

.NET 測試雙引擎：打造極致開發工作流

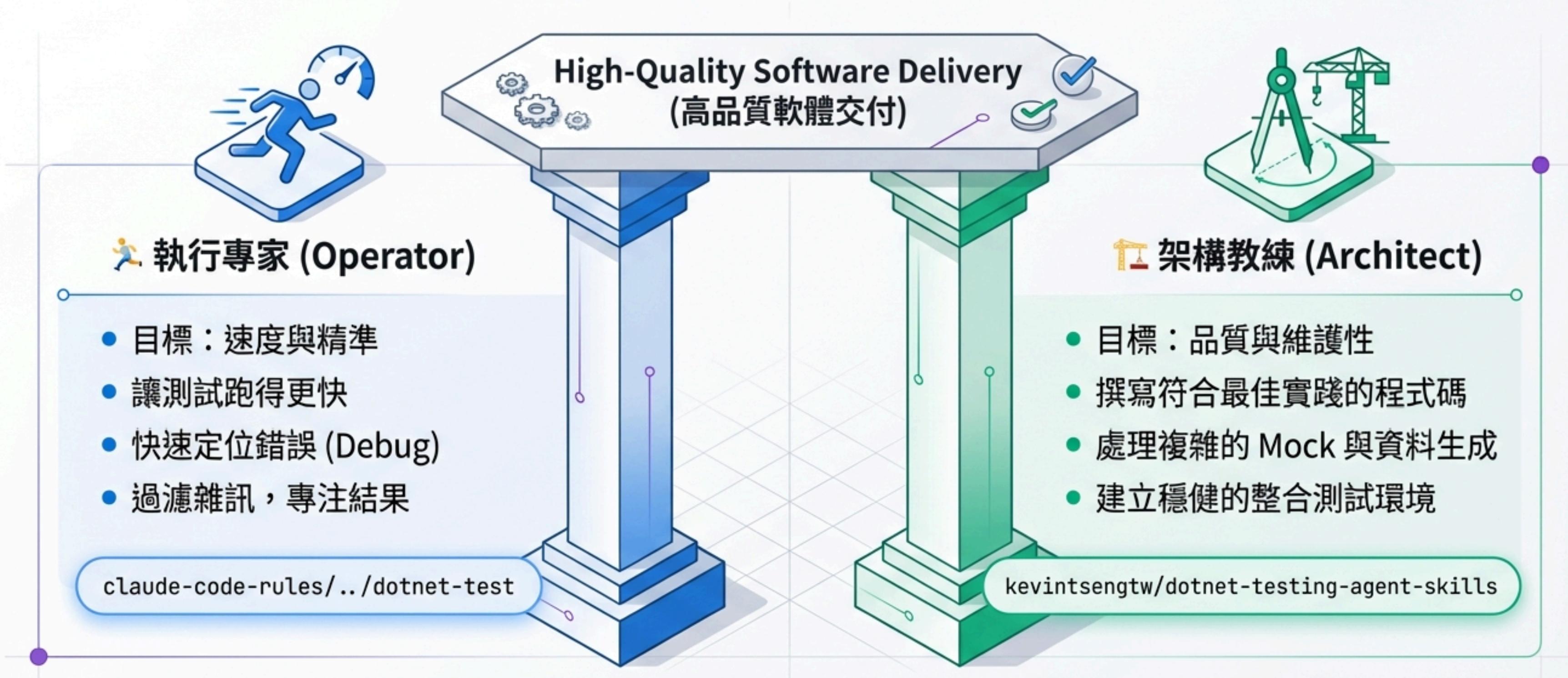
深入解析 CLI 執行專家 (dotnet-test) 與 測試開發教練 (kevintsengtw) 的完美協作



現代化測試的兩大挑戰：如何寫出高品質的測試？如何最有效率地執行它們？
本文將介紹兩個針對不同痛點的 AI Agent Skills，助您將 AI 助手升級為資深 SDET。

現代化測試的兩大支柱 (The Two Pillars)

一個完整的 AI 輔助測試策略，需要同時解決「如何寫」與「如何跑」的問題。



Part 1: CLI 指令專家

dotnet-test

您的 CLI 指令專家

專注於執行 (Execution)

這個 Skill 不教您寫程式，而是教您「如何像專家一樣執行測試」。

 **更流暢 (Workflow) :**

採用 Build-First 策略，大幅減少等待時間。

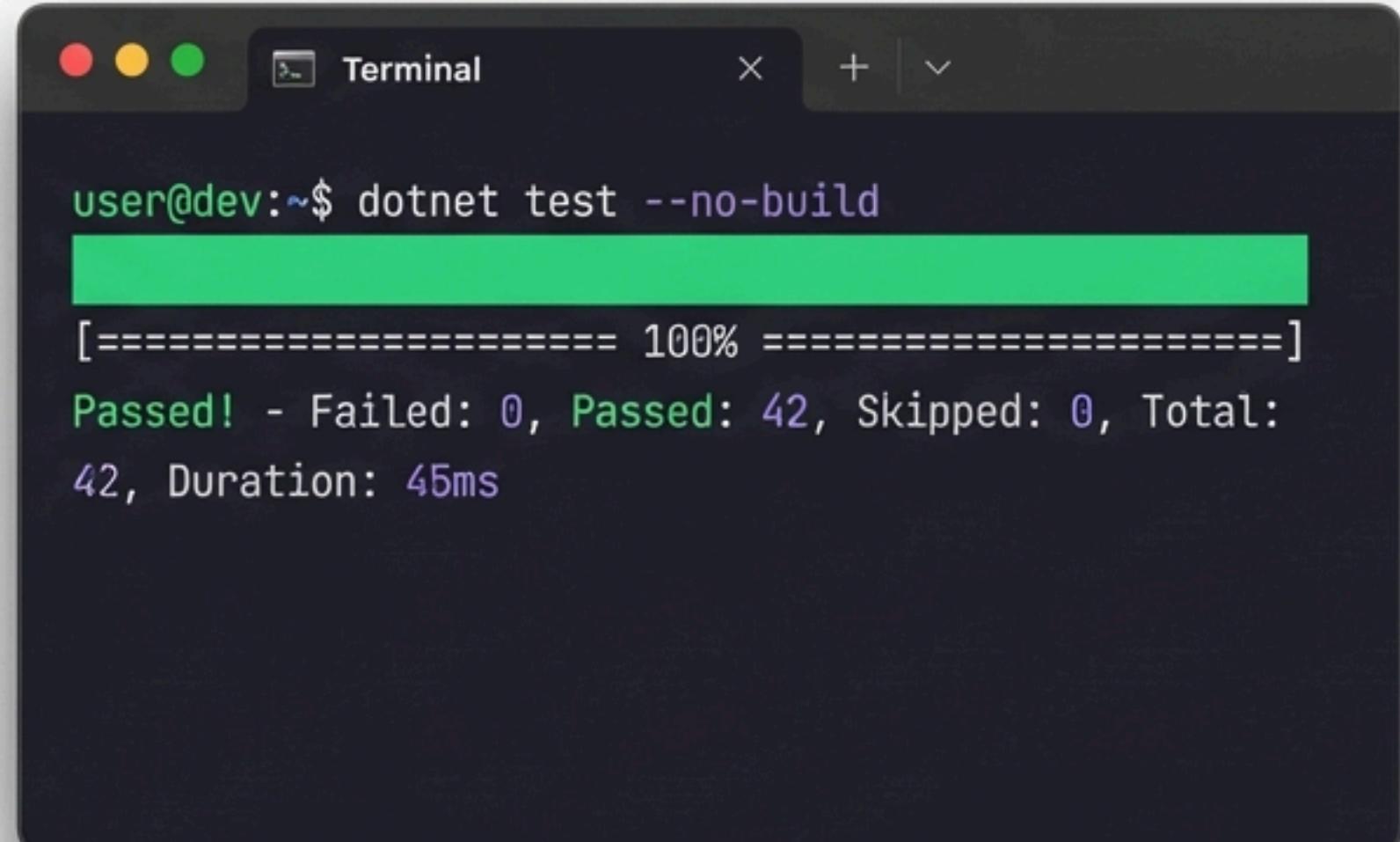
 **更精準 (Precision) :**

能夠精確過濾並鎖定特定的測試案例。

 **易除錯 (Debugging) :**

在需要時提供關鍵的 Log 資訊，自動過濾雜訊。

它將 dotnet test 指令的潛能發揮到極致。

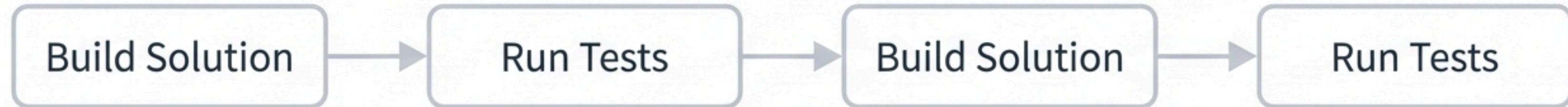


The screenshot shows a macOS Terminal window titled "Terminal". The command "dotnet test --no-build" is run, resulting in the following output:

```
user@dev:~$ dotnet test --no-build
[===== 100% =====]
Passed! - Failed: 0, Passed: 42, Skipped: 0, Total: 42, Duration: 45ms
```

核心策略：Build-First Workflow (建置優先工作流)

傳統方式 (Standard Way)



每次執行都重新建置，浪費時間 (Slow & Noisy)

專家方式 (Agent Way)

1. 快速建置 (Build Once)

先編譯整個解決方案，忽略警告，只看錯誤。

```
dotnet build -p:WarningLevel=0  
/clp:ErrorsOnly --verbosity minimal
```

Success



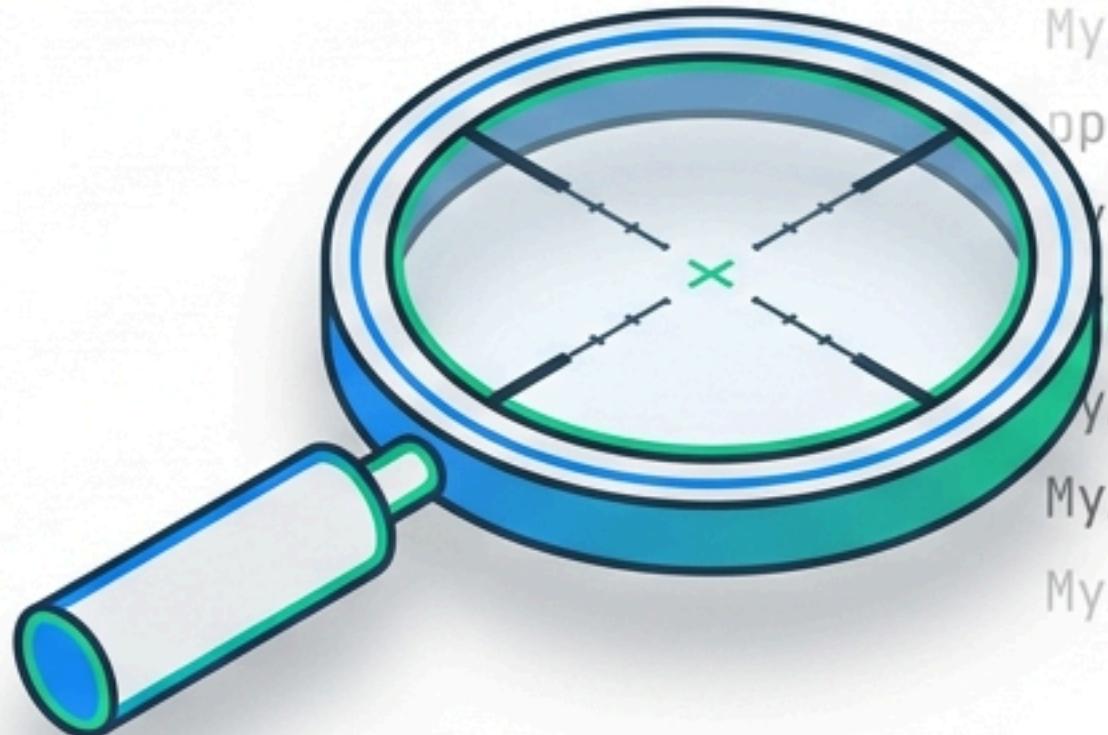
2. 執行測試 (Test Specific)

使用 --no-build 參數，直接針對特定專案執行，速度極快。

```
dotnet test path/to/project --no-build
```

精準打擊：過濾器的藝術 (The Art of Filtering)

不要浪費時間跑全套測試。Agent 懂得利用 FullyQualifiedName 進行最精確的鎖定。



MyApp.Tests.UserServiceTests.UpdateUser_Should_Fail
MyApp.Tests.ProductServiceTests.GetProduct_Should_Return
MyApp.Tests.OrderServiceTests.CalculateUser_Should_BeCorrect
MyApp.Tests.OrderServiceTests.CreateOrder_Should_Succeed
MyApp.Tests.OrderServiceTests.GetProduct_Should_Return
MyApp.Tests.UserServiceTests.Calculation_Should_BeCorrect
MyApp.Tests.OrderServiceTests.UpdateUser_Should_Fail

過濾情境 (Scenario)	建議指令參數 (Command Flag)	備註 (Note)
依方法名稱 (Method Name)	"FullyQualifiedName~MyTestMethod"	推薦，最穩定
依類別名稱 (Class Name)	"FullyQualifiedName~MyTestClass"	鎖定單一類別所有測試
依 Theory 參數 (Theory Params)	"DisplayName~paramValue"	針對 xUnit Theory，例如 "User=Admin"
排除慢速測試 (Exclude Slow)	"Category!=Slow"	排除特定 Category 標籤



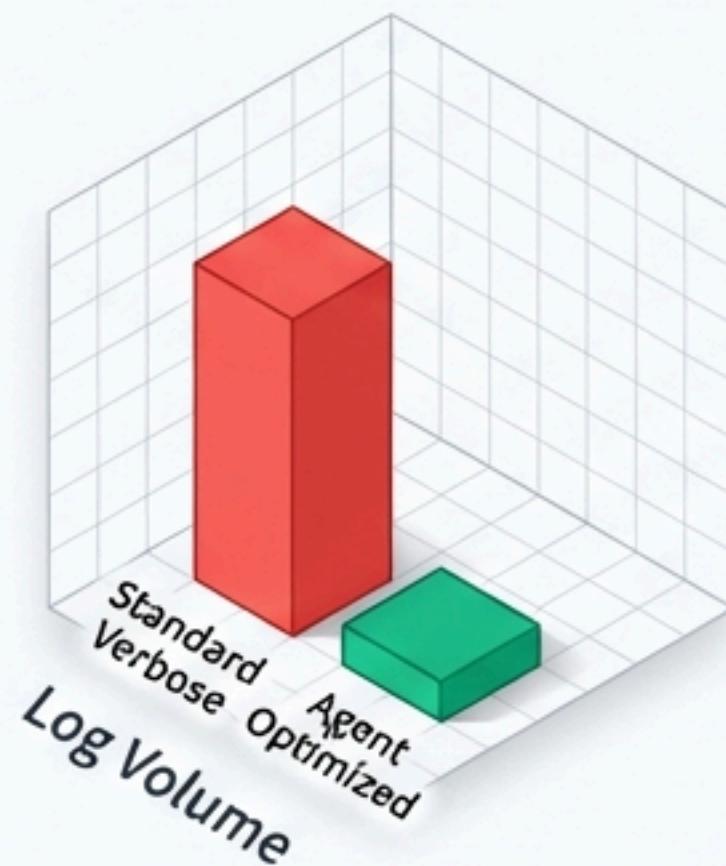
專家提示 (Pro Tip):

針對 xUnit 的 Theory 測試，透過 DisplayName 可以只執行特定參數的案例。這是除錯時的神器。

除錯與日誌：看見該看見的 (Signal vs. Noise)

雜訊困境 (The Noise Problem)

測試失敗時，預設輸出往往資訊不足；開啟詳細模式後，又會被成千上萬行 'Discovered' 訊息淹沒。



精準降噪 (The Solution)

Step 1.

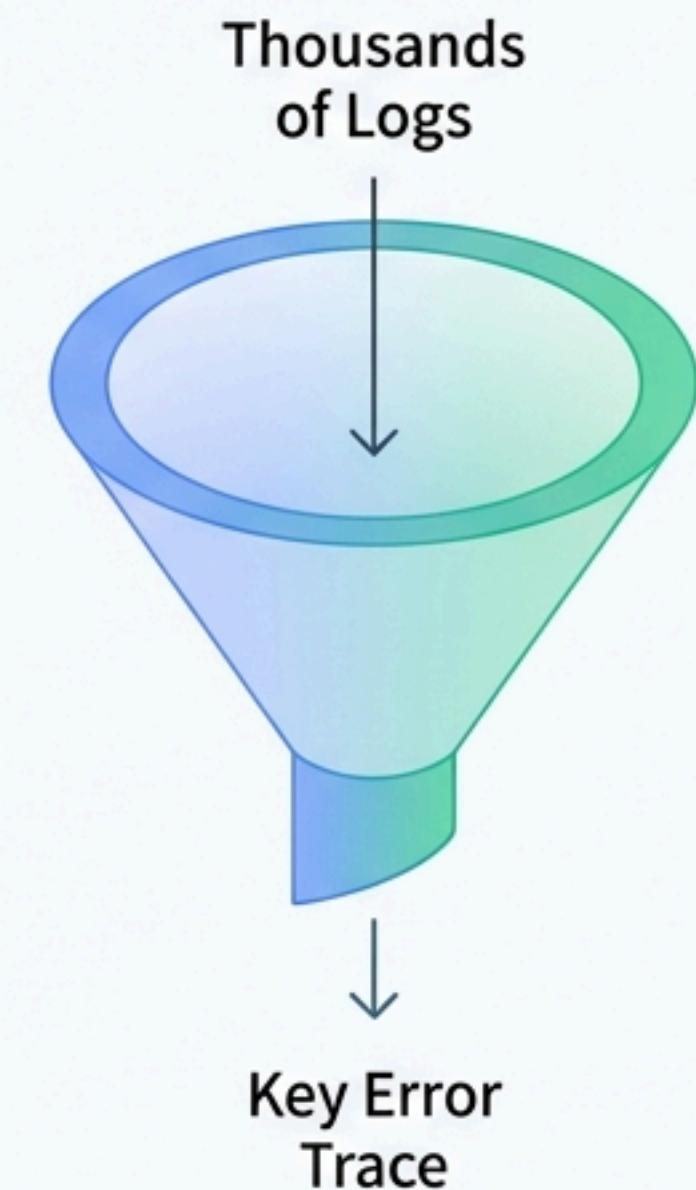
開啟 ITestOutputHelper: 為了看到 `output.WriteLine()` 的內容，Agent 會自動加入參數：

```
--logger "console;verbosity=detailed"
```

Step 2.

自動降噪: 使用 `grep` 過濾掉無用的執行訊息，只保留錯誤堆疊。

```
| grep -v "Discovered"
```



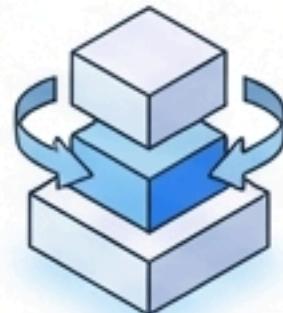
Part 2: 測試開發教練

kevintsengtw/dotnet-testing-agent-skills

您的測試開發教練 (Your Test Development Coach)



基於 2025 iThome 鐵人賽冠軍作品
「老派軟體工程師的測試修練」
這是一套完整的**測試知識庫**，不只幫
你寫 Code，還教你最佳實踐。



標準化 (Standardization)

生成符合 3A Pattern
(Arrange, Act, Assert)
的結構。



工具箱 (Toolbox)

內建 27 個精煉技能，
從單元測試到整合測
試。

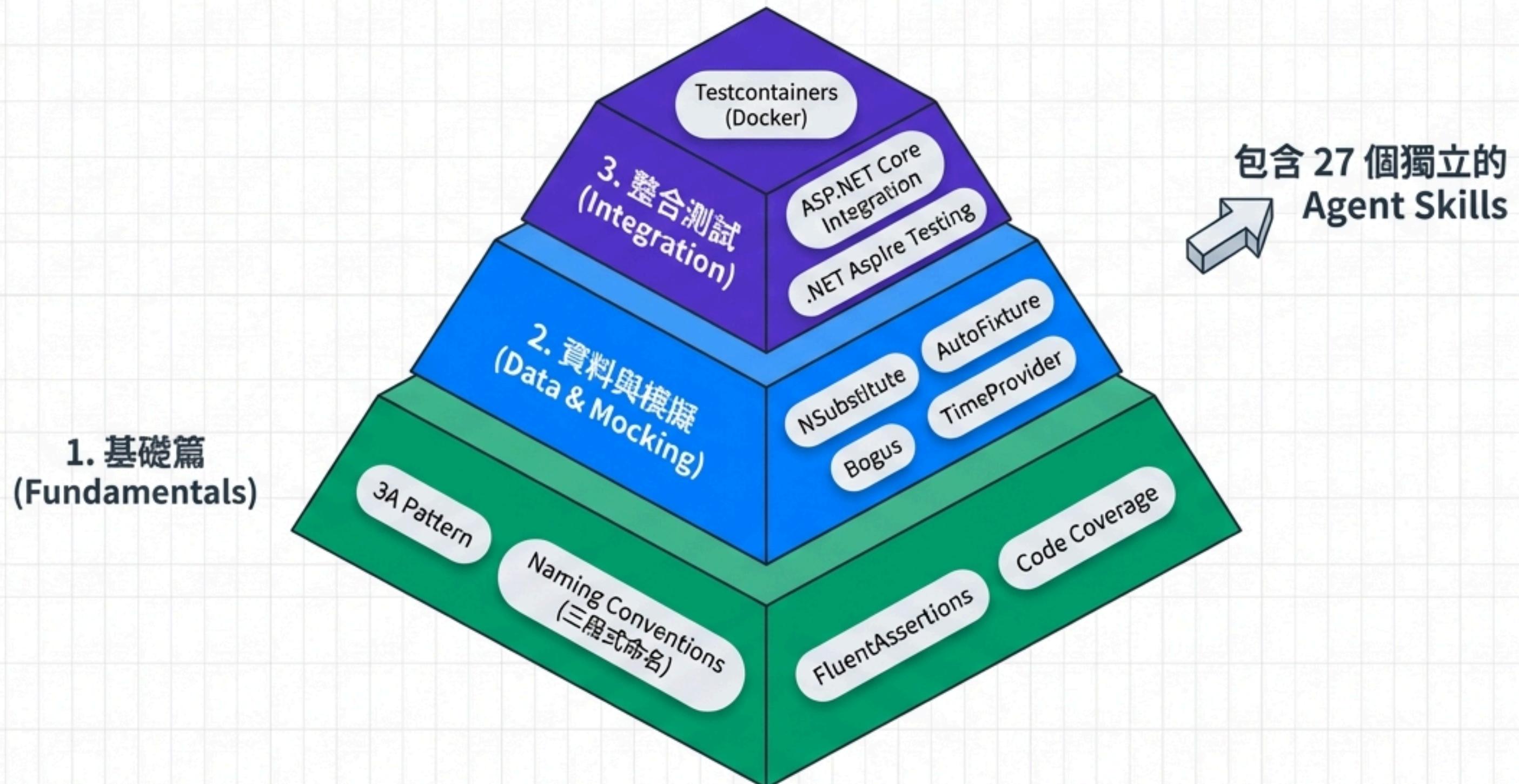


在地化 (Localization)

提供繁體中文的說明
與命名建議。

完整的技能樹 (The Skill Tree)

從最基礎的單元測試到現代化的容器整合測試，無所不包。



超越語法：堅持最佳實踐 (Best Practices)

Before (Messy & Fragile)

```
[Fact]
public void Test1() {
    var s = new Service();
    var result = s.DoWork(null);
    → Assert.Equal(true, result); // 斷言不清晰 ✗
}
```

After (Agent Generated)

```
[Fact]
public void CreateOrder_WithValidInput_ShouldReturnSuccess() {
    // Arrange (3A Pattern)
    var fixture = new Fixture();
    var order = fixture.Create<Order>();
    var sut = new OrderService();

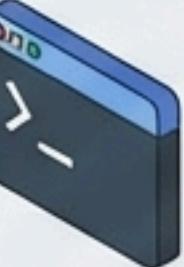
    // Act
    var result = sut.Create(order);

    // Assert (FluentAssertions)
    result.Should().BeTrue();
}
```

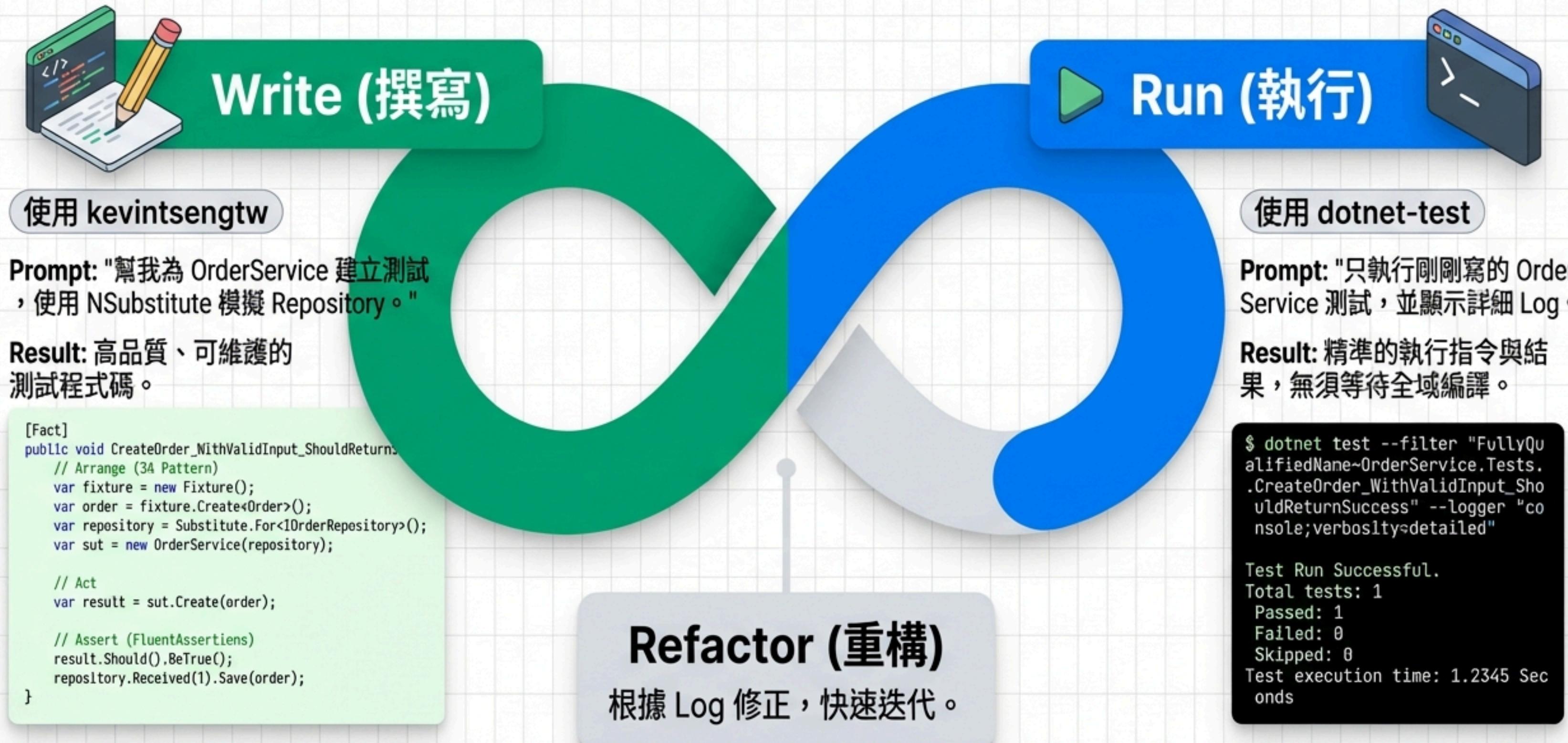
關鍵實踐 (Key Practices) :

- 🔑 命名慣例：嚴格遵守「三段式命名法」。
- 🔑 斷言風格：使用 FluentAssertions 提升可讀性。
- 🔑 資料生成：使用 AutoFixture 取代寫死的測試資料。

兩者的差異 (The Differences)

特性 (Feature)	dotnet-test (CLI 專家) 	kevintsengtw (測試教練) 
主要角色	執行者 (Runner / Operator)	創作者 (Creator / Architect)
解決痛點	測試跑太慢、Log 看不懂、指令複雜	不知道怎麼寫、Mock 很難搞、架構混亂
輸出產物	終端機指令、測試報告、除錯訊息 	C# 程式碼檔案、專案結構、設定檔 
互動方式	"\$ 幫我跑失敗的測試" 	"\$ 幫我寫這個 Service 的單元測試" 
核心技術	CLI, Bash, MSBuild	Roslyn, xUnit, NSubstitute

互補效應：完美的開發內圈 (The Perfect Inner Loop)



實戰情境演示：資料庫整合測試



1. 建立測試 (The Coach)

.NET Purple

User:

我有需要一個整合測試，使用 Testcontainers 啟動 SQL Server。

Action:

Agent 載入 testcontainers-database 技能，生成包含 Docker 設定的 C# 程式碼。

```
var container = new PostgreSQLBuilder()
    .WithDatabase("db")
    .WithUsername("user")
    .WithPassword("password")
    .Build();
await container.StartAsync();
```



2. 執行除錯 (The Runner)

Electric Blue

User:

測試在 CI 上失敗了，幫我用詳細模式在本地跑一次，我想看 SQL Log。

Action:

Agent 執行 dotnet test --no-build --logger "console;verbosity=detailed"

```
$ dotnet test --no-build --logger "console;verbosity=detailed"
Test Run Successful.
Total tests: 1
Passed: 0
Failed: 1
Skipped: 0
Test execution time: 1.2345 Sec
```



3. 問題解決 (Success)

Emerald Green

User:

(Agent 過濾雜訊，找出連線字串錯誤。)

Action:

測試通過。

```
Test Run Successful.
Total tests: 1
Passed: 1
Failed: 0
Skipped: 0
Test execution time: 1.2345 Sec
```

總結：為何您需要這兩個 Skill？



單獨使用...

Noto Sans TC

- 只用 CLI 專家：跑得很快，但可能寫出難以維護的爛測試。
- 只用 測試教練：寫得很漂亮，但每次執行都要等很久，且除錯困難。



只用測試教練

JetBrains Mono

- 只用 CLI 專家：跑得很快，但可能寫出難以維護的爛測試。
- 只用 測試教練：寫得很漂亮，但每次執行都要等很久，且除錯困難。



同時使用 (Combined)

同時安裝並啟用這兩個 Skill，讓 AI 成為您的資深測試架構師 兼 DevOps 專家。

Write Better. Run Faster.

資源連結 (Resources)

立即安裝，升級您的開發體驗。

CLI 指令專家

GitHub Repo: NikiforovAll/clause-code-rules

Path: /plugins/handbook-dotnet/skills/dotnet-test/



專注於 dotnet test 指令優化。

測試開發教練

GitHub Repo: kevintsengtw/dotnet-testing-agent-skills

2025 iThome 鐵人賽冠軍作品 2025 Champion



27 個單元/整合測試技能，繁體中文友善。

讓 AI Agent 協助您掌握 .NET 測試的每一個環節。