

章節	學習成效承諾	具體內容 v1.0	進度	內容上線時程	單元教學影片	圖文教學實戰	實作練習題	Github 專案 & Prompt 實戰教材	共學與線上 FAQ	【加碼工具！即將推出！】網頁 AI 互動學習模組
Ch1-規格驅動開發的前提	理解課程架構與規格光譜	理解整門課程的架構與學習路徑。掌握規格光譜的核心概念(從 Vision/PRD → Markdown → DSL Feature → ISA Feature → Test Code 的完整翻譯鏈)。理解為何需要整合軟工方法論(TDD, BDD, DDD)來達成 60% →100% 自動化開發的漸進式路徑。	100%	已完成	v	v		v		
Ch1-規格驅動開發的前提	認識真正的規格驅動	理解真正的規格驅動定義,不是「把規格寫清楚」而是「建立完整的翻譯鏈」。從 Problem Domain 精準翻譯到 Solution Domain, 市面工具(Cursor, Spec-Kit, Kiro)為何做不到真正的規格驅動? 因為它們依賴 LLM 去「猜」缺失的資訊, 而 LLM 是非決定性的。真正的規格驅動需要事件驅動型(Event Storming, Gherkin, ISA)確保資訊完整, 用測試驗證翻譯的正確性(Auto-Feedback Loop)。	100%	已完成	v					
Ch1-規格驅動開發的前提	識別現有工具極限	識別現有 AI 輔助開發工具的瓶頸。理解為何卡在 60% 自動化。問題核心在於:(1) 沒有結構化的規格模型。依賴自然語言的模糊描述(2) 缺乏精準的翻譯機制。LLM 需要「猜測」缺失資訊(3) 沒有 Auto-Feedback Loop 來驗證正確性。這些限制使得 AI 無法持續、穩定地完成開發任務。需要人工介入修正。	100%	已完成	v	v		v		
Ch1-規格驅動開發的前提	破除規格清楚的迷思	理解為何「把規格寫清楚」是錯誤的。真正的重點是面對問題。軟體開發的歷史演進: 1960-1970 年代, 機器語言太複雜, 沒有「抽象」與「協作」。軟體 = 解決問題的一組描述(不是程式碼)。包含對老闆的描述(KPI)。對 PM 的描述(PRD)。對 QA 的描述(測試案例)。對工程師的描述(程式碼)。抽象讓我們能夠協作, 但抽象會損失資訊, 因此需要「翻譯」來補齊。	100%	已完成	v			v		
Ch1-規格驅動開發的前提	掌握 Discovery → Formulation 流程	掌握用模型思維釐清規格的實戰技巧。Discovery(探索)階段: 透過 Event Storming 等技術發現問題領域的核心概念。Formulation(公式化)階段: 將探索結果轉化為結構化的規格(Gherkin Feature File)。這個流程確保從業務需求到技術實作的精準對齊。避免「規格不清」導致的反覆溝通與返工。	100%	已完成		v		v		
Ch2-規格光譜	理解抽象與翻譯的雙核心概念	深入理解軟體工程的兩大支柱:(1) 抽象(Abstraction): 沒有抽象就沒法協作。機器語言太瑣碎, 企業無法運作。透過抽象, 老闆能看懂 KPI, PM 能看懂 PRD, QA 能看懂測試案例。工程師能看懂程式碼。每個角色都能用自己熟悉的語言參與軟體開發。(2) 翻譯(Translation): 有了抽象後, 需要翻譯來做到完全自動化開發。每一層抽象都會損失資訊(如「支援紅色」沒說紅色可否疊加、是否有期限)。要讓軟體能做和實作, 必須把資訊「翻譯」回來。規格驅動開發的本質: 建立一條完整的翻譯鏈。從 Problem Domain 精準翻譯到 Solution Domain, 不遺漏任何資訊。	100%	已完成	v	v		v	2025/11/4	
Ch2-規格光譜	認識領域驅動設計(DDD)的核心概念	理解 DDD 的兩大領域區分:(1) Problem Domain(問題領域): 關注「要解決什麼問題」。參與者包含老闆、業務、PM、QA、使用者。特性是結果導向、抽象程度高、歧義性高。(2) Solution Domain(解決領域): 關注「如何解決問題」。參與者包含架構師、工程師、DevOps、機器。特性是過程導向、抽象程度低、歧義性低。理解兩個領域之間的模型如何垂直追蹤(從 Event Storming → Gherkin → ISA → Code)和水平追蹤(同一層級的不同面向, 如前端 ISA 與後端 ISA)。確保 Problem Space 與 Solution Space 的精準對齊。	100%	已完成		v		v	2025/11/4	
Ch2-規格光譜	理解事件風暴(Event Storming)模型所呈現的系統本質樣貌	掌握 Event Storming 如何呈現系統的運作本質。系統的行為本質: Command(指令)作為 Input, Event(事件)作為 Output, Command Handling(指令處理)與 Event Handling(事件處理)作為系統行為。透過 Event Storming Workshop, 團隊協作識別出: Domain Event(領域事件)、Command(指令事件的命名)、Aggregate(聚合名、處理指令的實體)、Policy(政策、事件驅動的自動化反應)、Read Model(查詢模型)。這種模型化方式讓業務邏輯視覺化、可追溯。並且能直接映射到 Gherkin 的 Given-When-Then 結構。	100%	已完成		v		v	2025/11/4	
Ch2-規格光譜	認識規格光譜五層結構及其映射關係	理解規格光譜的完整結構與每一層的映射關係: Vision/PRD(遠抽象, 給老闆/PM 看) → Markdown 規格(結構化的業務描述) → Gherkin DSL(Domain-Specific Language, 業務導向的可執行規格, 使用自然語言) → ISA (Instruction Set Architecture, 技術導向的可執行規格, 使用技術術 API/CLL) → 測試程式碼(可執行的驗證邏輯) → 程式碼(業務邏輯實作) → 機器碼(最具體, 給機器執行)。重點理解: Event Storming 產出的領域模型如何映射到 Gherkin 的 Scenario, Gherkin 如何映射到 ISA(如後端需要 API Spec + Entity Spec), ISA 如何產生測試程式碼與實作程式碼。	100%	已完成		v		v	2025/11/4	
Ch2-規格光譜	理解從 DSL 翻譯到 ISA 的意義與世界趨勢	理解軟體語言演進的歷史趨勢與 DSL→ISA 翻譯的重大意義。軟體語言演進:(1) 過程導向時代: 機器語言 → 組合語言 → 高階語言(C/Java/Python)。工程師需要明確告知機器「怎麼做」。(2) 結果導向時代: DSL(Compiler, 編譯器層級) → ISA(Assembler, 組譯器層級) → Codegen(直接生成)。工程師只需描述「要什麼」。系統自動翻譯成實作。對比: ISA 之於 DSL, 如同組合語言之於高階語言。Gherkin DSL 是業務語言, ISA 是技術指令集。透過自動化翻譯(Codegen + LLM)達到 30% →100% 自動化。這是軟體工程的典範轉移: 從 Coding 到 Spec-Writing。	100%	已完成		v		v	2025/11/4	
Ch3-TDD實戰 (70%)	掌握 TDD 基本流程與單元測試	以實際案例(水球軟體學院課程平台)學會 TDD 的基本流程: Red-Green-Refactor cycle。(1) 紅燈(Red): 先寫 failing test, 定義預期行為。(2) 綠燈(Green): 寫最少的程式碼讓測試通過。(3) 重構(Refactor): 在測試保護下改善程式碼品質。重點學習 Service Layer 的規格。從 Service Layer 的規格開始。理解測試驅動開發(Specification)。作為回歸保護(Regression Protection)。作為設計工具(Design Tool)。掌握從零開始建立測試基礎設施(test runner, assertion library)。	95%	2025/11/3-2025/11/18 影片 2025/11/25	v	v	v	v	2025/11/4	
Ch3-TDD實戰 (70%)	理解 TDD 與 Gherkin 思維的結合	學習如何應用 Given-When-Then 框架來思考與撰寫單元測試。取代傳統的 3A(Arrange-Act-Assert)模式。Given(前置條件): 系統的初始狀態、測試資料的準備。When(執行動作): 觸發要測試的行為。Then(後置條件): 驗證預期結果。重點理解:(1) 前置條件在完整條件: 如何驗證行為。(2) 後置條件的測試規則: 如何驗證「執行後」系統狀態的變化。(3) 不同條件下的測試規則: 如何針對不同的 context(如登入 vs 未登入、有權限 vs 無權限)設計不同的測試。這種思維訓練為後端的 BDD(Behavior-Driven Development)奠定基礎。	95%	2025/11/3-2025/11/18		v		v	2025/11/4	
Ch3-TDD實戰 (70%)	掌握 AI 輔助 TDD 的三階段實戰	學習如何讓 AI 協助執行完整的 TDD 流程。(1) 紅燈階段: 提供專門的 prompt 讓 AI 撰寫 failing test。根據規則在前置或後置條件使用不同的 prompt(因為前置與後置的驗證重點不同)。(2) 綠燈階段: 同一個 prompt 讓 AI 持續工作。直到所有測試通過為止。驗證測試驅動的自動化開發流程。(3) 重構階段: 提供專門的 prompt 要求 AI 在回歸測試下進行重構。確保程式碼品質提升但行為不變。重點展現: 將大任務(成功功能)切成小任務(紅→綠→重構)的效益。以及如何透過 prompt engineering 控制 AI 的行為。	80%	2025/11/3-2025/11/18		v		v	2025/11/4	
Ch3-TDD實戰 (70%)	掌握從單元測試到端端測試	學習如何從 Service Layer 的單元測試擴展到完整的後端測試與前端測試。(1) 後端測試: 從單元測試(測試單一 class/function)擴展到整合測試(測試多個元件協作)與 E2E 測試(測試完整的 API 請求-回應流程)。學習如何測試 Controller, Repository, 以及完整的 API endpoint。(2) 前端測試: 學習從高階的 E2E 測試(使用 Playwright/Cypress 等工具)。模擬使用者操作(點擊、輸入、導航)。驗證 UI 行為與狀態變化。理解不同層級測試的適用場景與測試金字塔原則(單元測試多、整合測試中等、E2E 測試少)。	15%	2025/11/3-2025/11/20		v	v	v	2025/11/4	
Ch3-TDD實戰 (70%)	掌握進階測試技巧	學習進階的測試技術與模式。(1) Test Double 概念: 理解為何需要測試替身(隔離依賴, 提升測試速度, 模擬難以實現的場景)。(2) Mock(模擬物件): 驗證互動行為。如「是否呼叫了某個 method」、「呼叫了幾次」、「參數是什麼」。(3) Stub(導物件): 提供預設回應。讓測試聚焦在受測對象而非依賴。(4) Spy(間諜物件): 記錄互動資訊供後端驗證。(5) 讓 AI 精準撰寫 Mock: 透過明確的 prompt 指示。讓 AI 理解何時該用 Mock。如何設定預期行為、如何驗證互動。實戰演練: 測試需要呼叫外部 API、資料庫、第三方服務的場景。	0%	2025/11/10-2025/11/23		v		v	2025/11/11	
Ch3-TDD實戰 (70%)	提升測試判斷力能夠拆解 AI 的測試程式碼	培養評估測試品質的能力。能夠判斷 AI 產生的測試是否完整、正確、有價值。學習如何(1) 針對不同的規則、不同的 context 規劃不同的測試(如註冊功能要測: 正常註冊、重複 email、無效密碼、欄位缺失等)。(2) 理解不同驗證點的差異(狀態驗證 vs 行為驗證、精確匹配 vs 約束匹配)。(3) 識別測試的壞味道(過度耦合實作細節、測試不穩定、測試意圖不清)。(4) 評估測試覆蓋率的意義(不只是程式碼覆蓋率, 更是測試資料覆蓋率)。核心目標: 你能拆解 AI 的測試程式碼。就代表你具備了設計自動化流程的能力。	95%	2025/11/3-2025/11/18		v		v	2025/11/11	
Ch4-BDD實戰 (80%)	理解測試驅動開發規格的問題與 Gherkin 的必要性	認識以測試程式碼作為規格的根本問題:(1) 太瑣碎: 一個簡單的業務場景可能需要數十行測試程式碼。充斥著技術細節(如: setup, teardown, mock 設定)。(2) 難讀: 非技術人員(PM, QA, 領域專家)無法理解測試驅動什麼業務行為。(3) 難改: 業務邏輯變更時, 需要改動大量測試程式碼。維護成本高。理解為什麼需要 Gherkin 來做關注點分離: 將「業務意圖」(What)與「技術實作」(How)分離。業務層用自然語言描述場景。技術層用 Step Definition 實作細節。充分將技術測試難度與複雜度有效下降。以及業務人員與技術人員的合作效率提升。	30%	2025/11/15-2025/11/30		v		v	2025/11/11	
Ch4-BDD實戰 (80%)	掌握 BDD 的 Test-First Workflow 與自動化開發	體驗把規格層級從測試程式碼拉高到 DSL level 可執行規格的威力。BDD 的 Test-First Workflow:(1) 先寫 Gherkin Feature File(描述業務行為)。(2) 執行測試(Red: Step Definition 尚未實作)。(3) 讓 AI 實作 Step Definition 與業務邏輯(Green: 測試通過)。(4) 重構與優化。透過兩個 prompt 完成 BDD 的基本開發: Prompt 1 讓 AI 讀取 Feature File 並生成 Step Definition, Prompt 2 讓 AI 實作業務邏輯。感受給老闆向的自然自動化開發(AI 只需要 Gherkin, AI 自動產生實作)。解放雙手。專注在規格定義而非程式碼撰寫。	30%	2025/11/15-2025/11/30	v		v		2025/11/11	
Ch4-BDD實戰 (80%)	能夠 Review AI 產生的測試程式碼	運用第二章學到的測試評估能力。檢視 AI 在 BDD 流程中產生的 Step Definition 與測試程式碼品質。評估重點:(1) Step Definition 是否正確實作 Gherkin 的語義。(2) 測試資料的準備是否完整(Given 步驟)。(3) 驗證邏輯是否精準(Then 步驟)。(4) 是否有不必要的耦合或重複。(5) 錯誤處理是否恰當。學習如何給 AI 回饋。讓 AI 修正問題(如「這個 Step Definition 應該要驗證 HTTP status code 是 201」、「Given 步驟缺少初始化資料狀態」)。目標: 確保 AI 產生的測試是高品質、可維護的。	30%	2025/11/15-2025/11/30		v		v	2025/11/11	
Ch4-BDD實戰 (80%)	掌握前端與後端的 BDD 落地實戰(結合 AI)	學習在不同技術堆疊(前端/後端)都能進行 BDD 開發並結合 AI 自動化。(1) 後端 BDD + AI: 撰寫 API 層級的 Gherkin Feature File, 使用 API 呼叫的語言(如「When 發送 POST /api/users 請求, with body...」)。讓 AI 產生 API 測試與 Controller/Service 實作。(2) 前端 BDD + AI: 撰寫 UI 層級的 Gherkin Feature File。使用使用者行為的語言(如「When 用戶點擊登入按鈕」)。讓 AI 產生 E2E 測試(Playwright/Cypress)與元件實作。掌握不同層級 BDD 的 Step Definition 撰寫模式。以及如何設計 prompt 讓 AI 理解前端與後端的技術差異。	0%	2025/11/15-2025/12/20		v		v	2025/11/11	
Ch5-ISA可執行規格 (90%)	掌握如何撰寫能產生 99% 正確率測試程式的 ISA feature file	理解 ISA level 與 DSL level 的核心差異。DSL-Level(Domain-Specific Language): 目標讀者是 PO、BA、業務人員。語言風格是自然語言、業務術語(如「當玩家 Alice 加入遊戲 G001」)。細節程度是高階業務意圖。ISA-Level(Instruction Set Architecture): 目標讀者是開發者、測試工程師。語言風格是技術術語。API 呼叫(如「When 玩家加入遊戲, call table: {gameId: {playerName: J}}」)。細節程度是低階技術指令。感受撰寫完 ISA level feature file 後如何達到 99% 正確率: 因為 ISA 消除了歧義。AI 不需要「猜測」而是「執行指令」。	0%	2025/11/20-2025/12/10			v		2025/11/11	

章節	學習成效承諾	具體內容 v1.0	進度	內容上線時程	單元教學影片	圖文教學實戰	實作練習題	Github 專案 & Prompt 實戰教材	共學與線上 FAQ	【加碼工具！即將推出！】網頁 AI 互動學習模組
Ch5-ISA可執行規格 (90%)	透過原子化思維將 AI 產出的測試程式碼精準度提高至 99%	學習原子化測試的設計原則與實踐。原子化 (Atomic) 意義：每個測試步驟只做一件事，驗證一個概念，避免複合邏輯導致的歧義。如何原子化：(1) 將複雜的業務場景拆分成多個原子步驟。(2) 每個 Step Definition 只呼叫一個 API 或執行一個明確動作。(3) 驗證步驟分相關注點 (如分別驗證 HTTP status, response body, database state)。持續調整直到達到 99% 精準度的方法：執行測試 → 檢視 AI 產出的程式碼 → 覆寫該驗證點 → 在 ISA Feature File 中增加約束或拆分步驟 → 重新生成 → 驗證。這個迭代過程訓練你的「規格設計能力」。	0%	2025/11/20-2025/12/10		v	v	v	2025/11/18	
Ch5-ISA可執行規格 (90%)	用 ISA-Feature File 來產出高精度的後端 E2E Test	掌握撰寫後端 ISA-Level Feature File 的技巧，以技術棧 API/CLI 的方式描述後端行為。例如：Given 步驟用 Database seeding (「Given 資料庫中存在, with table: {id   name   email  }」)。When 步驟用 API 呼叫 (「When 發送 POST /api/users, call table: {name   email   password  }」)。Then 步驟用 Response 驗證與 Database 查閱 (「Then response status is 201, {「And 資料庫中存在, with table: {name   name   email  }」}」)。學習如何設計 ISA 讓 AI 能產生完整的 E2E Test，包含：API client setup, HTTP request/response handling, Database assertion, Error case handling。	0%	2025/11/20-2025/12/10	v	v	v	v	2025/11/25	
Ch5-ISA可執行規格 (90%)	用 ISA-Feature File 來產出高精度的前端 E2E Test	掌握撰寫前端 ISA-Level Feature File 的技巧，以技術棧 API/CLI 的方式描述前端行為。例如：Given 步驟用頁面導航與初始狀態 (「Given 用戶在登入頁面」, 「And 表單欄位為空」)。When 步驟用 UI 操作 (「When 用戶輸入 email: test@example.com」, 「And 用戶點擊登入按鈕」)。Then 步驟用 UI 狀態驗證 (「Then 頁面導航至 /dashboard」, 「And 顯示歡迎訊息: Welcome, test@example.com」)。學習如何設計 ISA 讓 AI 能產生完整的前端 E2E Test (使用 Playwright/Cypress)，包含：Browser automation, Element locator, User interaction simulation, Assertion on UI state。	0%	2025/12/22		v	v	v	2025/12/22	
Ch5-ISA可執行規格 (90%)	掌握 ISA-Level 變數系統 (Var-system)	深入理解 ISA-Level Var-system 如何處理動態產生的值。三種符號：(1) >contextKey (大於符號)：定義要存入 scenario context 的變數名稱，出現在 data table 的 header row。(2) <executionKey (小於符號)：指定要從 execution variable 取得哪個值的值，出現在 data table 的 data row。(3) \$variableName (美元符號)：從 scenario context 中引用之前保存的值。可出現在任何需要引用的地方。實際應用場景：處理系統自動生成的 UUID、JWT Token、時間戳等無法預知的值。實現跨步驟的變數引用與傳遞 (如註冊後取得 userId，在後續步驟中用 SuserId 引用)。變數生命週期：產生 (execution variable) → 保存 (<key 與 <key → 引用 (\$key))。	0%	2025/11/10-2025/11/20		v	v	v	2025/12/2	
Ch5-ISA可執行規格 (90%)	Design by Contract - 掌握 ISA-Level 約束系統 (CAS-system)	學習 Cherkon Constraint Assertion System (CAS) 如何實現面向語言的八大類別：(1) 數值比較 (&eq, &gt, &lt, &gt;=, &lt;=, &neq)。 (2) 類型檢查 (&isNum, &isStr, &isBool, &isNil, &isDef)。 (3) 字串操作 (&contains, &startsWith, &endsWith, &matches, &length)。 (4) 集合/陣列 (&hasItem, &size, &isEmpty)。 (5) 存在性 (&exists, &hasField)。 (6) 範圍檢查 (&between, &noneOf)。 (7) 時間相關 (&before, &after, &withinDays, &sameDay)。 (8) 函數訪問 (&called, &calledWith)。核心價值：驗證動態數據而不硬編碼具體值。如「age   &between(18,65)  」取代「age   25  」。讓測試更真實、更穩定、更易維護。	0%	2025/11/10-2025/11/20		v	v	v	2025/12/2	
Ch5-ISA可執行規格 (90%)	Contract Testing - 透過合約測試做到分散式系統的整合測試驅動開發	理解合約測試 (Contract Testing) 在微服務架構中的重要性與實作方式。問題背景：在分散式系統中，服務之間透過 API 互相依賴。如何確保 Provider (提供者) 的變更不會破壞 Consumer (消費者)？傳統的 E2E 測試成本高、速度慢、難以維護。合約測試的解決方案：(1) Consumer 定義期望的 API 合約 (request/response 格式)。(2) 從 ISA feature file 生成 Pact 合約測試。(3) Provider 驗證是否滿足所有 Consumer 的合約。(4) 在 CI/CD 中自動執行合約驗證。學習如何從 ISA Feature File 映射到 Pact 合約定義，以及如何進行 Provider 與 Consumer 的契約驗證，達到分散式系統的測試驅動開發。	0%	2026/1/20		v	v	v	2025/12/9	
Ch5-ISA可執行規格 (90%)	掌握 DSL level 翻譯成 ISA level 的實戰	以水球軟體學院課程平台為真實案例。學習完整的 DSL → ISA 翻譯流程。翻譯步驟：(1) 讀取 DSL-Level Feature File (業務導向的 Gherkin)。(2) 識別業務實體與動作 (如「用戶註冊」, 「課程購買」)。(3) 映射到技術實體與 API (如 POST /api/users), 「POST /api/orders」)。(4) 設計 ISA-Level Feature File (技術導向的 Cherkon，包含 API endpoint, request/response 格式, database schema)。(5) 使用 DSL-to-ISA Translator 工具自動化翻譯 (基於 YAML 配置的規則引擎)。(6) Review 翻譯結果並手動調整，透過實戰演練，理解翻譯過程中的關鍵決策點，常見問題與最佳實踐。	0%	2026/1/1		v	v	v	2025/12/9	
Ch6-自幹Agent (99%)	學會自幹一個可以自動化執行 BDD ISA Feature File to Auto Feedback Loop 的 workflow	學習如何用 Python 脚本控制整個 BDD 流程，達到 99% 自動化。核心 workflow：(1) 從 ISA Feature File 讀取規格 (parsing Gherkin syntax)。(2) 自動產生測試程式碼 (透過 LLM 或 Codegen)。(3) 執行測試並取得結果 (test runner integration)。(4) 根據結果自動回饋修正 (如果測試失敗，讓 AI 讀取錯誤訊息並修正程式碼)。(5) 持續循環直到所有測試通過 (Auto-Feedback Loop)。實作技術：使用 CLISDK 工具 (如 pytest, behave)，腳本化控制流程 (Python/Shell)，錯誤訊息解析與 AI prompt engineering，測試結果追蹤與報告。這個 workflow 是企業級 BDD 自動化的基礎。	0%	2025/12/8	v	v	v	v	2025/12/16	
Ch6-自幹Agent (99%)	學習如何使用 SDD.os 框架來做到全自動化開發	掌握 SDD.os (Specification-Driven Development Operating System) 的核心架構與運作機制。SDD.os 的五大核心元件：(1) SpecReader: 讀取與解析 Feature File (支援 DSL 與 ISA level)。(2) InstructionArchitecture (ISA): 定義 Step 對 Executor 的映射規則 (使用正則表達式匹配 Gherkin Step，映射到對應的 Executor)。(3) Scheduler: 管理測試執行順序與依賴關係。(4) Assembler: 將 Feature File 翻譯成可執行的測試程式碼 (透過 Jinja2 模板 + Codegen)。(5) Executors: 執行具體的測試動作 (API call, Database query, UI interaction)。學習如何配置和使用 SDD.os 進行規格驅動開發，達到 99% 的全自動化開發能力 (僅需人工撰寫 Feature File 與 Review 結果)。	0%	2025/12/8		v	v	v	2025/12/16	
Ch7-超AI化 (100%)	學會如何透過 SDD.os 來做到 100% 絕對不可能寫錯的測試程式碼生成達到 100% 全自動化開發	理解終 Codegen (不依賴 LLM) 的運作原理與優勢。為什麼需要 100% Codegen？LLM 有機率產生錯誤的程式碼 (即使是 99% 也代表每 100 次有 1 次錯)，在關鍵系統中不可接受。Codegen 的運作方式：(1) 定義 ISA 映射規則 (YAML 配置檔，明確規定每個 Gherkin Step 對應的程式碼模板)。(2) 使用 Jinja2 模板引擎生成測試程式碼 (將 Step 的參數注入模板)。(3) 確保生成的程式碼 100% 符合規格 (規則映射是決定性的，沒有 LLM 的隨機性)。優勢：100% 準確度、Token 成本低、不需要呼叫 LLM、執行速度最快、可預測性最高、適用場景：高度結構化的測試 (如 API CRUD 測試, Database migration 測試)，達到真正的「寫完規格，程式自動生成」。	0%	2025/12/15	v	v	v	v	2025/12/23	
加碼課程 Ch8-敏捷整合	學會如何把全自動化開發導入 DevOps, Agile, Scrum 讓團隊軟體產值放大數倍	學習全自動化開發在敏捷團隊中的應用與整合方式。(1) DevOps 流程整合：將 SDD.os 整合到 CI/CD Pipeline (如 GitHub Actions, GitLab CI)，實現規格變更後自動生成測試、自動執行驗證、自動部署。設計 Spec-Driven CI/CD 流程：Commit Feature File → Trigger Pipeline → Codegen 生成測試與實作 → 執行測試 → Deploy。(2) Agile/Scrum 框架應用：在 Sprint Planning 中用 Feature File 作為 User Story 的 Acceptance Criteria。在 Daily Standup 中追蹤 Feature File 的完成度。在 Sprint Review 中 Demo Feature File 驅動的功能。(3) 團隊協作模式：BA/PO 負責撰寫 DSL Feature File, Tech Lead 負責翻譯成 ISA Feature File, Developer 負責 Review AI 產出的程式碼。QA 負責補充邊界測試，讓團隊整體產值放大數倍。減少 70% 的 coding 時間，提升 90% 的測試覆蓋率，降低 50% 的 bug rate。	0%	2026/2/1		v		v	2026/2/3	