thoughtworks

让混乱消散

AI赋能的软件研发熵减实践

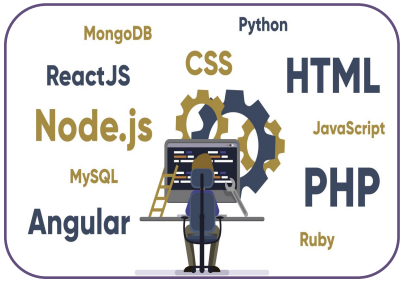
张思楚

May 2025

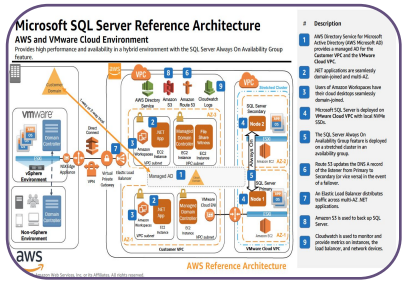
讲师简介

张思楚
Technical Principal

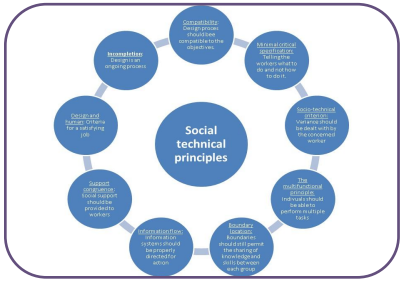
海外大项目的离岸交付中心技术负责人，全栈工程师、畅销Web产品SpreadWeb架构师，3项Web专利技术发明人。目前专注于提升软件交付效能，致力于持续集成和持续交付的有效实施，推动适宜的软件效能度量体系的落地。擅长运用 AI 技术赋能软件开发流程，具备深厚的技术背景，能够引导团队实现AI智能化的开发与运维流程，提升团队整体协作效率和软件质量。



Full Stack Developer



Architect



Technical Principal

AI 赋能软件研发的成果惊艳，让我们对AI充满憧憬

许多企业和团队持续投入资源，期待通过AI赋能软件研发，开启未来新篇章

在当前的技术发展趋势下，AI技术正以惊人的速度改 变着软件研发的方式。AI赋能软件研发不仅带来了新的机遇，也带来了一系列需要深入思考的 问题。引入 AI 通常被视为增加了开 发过程的无序度，可能 导致“熵增”现象。然而当AI应用于各个研 发阶段时，借助于AI的能力，带来了熵减。那么 总体来说具体是熵增还是熵减？是否提升了整体的效率和 质量？本话题意旨在讨论 AI 赋能软件研发过程中“熵增”与“熵减”的应对策略，提出一些思考和 见解，以此促 进AI技术在软件研发领域的发展。

软件熵

分析体系

引入熵

三种模式

赋能熵

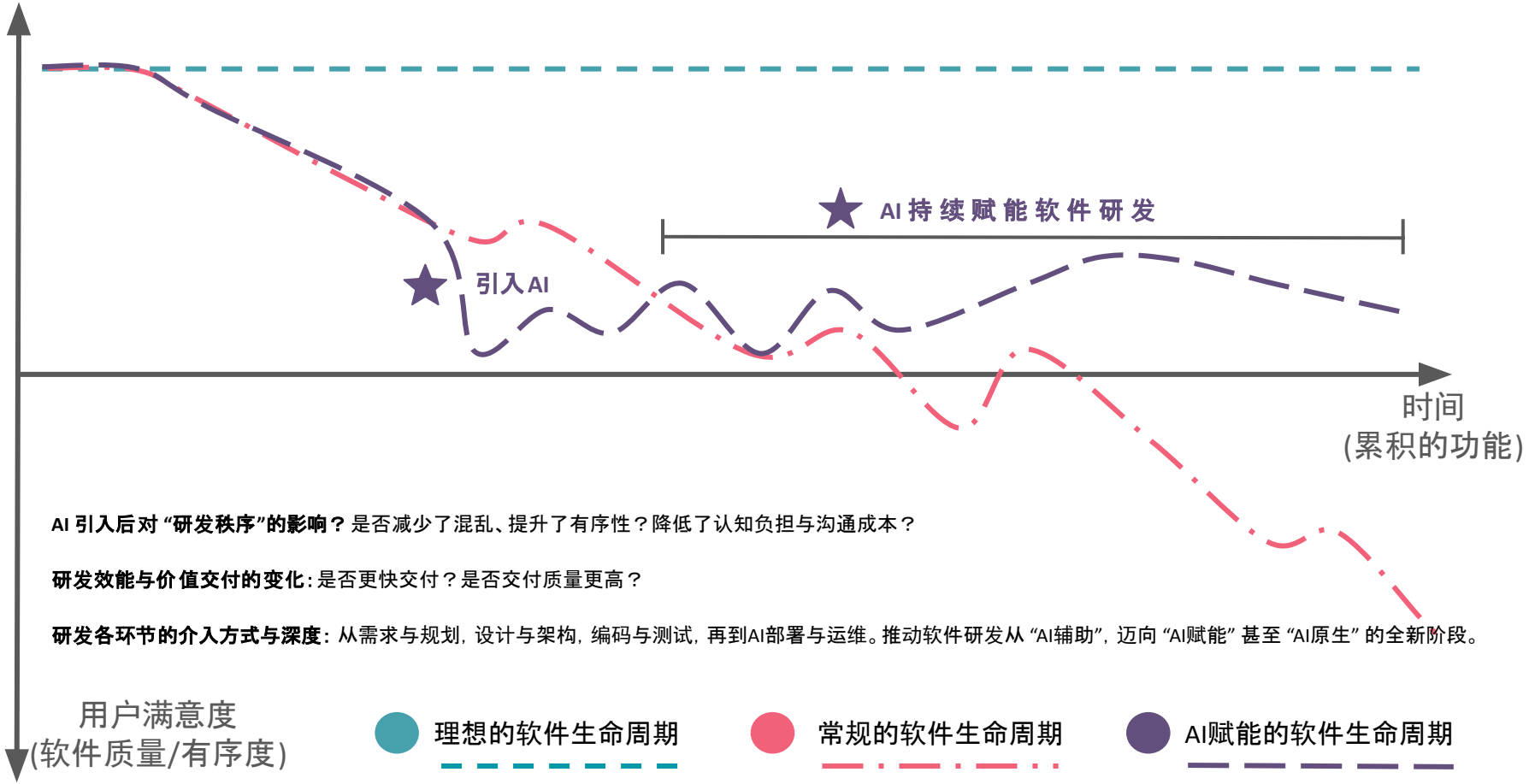
多个阶段

治理熵

以人为本

在这个AI 爆炸的时代，当我 们讨论AI 赋能软件研发这个话题时，我们在讨论什么？和我有什么关系？我 为什么要参与和关注，能 给我带来什么？

当我们讨论AI赋能软件研发这个话题时，我们在讨论什么？



AI引入后对“研发秩序”的影响？是否减少了混乱、提升了有序性？降低了认知负担与沟通成本？

研发效能与价值交付的变化：是否更快交付？是否交付质量更高？

研发各环节的介入方式与深度：从需求与规划，设计与架构，编码与测试，再到AI部署与运维。推动软件研发从“AI辅助”，迈向“AI赋能”甚至“AI原生”的全新阶段。

因为AI的介入，是带来了混乱，还是赋能并加速了软件研发？

德鲁克、戴明 ：If You Can’t Measure It, You Can’t Improve It.

开尔文（英国物理学家）：当你能 够量化你谈论的事物，并且能用数字描述它 时，你对它就确实有了深入了解。

但如果你不能用数字描述，那么你的 头脑根本就没有 跃升到科学思考的状 态。

“经验”未必可靠，你能看到“真相、本质”吗？

抓手 抓手 抓手

熵

If You Can’t Measure It, You Can’t Improve It 不等价于 **If You Can Measure It, You Can Improve It**

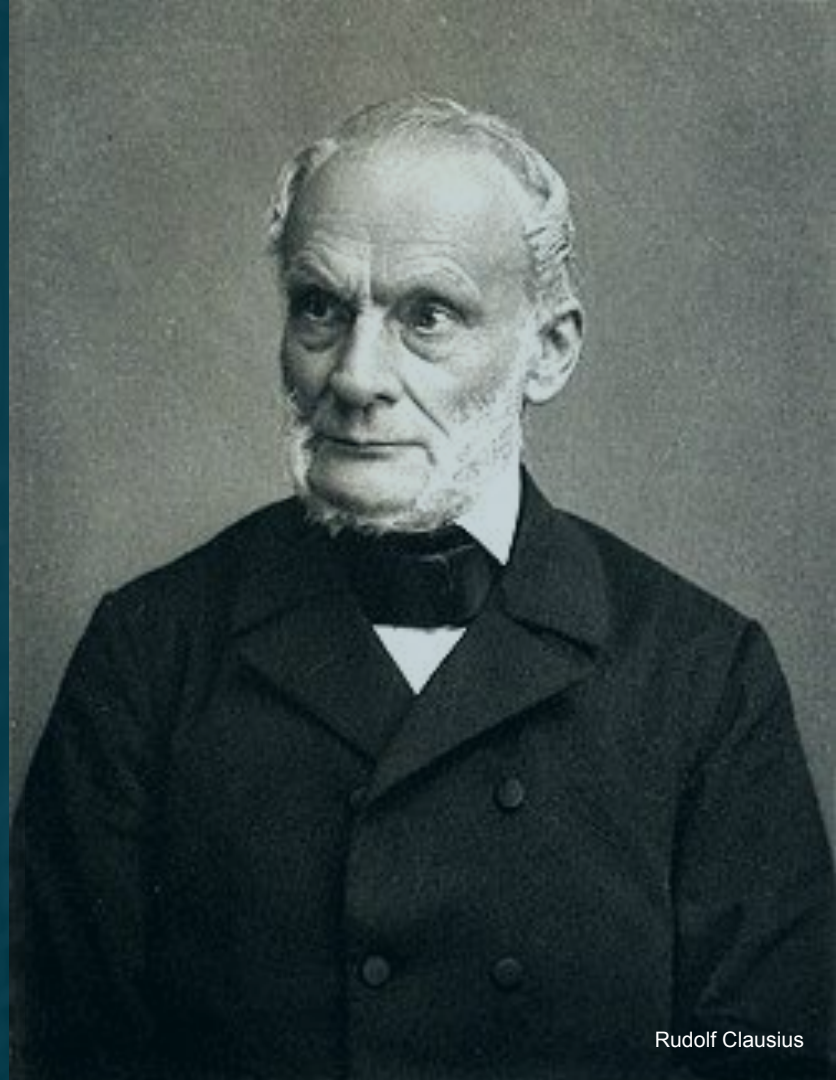
什么是熵

熵 - 根据热力学第二定律描述

- 系统 **无序性的量度**

熵的特点

- 系统本质上 **趋向于无序**
- 无序性越强，可用于 **做功的能量就越少**



Rudolf Clausius

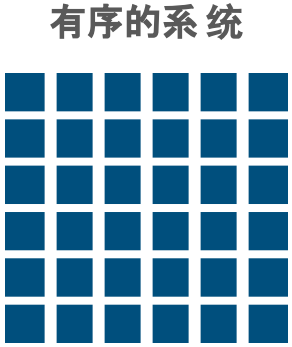
“对于大多数软件系统来说，随着时间的推移，添加新功能 变得越来越困 难。”

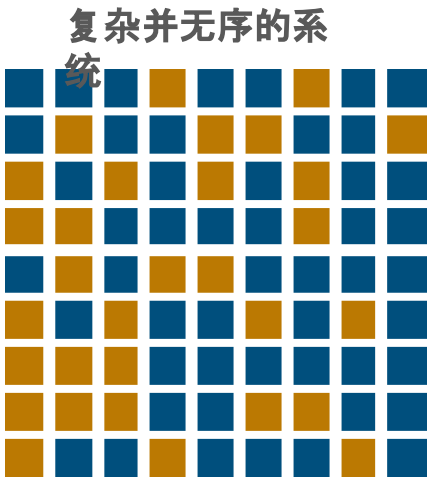
Martin Fowler, 2019

- **Lehman的软件演化定律** 指出，任何持续使用的软件系统都需要不断更新以满足快速变化的需求，否则其性能和可维护性会逐渐退化，这种现象也被称为“**软件衰变**”。

- 如果衰变不施加有效的重构与治理，系统的复杂度（即“熵”）就会不可逆地增加，这一现象与热力学中的熵增定律异曲同工。

- 研究表明，信息熵可用于评估软件架构，引入一致性规则能显著降低系统熵值，提升软件架构的有序性和信息含量，证实了**软件熵的可量化性**。





软件熵可量化的指标体系

因为AI的介入，是带来了混乱，还是赋能并加速了软件研发？

需求与规划	设计与架构	编码与测试	部署与运维
周需求变更次数 统计在一周内提出并批准的需求变更数	**模块依赖膨胀度** 模块之间互相调用或引用的关系数量随时间的变化率	**代码重复率** 项目中重复代码行数与总代码行数之比	**部署失败率** 自动化或手动部署操作中失败的比例
未确认需求比例 待定需求数占总需求数的百分比	**架构文档同步延迟** 架构更改后文档更新的平均时长	**代码审查(Code Review)时长** 一个周期内多个代码审查所需的平均时长	**平均修复时长（MTTR）** 从故障发生到恢复服务的平均时长
优先级分歧程度 衡量不同利益方对同一需求优先级评分的差异程度	**设计变更回退率** 因设计缺陷导致的重构或回退次数占比	**测试用例覆盖率** 已执行的测试用例（如需求项、功能点等）与总需求用例数之比	**告警噪声比** 无用或重复告警与总告警数的比值
	接口稳定性指数 衡量公共接口在版本迭代中的变动频率	**缺陷漏检率** 测试阶段未发现但在生产环境中出现的缺陷占总缺陷的比例	**知识库利用率** 故障排查时团队从知识库检索获得解决方案的次数与总排查次数的比值

构建度量基线 减少对猜测和直觉的依赖

AI介入的三种模式及 熵增治理

AI辅助
(AI-Assisted)

在既有流程中嵌入 AI 建议，由人主导决策与执行，侧重对特定岗位或环节的效率增强。

AI赋能
（AI-Empowered）

深度融合 AI 与主流程，AI 驱动从需求到交付的端到端 闭环，注重跨环节协同与持续优化。

AI原生
（AI-Native）

从架构设计起即将 AI 能力作为核心，自优化，自愈与自 监控成为系统全生命周期的常 态化能力。

典型熵增来源

- **工具学 习与上下文切 换成本**
- **模型协作和上下文共享困 难**
- **内部数据依 赖和安全风险**
- **模型训练与治理流程的管理开 销**
- **基础设施与数据管道建 设成本**
- **组织变革阻力**

通过在常用IDE或平台中集成AI辅助功能，配合渐进式引导、设定能力构建路径，最佳实践和案例推广，Social Learning，降低学习曲线，实现开箱即用。

抽象统一AI服务适接口，使用 MCP，降低开发成本并、准确地共享上下文信息，实现协同推理和响应。

构建标准化 MLOps 和 LLMOps 平台，利用流水线、CI/CD、模型注册表和监控优化资源，以降低模型训练与治理的管理开销。

通过契约、抽象、Schema Registry、数据血缘、合成/脱敏数据等措施，隔离上游变化，保障数据依赖安全合规。

优先采用托管云服务降本，通过IaC模板化管道、自动伸缩和分层存储实现ETL、按需释放，结合成本可视化与预算警戒确保高效稳定。

制定企业AI战略和治理，建立变革愿景，组建跨职能团队统筹技术与流程变革，配合培训、持续沟通并激励，实现组织、技术、流程协同进化。

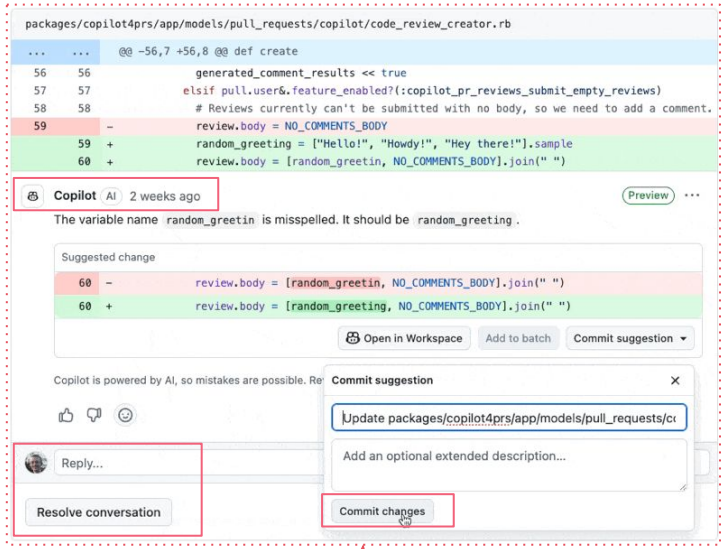
案例剖析

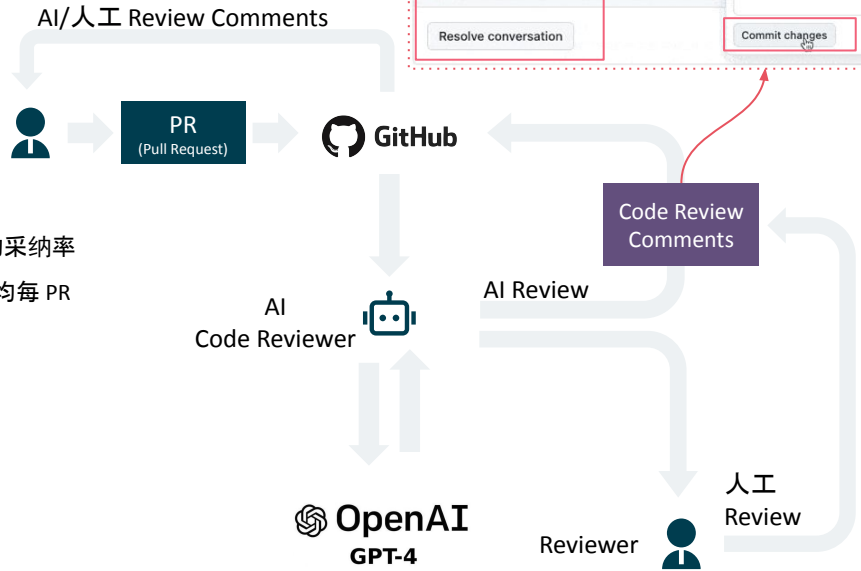
AI 辅助 - 代码审查(Code Review)

引入AI的熵增来源 自动化代码 Review，但是需要在PR (Pull Request) 里添 Reviewer，在流程中引入新的 Review 方式，需要增加 Review 规则和模版的制定。

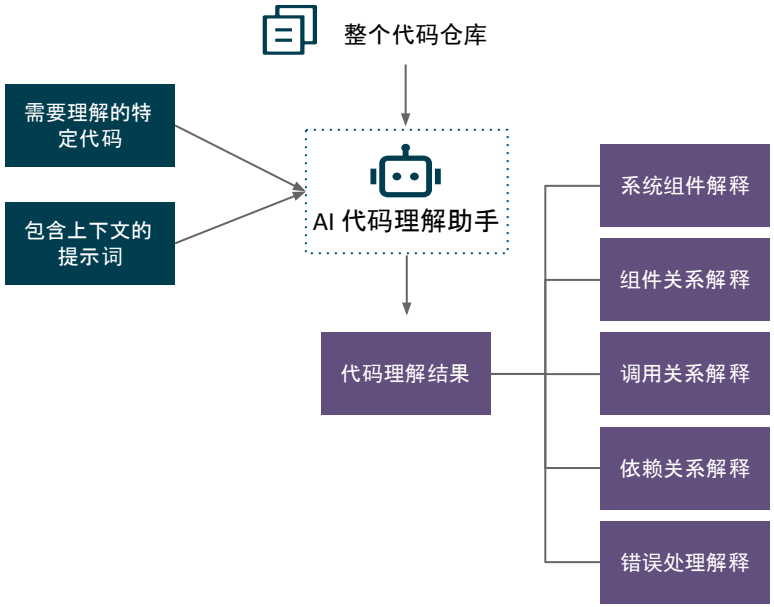
度量指标 代码重复率 | 代码审查(Code Review)时长

熵减实践及成果 重复代码在审查过程中更快速的被识别，审查建议的采纳率达70%，PR 平均审查时长从 48 小时降至 24 小时以内。安全合规扫描平均每 PR 触发的修复次数减少 40%。





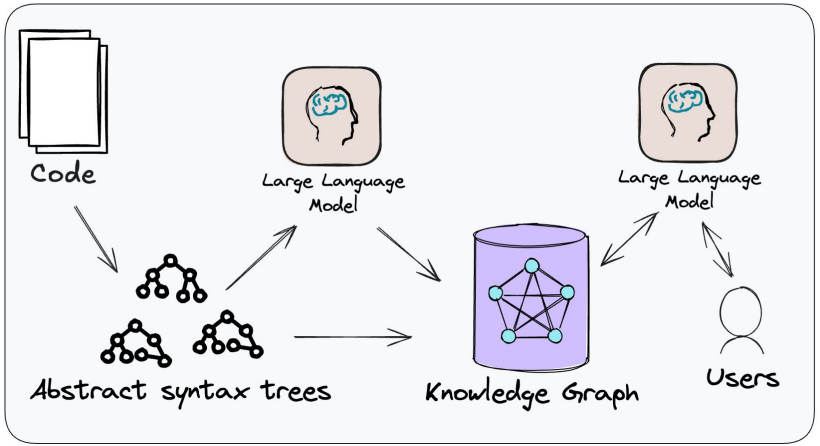
AI 辅助 - 理解遗留老代码



引入AI的熵增来源 团队将 Github Copilot 嵌入日常开发环境，利用其提示词和代码解释能力，针对无文档、难理解的遗留模块进行系统化解读与注释，但是提示词需要具体且明确，不同开发者对同一模块使用各自提示词，生成的注释风格不一致，增加了文档与代码一致性校对成本，在采纳AI 建议前，往往需要几轮校对与修正，延长了理解时间。

度量指标 架构文档同步延迟 设计变更回退率

熵减实践及成果 团队在理解和维护遗留代码的效率上提升了约 3 倍，开发者将原本需数小时的研读时间，缩短至数十分钟，并显著降低了因代码误解带来的设计缺陷风险，减少了文档更新耗时，保证了设计一致性和准确性，同时团队也使用这个功能作为收集业务信息的有效方式。

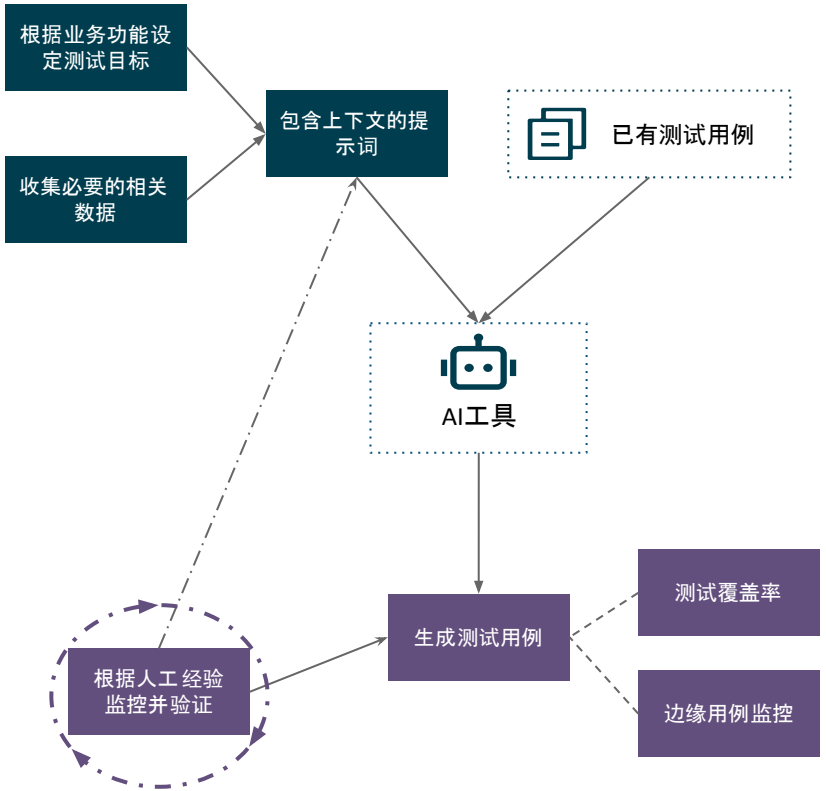


AI 辅助 - 编写测试用例(Test Case)

引入AI的熵增来源 根据业务需求上下文通过提示词和 ChatGPT LLM 交互生成 Test Case，需要有良好的提示词能力，把需要生成的TestCase表达清楚，提示词需要具体且明确(如：性能、等价类、边界值、TestCase模版 等需求)，同时提供良好的业务上下文 (用户故事UserStory)，有时需要处理生成的重复测试用例。

度量指标 测试用例覆盖率 缺陷漏检率

熵减实践及成果 测试用例覆盖率从 70% 提升到了 90%，编写时间缩短了 90%（从2个小时到10分钟），平均一个故事卡 (User Story) 可多识别一个到两个边缘用例 (edge case)。

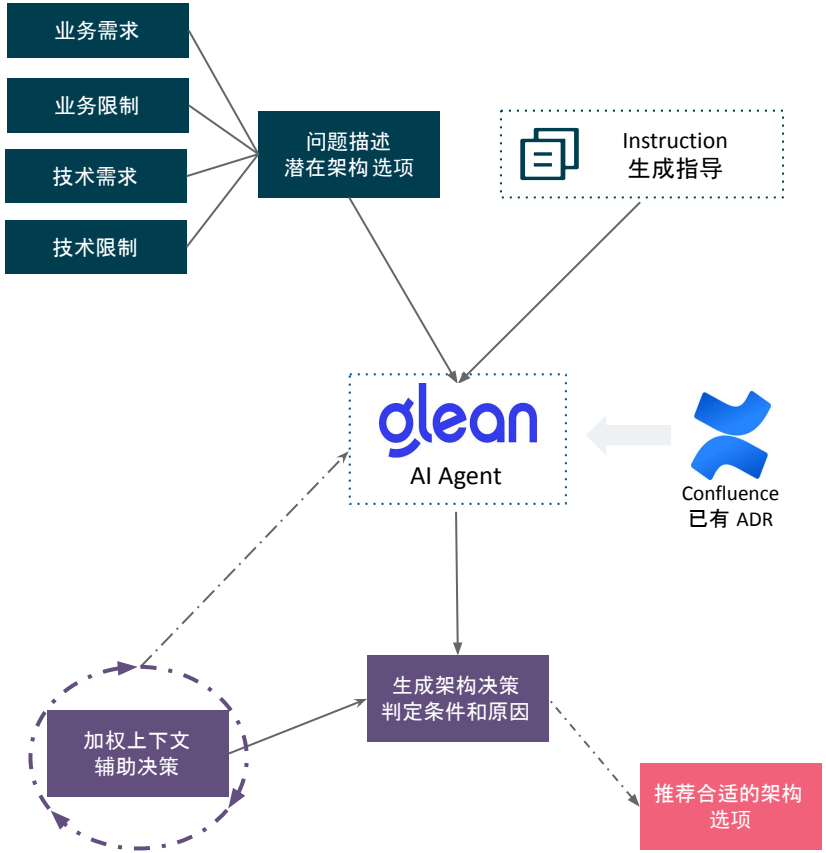


AI 辅助 - 编写架构决策文档 (ADR)

引入AI的熵增来源 根据业务需求，业务限制，技术需求，技术限制，通过提示词和 Glean LLM 交互生成架构决策的判定条件和推荐选项。需要有良好的问题定义能力，把要解决的问题表达清楚 (如：交互的子系统、业务和技术的限制、各种推荐的架构选项等)，Glean LLM需要已经学习了继往的架构决策，知道本企业内合适的架构判定条件，有时需要人工加权，帮助产生合适的架构推荐选项。

度量指标 模块依赖膨胀度 | 架构文档同步延 迟 | 设计变更回退率

熵减实践及成果 编写架构决策文档 (ADR) 的决策判定部分，从4个小时，降低到几分钟，同时为每个决策项生成了更多可用的决策依据，有效的加速了架构决策过程。



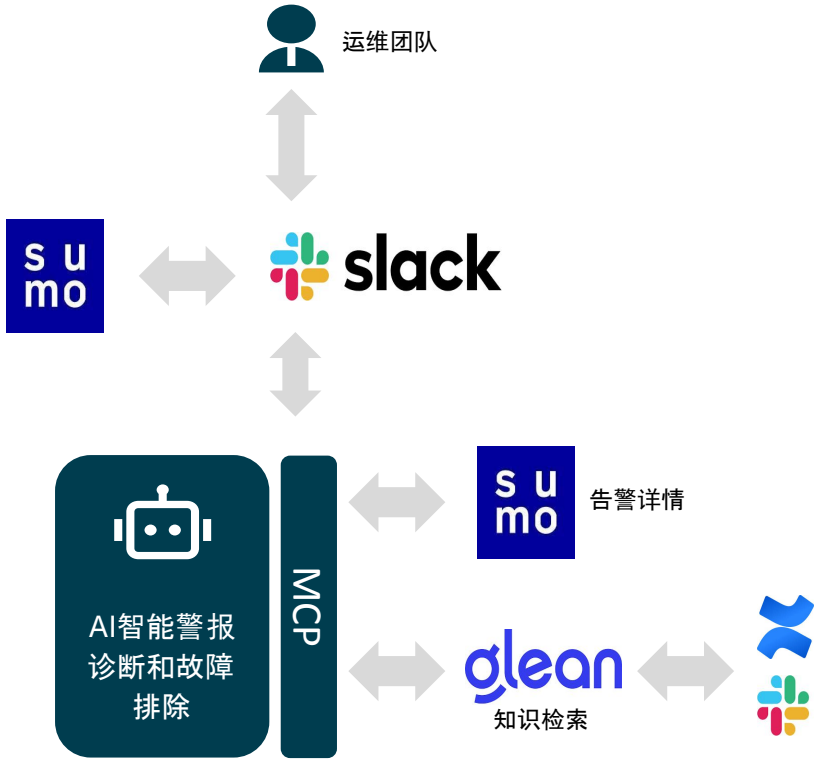
AI 赋能 - 智能警报诊断和故障排除

使用Sumo(监控系统) + 团队知识库 (Confluence) + Glean(AI) 快速处理系统告警

引入AI的熵增来源 集成各种监控指标、自动获取系统当前状态，将这些信息与 Glean (AI) 集成，根据现有团队知识库生成最终的故障排除步骤，并通过 Slack 提供建议。需要根据 Glean (AI) 集成各个系统 (MCP)，管理好接口和数据复杂度，实时的人工校准，保证良好的工程实践，持续治理一个良好的团队知识库。

度量指标 平均修复时长（MTTR） 知识库利用率

熵减实践及成果 故障识别时间从 30 分钟缩短至 2 分钟，80%以上的故障可以在团队知识库内查询到，并实现 99.97% 的系统可用性。

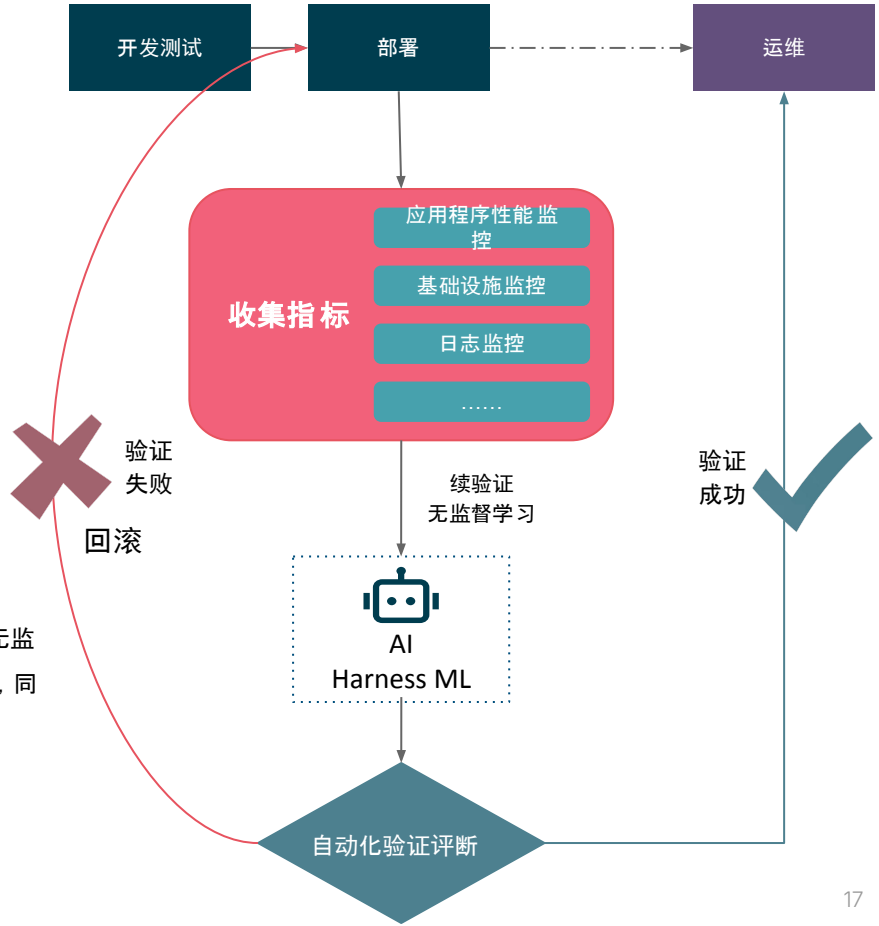


AI 赋能 - 部署后验证驱动 自动化闭环

引入AI的熵增来源 Build.com 引入 **Harness ML** 机器学习模型对部署后性能进行实时验证，需要额外搭建数据采集与模型运行管道，增加了流程治理的开销和数据安全的风险。

度量指标 **缺陷漏检率** **平均修复时长（MTTR）** **部署失败率**

熵减实践及成果 Build.com 在其三周一次的生产发布后，利用 Harness ML 通过持续验证和无监督学习实现了自动化性能验证，将每次验证工时从约 21 小时减少至 3 小时，工作量缩减 85%，同时当出现问题时可以智能的自动化回滚。

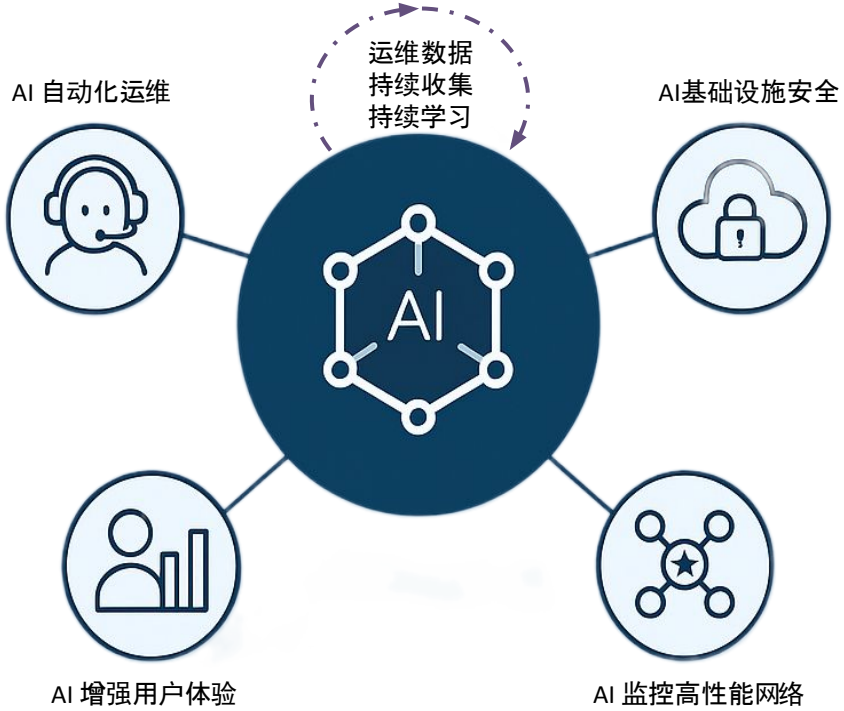


AI 原生 - 自优化自愈的网络运维

引入AI的熵增来源 **Juniper Mist 网络运维** 将 AIOps、微服务架构与实时分析引擎深度融合，进行自动化监控与优化。将 AI 模型与系统控制集成，通过数据流持续训练与反馈，操控系统，需要治理数据、模型训练，需要为组织引入新的工作方式，产生一些组织变革。

度量指标 **平均修复时长（MTTR）** **告警噪声比**

熵减实践及成果 可将告警误报率降低 90%，平均故障定位时间从数小时降至分钟级，MTTR 缩短 70%，减少了大量人工排查与跨系统协调带来的无序成本。



AI 加速软件研发 Do’s & Don’ts

AI 介入方式	✅ 要做的（Do’s） （有助于熵减、提升系统有序性）	❌ 不要做的（Don’ts） （易导致熵增、增加系统混乱）
AI 辅助 (AI-Assisted)	利用 AI 自动补全代码、生成文档，减少重复劳动，提升文档与代码一致性，有效**抑制“编码熵”**	**不经人工校验地直接采纳** AI 生成内容，容易引入潜在缺陷，**放大系统无序度**
	使用 AI 自动生成注释、知识总结，提升知识留存率，**降低“知识孤岛”带来的熵增**	**依赖 AI 替代专业判断**，忽视基本设计原则，可能**引发架构方向性混乱**
	将 AI 输出视为“第一稿”，进行人工审查与改进，保持质量闭环	视 AI 为“正确答案提供者”，导致**开发者心智萎缩与协同效率下降**
AI 赋能 (AI-Empowered)	使用 AI 生成测试脚本、识别测试盲区，**增强覆盖率，减少“测试熵”**	**盲目自动化而缺少人工 审核机制**，容易导致自动错误传播，**导致熵迅速积累**
	引入AIOps 聚合告警、辅助定位根因，**提升可观察性，减少“运维熵”**	**忽略 AI 的上下文局限性 (MCP)**，导致误判结果被采纳，**增加不确定性**
	用 AI 进行安全审查、依赖分析与编码规范治理，**在 CI/CD 体系内控制“技术债”增长**	在流程中引入AI 却**缺乏对输出质量的校验机制**，破坏已有稳定链路
	结合静态分析与智能质量门禁，**阻断“缺陷传染链”，提升交付 稳定性**	**忽视数据透明性与上下文 记录**，AI 决策“黑盒化”，**难以溯源与 调优**
AI 原生 (AI-Native)	从系统设计阶段即考虑 AI 驱动，如自愈系统、智能调度等，**构建“主动有序系统”**	**完全依赖 AI 做出决策，缺乏兜底与回退机制** ，易在关键时刻导致系统失控
	构建 MLOps 流水线，支持数据闭环与模型迭代，**保证“智能”能力可持续演进**	**数据未治理即喂 给模型**，输入混乱则输出混乱，**“数据熵”通过 AI 被放大**
	融合 AI 与架构治理策略，使 AI 融入变更管理、配置管理流程，**减少“架构熵”波动**	**忽视模型解释性与决策可追溯性** ，使 AI 行为难以理解与纠偏，**影响系统可维护性与可信度**

观点摘要

1. AI 赋能研发是一个“行易知难”的过程，需要实践先行，归纳总结（Dos & Don'ts），再规模化。
2. 从全链路视角思考，AI 是提升研发“有序性”的生产力工具。
3. 当 AI 成功赋能软件研发时，研发活动将变得更加高效、有序、自适 应，团队释放更多创造力，系统表现出持续自优化的能力——即“以 AI 为杠杆，实现逆熵增式进化”。
4. 对你来说是复杂的，对AI来说也是复杂的。AI 模型现在能力有限，但未来可期。

Q&A

张思楚
May 2025