AI 時代的軟體工程

從 Vibe Coding 到 Vibe Engineering

時程

時間	內容
9:00–9:30	自我介紹 + 環境安裝
9:30–10:30	AI 未來 + 原則
10:30–12:00	Live Demo：數位勞報單系統
13:00–14:00	產品思維 + 發想
14:00–15:30	實作：旅遊規劃網站
15:30–18:00	分享 + Feedback + Q&A

環境安裝

請確認以下工具已安裝：

```
# Docker
docker --version

# Node.js (v18+)
node --version

# Claude Code
claude --version
```

安装指引

Docker Desktop

https://www.docker.com/products/docker-desktop

Node.js

https://nodejs.org/

Claude Code

```
npm install -g @anthropic-ai/claude-code
```

VS Code 擴充套件

建議安裝：

- **Marp for VS Code** — 預覽簡報
- **ESLint** — 程式碼檢查
- **Prettier** — 格式化
- **Tailwind CSS IntelliSense**

我是誰

葉又銘

台大資工碩一｜App & AI & Agent 開發、研究

我在做什麼

- 實驗室在幫助上市櫃公司建立 **AI Code Review Workflow**
- 經營 **Kardomo** — K-pop 社群平台（2 個月開發上線），目前 25k 用戶
- 創辦 **Peeps** — Gen Z 社交 App（1 週獲得 10K 用戶）
- 大量接案、AI 開發陪跑

(以上都在去年上半年)

App 獨立開發者、接案仔

同時經營多個專案（已上線五個有賺錢的 App）
全端 + 產品 + 設計 + 維運

一個人怎麼做到的？

我與哥布林

接案時發現的問題：

- 步調太慢，無法快速迭代
- 技術老舊，包袱過重
- AI 尚未完全導入
- 部署手動成分過高
- 上一個案子拖了好幾個月...

我的觀察

> 系統全包給我一個月就能做完一個可以賣錢的產品
> 我遇過的所有資訊系統廠卻都要安排前端、後端、UIUX、PM 才能完成...

差別在哪？

今天的目標

1. 理解 AI 開發工具的演進與瓶頸
2. 學會用 Claude Code 高效開發
3. 建立正確的工程與產品思維
4. **親手做出一個可上線的產品**
5. **培養未來時代的思維 Mindset**

準備好了嗎？

裝好的請舉手

有問題的請舉手

讓我們開始

The Future of AI

AI 開發工具的演進與瓶頸

AI 開發工具演進

你用過什麼工具？經驗是什麼？

2024：自動補全

GitHub Copilot 時代

- Inline suggestions
- 一次補一行、一個函數
- 你還是要寫大部分的 code

一個工程師的效率可以提升個 1.1 倍

2025：一個檔案

Cursor / Windsurf

- Composer 模式
- 可以改整個檔案
- Chat in editor
- 但還是一次處理一個檔案

一個工程師的效率可以提升個 1.5 倍
UIUX、PM 可以加入協作

2026：多個檔案

Claude Code / Codex

- Terminal-first、Agentic coding
- 200K token context window
- 跨檔案重構
- 可以執行 shell commands
- SWE-bench 80.9%（Opus 4.5）

可以完成一或多個 Junior SWE 的工作量（根據操作者能力）

2027：架構 + 設計 + QA

我的預測

- Spec-driven development
- AI 做架構設計
- AI 做 QA（自動測試、自動驗證）
- Background agents — Fire and forget
- Overnight PR

Junior、IT 可以正式取代 Senior SWE 的工作

演進的關鍵

年份	能力	範圍
2024	補全	一行
2025	編輯	一個檔案
2026	重構	多個檔案
2027	工程	整個專案

為什麼會這樣演進？可以觀察到什麼？

為什麼哪些做得好？哪些做不好？

目前所謂的「AI」有瓶頸

你認為 AI 是什麼？你怎麼定義？

這次工作坊我的定義

AI = 超強的文字接龍機器

- 根據前文預測下一段文字
- 不是「思考」，是「預測」
- 沒有真正的理解，只有模式匹配

但這已經夠強了

瓶頸 1：長期思維

- AI 只看當下的 context
- 不知道你昨天做了什麼
- 不知道你下週要做什麼
- **沒有長期記憶**

為什麼？

瓶頸 2：視覺能力

- AI 看不懂 UI/UX
- 不知道設計好不好看
- 不知道使用者體驗如何
- **需要人類判斷**

為什麼？

瓶頸 3：自我學習

- AI 不會從錯誤中學習
- 每次對話都是全新開始
- 重複的錯誤會一直犯
- **你要幫它學（CLAUDE.md）**

為什麼？

瓶頸 4：刪減能力

- AI 很會加東西
- 但不太會刪東西
- Over-engineering 傾向
- **需要人類簡化**

為什麼？

瓶頸 5：Bias

- 訓練資料的偏見
- 過時的 best practices
- 獎勵機制導致的行為
- **需要人類校正**

為什麼？

為什麼有這些瓶頸？

1. **訓練的獎勵機制** — 目前仍然在解決急迫的問題，還沒有為長期思維做優化，加上現在訓練多不依靠資料，而是依靠人類反饋（RLHF）
2. **訓練資料** — 過時、有偏見、不完整（其實很多 AI 現在想做到的事，網路上根本沒這種資料，e.g. QA、Reasoning、Tool Calling、Multi-step...）
3. **架構限制** — Context window、無狀態，以語言為載體而非在神經中的知識

Questions so far?

Agents

什麼是 Agent？

Agent vs AI

傳統 AI：**生成文字**

Agent：**Write code → Call tools → Execute → Loop**

現在 AI （LLM）到底怎麼運作

其實在模型看到的是長這樣：

```
System: You are a helpful assistant.
User: Please write a function that adds two numbers.
Assistant: Here is a Python function that adds two numbers:
def add(a, b):
    return a + b
User: Now, please save this function to a file named add.py.
Assistant:  (這裏 LLM 會根據前面的文字預測下一段文字)
```

是在「文字接龍」，不是在「思考並回答」。

問：為什麼 AI 現在可以調用工具、生成圖片？猜一下

Agent 的能力

```
接收指令
    ↓
分析任務
    ↓
呼叫工具（讀檔、寫檔、執行指令）
    ↓
檢查結果
    ↓
決定下一步
    ↓
重複直到完成
```

跟 RAG 沒有任何關係

問：為什麼 2025 是 Agent 的元年？

Claude Code 就是一個 Agent

- 讀寫檔案
- 執行 shell commands
- 搜尋 codebase
- 呼叫 MCP tools
- **自主決定下一步**

Context Engineering

> Tobi Lütke (Shopify CEO), 2025

從 Prompt Engineering → **Context Engineering**

Context Engineering 是什麼？

不只是寫好的 prompt
而是**管理 AI 能看到的所有資訊**

- System prompts
- Tools / MCP
- Message history
- Retrieved data (RAG)
- Memory

為什麼重要？

> "The art of providing all the context for the task to be plausibly solvable by the LLM"

AI 的能力 = 模型能力 × **Context 品質**

實作：MCP (Model Context Protocol)

讓 AI 能夠連接外部工具和資料

推薦 MCP Servers

MCP	功能
Context7	即時取得最新文件（Next.js、React...）
shadcn	查詢 shadcn/ui 元件、範例
Next.js DevTools	Next.js 開發工具、錯誤偵測
Playwright	瀏覽器自動化測試
Supabase	資料庫操作、Schema 管理

Context7 MCP

解決 AI 文件過時問題

```
claude mcp add context7 -- npx -y @anthropic/context7-mcp
```

使用方式：prompt 加上 `use context7`

> "Create a Server Component using Next.js 15 – use context7"

shadcn MCP

直接查詢元件 registry

```
claude mcp add shadcn -- npx -y shadcn-ui-mcp-server
```

- 取得正確的 props
- 查看使用範例
- 自然語言安裝元件

Next.js DevTools MCP

```
claude mcp add next-devtools npx next-devtools-mcp@latest
```

- 錯誤偵測
- Live state 查詢
- 開發 logs
- 升級指南

Playwright MCP

瀏覽器自動化

```
claude mcp add playwright -- npx -y @anthropic/playwright-mcp
```

- E2E 測試
- 截圖
- 無需視覺模型

Claude Code 進階功能

Hooks：事件自動化

在特定時機自動執行腳本

```
SessionStart  → 注入 git status、TODO
PreToolUse    → 阻擋危險命令
PostToolUse   → 自動格式化程式碼
Stop          → 強制完成檢查
```

Hooks 範例：自動格式化

```
{
  "hooks": {
    "PostToolUse": [{
      "matcher": "Write|Edit",
      "hooks": [{
        "type": "command",
        "command": "npx prettier --write $FILE"
      }]
    }]
  }
}
```

每次寫檔案後自動跑 Prettier

Hooks 範例：自動批准測試

```
{
  "hooks": {
    "PermissionRequest": [{
      "matcher": "Bash(npm test*)",
      "hooks": [{
        "type": "command",
        "command": "echo '{\"decision\": \"allow\"}'"
      }]
    }]
  }
}
```

測試命令不用每次確認

LSP：語言伺服器協定

讓 Claude 有 IDE 等級的程式碼理解

功能	沒有 LSP	有 LSP
找函數定義	grep 搜尋（45s）	直接跳轉（50ms）
找所有引用	猜測	精確列出
型別資訊	讀完整個檔案	即時顯示

效能提升 900 倍

LSP 啟用

```
# 啟用 LSP 功能
ENABLE_LSP_TOOLS=1 claude

# 安裝語言 Plugin
/plugin marketplace add boostvolt/claude-code-lsps
/plugin install pyright@claude-code-lsps
```

支援：Python、TypeScript、Go、Rust、Java...

Plugins：擴充系統

打包 Commands + Agents + Skills + Hooks + MCP

```
# 安裝 Plugin
/plugin install code-review@anthropics/claude-code

# 列出已安裝
/plugin list
```

官方 Plugins

Plugin	功能
code-review	5 個並行 Agent 做程式碼審查
commit-commands	`/commit` 、 `/commit-push-pr`
feature-dev	7 階段功能開發流程
security-guidance	偵測安全漏洞

Plugins Marketplace

```
# 新增 Marketplace
/plugin marketplace add anthropics/claude-code
/plugin marketplace add claudebase/marketplace

# 搜尋 Plugin
/plugin search review

# 安裝
/plugin install code-review@anthropics/claude-code
```

Commands vs Skills vs Hooks vs Plugins

類型	觸發方式	用途
Commands	`/command` 手動	重複的 workflow
Skills	AI 自動判斷	情境規則
Hooks	系統事件自動	事件驅動自動化
Plugins	打包上述全部	分享與分發

MCP 設定檔位置

```
# macOS
~/Library/Application Support/Claude/claude_desktop_config.json

# 或用 CLI
claude mcp add <name> -- <command>
```

Vibe Coding vs Vibe Engineering

Vibe Coding

> Andrej Karpathy, 2025/2

- 完全交給 AI
- 不看 code
- 用執行結果判斷對錯
- **快速原型**

Vibe Coding 的問題

- 19% AI 生成的 code 有安全問題
- 不理解就無法維護
- "Development hell"
- 技術債爆炸

> 我們稱為屎山代碼

Vibe Engineering

> Simon Willison

- 用 AI **加速**
- 但**對產出負責**
- 審查每一行 code
- 理解架構

問：AI 寫 Code 到底意味著什麼

A：取代工程師？

B：加速工程師？

C：改變工程師的工作方式？

我的觀點

軟體開發是工程問題

不是創造、不是生成
是**蓋房子**

軟體資訊是工程問題不是發明東西

蓋房子的邏輯

1. 基建要先蓋好
2. 每蓋完一小區塊就要檢查
3. 有問題馬上修
4. 才能往上蓋

你是工頭

AI 是你的工人

你要：

- 規劃藍圖
- 分配任務
- 檢查成果
- 確保品質

你需要有**長期思維**、**解決問題的能力**、**制定策略**、**標準**、**流程**

Questions so far?

接下來：這個時代，你如何比別人強？

如何比別人強

這個時代的生存法則

問：你覺得呢？

核心：時間與速度

做產品就是在拼速度

拼不過別人就不用玩了

技術不是護城河、資本才是（但跟你也沒關係）

什麼重要？

了解技術 > 追隨潮流

- 思考新技術的存在意義、原因
- 暫停、思考
- 穩定 > 新穎

提醒：軟體是工程問題。

模型 >>> 工具

- 換一個更好的模型 → 立刻提升，真的有效
- 換一個更好的工具 → 學習成本，而且大多數時候沒用
- 遇到 Bug 大多數時候換一個模型就可以解決（Bias 問題）

舉例：

1. Claude Code 好處是模型便宜，對開發者友善
2. Cursor UI 好看還可切換模型

跟他們自家 Agent 能力沒有任何關係

選對模型比選對工具重要 10 倍

問：Gemini vs GPT vs Claude？

速度 > 技術

- 開發產品要思考的是「怎麼最快做出來」
- 也許最新技術可以、也許不行，沒有答案
- 你要思考「你要做到什麼程度」、「怎麼最快做到」

討論：Saas vs Blog

偷前人的程式、貼網路資源

- 不要重造輪子
- 找現成的解法
- 複製、修改、ship

AI 的缺陷就是他現在的腦子已經固定了，有些任務就是做不好，給他翻書吧。

Open source 的意義

以 Shadcn、Supabase 為例。

什麼不重要？想想看你在蓋房子

- 完整的功能（先上線）
- 最擅長的技術？最新的技術？都不重要（先能用）

要學什麼？

1. 觀察前沿

- 追蹤最新的 AI 發展
- 判斷什麼會留下來、其他人有沒有跟進這個趨勢

2. 保持冷靜

- 不要 FOMO
- 不要追每一個新工具
- 專注在能解決問題的

論為什麼 Claude 依然是最好用的開發模型

（現場討論）

3. 用數據說話

- 不要用感覺判斷
- 網路上有統計分析你可以參考
- 新工具真正有多少人在用？他們是解決什麼問題？

4. 打造你自己的工程環境

- 建立、迭代給 AI 用的規則
- 重複利用的提示詞

不學什麼？

- 現在模型做不好，但以後會做好的事。

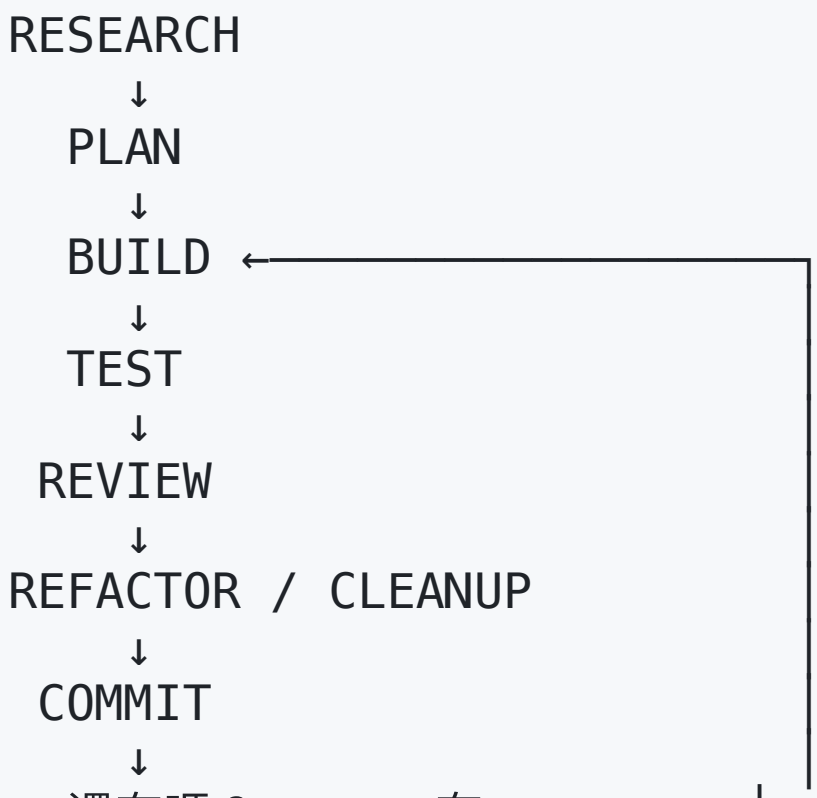
Questions so far?

我的一些原则

我的開發 SOP

也是大多數開發者的流程

流程圖（盡量自動化）


RESEARCH
PLAN
BUILD
TEST
REVIEW
REFACTOR / CLEANUP
COMMIT
還有嗎？
有
沒有
SHIP

RESEARCH

- 先搞清楚要做什麼
- 找現有的解法、開源程式
- 看別人怎麼做的

PLAN

- 讓 AI 先規劃
- 用 Plan Mode
- 沒想好怎麼辦？先 SHIP

BUILD

兩種方式：

1. **複製現成的** — 找別人做過的、改成你要的
2. **直接 ship** — 先做出來、看結果再調整

TEST：用 Docker Build 測試

不要用 Hot Reload 做 QA

```
docker build -t myapp .
docker run -p 3000:3000 myapp
```

為什麼？

- 環境跟 Production 一樣
- 不會有「我電腦可以跑」的問題
- AI 改的東西直接在真實環境測

REVIEW → REFACTOR → CLEANUP

每 1–2 個 feature 就要：

- `review` — 檢查品質
- `refactor` — 重構結構
- `cleanup` — 清理垃圾
- `simplify` — 簡化複雜度

不要累積技術債
就跟蓋房子一樣

COMMIT：我的 Git 工作流

簡單就好：

1. **用 VS Code 產生 commit message** — AI 幫你寫
2. **只在穩定時 commit** — 確定可以跑再 commit
3. **出問題就 revert** — 隨時可以回到上一個穩定版本

Commit = 存檔點

LOOP

```
Feature 1 → Review → Commit
Feature 2 → Review → Commit
          ↓
    Refactor / Cleanup
          ↓
Feature 3 → Review → Commit
Feature 4 → Review → Commit
          ↓
    Refactor / Cleanup
          ↓
        ...
```

遇到 Bug 解不掉？

1. 換 model 重試
2. 給更多 context
3. 拆成更小的問題
4. 問不同的問題

Debugging with AI：詳細技巧

怎麼貼錯誤訊息

Bad：

```
幫我修這個 bug
```

Good：

```
我在執行 npm run dev 時遇到這個錯誤：

[貼完整錯誤訊息]

相關程式碼在 src/app/page.tsx
我剛剛改了 XXX
```

給 Context 的技巧

1. **貼完整錯誤** — 不要只貼最後一行
2. **說明你做了什麼** — 「我剛剛加了 XXX」
3. **說明預期行為** — 「我期望它應該 XXX」
4. **說明實際行為** — 「但它卻 XXX」

什麼時候該放棄問 AI？

- 同一個問題問了 3 次以上沒解決
- AI 開始重複同樣的建議
- AI 給的答案越來越離譜

這時候：

1. Google 搜尋錯誤訊息
2. 看 GitHub Issues
3. 問 Stack Overflow
4. 問真人

Debug 的心法

AI 不是萬能的

有時候傳統方法更快：

- `console.log` 大法
- 二分法找問題
- 看官方文件
- 讀 source code

目標是解決問題，不是證明 AI 有多強

關鍵原則

瓶頸永遠要是 AI 生成程式的時間，不是你

- 你不應該在等 AI
- AI 應該一直在跑
- 同時開多個任務

怎麼做？

1. **多開 Terminal** — 同一個 repo，不同 feature
2. **多開 Repo** — 兩個資料夾，用 Git + GitHub 管理合併
3. **Cloud Background Agents** — Claude Code / Codex，開 PR 再 merge
4. **Plan Mode** — 同時開發同模組時先規劃

你是調度員，不是工人

請盡量把所有東西放同一個專案

- 讓 AI 有足夠 context
- 不要分太多 repo

請利用開源程式庫、文件、範例直接讓他參考

- 現在大多數文件都可以讓你直接複製成提示詞
- 很多很複雜的東西都有人做過了

請讓 AI 知道你的規範

- 建立 CLAUDE.md
- 定義專案規範、常見錯誤（當每次遇到都要記錄）
- 建立經常使用的提示詞，變成 Commands / Skills

根據結果調整 CLAUDE.md

AI 犯錯 → 更新 CLAUDE.md → AI 學會

```
# CLAUDE.md

## 專案規範
- 使用 TypeScript
- 用 shadcn/ui 元件
- 不要用 any

## 常見錯誤
- 不要直接 import from @radix-ui
- 記得加 "use client"

## 流程
- 每次完成一個 feature 要 test、review、refactor、cleanup
- 使用 npm run build、npm run lint、npm run test 自動檢查程式碼並修復問題
```

寫 Commands / Skills 加速

重複的 workflow → 寫成 command

```
# .claude/commands/review.md

請檢查最近修改的檔案：
1. 程式碼品質
2. 潛在 bug
3. 效能問題
4. 安全漏洞
```

把東西變成可擴展的

- 不要用時間換錢
- 建立可複用的流程
- 自動化重複的工作
- 就跟以前你在打造一個團隊一樣，你會制定規則、寫文件，現在是打造你的 AI 團隊

接下來：Live Demo

Live Demo

數位勞報單系統

從 0 到上線

先想一下

如果你要做一個「數位勞報單系統」
你會怎麼開始？

問：你會先做什麼？

A. 直接開始寫 code
B. 先畫 UI 設計稿
C. 先規劃資料結構
D. 先問 AI 怎麼做
E. 先 Google 看別人怎麼做

我的答案

E → D → C → 開始寫

1. 先看別人怎麼做（Research）
2. 問 AI 幫我規劃（Plan）
3. 確認資料結構
4. 開始寫（Build）

我們要做什麼

現場發想、現場設計、現場開發

你會看到我**真實的開發流程**

技術棧

- **Next.js** — React 框架
- **shadcn/ui** — UI 元件庫
- **Tailwind CSS** — 樣式
- **Supabase** — 後端 + 資料庫
- **Docker** — 容器化
- **Vercel** — 部署

工具

- **Claude Code** — AI 開發助手
- **VS Code** — 編輯器
- **Warp** — Terminal
- **Raycast** — 快速啟動
- **Git + GitHub** — 版本控制

Claude Code 功能

CLAUDE.md

專案的「憲法」

```
# CLAUDE.md

## 專案資訊
這是一個數位勞報單系統...

## 技術棧
- Next.js 14 + App Router
- shadcn/ui + Tailwind

## 開發規範
- 使用 TypeScript strict mode
- 元件放在 components/
```

Commands

自訂指令，放在 `.claude/commands/`

```
# .claude/commands/new-feature.md

請幫我開發新功能：$ARGUMENTS

1. 先分析需求
2. 設計資料結構
3. 建立 UI 元件
4. 實作邏輯
5. 測試
```

使用：`/project:new-feature 使用者登入`

Plan Mode

大任務先規劃

```
> /plan 實作勞報單提交流程

Claude 會：
1. 分析需求
2. 列出步驟
3. 等你確認
4. 再開始執行
```

Skills

自動套用的工作流程

```
# .claude/skills/SKILL.md

---
description: 當使用者提到 API 時自動套用
---

API 開發規範：
- 使用 Route Handlers
- 回傳統一格式
- 錯誤處理...
```

MCP (Model Context Protocol)

擴充 Claude 的能力

- **shadcn MCP** — 查詢元件、取得範例
- **Playwright MCP** — 瀏覽器自動化
- **Supabase MCP** — 資料庫操作

Hooks

事件驅動自動化

```
{
  "hooks": {
    "PostToolUse": [{
      "matcher": "Write|Edit",
      "hooks": [{ "command": "prettier --write $FILE" }]
    }]
  }
}
```

- **PostToolUse** — 寫檔後自動格式化
- **SessionStart** — 注入 git status
- **PermissionRequest** — 自動批准測試

Subagents

Claude Code 內建的專業 Agent：

- `review` — 程式碼審查
- `refactor` — 重構
- `cleanup` — 清理
- `simplify` — 簡化
- `debug` — 除錯

問：Commands vs Skills vs Hooks vs Subagents vs MCP?

答案

類型	觸發方式	用途
Commands	`/project:xxx` 手動	重複的 workflow
Skills	AI 自動偵測	情境規則
Hooks	系統事件自動	事件驅動自動化
Subagents	AI 自動委派	專業任務
MCP	AI 自己呼叫	外部工具連接

Subagents 做什麼？

- **review** — 檢查當前改動的品質
- **refactor** — 重構程式碼結構
- **cleanup** — 清除死碼、整合重複邏輯
- **simplify** — 簡化複雜度

Subagents 的原則

遵循 KISS、SOLID、CLEAN 原則

絕對不要 Over-engineering

目標：

- 長期可維護性（像資深工程師思考）
- 更短的 Context → AI 表現更好
- 乾淨的 Codebase → Debug 更容易

開發流程示範

Step 1: Research

- 勞報單是什麼？
- 有哪些欄位？
- 參考現有系統

Step 2: Plan

- 讓 Claude 規劃
- 確認功能範圍
- 設計資料結構

Step 3: Build

- 建立專案
- 開發功能
- **邊做邊學**

Step 4: Review & Refactor

每做完一個功能：

```
/review   # 檢查品質
/refactor # 重構
/cleanup  # 清理
```

Step 5: Ship

- 部署到 Vercel
- 連接 Supabase
- 上線！

開始 Live Coding

請看螢幕

Product Mindset

怎麼做出真正有人用的產品

問：你做過產品嗎？經驗是什麼？

做產品 ≠ 寫程式

技術只是手段
產品才是目的

你的目標是解決問題，不是寫 code

Ship First

問：你通常花多久時間規劃？

不管如何，先上線

- 不要想太多
- 不要規劃太久
- 先做出來再說

> 完美是好的敵人

邊做邊想

- 想破頭想不到？
- 先讓 AI 寫點東西
- 看到成果會有新想法

動手才是最好的思考方式

卡住了？

1. 先做其他專案
2. 一次開多個
3. 讓 AI 幫你打工
4. 你是工頭

不要停下來

Talk to Users

問：你怎麼知道使用者要什麼？

使用者說什麼？

- 不要假設需求
- 直接問使用者
- 觀察使用者行為

不要在房間裡想像

快速驗證

1. 做最小版本（MVP）
2. 給使用者試
3. 收集回饋
4. 調整方向

這個循環越快越好

Less is More

問：你的產品有多少功能？

少即是多

- 功能越少越好
- 只做核心的
- 其他都不要

一個功能做到極致 > 十個普通功能

為什麼？

- 開發時間少
- 維護成本低
- 使用者更容易懂
- 更快上線
- **更容易測試假設**

Easy ≠ Simple

Easy（容易） — 用起來不費力
Simple（簡單） — 沒有複雜的糾結

例子

Easy	Simple
用現成框架	只用需要的功能
複製貼上	理解後精簡
加更多套件	用最少依賴

追求 Simple，不只是 Easy

Easy 會堆積複雜度
Simple 才能長期維護

Design is How It Works

> Steve Jobs

設計 ≠ 好看

設計 = **好用**

- 使用者能完成任務嗎？
- 過程順暢嗎？
- 會困惑嗎？

好的設計

- 不需要說明書
- 一看就知道怎麼用
- 符合直覺

如果需要解釋，就是設計有問題

產品思考框架

問自己這些問題

1. **誰是使用者？** — 越具體越好
2. **他們的痛點是什麼？** — 真的痛嗎？
3. **我的解法是什麼？** — 最簡單的解法
4. **為什麼他們要用我的？** — 差異化

驗證想法

1. **問題存在嗎？** — 真的有人有這個痛點？
2. **解法可行嗎？** — 技術上做得到？
3. **使用者會用嗎？** — 有人願意付錢/時間？

先驗證再開發

Questions so far?

下午實作

旅遊規劃網站

你們來發想！

討論方向

1. **目標使用者** — 誰會用這個網站？
2. **核心功能** — 最重要的 1–2 個功能
3. **差異化** — 跟別人有什麼不同？

（15 分鐘討論）

第一階段：靜態網站

- 展示旅遊規劃
- 行程一覽
- 漂亮的 UI

目標：讓人看到就想用

第二階段：加後端

- 可以新增行程
- 可以修改內容
- 使用者登入
- 資料存 Supabase

目標：變成真正可用的工具

技術棧

```
Next.js + shadcn/ui + Tailwind
            +
         Supabase
            +
          Docker
```

跟上午 Demo 一樣

開始討論：你們想做什麼？

Build Time

實作：旅遊規劃網站

目標

14:00 – 15:30（1.5 小時）

做出一個可以展示的旅遊規劃網站

記住上午學的

1. **RESEARCH** — 先看別人怎麼做
2. **PLAN** — 讓 AI 規劃
3. **BUILD** — 複製現成的 / 直接 ship
4. **REVIEW** — 每個 feature 都要檢查
5. **SHIP** — 上線！

第一階段目標（必做）

靜態網站：

- 首頁
- 行程列表
- 行程詳情頁面
- 好看的 UI

時間：約 45 分鐘

第二階段目標（加分）

加後端：

- 使用者登入
- 新增/編輯行程
- 資料存 Supabase

時間：約 45 分鐘

建立專案

```
# 建立 Next.js 專案
npx create-next-app@latest travel-planner

# 選項：TypeScript、ESLint、Tailwind、App Router

# 進入專案
cd travel-planner

# 初始化 shadcn
npx shadcn@latest init
```

加入常用元件

```
npx shadcn@latest add button card input form
npx shadcn@latest add navigation-menu avatar
npx shadcn@latest add dialog sheet
```

或讓 Claude 幫你加

啟動 Claude Code

```
# 進入專案目錄
cd travel-planner

# 啟動 Claude Code
claude
```

建議的第一個 Prompt

```
我要做一個旅遊規劃網站。

功能：
1. 首頁展示旅行概覽（目的地、日期、人數）
2. 行程列表（每天的安排）
3. 每個行程的詳細資訊（時間、地點、備註）

技術：Next.js 15 + shadcn/ui + Tailwind

請先：
1. 建立 CLAUDE.md
2. 規劃資料結構
3. 列出要建立的檔案

確認後再開始實作。
```

開發流程提醒

```
寫完一個功能
      ↓
  /review
      ↓
  修正問題
      ↓
  git commit
      ↓
 下一個功能
```

不要一次寫太多再 review

善用上午學的指令

```
/review     # 檢查品質
/refactor   # 重構
/cleanup    # 清理
/simplify   # 簡化
/debug      # 除錯
```

善用 MCP

```
# 查 shadcn 元件
"幫我找適合的 shadcn 元件"

# 查最新文件（如果有裝 context7）
"use context7 – Next.js 15 的 App Router 怎麼用"
```

遇到問題？

1. **換個問法** — 重新描述問題
2. **給更多 context** — 貼錯誤訊息、截圖
3. **換 model** — 試試 GPT 或 Gemini
4. **問 Google** — 有時候傳統方法更快
5. **問我** — 舉手！

常見問題

Q: shadcn 元件樣式怪怪的

A: 檢查 `globals.css` 有沒有 Tailwind directives

Q: "use client" 錯誤

A: 有用到 hooks 或事件的元件要加 "use client"

Q: 部署失敗

A: 先 `npm run build` 本地測試

Supabase 設定（第二階段）

```
# 本地啟動（需要 Docker）
npx supabase init
npx supabase start
```

或直接用雲端版：https://supabase.com

Supabase 基本表格

```sql
-- 行程表
create table trips (
  id uuid primary key default gen_random_uuid(),
  title text not null,
  destination text,
  start_date date,
  end_date date,
  created_at timestamp default now()
);

-- 活動表
create table activities (
  id uuid primary key default gen_random_uuid(),
  trip_id uuid references trips(id),
  day int,
  time text,
  title text,
  description text
);
```

部署到 Vercel

```
# 方法 1：CLI
npm i -g vercel
vercel

# 方法 2：連 GitHub
# 推到 GitHub 後在 Vercel 網站 import
```

時間分配建議

時間	任務
14:00–14:10	建立專案、寫 CLAUDE.md
14:10–14:20	規劃、設計資料結構
14:20–14:50	開發核心頁面
14:50–15:10	完善 UI、review、refactor
15:10–15:25	加後端（optional）或 polish
15:25–15:30	準備分享

分工建議（如果組隊）

- **一個人** — 全部自己來，專注核心功能
- **兩個人** — 一人前端、一人後端
- **三個人** — 前端 / 後端 / 設計+測試

開始吧！

有問題隨時問
15:30 分享成果

記得：Ship first, perfect later

分享時間

15:30 – 16:30

分享格式

每人/組 5–8 分鐘：

1. **展示成果** — 實際 demo
2. **遇到的問題** — 卡在哪裡？
3. **怎麼解決的** — 用什麼方法？
4. **學到什麼** — 最大的收穫

評分標準

沒有評分

純粹分享，沒有競爭

重點是

- 你嘗試了
- 你學到了
- 你完成了（多少都算）

過程比結果重要

開始分享

Feedback

16:30 – 17:00

今天的回顧

1. AI 開發工具的演進（2024 → 2027）
2. AI 的瓶頸與限制
3. Agents 與 Context Engineering
4. Vibe Coding vs Vibe Engineering
5. 開發 SOP 與原則
6. Claude Code 實戰
7. Product Mindset
8. 親手實作

核心觀念複習

- **軟體開發是工程問題** — 像蓋房子
- **你是工頭，AI 是工人**
- **瓶頸要是 AI，不是你**
- **Ship first, perfect later**
- **模型 >>> 工具**

你學到了什麼？

請分享：

- 最有用的一個觀念
- 最想回去試的一件事
- 還有什麼想學的

給我的 Feedback

- 哪裡講得好？
- 哪裡可以改進？
- 什麼沒講到但你想知道？
- 時間安排 OK 嗎？

Q&A

17:00 – 18:00

問任何問題

- AI 工具相關
- 開發流程
- 產品思維
- 職涯建議
- 接案經驗
- 創業經驗
- 其他任何事

資源整理

AI 開發工具

工具	連結
Claude Code	https://claude.ai/code
Cursor	https://cursor.sh
GitHub Copilot	https://github.com/features/copilot
Codex	https://openai.com/codex

前端框架 & 元件庫

資源	連結
Next.js	https://nextjs.org
shadcn/ui	https://ui.shadcn.com
Tailwind CSS	https://tailwindcss.com
Radix UI	https://www.radix-ui.com

後端 & 部署

資源	連結
Supabase	https://supabase.com
Vercel	https://vercel.com
Railway	https://railway.app
Cloudflare	https://cloudflare.com

MCP Servers

MCP	用途
Context7	即時文件
shadcn	元件查詢
Next.js DevTools	Next.js 開發
Playwright	瀏覽器自動化
Supabase	資料庫操作

學習資源

資源	連結
Claude Code Docs	https://code.claude.com/docs
Anthropic Engineering Blog	https://anthropic.com/engineering
Prompt Engineering Guide	https://promptingguide.ai
MCP Servers Directory	https://mcp.so

Claude Code 進階功能

功能	說明
Hooks	事件驅動自動化（格式化、批准、注入）
LSP	語言伺服器整合（900x 效能提升）
Plugins	打包分享 Commands + Agents + Hooks

Claude Code Plugins

官方目錄

```
/plugin marketplace add anthropics/claude-code
```

社群目錄

- `claudebase/marketplace`
- `boostvolt/claude-code-lsps`（LSP 支援）
- `claude-plugins.dev`

回去可以做的事

1. 把今天的專案繼續完成
2. 建立自己的 CLAUDE.md 模板
3. 寫幾個常用的 Commands
4. 設定 Hooks 自動格式化
5. 裝幾個 MCP servers 試試
6. 嘗試 LSP 提升程式碼理解

建立你的 AI 工程環境

1. **CLAUDE.md** — 定義專案規範
2. **Commands** — 重複的 workflow
3. **Skills** — 自動套用的規則
4. **Hooks** — 事件驅動自動化
5. **MCP** — 擴充工具能力

這是你的競爭優勢

保持聯繫

有問題隨時問

Email: ym911216@gmail.com

謝謝大家

祝開發順利！

記得：你是工頭，AI 是工人