超越**PingCAP**，打造你公司的**AI**原生开发团队

作者：瑞典马工 | 2026年1月25日

本文是对黄东旭（PingCAP CTO）最近的文章《Vibe Engineering 2026.1》的一个回应。如果你还没有读过他这篇文章，我强烈推荐你去读一遍。

黄东旭的核心观点：

- **AI**编程工具已经跨过临界点。去年12月开始，头部模型（GPT-5.2、Opus 4.5）在复杂任务处理上有了跳跃式进步，“100%无人工干预下完成长时间Agentic Loop成为可能”
- 顶级模型能写出比大多数工程师更好的代码。他用AI重写TiDB的PostgreSQL兼容层，代码质量”已经接近生产水平”
- 人的瓶颈在验收。他90%的时间花在评估AI的工作成果，而不是写代码
- **Token**消耗呈二八分布。用得最好的工程师消耗的token比剩下80%的人加起来还多，产出差距是10x vs 10%

他对未来软件公司形态有一个判断：

> 这种工作方式更像是头狼带着一群狼群（Agents），在一片自己的领地里面耕耘，但是同一片领地里很难容纳两匹头狼，会造成 1+1 < 2。 但是PingCAP是中国最优秀的基础软件公司之一，黄东旭是中国最优秀的程序员之一，PingCAP的这种实践有多大的通用性？如果你我直接抄袭他的做法，究竟是见贤思齐，还是东施效颦？本文将尽力回答这个问题。

头狼是精英路线

黄东旭的”头狼+狼群”模式，是一个公司内部的超级个体，让一个顶级工程师端到端负责产品。

七牛云也在尝试类似的方案，叫产品架构师：一个人负责PRD、架构、实现、测试的全流程。CEO 许式伟的评估标准很直接： > 如果一定时间内做不到端到端负责，就淘汰，换能做到的人来。

但是，产品架构师需要传统产品经理 + 架构师 + 主程序员 + 测试Team Leader的综合能力。能接近这个定位的人都是凤毛麟角。大多数公司的大多数团队，找不到这样的人。笔者运营一个世界一流的AI Coding社区：Agent管理学论坛，里面敢于说自己满足这个条件的，也就三个人。

即使找到了，你怎么知道他/她下个月不会去做一人公司？上面说的那三个人，都是自己开公司的 CEO，没有一个给别人打工的。

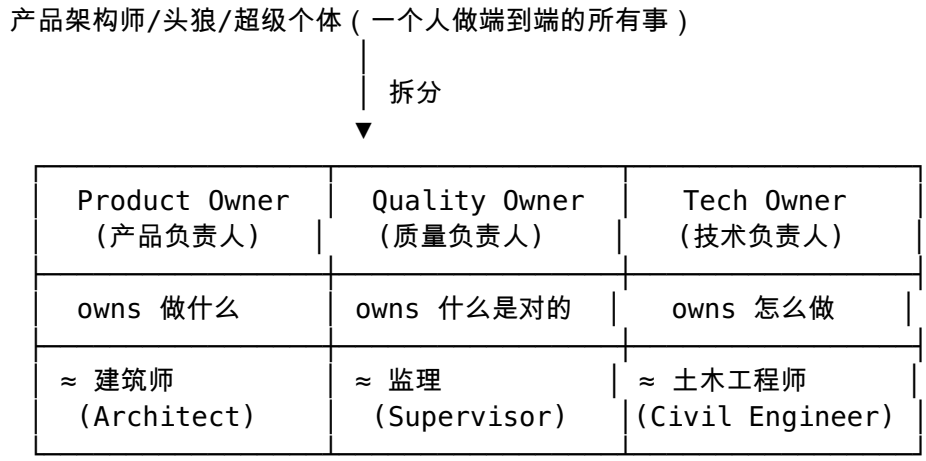
因此,这个精英路线的致命问题是它依赖一个几乎不可及的人才市场。

把超级个体拆成产品端到端三人组（**Product Tri-Ownership**框架）

在软件工程之外，建筑行业早已解决了类似的问题。他们不依赖超级个体，而是通过角色分工来保证质量。建筑师定义空间，土木工程师设计结构，监理独立验收，施工方负责执行。四个角色，四种专业，缺一不可。

我和我的朋友们借鉴这个模式，发明了产品端到端三人组框架（Product Tri-Ownership）。

PTO框架把超级个体的工作拆开：



细心的读者会问：施工方去哪了？

答案很直接：施工方就是AI Agent。它们按指令写代码、跑测试、生成文档，就像施工方按图纸砌墙。它们不配有署名权。

Product Owner：负责做什么

Tri-Ownership的PO是一个扩展的角色，合并了传统PM和PO的职责：

职责领域	具体内容
价值定义	产品愿景、商业价值判断、说No
需求管理	User Story、验收标准、优先级
版本规划	版本节奏、迭代计划、里程碑
干系人沟通	向上汇报、跨团队协调、客户沟通

这和建筑师的职责高度对应：建筑师不只是画图，他们也管理项目分期、与业主沟通、协调工程师和施工方。

PO对产出的价值负责（**accountable for outcomes**），而不是对产出本身负责。PO不是写需求文档的秘书，而是决定”什么值得做”的人。不仅定义”做什么”，更要有勇气说”不做什么”。没有合格的PO做第一层质控，后面的质量链条都会出问题。

Tech Owner：负责怎么做

TO对技术实现负责，他/她带领一群不同角色的AI Agent写出正确的软件。负责详细设计、代码review报告（不是代码本身）和工具选型，但最重要的职责是编排多个**Agent**的工作流。

黄东旭承认： > 当前的 Coding Agent 在面对单一模块复杂度超过大约 5 万行代码之后，就开始很难在 1-shot 里把问题一次性解决掉。 > Agent 通常不会主动去做项目结构和模块边界的治理。

下面是我团队的Tech Owner设计的一个AI Agent的工作流程

```
标准开发流程
├── story → 从Story生成详细设计, as a skill
├── tester → TDD红阶段(写失败测试), as a subagent
├── coder → TDD绿阶段(让测试通过), as a subagent
├── reviewer → 对抗式Review, as a subagent
├── qc → 质量检查+提交, as a skill
└── deployer → 部署, as a subagent
```

不同任务类型需要不同的工作流。Bug fix流程不同于新功能流程，绿地项目不同于棕地项目。Tech Owner负责选择和调整正确的工作流。

在此之外，Tech Owner负责解决过程中的任何异常。比如在我们的一个集成项目中，合作方没有提供API，TO不得不想尽办法去拼凑出一个openapi.yaml。

Quality Owner：负责做得对不对

QO对交付质量负责，不同于建筑行业的监理，QO要设计质量流程，全流程管理质量。

独立的QO解决了几个问题

卸载**Tech Owner**的职责

黄东旭承认： > 在我个人和 AI 开发项目的过程中，我 90% 的时间和精力都花在了这个阶段：也就是如何评估 AI 的工作成果。 > > There’s a test, there’s a feature，你只要知道如何评估和测试你要的东西，AI 就一定能把东西给你做出来。 > > 我在完成大目标前，我一定会先和 AI 一起做一个方便的集成测试框架，并提前准备好测试的基础设施。

这正是**QO**的工作。 黄东旭个人能力强，所以他扛了90%的验收工作，但是对于大多数人来说，这很容易成为瓶颈。

Tri-Ownership框架通过设置专职Quality Owner来卸载一部分工作。

通过对抗式测试保证质量

NASA在挑战者号灾难后建立了独立验证与确认（IV&V）项目，核心原则是软件验证必须由既不是开发者也不是采购方的独立组织执行。QO的定位完全一致。有一定AI Coding经验的朋友都知道，LLM很擅长投机取巧。如果多次无法通过一个测试用例，它很可能注释掉测试用例以满足用户期望。独立的QO和他的agents能够在一定程度上阻止TO的agents这样作弊。

全流程的质量控制

不同于传统的测试人员，QO需要参与SDLC的每个环节，并且设计多层次的测试体系（单元测试 → 集成测试 → E2E测试）嵌入到每个环节。 比如，Product Owner在提供产品需求说明书的时候，QO就会参与验收标准（Acceptance Criteria）的编写，确保需求是可验证的。 又比如，在我项目中，我们选择Makefile作为golang项目的测试入口，禁止Claude Code自行调用go test命令。并且把make test嵌入到git commit hook和CI workflow中，确保低级错误在早期得到纠正。 头狼模式把所有质量压力集中在最后验收（黄东旭说的90%），这极度依赖于头狼的个人素质。Tri-Ownership框架通过覆盖SDLC全流程的测试体系，把质量工作的门槛降低了。

每天完成一个用户故事

就节奏来说，Tri-Ownership框架应该能做到一天完成一个**Story**。

这个速度有多快？看看Claude Code的发布节奏：10天内发布了10个版本，有些天甚至发布2-3个版本。不能满足这个速度的团队，是低于业界平均水平的。

正常的节奏应该是： - 上午：PO写完User Story，QO同步定义验收标准 - 中午：Tech Owner完成详细设计，触发AI开发流程 - 下午：AI完成代码+测试，Tech Owner审核Code Review报告 - 傍晚：Github Action自动触发E2E验收，通过即合并上线

为什么能这么快？ - PO和QO同时工作，不是串行等待 - Tech Owner review设计而不是代码，AI处理实现细节 - QO提前准备好测试框架，验收不需要临时搭建 - 自动化工作流减少人工交接的等待时间

关键：三个**Owner**必须坐在一起。 物理上坐在一起，有问题随时讨论，不需要预约会议。每日站会不够，迭代速度快意味着决策频率高，等到明天站会就太晚了。

如果一个Story超过一天还没完成，说明Story太大，需要拆分。AI时代的Story应该比传统开发小得多。

传统实践成为瓶颈。

很多团队引入AI后速度没有提升，因为传统流程拖后腿： - **4-eyes code review**：每行代码必须两人审核。当AI一天能写完一个功能，等两个人排期review就要三天 - **Change Advisory Board**：每次上线都要开会审批。AI可以一天部署十次，CAB一周才开一次 - 手工部署：AI写完代码还要等运维排期手工部署。AI可以一天迭代十次，手工部署一周才排一次

Tri-Ownership的解决方案：

- Code review → AI做第一轮筛查，Tech Owner只看报告
- CAB → QO的E2E测试通过即可上线，出问题一键回滚
- 手工部署 → CI/CD自动化，QO的E2E测试通过即触发部署

支撑角色

Tri-Ownership之外，还需要一些支撑角色：

Sponsor（决策者）：提供资源，解决冲突。首先要保证充足的token预算和合适的硬件，因为AI原生开发的瓶颈往往不是人力成本，而是token成本。其次，当三个Owner之间发生分歧时，Sponsor做最终决策。他应该是产品线负责人或技术VP，不参与日常工作，但是关键时刻要有决策权限。

Architect（架构师）：定义系统架构、模块边界、接口契约、代码规范、技术栈选型（语言、框架、数据库等）。黄东旭说Agent在5万行以上的模块就很难一次搞定，Architect预先拆分，让每个模块保持在AI可处理的规模。例如，我们团队使用mono repo，每个应用天然继承Architect选定的Gin、Gorm、Observability等技术栈。小项目可由Tech Owner兼任。

Platform Engineer（平台工程师）：负责生产环境运维、监控告警、安全合规、成本优化。CI/CD pipeline由QO负责，但生产环境的稳定性、可观测性、灾备方案是Platform Engineer的领域。

Specialty Expert（领域专家）：按需咨询，不参与日常开发。比如我的朋友Nick在开发一个To C服务的时候，需要一个UI/UX设计师。他不懂代码，但可以跟Lovable这样的AI agent协作，把UI Kit做出来交给开发。Security、DBA等领域专家也是类似的模式：需要的时候介入，不需要全职配备。

另一种方法：培训模式

不是所有公司都能重组团队结构。我的朋友杨工使用了一个更温和的替代方案：培训模式。

- 不动组织结构
- 他定制skills/agents/workflows
- 实施团队按流程执行
- 用便宜的模型降低成本门槛

优点：门槛低，不需要组织变革，Team leader就可以实施。缺点也很明显，这个模式只覆盖SDLC的很小一部分，产生的价值被限制在一个不高的天花板下。

AI Coding的下一个核心问题

个人与AI结对编程已经是成熟的实践。Claude Code、Cursor等工具的普及证明了这一点。但是，团队如何与AI协作？这是一个尚未解决的问题。个人与AI协作时，责任边界天然清晰。团队引入AI后，传统的角色分工和责任归属都需要重新定义。

AI Coding的下一个核心问题，是在AI agents极大地降低实现成本之后，如何系统性地构建人机协作团队，以可持续的方式保证交付质量，并且将生产力的提升转换为市场竞争力。

黄东旭的头狼模式证明了AI原生开发的可行性。但头狼太稀缺。

Product Tri-Ownership的价值：提供一个可复制的框架，把”不可能的超级个体”拆成”三个可培养的专业角色”，让普通团队也能实现AI原生开发。

你们公司的AI原生团队是什么结构？遇到了什么问题？欢迎留言讨论。

引用来源

- 黄东旭《Vibe Engineering 2026.1》：https://www.infoq.cn/article/k05gRzGFhz4QerCz4ARl
- NASA IV&V Program: https://www.nasa.gov/ivv-overview/