课程讲义

生成式软件工程 · 第一讲：欢迎来到未来

从历史的交叉路口到三个空间的 Gap

cnfjlhj & Pi

2026 年 9 月 7 日

来源类型：课程第一讲 / 讲座
作者/频道/节目：绿导师原谅你了（南京大学）
发布日期：2026-08-26
时长：约 100 分钟
来源链接：https://www.bilibili.com/video/BV1pb8o6yE8f

目录

1 站在历史的交叉路口

一个用 AI 生成的开场。老师按 F5，浏览器里播放出一份幻灯片。它的**源头是一份 Markdown 讲义**（他称为 source of truth），紫色部分是模型“添油加醋”扩写的，图片全部是 AIGC。左边那张 1975 年 Altair 8800 的图也是 AI 生成的，“有非常明显的 AI 味”。这第一屏就在示范他后面要讲的 **content as code** 工作方式。

主线：把昂贵的能力变成日用品。老师把计算机发展史压缩成一条反复出现的规律——每一次重大变革，本质上都是“把某种昂贵的能力变成日用品”：

- **1975 年，第一台 PC（Altair 8800）**：计算能力从机房走向桌面。
- **互联网**：连接所有人。他回忆小学快毕业时装了 ICQ（腾讯山寨版还叫 OICQ，后改名 QQ），能随机加到地球另一端的 native speaker。
- **移动互联网**：“一夜之间，所有菜场卖菜的人都有支付宝和微信两个二维码。”
- **2022 年，ChatGPT 与 AI**：人类历史上增长最快的产品。

核心判断

我们正站在“人类命运的交叉路口”。在这样的背景下，**我们该学什么、怎么上课**，本身就是一个需要重新思考的问题。这也是这门“生成式软件工程”课之所以要开的理由。

老师强调上课风格是“松散的讨论和交流”，并会在课后把录像“重新切片、重新整理”。这第一讲本身是一次**现场表演 + 事后复盘**的实验。本讲义忠实复原这场“表演”的内容与逻辑，并纠正语音转写中的明显错字。

语音转写说明

本讲义基于自动语音识别（ASR）整理。讲者带口音，转录有大量同音错字，例如“操系统”应为“操作系统”、“大圆模型”应为“大语言模型”、“派”指 agent 工具 Pi、“agent.md”指 `AGENTS.md`、“QEmail”指 QEMU、“得办 13”指 Debian、“签问”指千问（Qwen）、“欧拉玛”指 Ollama、“人乐神话”指《人月神话》等。下文一律使用正确术语。

2 两种焦虑与这门课的回应

老师先讲**学生的两种焦虑**——这是整门课的问题意识。

第一种：“AI 替我上大学”。老师用豆包出题、学生用豆包做作业，“整个闭环里只有学生没有受到训练”。操作系统课曾用痛苦折磨换来提升，现在直接把作业链接丢进 Codex，甚至说“蒋老师操作系统课第八个实验帮我实现一下”就结束了。焦虑在于：每门课都用 AI 答题、都答对了，但“出来以后我被 AI 第一个取代了”。

第二种：听话学生的迷茫。延续中学“老师说什么就做什么”的习惯，每堂课认真听、每次作业认真做，毕业发现“你被大学骗了”——你学会的东西 AI 都会做。

招聘的痛点：无法区分的简历。老师展示一份“完美简历”：CCF 期刊论文、accepted、一堆实习奖项，全由 AI 生成，个人信息替换成 Unicode 方块脱敏。但这被他判为**零分**——没有 GitHub 链接、没有个人主页、没有 preprint。他收到过无数份这样的简历，“可能你们的简历多多少少也是长这个样子”。当所有简历都是“一大堆实习、奖项、论文、赛投”，竞争者已无法区分。

问题的矛头

“是什么能让你从这些简历里脱颖而出?”这不是焦虑的终点，而是这门课的起点。老师希望用这门 web coding 课**缓解焦虑**、**还原一种快乐**：在 web coding 中理解软件工程是怎么工作的，做出有趣的东西，让简历“写得更好看”。

3 课程的政策与考核哲学

禁止古法编程。学生“个个都是古法编程的好手”——因为要考试、保研、机试，平时在 LeetCode 刷题。但他认为“总有一天这个事情会改变”，从上而下传导不会那么快，所以他“非常想要撕开一个小的口子”。

Token 自费。他本想拉 token 赞助，看到选课人数后放弃。他有一个犀利观点：“如果你们手上没有付费的 token，同学应该立即退学”——没有 token 就没有 AI native 的训练量。但他承诺课上所有演示（除 slides 生成用更聪明的模型换质量外）都用 DeepSeek V4 Flash 这种“足够便宜但是智能相对有一些”的模型，并建议夜猫子利用 DeepSeek 的**优惠时段**（白天上课、晚上做作业正好是便宜时段），每人充 100 块钱。

通过 / 不通过考核。理由很实际：这个时代分数已经不那么重要，而且**无法制定公平的评判标准**——同一个任务，让更好的模型（如 Claude）做，就是比 DeepSeek V4 Flash 又快又好，没办法限制大家用什么模型。

建议的实践项目。实现一个**自己的电子邮件管理系统**：跑在宿舍常开的小电脑上，能回老师邮件。老师会用指定邮箱发邮件，学生回回来。这是不错的 vibe coding 尝试。

最终检验是简历，不是 GPA。老师承认现实：为了走到保研/找工作那一刻，你还需要不错的 GPA，还得 reward hacking、overfit 考试题目。但“最终检验你大学阶段的，是那一份交出去的简历——它能不能带你到下一个更高的平台”。

一个“恐怖”的设想

老师抛出思想实验：如果每个人电脑上都装一个监控，AI 就能评判你大学四年到底度过了怎样的四年。他随即承认这“非常非常恐怖”。这个设想不是建议，而是点出：**评判标准正在从“结果分数”滑向“过程与产出”**，而边界尚未被社会定义清楚。

4 Everything is Code：课程数字永生的构想

slides 是怎么生产出来的。老师的幻灯片工作流是：先写一份 **Markdown 讲义**作为 source of truth，预先想好每个时间点讲什么；用一个叫 `makeslides` 的 skill 让模型“添油加醋”扩写成 markdownslides；其中“AI 梗同学” skill 会**生成三个 subagent**、**赋予不同 personality**，在隔离上下文里各自干活，最后主 agent 回来汇总——“通过算力换智力”；他会换几个版本提示词，每个都重新生成，最后由**另一个不同模型盲评**哪个 slides 质量更高；所有图片都是生成的，在浏览器 F5 播放。

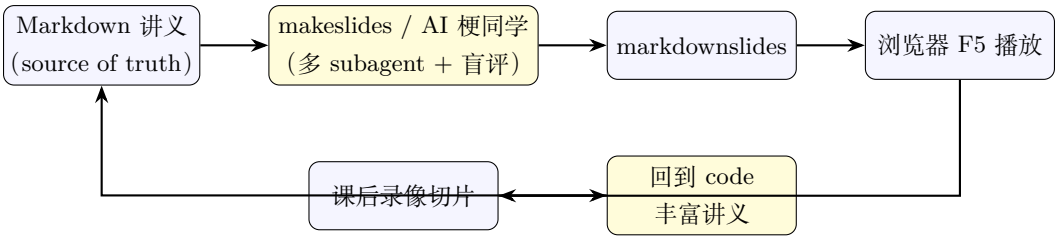

图 1: “Everything is Code” 工作流：Markdown 是唯一信源，其余皆由 AIGC 生成；课后录像切片回到 code，形成课程的 “数字永生”。重绘示意图，并非源画面。

上课是一场表演，课后要 review。老师把上课比作当演员：要调动情绪、按预先设定的剧本演出。但即兴是不完美的，所以课后会 review 录像—— “也许讲另外一个案例大家更能感同身受”。因为 everything is code，他可以把录像 “切片重新整理成片段”，回到 code 上把它变得更丰富。“我不就实现了数字永生了吗?”

愿景：杀死所有大学的教课方式。他希望把一门课的要点、逻辑、每一节怎么上，**沉淀成一个程序**。有一天他不维护了，可以把 B 站录像整理成切片，由全世界的人来维护、定制—— “就把所有的大学教课的方式给颠覆了”。他承认这是 “可能实现的路线”，会在本课做一次比较轻的尝试。

为什么这段值得高保真复原

这不是工具教程，而是**把教学本身当作可演化代码**的主张。关键不在于 “用 AI 做 slides”，而在于：信源唯一（Markdown）、生成物可重建、过程可回放（录像切片回到 code）、产物可被他人继续维护。这四点合起来，才是 “数字永生” 的真正含义。

5 大语言模型与 Agent 的本质

进入 “技术性内容” 前，老师先给一个**刻意压低门槛**的定义。

大语言模型是一个学到的概率分布。他说 “大语言模型就是一个概率分布，跟你们在概率统计课上学的抛硬币没有任何区别” ——已知抛了 6 次硬币加起来是 4，问第一次抛出 1 的概率，这就是条件概率；大语言模型做的是同一件事：给一段很长的文本，在结尾接上 “能不能帮我总结一下这本书”，然后**不断预测下一个 token 的概率**，每次输出一个字。模型很大，见过世界上所有东西，训练很难。但他强调：**这门课站在使用者视角**，打开网页 API 就能体验，不需要任何大语言模型的知识。

从预测 token 到 Agent。随着模型能输出越来越长的文本、接受越来越长的上下文，Agent 出现了：让模型输出一个**特殊的 token**，表示 “调用一个 Bash 程序”；Bash 的运行结果被重新送回到 Agent 的 loop 里，于是 Agent 可以自主运行——给它一个复杂任务，它可以 “不断地尝试、尝试、尝试”。

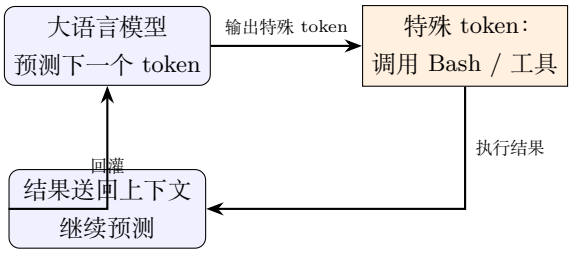

图 2: Agent 的自主循环：模型预测 token → 输出特殊 token 调用工具 → 工具执行结果回灌上下文 → 继续预测。重绘示意图，并非源画面。

一个被“按在地上打”的比方。老师回忆 2023 年 GPT-4 时代，让模型给五岁的孩子解释“扫描电子显微镜”。GPT-4 给了一个让他震惊的比方：你有一座看不见形状的山，闭着眼睛对它扔石头——有的方向石头穿过去了，有的方向石头弹回来了，你听到声音，就像 **ray tracing（光线追踪）**一样，通过哪些地方穿过去、哪些没有，推测出山的形状。他后来特意查资料，发现这个比方打得是对的。到了 GPT-5.6，因为带 reasoning，给了更严谨的解释。

未来远未定型。老师认为 next token prediction 其实是“效率很低的模型”，并提到若干正在被研究的方向：**Diffusion 类型的语言模型**（从随机噪声开始，一点一点生成很长的东西，“就像人做梦一样”）、甚至可以**回过头去改上下文**、把很长的上下文裁短。“摩尔定律还在”，整个 AI research 都在赌突破会发生，他无法预测哪条路线会胜出，但“这个世界肯定会变得非常不一样”。

6 Agent 实操：从装机到日常事务

老师切换到“普通人视角”，给出使用 agent 的几条原则，并用一串真实案例展开。

两条最小原则。第一，**现在就开始用**；第二，**随时问它**。在这个意义上，“是你的想象力限制了你的 agent 的上限”。他给 agent 配置了 `AGENTS.md`，问“你是谁”，agent 回答“我是生成式软件工程的教学辅助微臣”——之所以知道这门课的仓库，是因为 `AGENTS.md` 的配置（这部分是临时的）。

案例一：U 盘上的操作系统（“装机仙人被 AI 取代”）。老师上操作系统课带一个装了 Linux 的 U 盘，插教室电脑直接启动。他自嘲曾是“装机仙人”，2021 年知乎链接里各种定制系统。但现在只用一句话：**“你去 mirror.nju.edu.cn 上找一个 Debian 的镜像，把它装到我的 U 盘上，确保这个 U 盘可以启动，然后用模拟器验证。”**点睛之笔是“用模拟器（QEMU）验证”——装完后真的在 QEMU 里启动到登录界面，创建用户、设密码全部做好。接着又让 AI 装 tile-based 的窗口管理器，课堂上演示了丝滑的窗口操作（Super+Enter 开终端、Super+D 开应用启动器）。所有配置文档都是让 DeepSeek V4 Flash 生成的，“我没有看过任何一眼配置文件”。

学习方式的转变

老师把毕生修炼（大四时看 Intel 几千页 CPU specification）与今天对比：“有一个人可以把这些都看了、都懂、能做对，而我现在需要的是**解锁它的能力**。”结论是：**人现在唯一的功能，就是带着 AI 做 AI 现在做不到的事情；一旦你没有这个能力，你就立即已经被淘汰了**。

案例二：收发邮件与日常事务。老师有一个自己的 skill：数据库里存有他历史上回过的所有邮件，每当收到邮件就自动转已读、用他的口吻写好回复草稿。虽然大部分时候还得改，但他得到了 AI 的 bot。发票处理更具体：收到发票自动保存到指定归档位置；他可以在 bot 里加一个 todo“把我收到的上一个发票打印出来”，点勾就 commit，后台处理后通过 Apple Reminders → 手表推送通知他。他认为这比 Telegram 聊天更好，且 OpenClaude / Hermes 这类产品“不是个人 agent 的终

极形态，还可以做得更好”。

案例三：用 AI 读书、读论文、看 B 站。老师说自己“已经不看书了”，但为备课又买了几本，包括 Fred Brooks 的《人月神话》（Mythical Man-Month）和 John Ousterhout 的《A Philosophy of Software Design》。他读论文的方式是：让 AI **从人类已有的知识开始**，假设已有的一切都是已知的、trivial 的，再把这篇论文里**最少的新东西剥离出来**——一篇 12 页的论文就变得非常短。他的“AI 梗同学”流程：多个 personality 的 subagent、随机抽卡、讨论、分歧观点送回再讨论直到收敛，以此审稿。对 B 站视频，给一个 BV 号，AI 梗同学消化一个多小时的视频，去掉他已经清楚的 trivial 部分，“几分钟看完两个小时的视频”，并确信自己掌握了“里面不一样的东西”。

一个被反复强调的判断

“会用 agent 学习的人和不会用 agent 学习的人，学习效率差距是 10 倍 100 倍。”老师把这个判断接回开场的简历问题：**那个学习效率是你 100 倍的人，他的简历长什么样？**这是这节课埋下的第一个“更可怕的”悬念，留到下一节课。

案例四：现场修 USB 采集卡。直播时教室的 USB 视频采集卡断开了。老师立即让 Agent 修：一个 prompt 让 DeepSeek V4 Flash 看系统 USB 设备情况、找出设备、尝试 reconnect。它用“前所未见的 Linux 命令”以软件方式把采集卡“拔出来又插进去”——因为物理接口在机器后面碰不到。然后就好了。这是 Agent 在真实场景里“边反思、边试错、最终走通”的一个现场样本。

7 浏览器即平台：Chrome DevTools 与自动化

浏览器是软件工程奇迹。老师称浏览器“可以说是人类软件工程史上的奇迹之一”，代码量甚至远超操作系统内核，因为它要和用户打交道、有大量图形和显示相关的东西、性能优化等等。它有一个专为开发者设计的接口：**Chrome DevTools**（F12）。

Pre-AI 时代 vs AI 时代。以前开发者会从 console 里手动提取 HTTP 头、把请求的 curl 命令 copy 出来，带着 cookie 和状态去重放。但在 AI 时代，**普通人也能享受 Chrome DevTools 的厉害之处**。

一个 2020 年的疯狂项目。老师在 OpenJDK 上运行 Android App：Java/Kotlin 编译到 JVM，于是可以在本机跑 Android App；把界面上每个按钮的 bounding box 坐标和文本都算出来，但不走 Android 的渲染，**直接在浏览器里渲染**出那个计算器，每个按钮、每个字都对，可以点，点了状态就变。这是一个非常 powerful 的工具。

现场演示：让 Pi 操作浏览器自动填教学周历的 AI Slot。老师在 Pi 里说“帮我打开浏览器”，这个浏览器属于 Pi。他让 AI 填写教学周历的 AI Slot，但 DeepSeek V4 Flash **翻车了**——它不听指令，跑去读课程仓库的 lecture notes。老师吐槽：“它就是不听指令……翻车也是经常有的。”但它会不断反思“这件事合不合理”、找到正确方式，最终发现直接注入事件不行、点鼠标可以，于是走通并开始填 AI Slot。老师还指出：用一线模型会更好，他用 Flash 是因为课上更快。

从一次走通到确定性脚本

走通一次后，这个 session 里的路径可以**沉淀成确定性脚本**——它形成了记忆，知道哪条路径能成功、哪条会失败，把走过的网页变成脚本。“你就得到了一个自动选课抢课的系统。”这是从“Agent 偶然走通”到“可复用自动化”的关键跃迁。

生态：skill 互相组合。老师把这一切接回 UNIX 哲学：操作系统能成立靠的是管道，实现所有程序的自由组合和无限复用，1+1 可以大于 2。而浏览器 skill、记忆 skill、浏览 skill 可以互相调用

—— “现在这个 AI agent 的生态其实已经达到了一个临界点”。他描绘了一个**信息处理飞轮**：大量信息源（局压的 AI 早报 RSS、B 站推送、arXiv 推送、Hacker News 推送）进入处理系统，由各种自然语言 skill 构成、一行代码都不用写，但可以复用、不断调整工作流，产生有用的知识。

飞轮的可怕之处

“你想象一个拥有这样飞轮的人，和你这个没有飞轮上的人——这是一个多么可怕的事情。”这也解释了为什么 OpenClaude / Hermes 是现象级产品：它们给人惊喜。老师作为 CS professional 其实不惊讶，甚至对它总结 memory 的方式有些不满（会把不想记的记住、想记的不记），所以更多时候自己调用流程写到数据库里。但他承认：当模型过了某个临界点，他比 AI 多的那一点点也被取代以后，“以后我也被消灭了”。

8 CS 从业者视角：知道能力的边界

老师从“普通人视角”切到“CS 从业人员视角”。核心区别在于：一般人提需求，Agent 可能做不好、做不了，或不知道怎么让 Agent 确定性快速完成；**但 CS 的人知道编程的能力界限在哪里**——知道什么该以代码形式复用、什么该以 skill 形式复用，知道怎么把需求“编译”成可复用的东西。

改卷统分流水线。老师今年尝试了这样一条流水线来统分：

1. 办公桌上有**顶向下的摄像头**，拍到试卷的固定位置，调好后按一个回车就拍一张照。
2. 拍照后 offline 用**千问（Qwen）OCR** 识别姓名、学号、每个小分，写一行到文本文件里。一张卷子按一个回车，两手都闲不住。
3. 第二阶段按反序，左手按回车、右手拿笔。用**语音播报**学号和每个小分，他一边听一边检查——因为 OCR 会错，他要对每个同学负责。
4. **加法用脚本算**，不让大模型做加法（“我对所有同学是负责的”），每个分数他都看过。
5. 最终分数自动汇总到教务处的 Excel 模板上，上传后再**抽样检查**确保没问题才提交。

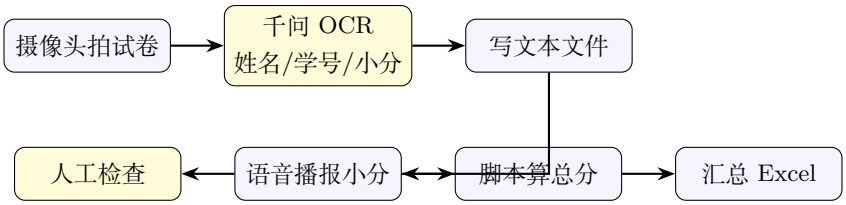

图 3: 改卷统分流水线：OCR 识别小分 → 语音播报人工复核 → 脚本算总分（不让大模型做加法）→ 汇总 Excel。重绘示意图，并非源画面。

为什么一般人做不到这么快收敛。老师指出：你哪怕跟最好的模型说“我改了 140 张卷子，小分都写上了，帮我统分”，也到不了这个收敛速度。关键是他知道本地有一个 27B 的千问 3.8、Ollama 可以用，知道用怎样一个最舒服的流程来减轻人和 Agent 之间同步的代价。“这些东西都是如果一般人，你可以试一下，到不了一个这么快的收敛方案。”他赌有一天 AI 能力强到能感知他的一切、帮他做完美的东西，但**现在还需要人和 AI 共建**。

9 QRGen：一个数据偷渡案例

在进入软件工程正题前，老师用一个“听起来像黑客故事”的案例做引子。

场景。假设你在一个金融公司，机器被管控：数据可以带进去（插 U 盘拷一个 Word/Excel 进去就行），但**数据出不来**——不能执行任何不信任程序、不能显示不信任网页，唯一能做的就是远处悄悄摄屏。你想把一个很大的文件偷出来，怎么办？

方案：二维码动画。老师做了一个程序：把任意文件内容（比如一串 Hello World）先用 Base64 编码，再转成二维码；文件变长，二维码就变成一个**循环播放的二维码动画**。只要偷偷摄屏，把每一帧都拍到，就把数据偷走了。关键约束是：全部用 **Excel 公式**实现，而且是 2019 之前兼容的公式——合法的 Excel 公式。

进阶：抖鼠标 + 远程桌面。如果能在机器里装 exe（一般不行），还可以用“抖鼠标”传输：上下维度抖动表示 0，左右维度抖动表示 1；用远程桌面连进去往里传数据，用二维码动画往外送数据；远程桌面本身可以变成虚拟网卡。“你是 Computer Science 的学生，你上过操作系统、上过计算机网络，你知道这里面需要的组件是什么，就可以把它 vibe coding 出来。”

为什么这是软件工程任务

老师强调：这**不是**一个炫技 demo，而是一个“典型的软件工程任务”——从一个明确的意图（偷数据），到一份一句话的规格（Base64 编码转 QR 动画、用 Excel 2019 公式），再到一个可运行的实现。它后面会直接用来暴露 AI 在“软件工程”上的真实短板。

10 三个空间：软件工程真正的 Gap

这是全课的理论核心。老师提出，当我们用 AI 解决问题时，存在三**个空间**：

- **Intention space（意图空间）**：你脑子里想的是什么。比如老板说“我要做一个国民级应用”“同学们点外卖不方便，我要做一个点外卖的小程序挣一个亿”。
- **Specification space（规格空间）**：程序的功能是什么、选用什么架构、有几个界面、每个界面什么功能、数据库 schema 怎么定、用 KV 数据库还是关系型数据库……每一个选择都带来不同结果。
- **Implementation space（实现空间）**：哪怕是满足同一个 spec，都有大量 implementation。

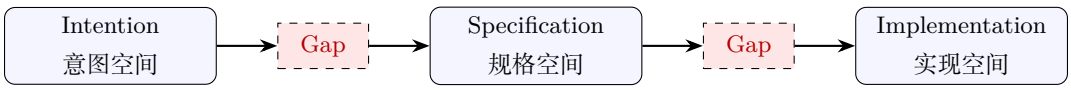

图 4: 软件工程的三个空间与它们之间的 Gap：意图 → 规格 → 实现，每一步翻译都可能出错。重绘示意图，并非源画面。

软件工程 = 补这些 Gap。老师引用一张经典梗图：程序员想的、产品经理想的、客户想的完全不一样；客户这样解释需求、project leader 这样理解、分析的人又这样想、宣传的人这样宣传，最后造出来的和客户真正要的差了十万八千里。“造成这个的根本原因，实际上就是这些 space 之间的这个 gap。”不管软件工程还是生成式软件工程，要做的就是**补上这些 gap**。

AI 是怎么补的？猜。老师指出，模型用的是“猜”——“你既然说了这句话，那我猜应该是这个吧”。因为后训练时模型有 reward，它被奖励“任务做对了、做完了、测试通过了”，于是形成一

种倾向：**疯狂地立即上手，尽可能快地把任务完成**，做得慢了要被惩罚。这和人解决问题的方式不一样。

11 AI 为什么会暴毙：品位与长程信号

QRGen 暴毙实验。老师做了一个受控实验：把 QRGen 的 specification 直接告诉 AI，看它会怎么做。

- **GPT-5.3（200K 上下文）**：第一次思考了 25 分钟，然后疯狂输出，写了超过 32K 的 output token，直接把 Claude Code 干爆（“已经超过默认 output 限制了”），被截断后重跑，**没有监管跑了 6 小时才跑完**，但最后是对的。
- **DeepSeek V4 Pro**：晚上跑，第二天早上看，给了一个**硬编码的 base64**——它用 Python 算了一个 base64 贴到公式里，明显不符合“用 Excel 公式编码任意输入字符串”的指令。

两者的共同问题：不可维护。虽然都完成了任务（Agent 完成任务的能力很强），但生成的都是“没有办法继续维护下去的代码”，和教务系统 `1.0.nju.edu.cn` 的死法完全一样——“软件工程里最经典的死法：上来我知道需求了，我把这个需求实现了就好了，直接就陷入了维护的泥潭。他们没有想一个好的软件架构。”

老师的正解：设计一个中间语言（DSL）。他只用少量 prompt，让 GPT-5.3 在 200K 上下文里完成第一个正确版本。做法是：让它先设计一个像编译器一样的基础设施——一个符号系统（XOR、concat 等位运算），这个中间语言**既对 Python 友好、又对 Excel 友好**。然后他只需验证：到 Excel 的翻译是正确的、到 Python 的结果是正确的；在 Python 世界里验证 Base64 编码算得对、二维码每个像素算得对；再单独验证每个东西和 Excel 公式翻译的正确性。“最后一把头生成就直接正确，代码非常干净，维护性远比（直接生成的）好。”

AI 容易在哪里犯错

“AI 容易在小 corner case 上犯错。GPT-5.6 第一秒钟就被带歪了——它希望做一个对于任意的 s 都满足正确性的东西，这已经错了。**软件工程的第一步：不要扩大，做最简单的东西、不会失败的东西**。它没有理解这一点。”这是“品位”问题，不是“能力”问题——模型有足够智能设计一个好的 DSL，但它的第一反应是像做题家一样直接上手。

《人月神话》的焦油坑。老师重温 Brooks 的经典论断：一个团队或软件产品已经开始失败了，**加进去更多的人只会让它变得更失败**。他把它类比到 AI：“当一个产品 AI 已经开始产生 slop 的时候，你投入更多的 token，也只会让它变得更失败——AI 会不断在它失败的东西上面再往上浮起来一点。”教务系统就是这样：每年都增加功能，学校又花了很多纳税人的钱。

教育体系的问题。老师指出一个深层问题：从高考过来，所有考试题都是“分钟级能解决的”，如果人五分钟解决不了，这题一般就不出了。长久以往，学生训练的就是“五分钟能不能解决”——这和 agent 训练的 reward 是一样的，而 agent 有更多数据、更明确反馈信号，所以做得比人好。**如果没有对“世界怎么运行、产品怎么开发”的好奇心与 self-motivation**，你就没有训练自己解决 long-horizon 问题，最后解决 long-horizon 问题还不如 agent。

软件工程本质是文科 / 管理学。老师认为软件工程有技术、也有人类学、社会学、管理学的部分；在生成式软件工程里，这些“人”被换成了 AI——你管一堆“不会和你抬杠、只会 you're absolutely right、会听你话”的 agent。软件工程要研究“这个人不听你话怎么办、离职了怎么办、跳槽到竞争对手怎么办”，现在这些都变成了对 AI 的风险管控。

失败的定义没变，但模型的短板变了

大语言模型时代，软件工程失败的定义没有变——从 intent 到 specification 到 implementation 的**翻译错了**（要么做出来的不是我要的，要么实现不满足 spec 的正确性/性能）。但今天，模型因为 reinforcement learning，对真正的长程信号 reward 做得不够好，更多还在“机械地完成任务”。所以不管是 GPT、Claude 还是 Gemini、DeepSeek，面对这类问题时都不会像老师那样“先设计一个 DSL”。**AI 现在最缺的是品位——好的工程品位**。

危机感。老师说他有危机感：“既然它能解决数学问题、能解决编程里很细碎的逻辑，它为什么不能从记忆库里找到那个好的设计来解决这个问题？我觉得是可以的。”他甚至有一些 skills 就在朝这个方向做。“我可能三个月以后，或者这门课还没结束的时候，我讲的所有东西都已经没用了。”

12 总结与延伸

两个结论。第一，**古法编程已经死了**——在完成一个单个任务上，AI 已经无人能敌。第二，**软件工程暂时还没有死**——因为 AI 还没有“品位”，还不会先做设计、还会陷入维护泥潭。但这不是永恒的。

这门课试图回答的问题。在保研的时候，导师问你：“为什么我要招你，而不是买一个 200 刀的 Codex?”你怎么接住这个问题？老师发现 GPT-5.6 回答得挺好：

- 能找到**值得解决的问题**；
- 把未知**拆成便宜的实验**；
- **用证据让坏方向及时停下**。

“但是很有意思，我发现 GPT-5.6 做不到。”——它能说出正确答案，但自己做不到。这正是这门课的张力所在：**知道该怎么做，和能做到，是两回事**。

课程产出。老师会把今天课程的内容重新切片、重新整理，但不一定马上——先会有一个“生肉版”（剪一下今天上课的流程发布），大家可以对比他精修过的版本。这本身又是“上课是表演、课后回到 code”的一次示范。

给读者的延伸思考。

1. 如果你的简历和所有竞争者的简历一样都是“一堆实习奖项论文”，那**那个学习效率是你 100 倍、且拥有信息处理飞轮的人**，他的简历长什么样？
2. 当模型连“品位”也学会以后，软件工程还会不会死？我们做一切的方式会发生什么变化？
3. 在 Intention → Specification → Implementation 的每一段 Gap 上，你能不能像老师设计 DSL 那样，**用最简单、不会失败的方式**补上它，而不是让 AI“猜”？

这三个问题没有标准答案，但它们正是这门“生成式软件工程”课想要陪伴大家去探索的。