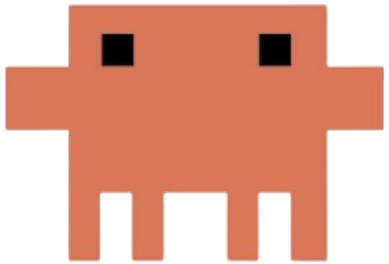


/pm



/arch

subagent



Implementer

根據架構文件進行實作，帶有開發哲學的 system prompt

- 「先讓改動變容易，再做那個容易的改動。」— Kent Beck
- 「重複遠比錯誤的抽象便宜。」— Sandi Metz
- 跟隨既有的開發風格
- 命名要讓讀者不打開實作就知道這段程式在做什麼
- 少寫一點 code，能重用就不新增，能移除就不保留

Test-reviewer Reviewer

如何 AI 賦能的克服困難

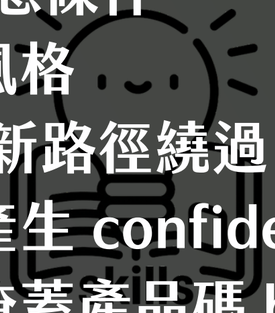
Hello World Dev Conference

根據 PM 文件進行檢查，帶有工程判準的 system prompt

- 是否有過度嵌套/競態條件
- 是否忽略專案既有風格
- invariant 有沒有被新路徑繞過
- 審查測試是否真的產生 confidence，抓到 regression
- 有沒有偷改測試來掩蓋產品碼 bug



/pm



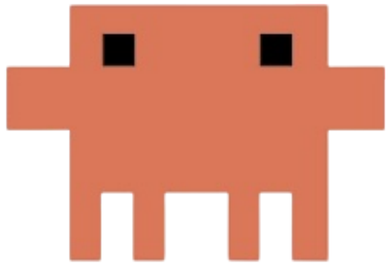
/arch

subagent



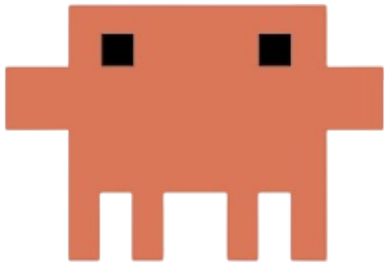
Implementer

subagent



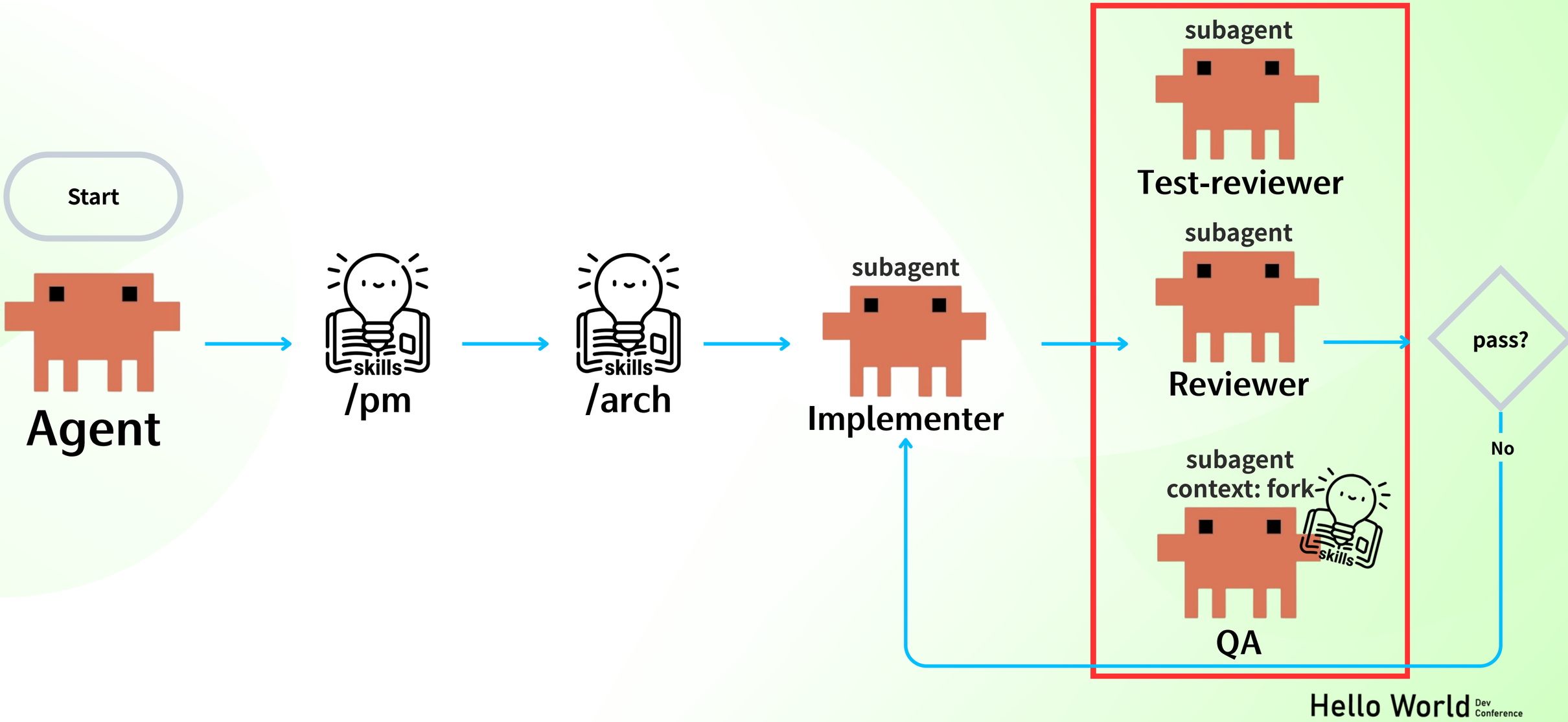
Test-reviewer

subagent



Reviewer

如何 AI 賦能的克服困難



修但幾勒



一個功能就要兩份文件

一支 API 不就要有好多份

修但幾勒



真的每份都有認真看嗎?

一個功能就要兩份文件

一支 API 不就要有好多份

讓自己的關注點縮小

需要確認的東西讓 AI 提問，幫你分段

讓自己的關注點縮小

提問節奏

- 已能從描述或既有文件推斷的維度跳過，只問不清楚的
- 用 `AskUserQuestion` 提問（一次最多 4 題），問完等回答再問下一輪
- 問具體情境：「當員工沒有種子追蹤權限但上傳照片時，Bot 應該怎麼回應？」而非「錯誤情境有哪些？」

組成 workflow 的元件



Skills



Subagents



Hooks

組成 workflow 的元件



Skills

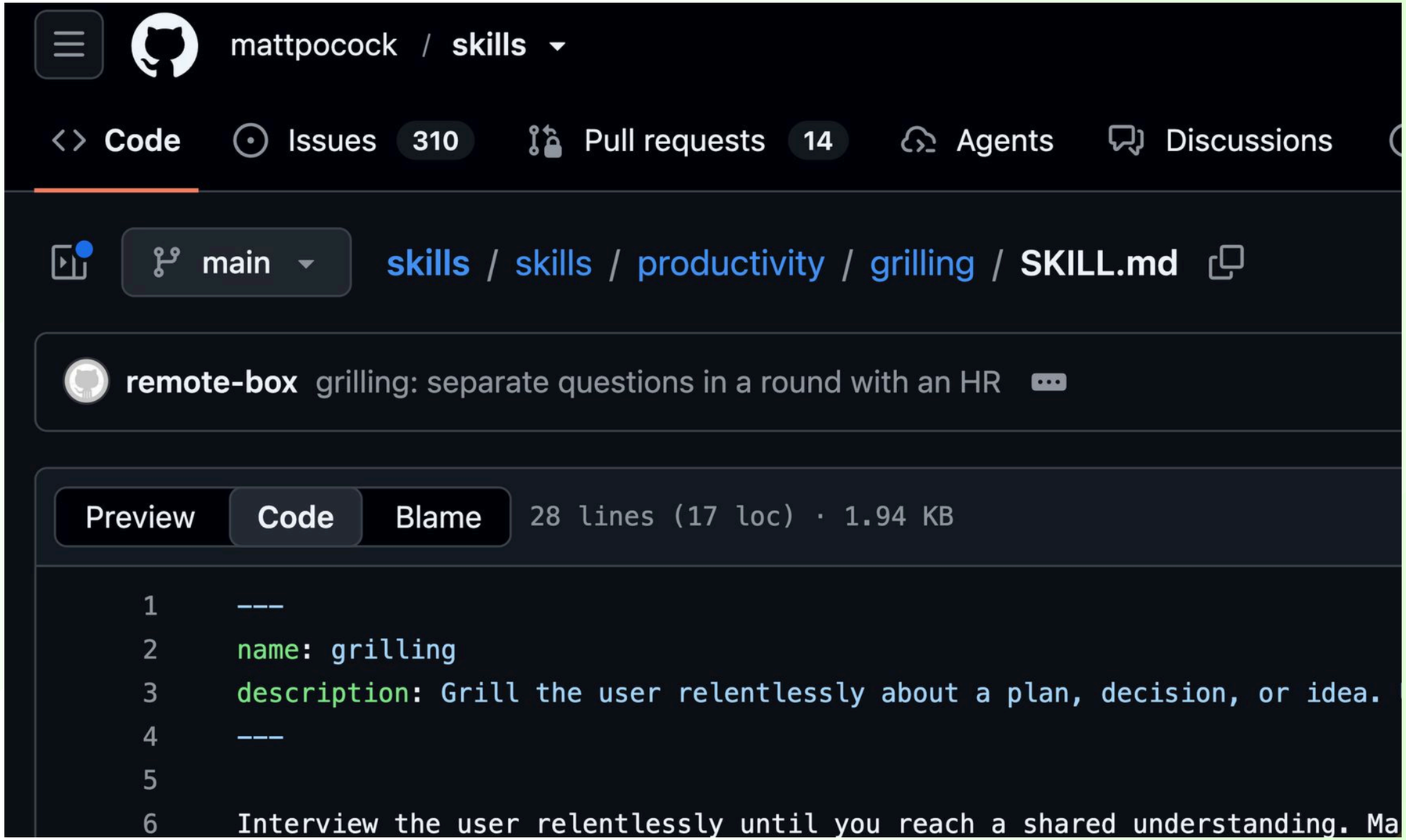


Subagents



Hooks

先認識 skills 的機制



先認識 skills 的機制

mattpocock / skills

Code Issues 310 Pull requests 14 Agents Discussions

main skills / skills / productivity / grilling / SKILL.md

remote-box grilling: separate questions in a round with an HR

Preview Code Blame 28 lines (17 loc) · 1.94 KB

```
1  ---
2  name: grilling
3  description: Grill the user relentlessly about a plan, decision, or idea.
4  ---
5
6  Interview the user relentlessly until you reach a shared understanding. Ma
```

但 skills 其實可以做更多事

```yaml
---
name: deep-code-review
description: 深度審查 PR 變更，產出結構化報告

disable-model-invocation: true
argument-hint: "[PR number]"
allowed-tools: Bash(gh *), Bash(git diff *), Read, Grep
paths: "src/**/*.ts"
user-invocable: true

context: fork
agent: Explore
model: sonnet
effort: high

hooks:
  PreToolUse:
    - matcher: Bash
      command: echo "Tool used: $TOOL_NAME"
shell: bash
---
```

但 skills 其實可以做更多事

```yaml
---
name: deep-code-review
description: 深度審查 PR 變更，產出結構化報告

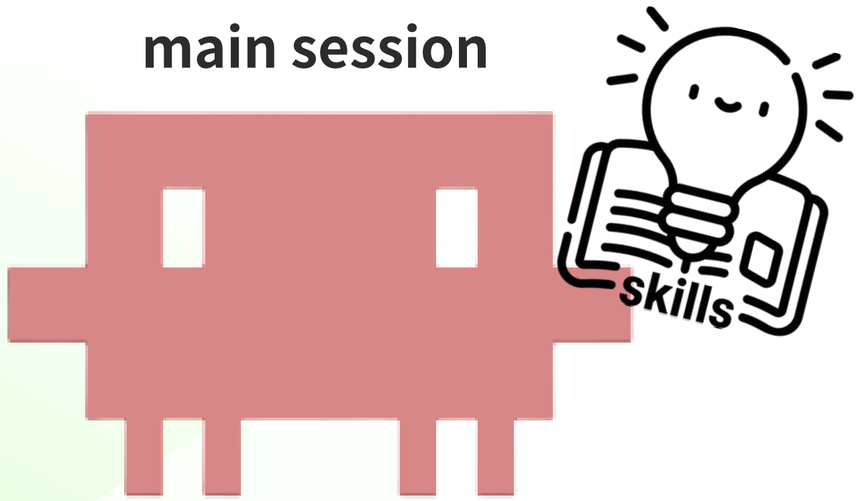
disable-model-invocation: true
argument-hint: "[PR number]"
allowed-tools: Bash(gh *), Bash(git diff *), Read, Grep
paths: "src/**/*.ts"
user-invocable: true

context: fork
agent: Explore
model: sonnet
effort: high

hooks:
  PreToolUse:
    - matcher: Bash
      command: echo "Tool used: $TOOL_NAME"
shell: bash
---
```

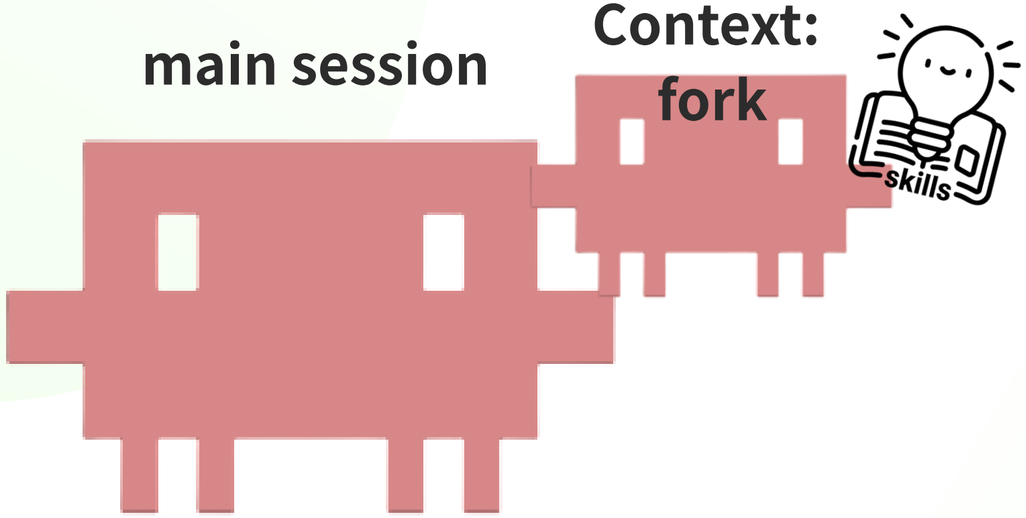
code-review: skills vs fork agent skills

main session

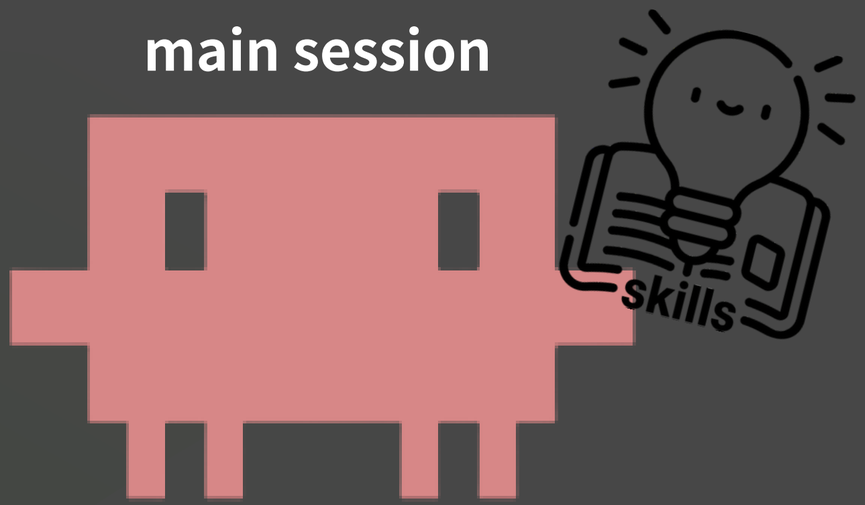


main session

Context: fork



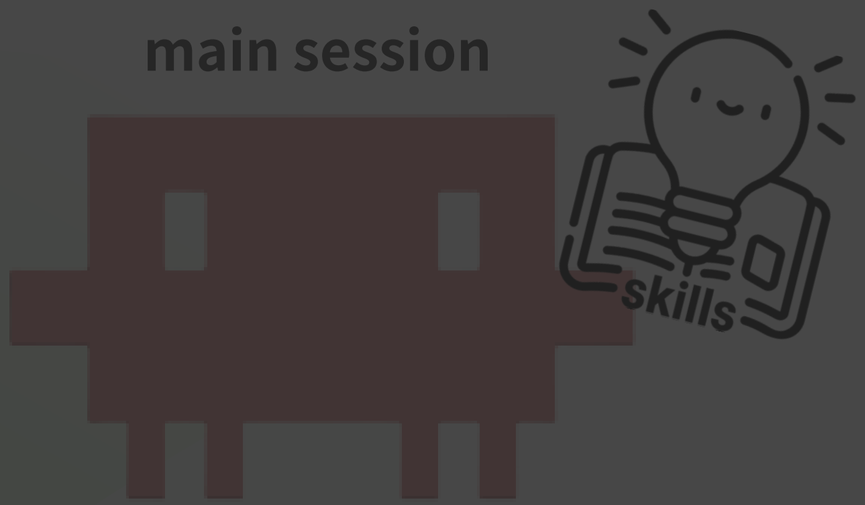
code-review: skills vs fork agent skills

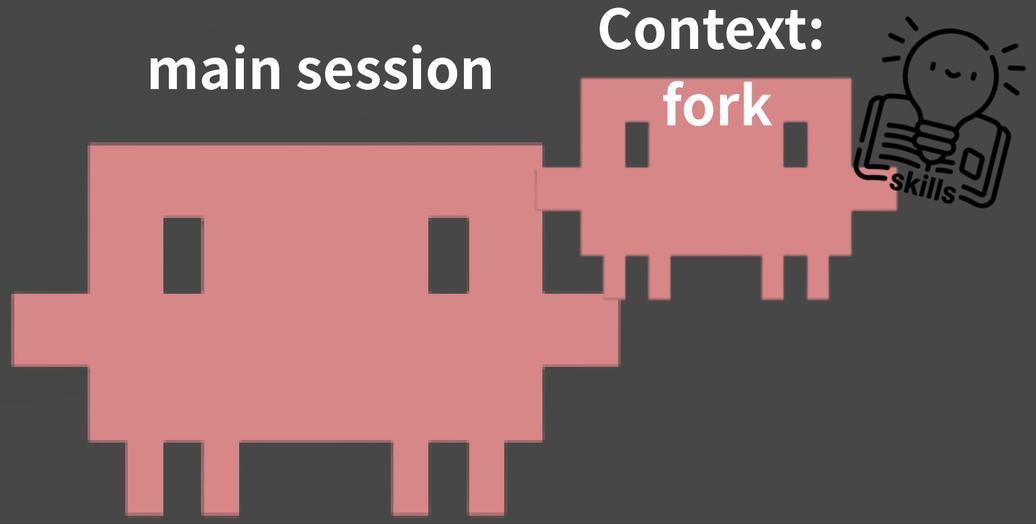


污染上下文



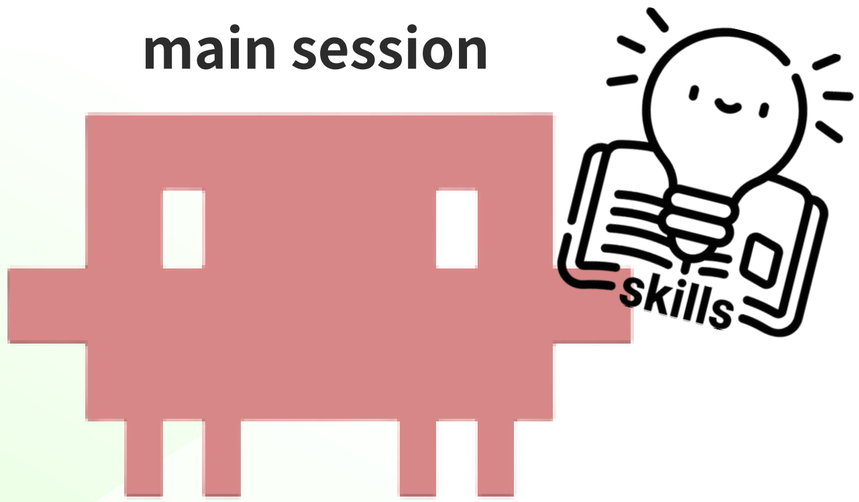
code-review: skills vs fork agent skills



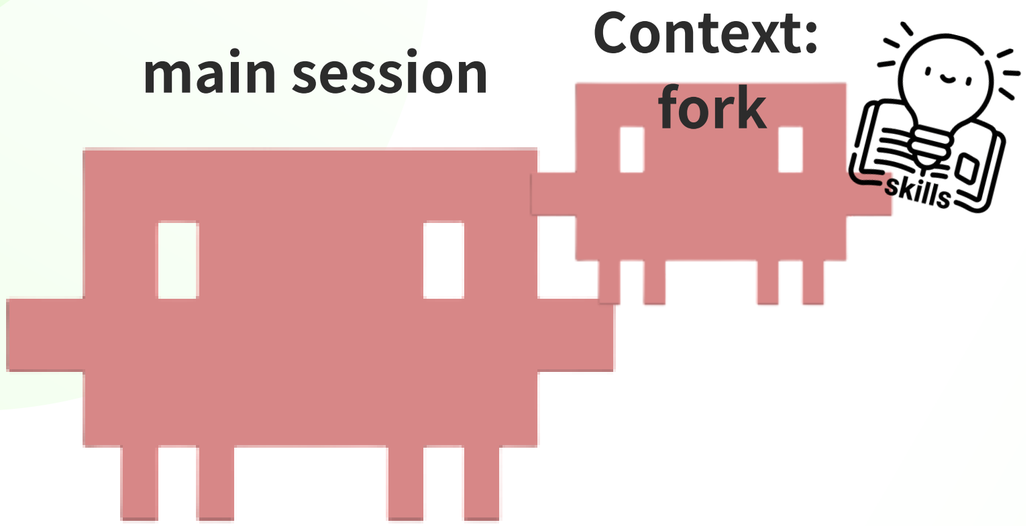


比較燒 token

code-review: skills vs fork agent skills



- 常用 prompt 的快捷鍵
- 能力擴充 e.g., /wt, /reddit-search
- 按需載入的背景知識擴充



- 通用性的重複任務
- 不在意過程只在意結果

組成 workflow 的元件



Skills

Subagents



Hooks

如何 AI 賦能的克服困難

Hooks = 當條件成立時自動發生的行為

如何 AI 賦能的克服困難

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "if": "Bash(rm *)",
            "command": "${CLAUDE_PROJECT_DIR}/.claude/hooks/block-rm.sh",
            "args": []
          }
        ]
      }
    ]
  }
}
```

如何 AI 賦能的克服困難

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "~/.claude/hooks/test-file-reminder.sh"
          }
        ]
      }
    ]
  }
}
```

如何 AI 賦能的克服困難



```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "bash .claude/hooks/test-file-remover.sh"
          }
        ]
      }
    ]
  }
}
```

撰寫測試時遵循規範

採用 BDD 命名、適度參數化

並為 Prod / QA 發現的錯誤補上防回歸註解

小結

- 對於古蹟翻寫，建立測試環境與確保資料庫是首要
- 善用 Hooks, Skills, MCP 等機制，來控制 AI 的輸出
- 讓 AI 幫你拆分關注的東西，一次一件事確認

SDLC



SDLC



SDLC 中被犧牲的受害者...

回滾到時再想

需求先做再確認

我這邊可以跑啊

審查先跳過

需求不會再改了

先上線再說

這個很簡單

架構之後再想

先這樣寫，以後再來重構

資安之後再補

不會有空值

改一行而已，應該不會壞

快取清掉就好

這段沒人會改

這次先不寫測試也沒關係

文件之後補

升版應該相容

重開就好了

不可能有人這樣操作

監控等出事再加

測過一次就夠了

備份應該有吧

明天一定修得完

交接口頭講就好

SDLC 中被犧牲的受害者...

回滾到時再想

審查先跳過

需求先做再確認

我這邊可以跑啊

需求不會再改了

先上線再說

這個很簡單

架構之後再想

資安之後再補

先這樣寫，以後再來重構

不會有空值

改一行而已，應該不會壞

快取清掉就好

這段沒人會改

這次先不寫測試也沒關係

文件之後補

升版應該相容

重開就好了

不可能有人這樣操作

監控等出事再加

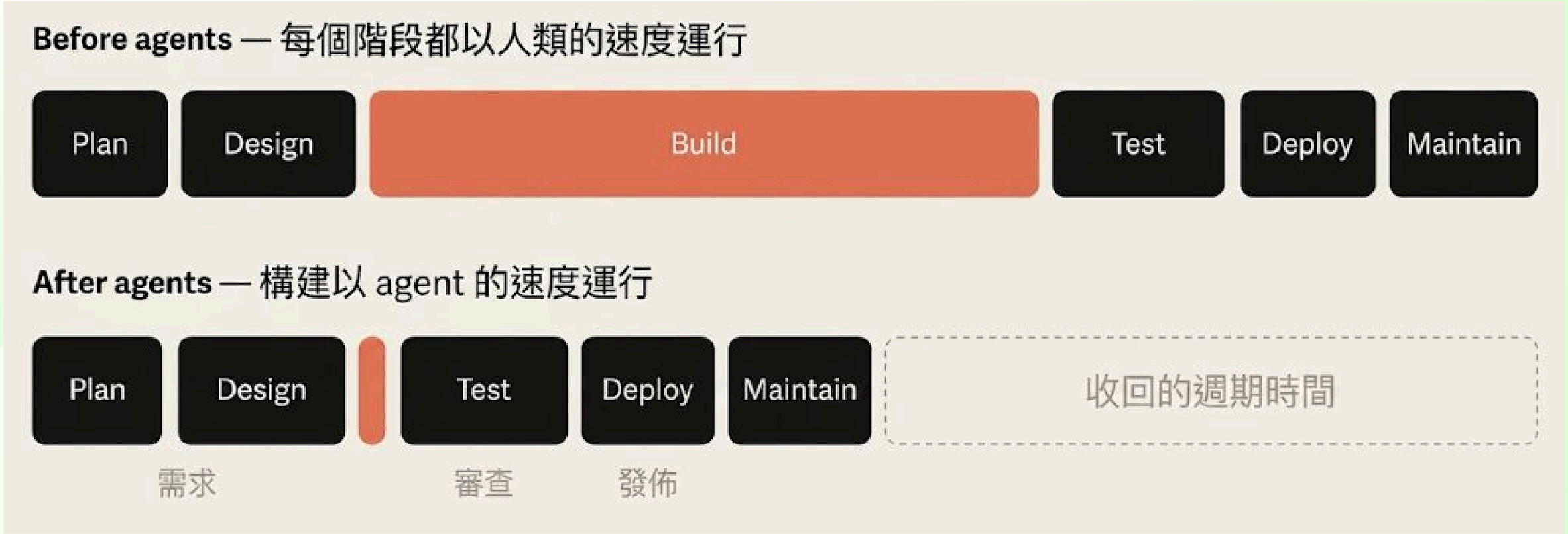
測過一次就夠了

明天一定修得完

備份應該有吧

交接口頭講就好

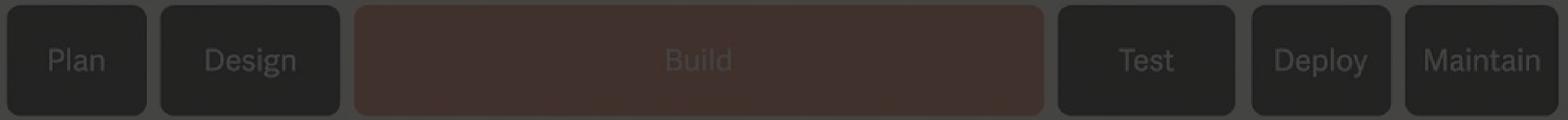
Agent 對 SDLC 的影響



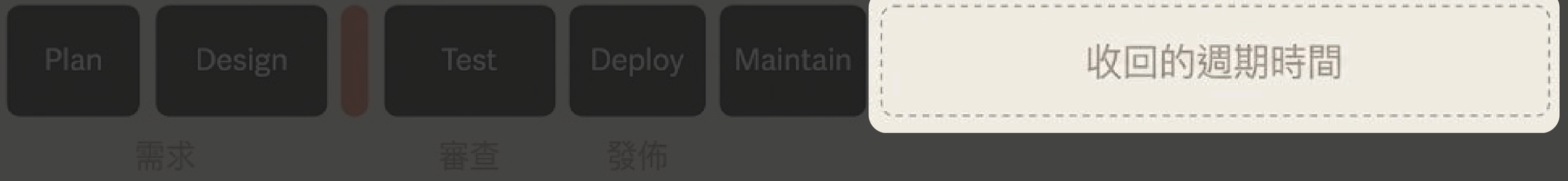
Source: “The AI-native SDLC playbook”

Agent 對 SDLC 的影響

Before agents — 每個階段都以人類的速度運行



After agents — 構建以 agent 的速度運行



Source: “The AI-native SDLC playbook”

SDLC



測試的初衷

確保軟體的行為品質、可靠性和穩定性

測試的初衷

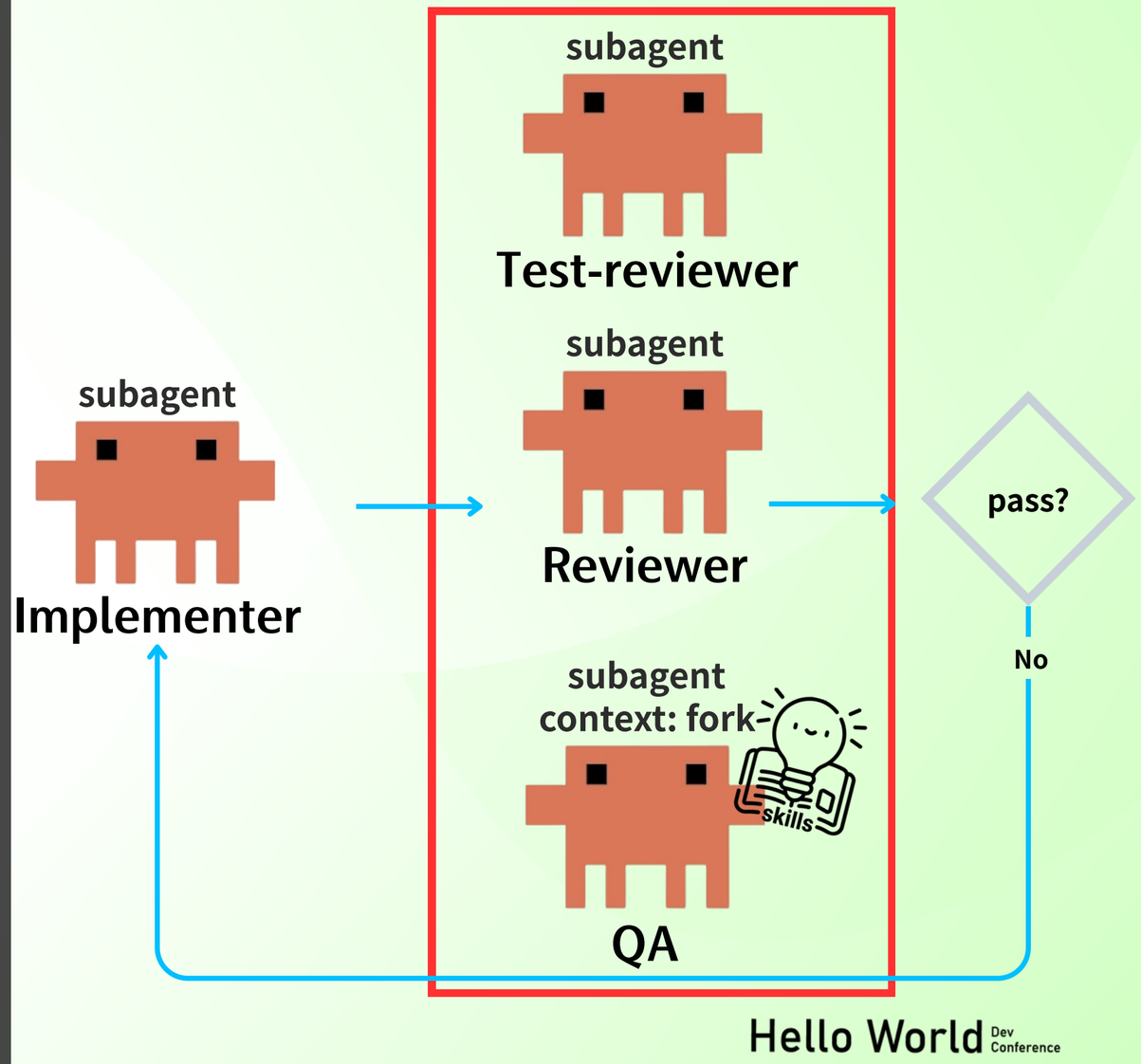
自動化測試最根本的核心在於
讓你能充滿信心地修改程式碼

——《The Practical Test Pyramid》

如何 AI 賦能的克服困難



可以在這個迴圈中納入重構



測試信心?



怎麼確認 AI 寫的測試真的有用?

需求是 SSOT



回歸需求規劃

提醒 AI 有這些方法（偏好）

等價分割 (Equivalence Partitioning)

同一類別內的輸入值，系統行為應該相同。
把輸入空間分成等價類，每類取一個代表值，就能用最少的 case 覆蓋最多情境。

適用時機：任何有輸入範圍或分類的欄位。

產出格式：

```
欄位：[欄位名]
├ 有效類 1：[範圍/條件] → 代表值：[值]
├ 有效類 2：[範圍/條件] → 代表值：[值]
├ 無效類 1：[範圍/條件] → 代表值：[值]
└ 無效類 2：[範圍/條件] → 代表值：[值]
```

範例 — 訂票日距乘車日天數：

```
欄位：訂票距乘車日
├ 無效類：>29 天（尚未開賣） → 代表值：35 天
├ 有效類：6~29 天（早鳥區間） → 代表值：15 天
├ 邊界類：5 天（截止邊界） → 代表值：5 天
├ 無效類：1~4 天（已截止） → 代表值：3 天
└ 無效類：當日 → 代表值：0 天
```

提醒 AI 有這些方法（偏好）

決策表 (Decision Table)

多個條件組合影響不同結果時，列出所有組合確保無遺漏。
n 個布林條件產生 2^n 種組合。條件超過 5 個時，先識別互斥或不可能的組合來縮減。

適用時機：權限 matrix、折扣規則、狀態轉換前置條件、多 flag 組合。

產出格式：

條件 A	條件 B	條件 C	→ 期望結果
T	T	T	[結果]
T	T	F	[結果]
...	...	...	...

提醒 AI 有這些方法（偏好）

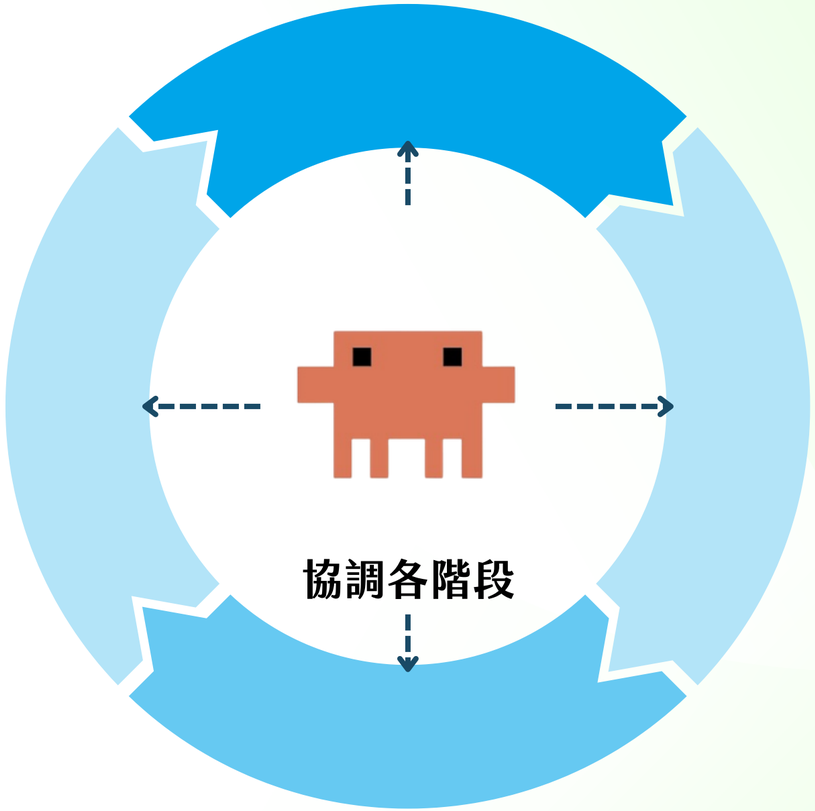
組合縮減：Pairwise

條件數超過 4 個時完整決策表指數爆炸。Pairwise 利用一個經驗法則縮減：
大多數 bug 由 1-2 個參數的交互觸發，3-way 以上極罕見。
保證每對參數的每種值組合至少出現一次，即可覆蓋絕大多數交互缺陷。

條件數	完整組合	Pairwise 約	策略
≤ 4	≤ 16	—	完整決策表，不縮減
5-7	32-128	8-12	先刪互斥組合，剩餘用 pairwise
> 7	> 128	12-20	直接用 pairwise

RD 自己 SDLC

需求規劃
/pm



審查驗收及重構
Reviewer
Test-reviewer
QA

系統設計
/arch

程式開發
Implementer

心法

假設 AI 什麼都知道，但他也什麼都不知道

總結

- 想像力就是超能力，描述目標與意圖，避免成為 AI 的限制器
- 善用 Hooks, Skills, MCP 等機制，來控制 AI 的輸出
- SDLC 最常被忽略的測試，是 AI 時代最不該被忽略的
- 透過撰寫測試，在開發後接著重構，維持程式品質
- AI 什麼（技術）都會，但他也什麼（需求、風格）都不會



Maple Kuo

AI 時代出身的軟體工程師

感謝你！

contact.maple.kuo@gmail.com