

VIBE CODING 方法论

—— SKILL 与 MCP SERVER 实战

LESSON 01 · 本节课

把「写代码 → 检查 → 验证」 闭环写成 Skill

Vibe coding 的正解 =

写得快 + 验得严

FAST TO WRITE – STRICT TO VERIFY. 而不是「随缘生成」。

“

Skill 是给 agent 的
实验操作规程（SOP）。

主讲人	待定
学期	2026 秋季
时长	45 分钟 · 含实验
前置	会用一种 agent CLI

课程地图

COURSE MAP

三节递进：方法论 →
能力 → 融合

01

写 Skill

方法论 · 闭环 SOP

本节

02

写 MCP Server

能力 · agent 的手脚

03

Skill × MCP 协同

融合 · 故障下不崩

课程地图：三节递进

方法论 → 能力 → 融合
每节课的作业都是下节课的输入

递进结构 · 正 → 反 → 合

课次	主题	一句话定位	实验
01	写 Skill 本节	把「写→查→验」固化成 agent 的 SOP	修复坏 skill（纯软件闭环）
02	写 MCP Server	把开发板 API 变成 agent 的手脚	补全 tools + 点灯
03	Skill × MCP 协同	纪律 + 手 = 可靠闭环，故障下不崩	协同改造 + 故障注入

DEFINITION OF WORK
「保证它能 work」的
两个可操作定义

1 换一个不知道内情的人，按 skill 也能复现闭环
→ 验收：跨组交换测试 **（本节）**

2 逆风局（故障注入）下行为可预期
→ 验收：混沌测试（第 3 节）

今日议程

七个环节
页码 = 本 deck 页码

页码	环节	内容
P04	开场演示	看 agent 按 skill 自动修完一个闭环
P05-06	概念与触发机制	skill 是什么 · description 决定触发 · 渐进式披露
P07-08	精读范例 Skill	五条设计原则 + 反模式清单
P09	怎么「用」	安装位置 · 触发确认 · 不触发时怎么调试
P10-12	实验一讲解	任务 · 验收 · rubric
P11 驻屏	实验一 动手	修复坏 skill + 跨组交换测试
P13	展示 + 小结 + 作业	完善本组 skill → 第 2 节课接上硬件

先看它跑一遍

现场演示 · 纯软件
不涉及硬件——硬件在第 2、3 节出场

演示脚本

LIVE DEMO · SCRIPT

按 skill 自动 · 全程无人插手

1 给 agent 一份带 bug 的 Python 练习

纯软件练习 · 零硬件依赖

2 跑 ruff 静态检查

查——`ruff check .`

3 跑 pytest → 发现 3 个失败

验——失败清单就是修复的输入

4 定位修复 → 重跑

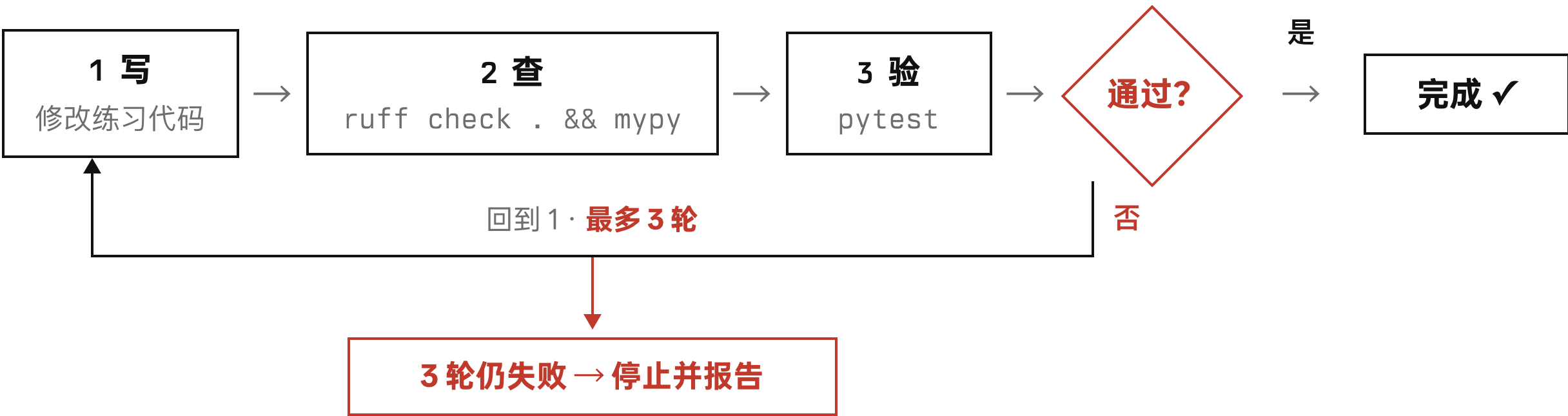
只改练习代码 · 不动测试

5 全绿 ✓

`3 passed · exit code 0`

agent 自动跑的闭环

WRITE → CHECK → VERIFY



—— 今天这节课要学的，就是把这张图固化成一份 SKILL.md

Skill = 给 agent 的 SOP

实验操作规程
不是知识文档

三个认识

WHAT A SKILL IS

1 一组按需加载的 Markdown 文件

SKILL.md (frontmatter + 正文) + references/ (细节按需读)

2 与 system prompt 的区别

system prompt 常驻占上下文; skill 由 description 触发、用时才加载 (渐进式披露)

3 放在哪

项目 .claude/skills/<name>/ (或所用 agent CLI 的对应目录)

在磁盘上长这样

ON-DISK LAYOUT · PROJECT ROOT

.claude/skills/

└─ firmware-vibe-workflow/

← 一个目录 = 一个 skill · 本课程的范例

└─ SKILL.md

← frontmatter: name + description = 触发; 正文 = 流程指令, 要短!

└─ references/pins.md

← 引脚表等细节, agent 按需读

description 决定触发

skill 不生效的第一大原因：
agent 根本没加载它

× 反例 —— 永不加载，或乱加载

反例

description: 帮助编写高质量代码

太泛、无场景词 → agent 无法判断何时用 → 永不加载或乱加载

✓ 正例 —— 一给场景就命中

正例

description: 修改或编写开发板固件/脚本时使用。[1]
执行 写代码→检查→验证 闭环，[2]直到硬件测试通过。[3]

[1] 场景 何时用 + [2] 动作 做什么 + [3] 成功标准 到什么程度

公式 好 description = 何时用 + 做什么 + 到什么程度

不触发时 · 调试三步

DEBUG · 按顺序

- 1 检查 description 是否含用户任务
说法的触发词
- 2 改写 description, 补场景词与动作
词
- 3 看 agent 回复是否引用 skill 内容, 没引用 = 没触发

范例 Skill 解剖

主稿：SKILL.MD · 15 行
边注：五条设计原则，按行号对位

.claude/skills/firmware-vibe-workflow/SKILL.md UTF-8 · MARKDOWN · 15 LINES

```
01 ---
02 name: firmware-vibe-workflow
03 description: 修改或编写开发板固件/脚本时使用。 — 1
04     执行 写代码→检查→验证 闭环，直到硬件测试通过。
05 ---
06 # 工作流（严格按顺序）
07 1. 写：修改 firmware/ 下代码，遵循 pins.md 引脚约定
08 2. 查：运行 `ruff check . && mypy firmware/` — 2
09 3. 验：运行 `make flash && pytest tests/test_hil.py`
10     成功标准：exit code 0 且输出含 "3 passed" — 3
11 4. 迭代：2-3 步失败回到 1，最多 3 轮 — 4
12 5. 停止：3 轮仍失败 → 停止，报告最后一次错误输出
13 # 禁止事项 — 5
14 ✗ 未跑 pytest 不得声称「应该可以了」
15 ✗ 不得改测试来「让它通过」
```

五条设计原则

FIVE PRINCIPLES · 边注对位

- 1

description 决定触发 L03-04

不写清「什么时候用」，agent 永远不会加载它。
- 2

验证命令可复制执行 L08-09

整行可直接粘进终端跑，不允许伪代码。
- 3

成功标准机器可判 L10

exit code / 期望输出，不接受「看起来对了」。
- 4

有限迭代 + 停止条件 L11-12

「保证 work」与死循环烧 token 的分界线。
- 5

负面清单堵捷径 L13-15

堵住 agent「改测试凑通过」的路。

五个反模式

对照自查
每一条都是真实的失败原因

NO.	反模式 · 症状	修复法	
01	description 太泛 agent 从不加载	补「修改固件时使用」类 场景词	× 反例
02	没有验证步骤 只有「写好代码」的愿望，agent 说「应该可以了」就收工	写 可复制执行 的验证命令 + 机器可判标准	× 反例
03	命令写成伪代码 agent 靠猜执行，次次报错	给能 直接粘进终端跑 的整行命令	× 反例
04	背景知识堆正文 正文又长又全，重点被淹没	细节移 references/ 按需加载	× 反例
05	无停止条件 失败无限重试烧 token	最多 N 轮 + 超限 停止并报告 最后一次错误	× 反例

会用：三步

写只是一半
会用才算 work

01

安装 INSTALL

放进项目 skill 目录：把 SKILL.md 移入所用 agent CLI 的对应目录。

```
$ mkdir -p \
.claude/skills/firmware-vibe-workflow
```

02

触发 TRIGGER

给 agent 一个**匹配场景**的任务，观察回复是否引用 skill 内容、声明按流程执行。

```
> 修改 firmware/ 里的点灯脚本
```

不触发 → 回到调试三步：改 **description** 补触发词

03

观察闭环执行 OBSERVE

盯住三个检查点，一个都不能少：

- ✓ 真的运行了 skill 里的验证命令吗
看命令记录
- ✓ 用标准判了成败吗
exit code / 期望输出
- ✓ 超过轮次上限时，停下来报告了吗

判据 「会用」的判据：你能说出 agent 每一步对应 skill 里的哪一条

实验一：修复坏 Skill

纯软件 · 零硬件依赖
12 分钟 + 跨组验收

场景 · SCENARIO

scaffold 里有一份故意写坏的 skill，和一份埋了 bug 的 Python 练习——模拟开发板 LED 控制器。

任务 · TASK

修好 skill，让 agent——而不是你手工——走完「写 → 查 → 验」闭环，把代码修到全绿。

验收方式 · CROSS-CHECK

跨组交换：A 组的 skill 拿到 B 组环境能复现闭环，才算「work」。

初始状态
BEFORE

2 处 ruff 报警 + 3 个 pytest failed · 恰好

验收标准
AFTER

0 exit code —— make check 全绿

实验材料

SCAFFOLD · 目录树

```
lesson1-skill-lab/
├── README.md 实验讲义
├── skills/firmware-vibe-workflow/
│   └── SKILL.md 坏 skill · 修复对象
│
│   提示 ✖ description 太泛 ✖ 无验证命令
│       ✖ 无停止条件
│
├── exercise/
│   └── board_sim.py
│       模拟开发板 · 内存态
│
├── led_controller.py 埋 bug
│   └── tests/
│       └── test_led_controller.py
├── Makefile
│   make check = lint + test
└── requirements.txt ruff · pytest
```

两个任务

先跑通，再交换验证
交换通过 = 真的 work

01

约 8'

修复闭环 FIX THE LOOP

改 skills/firmware-vibe-workflow/SKILL.md 的三处：

- + **description** 补触发场景
- + 写可复制执行的验证命令 **make check**
- + 加迭代上限与停止条件

然后 —— 让 agent 对 **exercise/** 走完「写 → 查 → 验」闭环，直到 **make check 全绿**。

提示 —— **五条原则**就是修复 checklist。 | **✗ 禁止改测试文件凑通过。**

02

约 4'

跨组交换测试 SWAP TEST

A 组把修好的 skill 交给 B 组环境（作者**不口头解释**）→ B 组的 agent 能复现闭环 = **通过**

这是「保证它能 work」的第一次落地 —— 换一个不知道内情的人也能跑通。

TIP · 卡住时 先看 README 讲义的 **FAQ**，再举手。

评分细则见下页 →

怎么评分

100 分制
skill 质量与闭环各占四成

评分项	验收标准	分值
闭环跑通	make check 全绿, agent 自主完成「写 → 查 → 验」	40
触发词	description 含场景 + 动作	10
验证命令可执行	整行能粘进终端跑	10
机器可判标准	exit code / 期望输出	10
停止条件	迭代上限 + 超限报告	10
跨组交换测试通过	别人也能复现	20

合计 TOTAL

100

前五项当场机器可判自查 · 最后一项由另一组的环境裁决

验收定义

WHAT 「WORK」 MEANS

OPERATIONAL DEFINITION

「保证它能 work」 =
换一个不知道内情的人，
按 skill 也能复现闭环。

闭环	40
skill 质量	40
交换测试	20
合计	100

红色 20 = 这条定义的直接验收

带走的：一个公式，五条原则

下节课：给 agent 装上手
MCP SERVER

TAKEAWAY · 核心主张

Vibe coding 的正解 =

写得快 + 验得严

FAST TO WRITE - STRICT TO VERIFY. 而不是「随缘生成」。

- 1 description 决定触发
- 2 验证命令可复制执行
- 3 成功标准机器可判
- 4 有限迭代 + 停止条件
- 5 负面清单堵捷径

今天对「work」的定义 = 可交换复现。

NEXT · LESSON 02 写 MCP Server —— 把开发板 API 变成 agent 的手脚
FastMCP 20 行 · 补全 pwm_set / adc_read · 点亮第一颗 LED

作业

HOMEWORK · 下节课前完成

- 1 完善你们组的 skill
下节课接硬件 —— 验证步骤将换成：
make flash && pytest tests/test_hil.py
- 2 安装第二节课 scaffold
按安装说明装好后跑 make doctor 自检（模拟器 + 半成品 MCP server）。

两份作业都是下节课的输入 —— 全程迭代同一份作品。

Skill 装到哪：非 .claude/ 速查

8 个 harness · 项目级为主
各家官方文档核实 · 2026-10

HARNESS	项目级目录（当前文件夹）	备注 / 用户级
Claude Code 基线	.claude/skills/<name>/SKILL.md	用户级 ~/.claude/skills/
Codex	.agents/skills/<name>/SKILL.md	从 CWD 向上扫到 repo 根 · 用户级 ~/.agents/skills/
OpenCode	.opencode/skills/<name>/SKILL.md	兼读 .claude/ 与 .agents/ · 用户级 ~/.config/opencode/skills/
Pi	.pi/skills/<name>/SKILL.md	或 .agents/skills/ · 用户级 ~/.pi/agent/skills/
WorkBuddy	.workbuddy/skills/<name>/SKILL.md	用户级 ~/.workbuddy/skills/ · 亦可 GUI 导入技能包
DeepSeek Harness dsh	.dsh/skills/<name>/SKILL.md	优先级最高；兼读 .agents/skills/ · 项目根 = 最近 .git 祖先 · 用户级 ~/.dsh/skills/
Qwen Code	.qwen/skills/<name>/SKILL.md	用户级 ~/.qwen/skills/
豆包	× 无项目级目录约定	PC 版「技能 → 我的技能 → 上传技能」；技能根在其 runtime workspace ~/.super_doubao/.../workspace/.skills/

SAFE DEFAULT 跨 agent 标准目录 .agents/skills/：Codex / OpenCode / Pi / dsh 都兼读——不确定装哪时，放这里一次覆盖四个 harness。