

[https://youtu.be/CRKeQXpDu1k?list=PLIAzIZzCjtLJQzoK-1n6R\\_KVTXm4tHzqP](https://youtu.be/CRKeQXpDu1k?list=PLIAzIZzCjtLJQzoK-1n6R_KVTXm4tHzqP)

這份原始資料是台大紅中陽教授「**Web Programming**」2025年課程導論的逐字稿，核心旨趣在於引導學生如何於 **AI 浪潮** 下重新定義軟體工程師的角色。課程從傳統的手寫代碼轉向以 **AI Coding** 為核心的開發模式，強調學生應扮演管理一群「**Super Junior**」代理人的高階主管，專注於邏輯規劃、需求描述與系統設計，而非瑣碎的語法撰寫。除了技術層面的全端開發與 **React** 框架學習，教授更分享了其子在矽谷新創公司的實戰經驗，藉此激勵學生培養**自學能力**與**對科技的好奇心**。課程設計極具挑戰性，要求學生透過七次高強度的作業與一個必須具備實際使用者的**期末專題**，將創意轉化為真正能運行的網路服務。

Discuss AI 輔助開發。

AI 輔助開發（AI-assisted development）正處於一場快速且劇烈的變革中，這項技術已經從單純的輔助工具演變成改變軟體工程習慣的核心力量，。

以下根據來源內容，從技術趨勢、能力重新定義、開發流程及學習方法四個維度來探討 AI 輔助開發：

### 1. 開發形態的典範轉移

過去軟體開發強調「手把手」地撰寫每一行代碼，往往需要花費數個月才能完成一項服務。然而在 AI 浪潮下，開發習慣已大幅改變：

- 從「手寫」到「指引」：現在開發者不再需要從頭撰寫語法，而是透過與 AI 沟通（如使用 **Cursor**、**ChatGPT** 或 **Claude**）來實踐想法，。
- AI 作為「**Super Junior**」：AI 的代碼編寫能力已達到「專業級」水準，相當於一名 **Super Junior 工程師**，能夠秒殺大多數傳統的資料結構或演算法作業，並極大機率取代一般的初級工程師，。
- **Agentic Mode**（代理模式）：這是目前盛行的開發模式。開發者可以將多個想法丟給 AI 代理人（Agent），讓其在背景自動執行，開發者只需在事後進行「收割」與調整，這能同時處理多項任務並完成七至八成的開發進度。

### 2. 開發者能力的重新定義

當 AI 承擔了大部分的編寫工作，人類工程師的角色轉向了更高等級的**管理與決策**：

- **溝通與任務拆解**：未來的核心能力不再是記憶語法，而是**如何與 AI 有效溝通，並有邏輯、有系統地將複雜需求轉化成一系列工項**。
- **需求定義的價值**：AI 雖然強大，但無法「腦補」人類心中複雜且細膩的需求（如像 Notion 這樣複雜的服務），。**人類的想法與對需求的描述才是最珍貴的資產**。
- **監督與指導（Senior 角色）**：開發者必須扮演「主管」的角色，管理這群不會累的 Super Junior。這需要開發者具備足夠的專業素養來判斷 AI 寫出的內容是否正確，並在出錯時給予指引（guidance），。

### 3. AI 輔助下的開發流程

在 Web 開發等實務場景中，AI 展現了極高的效率：

- **One-shot 實踐**：對於簡單的應用程式（如個人網頁、單人遊戲或五子棋），AI 往往能透過「一段話」的描述就直接生成可執行的成果，。
- **全端開發與部署**：藉由 AI 的幫助，開發者可以在短時間內（如兩個月內）從無到有建立起一個包含前後端、資料庫且成功部署到網路上的**全端服務（Full-stack Service）**，。
- **善用現成套件**：現代開發不再是「重新造輪子」，而是學會利用 AI 搜尋並整合正確的第三方套件與 API（如 Google Map API 或 OpenAI API）,,。

#### 4. 新的學習路徑：由果溯因

AI 也徹底改變了學習編程的方法：

- 「看代碼」比「寫語法」更高效：與其從抽象且痛苦的語法開始學起，現在建議先叫 AI 幫你寫出來，再透過閱讀與測試這些代碼來學習。
- 專注於高階概念：人類的記憶有限，應將容易忘記的細節語法交給電腦，人類則專注於學習**系統設計**、**演算法邏輯**與**軟體工程的架構觀念**,。

總結來說，AI 輔助開發讓開發者能從繁瑣的代碼編寫中解放，轉而專注於**商業模式**、**產品創新**與**人性化的服務設計**,。

若要理解這種關係，可以想像**開發者是一位「導演」，而 AI 是「技術極其精湛但缺乏靈魂的演職員工」**。導演不需要親自去架燈光或剪輯每一格底片，但他必須清楚劇本的走向，並知道如何指導員工將腦中的畫面精確地呈現在螢幕上；如果導演自己完全不懂運鏡與節奏，即便有再強大的團隊，也拍不出動人的電影。

本課程作業的評分機制與同儕審查規則為何？

課程中對於期末專題的組隊人數有何限制？

學生寄送問題郵件給助教時應注意哪些禮儀？

Discuss 全端網頁實作。

根據來源內容與先前的討論，**全端網頁實作（Full-stack Web Development）** 在 AI 時代已不再僅僅是學習程式語法，而是轉向以**AI 輔助為核心**，從需求定義到自動化部署的完整工程實踐過程。

以下詳細討論全端網頁實作的關鍵組成與實踐流程：

##### 1. 技術棧與核心架構

現代全端開發不再是「重新造輪子」，而是學習如何整合高效的工具與框架：

- **前端開發（Frontend）**：以 **HTML**、**CSS**、**JavaScript/TypeScript** 為基礎。其中，**React** 被視為經典且熱門的框架，能透過組件驅動（Component-driven）的方式打造服務。
- **全端框架（Full-stack Framework）**：來源特別推薦 **Next.js**，它能同時提供前端與後端服務，並實現伺服器渲染（Server-side Rendering）等先進技術。
- **單頁應用程式（SPA）**：這是現代網頁的主流觀念，僅更新畫面上需要變動的部分（如 YouTube 或 Notion），而非重新整理整個頁面，能大幅提升使用者體驗。

## 2. 後端邏輯與資料管理

- **伺服器與資料庫**：使用 **Node.js** 作為經典後端環境，並結合資料庫管理系統來處理數據。
- **即時通訊（Real-time Communication）**：針對如 IG 按讚通知或聊天室等需求，需運用 **WebSockets** 技術來實現資料的即時更新。
- **API 構桿運用**：優秀的全端開發者應學會構桿（Leverage）第三方服務，例如串接 Google Map API 製作旅遊規劃工具，或串接 OpenAI API 打造專屬的 LINE 機器人。

## 3. 開發流程與維運（DevOps）

在全端實作中，開發環境的建構與維護與撰寫代碼同樣重要：

- **環境建置**：需熟悉 **Local Host** 開發、使用 **Git** 進行版本控制，並將代碼託管於 **GitHub**。
- **自動化流（CI/CD）**：為避免複雜系統隨時間崩壞，需建立**持續整合與持續部署（CI/CD）** 流程，確保系統能穩定地進行更新與維護。
- **容器化技術**：當產品從專案轉向正式產品時，需使用 **Docker** 等容器化技術來進行服務管理與部署。

## 4. 以使用者為核心的實作哲學

成功的全端實作必須具備**實用價值**：

- **真實需求驅動**：開發應源於自己或身邊人會使用的需求。若產品僅是開發者的「腦補」，往往會走向失敗。
- **數據導向修正**：實作不僅是上線就結束，還需埋設追蹤代碼，利用 **Google Analytics 4** 或 **Mixpanel** 等工具觀察使用者行為，並根據真實回饋進行迭代（Iteration）。
- **部署至公網**：實作的目標是將服務部署到**公開網路（Public Service）**，讓所有使用者都能透過網址存取。

## 5. AI 在全端實作中的角色

在實作過程中，AI 扮演了 **Super Junior 工程師** 的角色。對於簡單的個人網頁或單人遊戲，AI 甚至能做到 **One-shot（一段描述即完成）**。然而，面對如社群論壇（如 X/Twitter）這類複雜的全端服務，開發者則需發揮 **Super Senior** 的管理能力，將需求拆解成一系列工項，並指引 AI 完成開發。

總結來說，全端網頁實作是將一個「想法」轉化為「可運作服務」的過程。

如果用一個比喻來理解：**全端開發者就像是一位「餐廳老闆」**。前端是客人的用餐空間與菜單設計，後端是廚房的運作邏輯，資料庫是食材倉庫，而 AI 則是聘請來的高效廚師。老闆不需要親自下廚處理每一道菜（寫代碼），但他必須精通整間餐廳的運作流程（軟體架構），確保食材（資料）新鮮、出菜順暢（部署），並根據客人的回饋不斷改良菜色（迭代），餐廳才能長久經營。

本課程使用哪種授權方式開放課程內容？

作業互評機制中，分數差距達幾分會由助教重評？

學生應如何與助教聯繫以處理課程相關問題？

Discuss 專案管理實踐。

在軟體開發的脈絡下，**專案管理實踐**已不再僅僅是進度追蹤，而是涵蓋了從需求定義、技術選用、協作流程到產品迭代的全面性管理。根據來源內容，現代的專案管理特別強調 AI 工具的整合與開發者角色的轉型。

以下分點討論專案管理的關鍵實踐：

### 1. 角色轉型：從開發者到管理者 (Super Senior)

在 AI 輔助開發的時代，專案管理者的角色被重新定義為 「**Super Senior**」，負責指揮一群不會累的「**Super Junior**」（即 AI 工具）。

- **任務拆解**：管理者的核心能力在於如何有邏輯、有系統地將複雜的需求轉化為一連串的工項 (**Work Items**)。
- **需求定義**：AI 無法自動補完人類心中複雜的想法。專案最珍貴的資產是管理者對**需求描述與規格定義**的精確度，這決定了專案的成敗。

### 2. 開發環境與協作流程 (CI/CD)

一個成熟的專案需要穩健的開發環境，以確保系統在功能增加時不會「爛掉」。

- **版本控制與協作**：使用 **GitHub** 進行代碼上傳與團隊協作是基本要求。
- **自動化流程**：實踐 **CI/CD**（持續整合與持續部署）是專案管理中極其重要的一環，能確保代碼在整合與發布過程中的品質穩定。
- **善用現成套件**：專案管理應避免「重新造輪子」，而是學會搜尋並整合正確的第三方套件與 API（如 Google Map 或 OpenAI API）來提升效率。

### 3. 品質控管與同儕審查 (Mutual Review)

為了確保專案品質並促進團隊成長，實踐**互評機制 (Mutual Review)** 是非常有效的方法：

- **多元回饋**：透過多位成員（或同學）互相審查代碼與成品，不僅能發現盲點，還能互相學習不同的開發技巧。
- **給予建設性評論**：審查時應包含質性的評語（如 100 字以上的評論），這能讓專案成員感受到支持並獲得具體的改進方向。

### 4. 以使用者為核心的產品迭代

專案管理不應止於「完成開發」，而應關注產品的生命週期：

- **解決真實需求**：建議開發者設計**自己或身邊的人會使用的應用**，這樣才能獲得最直接、最真實的使用者回饋。
- **數據驅動優化**：透過在服務中埋入程式碼追蹤使用者行為（如使用 Google Analytics），根據 **Log 資料**而非開發者的「腦補」來進行產品改進。
- **快速原型與迭代**：及早將 **Prototype**（原型）部署到網路上，並透過不斷的 **Iteration**（迭代）讓服務從洋春版進化為成熟的產品。

### 5. 敏捷開發的應用

在管理團隊時，可引入如 **Lean Startup**（精實創業）或 **Sprint**（衝刺）等快速開發的觀念，透過短週期的開發目標來維持專案的動能。

---

總結來說，現代專案管理實踐的核心在於：「將繁雜的編碼交給 AI，將人類的精力保留在思考、設計與理解使用者需求上」。

若要用一個比喻來理解，**專案管理就像是一場「交響樂團的指揮」**。指揮不需要親自演奏每一件樂器（就像工程師不必手寫每一行代碼），但他必須精確理解樂譜（需求規格），並知道如何協調各個聲部（AI 與團隊成員）在正確的時間點發出聲音；如果指揮不了解各樂器的特性或缺乏對整部作品的遠見，即便樂手技術再高超，也無法演出動人的樂章。

說明本課程採用 MIT 授權條款的具體規範。

AI 浪潮如何改變本學期的網頁開發教學模式？

簡述本課程作業的同儕互評機制與評分準則。

Discuss 自主學習精神。

根據來源內容，**自主學習精神**在 AI 浪潮襲來的今日，已不再僅是加分項，而是軟體開發者能否生存並維持競爭力的核心責任。

以下從必要性、實踐方法、心理素質及目標導向四個方面來探討：

### 1. 自主學習的必要性：老師與學校的局限

在技術爆炸的時代，知識更新的速度遠超想像。

- **責任移轉**：由於科技進步太快，老師可能也還在學習新技術，甚至許多教材仍停留在數十年前的基礎知識。因此，**未來學習的責任在於學生自己**。
- **開窗與探索**：課堂教學的角色僅是為學生「開一扇窗」，讓學生窺見科技的進步，窗外的世界則必須靠學生自己去探索。

### 2. 實踐自主學習的方法

現代學習者應善用數位資源與 AI 工具來進行高效學習：

- **做中學與查資料**：學習不應期待聽完課就能直接寫作業，而是要親自動手實作，遇到不懂的地方主動看影片或上網查資料。
- **投資學習資源**：來源中舉例，教授的兒子曾自行購買十幾門線上課程來學習演算法、Python 與 JavaScript，這種不吝於投入時間與少量金錢的精神是自學的關鍵。
- **透過 AI 「由果溯因」**：與其痛苦地死背抽象語法，不如叫 AI 先寫出來，再透過觀察 AI 產出的代碼來學習其邏輯，這比查閱繁瑣的文件更快速。

### 3. 必備的心理素質

- **謙虛與好奇心**：在浩瀚的知識面前應保持謙虛，若兩三年不進步，原本掌握的技術很快就會過時。

- **跳脫分數框架**：不應過度在意成績（如爭取 A+），而應專注於是否真的學到東西。
- **內在驅動力**：真正的自學者是因為覺得「創造新東西非常有趣」而主動投入，而非被公司或學校逼迫。

#### 4. 目標導向的自學：解決真實問題

自學應與實踐掛鉤，而非盲目吸收：

- **開發自己會用的應用**：設計一個自己或身邊的人每天都會用的服務，能讓你獲得最真實的反饋，並驅使你透過不斷的**迭代 (Iteration)** 去解決問題。
- **理解背後原理**：即便 AI 能代勞大部分的代碼（One-shot coding），自主學習者仍會花時間弄懂背後的原理，以具備指導 AI 的能力，避免成為只會出一張嘴卻沒有技術底蘊的主管。

---

總結來說，自主學習精神是從被動接收轉為主動獵取知識的過程。

若要用一個比喻來理解：**自學者就像是一位在原始森林中探險的「開路先鋒」**。老師提供的地圖（課程）只標註了森林的入口和大致方向，但真正的路徑需要探險者自己揮舞開山刀（AI 工具與自學能力）去開闢；如果只坐在入口等待別人鋪好柏油路，你將永遠無法看見森林深處最壯麗的風景。

課程對於「停修申請」的期限與規則為何？

本課程建議使用的 AI 程式編碼工具有哪些？

期末專題對於小組人數與作品部署有何要求？

Discuss 課程評分機制。

這門課程的評分機制設計相當嚴謹且強調**同儕互動**與**實作成果**，主要分為**作業 (50%)** 與**期末專題 (50%)** 兩大核心部分。

以下詳細說明各項評分標準：

##### 1. 作業評分 (佔總分 50%)

全學期共有 **7 個作業**，大約每兩週需要繳交一個。作業評分並非由助教逐一批改，而是採用 **Mutual Review (同儕互評)** 機制。

##### • 作業內容分 (40分) :

- 每個作業滿分為 6 分（7 個作業總計上限 40 分）。

##### ◦ 評分標準：

- **6分**：表現極其出色，超出預期。

- **5分**：非常優秀。

- **3分**：表現平平，尚可接受。

- **1分**：僅具備最基本功能。

- **0分**：程式無法運行。

- **互評責任分 (10分)**：

- 每繳交一個作業後，系統會分派 **3 份**同學的作業供你評論，全學期共需評論 21 份作業。
- 每份評論須包含**至少 100 字的質性評語**，且必須給予整數分。
- 每少完成一份評論扣 0.5 分，扣完為止。

- **防呆與複核機制**：

- 若三位同學給出的分數差距過大（如最高與最低差 4 分以上），或收到的評分不足 3 份，則由助教介入重新評分並以助教分數為準。

## 2. 期末專題評分 (佔總分 50%)

期末專題要求由 **2 至 3 人組隊**，嚴禁單人作業。

- **實作成果分 (40分)**：

- 由教授（佔 40%）與三位助教（各佔 20%）進行評分。
- 評分對象包含 **Deployment**（已部署至網路的正式版本）以及 **Local Host**（本地端運作版本），以確保代碼真實性，防止虛假成果。

- **流程參與分 (10分)**：

- **Proposal (提案)**：準時繳交提案得 2 分。
- **Prototype (原型)**：完成可點擊、已部署的初步原型得 3 分。
- **Mutual Review (互評)**：參與專題互評活動得 5 分。

## 3. 重要規定與考勤

- **零容忍政策 (Zero Tolerance)**：作業與評論的繳交截止時間為 **禮拜一晚上 9:00**，逾時不候，一旦錯過即為 0 分。
- **抄襲處置**：雖然不主動抓抄襲，但若被發現代碼與他人雷同且判定為抄襲，該門課將直接**當掉 (Fail)**。
- **不點名**：課程不採取傳統點名，學生可選擇看線上錄影或實體到課，分數取決於產出的品質而非出席率。

## 4. 成績分布趨勢

教授指出，本課程的成績分布通常呈現「**海嘯型**」狀態：

- **高停修率**：由於 loading 極重（被形容為 90 分的 loading 換 3 學分），會有大量學生選擇中途停修。
- **高分回饋**：只要能撐過「海嘯」、按時交作業並完成期末專題，最終拿到的成績通常不錯（平均約在 80-82 分，即 A- 左右），且 A 與 A+ 的比例相對較高。

**總結來說**，這門課的評分機制更像是一場「軟體開發馬拉松」。老師與助教並不扮演唯一的裁判，而是建立一個類似真實業界的「代碼審查系統」（Code Review System），讓你透過評價他人的作品來精進自己的眼光。只要你能在規定的時間內不斷產出成果並保持在浪頭上，就能順利通過這場海嘯，獲得豐碩的學術回報。

教授為何決定將 2024 年度的課程停開？

學生如何在 10 月 26 日前申請停修？

期末專題對於小組人數與成員身分有何限制？