

从 SCR 关系到搜索空间治理：AI 软件工程交付可靠性的基础分析框架

From the SCR Relationship to Search Space Governance: A Foundational Analytical Framework for Delivery Reliability in AI Software Engineering

作者：刘明

ORCID: 0009-0001-2816-538X

中文正式论文母版 V2.6.2

摘要

生成式 AI 已能低成本生成代码和功能，但复杂软件的可靠交付仍依赖对任务约束、组件依赖、变更影响与验证反馈的持续处理。本文考察由大语言模型、开发 Agent、工具、执行环境、验证反馈与多轮迭代共同组成的完整 AI 开发系统，并以从接受任务到完成交付、终止任务或重新拆分为止的任务会话为分析单位。在此基础上，本文依次定义搜索空间（S）、覆盖能力（C）和交付可靠性（R），提出**搜索空间—覆盖能力—交付可靠性关系（Search Space–Coverage Capability–Delivery Reliability Relationship, SCR）**，作为分析 AI 软件交付问题的基础框架。交付可靠性（R）描述一项具体 AI 开发结果能否被作为完成交付加以信赖，既不等于结果在已知标准下的正确性，也不等于多次任务的统计成功率。在 SCR 框架中，覆盖能力（C）与搜索空间（S）的匹配构成交付可靠性（R）的基础约束；在其他关键条件相近时，匹配越充分，具体交付通常越可能达到可被信赖的状态。由此可以形成两个基本干预方向：扩展覆盖能力（C），或调整搜索空间（S）。这一关系是框架中的定义性关系，不是已经完成操作化的封闭定量公式，也不预设搜索空间（S）和覆盖能力（C）可以被压缩为单一标量。本文聚焦后一方向，将搜索空间治理定义为持续调整任务边界与工程环境，使搜索空间（S）进入当前覆盖能力（C）可稳定处理范围的工程过程。搜索空间（S）虽难以绝对量化，但通常可以判断局部工程变化如何改变典型后续任务需要处理的范围、结构、候选可区分性与验证条件，并通过任务轨迹与交付结果逐步校准这一判断。由于 AI 生成的代码也构成后续任务的工程环境，搜索空间治理必须贯穿项目演化。对于大型、长期演化的 AI 软件项目，功能生成体现项目产生了多少直接结果，长期工程治理则影响这些结果能否被后续任务持续理解、修改和验证。SCR 关系由此提供描述搜索空间、系统能力与交付结果的共同结构；搜索空间治理说明工程上可以直接干预什么；对长期项目的分析则进一步表明，工程进展还受到当前结果能否被后续工作稳定继承的影响。这些内容共同解释传统软件工程与 AI 软件工程的连续性和变化，并为 AI 时代的工程化理论、系统实现与后续证据研究提供基础分析框架。

关键词：AI 软件工程；SCR 关系；搜索空间；覆盖能力；交付可靠性；搜索空间治理

1. 引言

AI 生成的代码不仅是当前任务的结果，也会成为后续任务所依赖的工程环境。一个局部可用的实现可能同时引入新的状态、依赖与隐含约束，改变未来开发需要理解和验证的范围。因此，当前功能能否

运行、当前结果是否足以被作为完成交付加以信赖，以及后续任务能否继续可靠完成，是彼此相关但并不相同的问题。

传统软件工程已经通过模块化、接口、类型、测试和重构改善工程环境，AI 开发又引入任务规范、上下文选择、检索增强、Agent 规划与执行反馈等机制。本文试图补充一种面向 AI 开发的统一判断结构：项目应被治理成什么状态，才能让 AI 在持续演化的工程环境中保持可靠交付？这一结构需要解释单次生成、上下文组织、仓库结构和 Agent 能力等处于不同技术层次的机制如何共同影响交付结果。

本文首先以搜索空间（S）与覆盖能力（C）的匹配解释交付可靠性（R），随后讨论如何通过调整任务边界和工程环境改善这种匹配，最后分析这一过程为何必须贯穿项目的长期演化。本文的贡献在于提出并界定这一分析框架，而不是给出可跨任务直接计算的定量函数。这一结构可以统一说明模型能力、Agent 机制、任务规范、上下文机制与传统软件工程实践如何影响交付结果，也可以解释 AI 软件工程为何仍需依赖模块、接口、类型和测试等工程结构，同时更直接地处理生成结果中的交付不确定性。

2. SCR 关系的构成

本文所称的 **AI 开发系统**，不是一次独立的大语言模型调用，而是基础模型、开发 Agent、检索与编辑工具、代码库与执行环境、测试与人工反馈，以及预算范围内多轮修正能力的总和。本文以**任务会话**为分析单位：任务会话从系统接受一个明确的工程目标开始，到任务完成交付、被终止或被重新拆分为止，可以包含多次模型调用、工具执行和上下文更新。

2.1 搜索空间（S）

搜索空间（S），是完成特定任务时，系统需要辨别和处理的、围绕需求解释、工程状态、实现路径与验证路径展开的可能性范围。

搜索空间（S）由任务要求、当前工程状态和任务治理方式共同塑造。它具体包括：对目标、非目标和验收标准的可能解释；可能相关的代码、状态、文档与依赖；尚未显式表达的约束；可能成立的实现路径；以及需要验证的受影响状态和执行路径。边界越模糊、相关状态越分散、候选路径越多且越难区分，系统需要处理的搜索空间（S）就越大。

搜索空间（S）不是单一标量。其工程形态同时取决于需要处理的范围、空间结构、隐含约束、候选路径的可区分性以及验证条件。候选数量相近的两个任务，也可能因依赖结构、反馈质量或关键约束是否显式而具有完全不同的处理难度。因此，本文所称搜索空间扩大、缩小或得到改善，均指这些维度在具体任务中的综合变化，而不是对所有任务通用的单值排序。

搜索空间（S）不能用代码量或某次调用的上下文空间直接表示。上下文描述模型当前可直接访问的信息，搜索空间（S）则描述整个任务会话中系统为完成交付而需要辨别和处理的可能性。上下文工程可以通过提供必要信息、排除无关候选和减少临时推断来改变搜索空间（S），但长上下文本身不能保证信息得到稳定利用[9]。

2.2 覆盖能力 (C)

搜索空间 (S) 描述任务需要处理的可能性范围，覆盖能力 (C) 描述完整 AI 开发系统处理这一范围的能力。覆盖只能相对于特定搜索空间讨论，因此，覆盖能力 (C) 不是脱离任务存在的通用数值，而是系统在特定任务类型、执行预算和反馈条件下表现出的稳定能力。

覆盖能力 (C)，是完整 AI 开发系统在给定任务类型、执行预算和反馈条件下，稳定识别搜索空间 (S) 中的关键条件、区分候选路径、排除错误方向并获取有效验证证据的能力。

这里的覆盖不要求穷举全部候选。可靠交付所要求的覆盖，是指交付所依赖的关键约束、相关状态、实现决策和影响范围没有被遗漏，不同候选路径得到充分比较，错误方向得以排除，最终结果获得与风险相称的验证。

模型提供知识、推理和候选生成能力；Agent 通过上下文组织、检索、规划、工具调用、执行循环与验证反馈，使这些能力作用于具体任务。验证反馈不仅检查结果，也帮助系统区分候选路径并排除已被证伪的方向；反馈越能准确暴露关键错误，有限的搜索过程越可能形成充分覆盖。

覆盖能力 (C) 以系统能够重复、稳定覆盖的关键环节为准，而不是偶然完成高难度任务时表现出的能力上限。同一系统面对边界清晰、反馈密集的任务可能表现可靠；面对目标含糊、隐含状态密集或难以验证的任务，覆盖能力 (C) 则可能显著下降。

2.3 交付可靠性 (R)

交付可靠性 (R)，是特定 AI 开发结果满足关键要求、其相关影响得到充分验证，并在预期条件下保持有效，因而能够被作为完成交付加以信赖的程度。

交付可靠性 (R) 是一个复合交付属性：它既涉及结果实际上是否满足关键要求，也涉及当前验证证据是否足以支持将其作为完成交付加以信赖。本文不进一步把二者拆分为独立变量；在经验研究中，则应分别记录结果状态与证据状态，避免把“未发现错误”直接等同于“结果不存在错误”。

交付可靠性 (R) 面向一次具体交付，回答“结果已经生成，但能否作为完成交付被信赖”。AI 产出可能已经实现主要功能，局部结果也可能正确；但如果系统遗漏关键约束、未处理相关状态，或没有充分验证变更影响，这一结果仍不足以被视为可靠交付。

覆盖能力 (C) 与交付可靠性 (R) 处于不同分析层次：覆盖能力 (C) 描述系统在一类任务条件下能够稳定完成何种处理，交付可靠性 (R) 描述本次具体结果在要求满足、影响处理与验证证据方面达到的状态。前者是过程能力，后者是交付结果属性。本文所称交付可靠性也不同于以特定时间和环境内无故障运行为中心的软件产品可靠性；软件的运行可靠性可以是交付要求的一部分，但不能单独代表一次开发结果是否足以被信赖。

交付可靠性 (R) 不等同于正确性。正确性描述结果是否符合一组已知判断标准；交付可靠性 (R) 还取决于这些标准是否覆盖关键要求，相关影响是否已经处理，以及支持判断的验证证据是否充分。正确性是交付可靠性 (R) 的必要组成，却不能单独穷尽后者。

交付可靠性（R）也不等同于成功率。成功率是系统在一组任务或多次运行中的统计结果，交付可靠性（R）则首先描述一次具体交付能够被信赖的程度。统计结果可以用于校准对交付可靠性（R）的判断，但不能替代对当前任务要求、影响范围和验证证据的分析。

2.4 SCR 关系

本文将搜索空间（S）、覆盖能力（C）与交付可靠性（R）之间的基础关系称为 SCR 关系：

在 SCR 框架中，系统覆盖能力（C）与任务搜索空间（S）的匹配构成交付可靠性（R）的基础约束。在其他关键条件相近时，匹配越充分，一次交付通常越可能达到可被信赖的状态。

本文所称“匹配”，不是两个标量之间的数值比较，而是指任务在关键约束识别、相关状态处理、候选路径区分和验证证据获取等方面提出的要求，是否进入当前系统能够稳定处理的范围。不同方面的具体划分和代理指标可以随任务类型变化，本文不预设一套跨项目通用的操作化方案。

同一项产品需求在不同工程状态和任务治理方式下，可能形成显著不同的搜索空间（S）。当搜索空间（S）相对于覆盖能力（C）扩大时，更多关键条件、候选路径和影响范围会共同争夺有限的注意力、搜索次数与验证资源，决定性遗漏更容易发生，交付可靠性（R）通常随之下降。反之，当覆盖能力（C）相对于搜索空间（S）增强时，更多关键条件能够得到稳定处理，错误路径也更容易被反馈排除，交付可靠性（R）通常随之提高。因此，搜索空间（S）的绝对大小或覆盖能力（C）的单独强弱都不能充分解释交付结果；关键在于当前覆盖能力（C）能否稳定处理当前搜索空间（S）。

搜索空间（S）的工程难度同时取决于候选数量、空间结构与候选可区分性。明确目标、清晰边界和稳定结构能够使关键判断局部化；高质量反馈则可以提高候选路径的可区分性，并增强系统排除错误方向的能力。

SCR 关系的理论性质。 SCR 关系是一种定义性分析框架，而不是由三个已经操作化的标量拟合得到的经验公式。它通过区分任务需要处理的搜索空间（S）、系统能够稳定形成的覆盖能力（C）和一次结果的交付可靠性（R），建立描述 AI 软件交付问题的共同结构。框架本身需要接受概念区分、逻辑一致性、解释范围与分析效用的检验；由框架提出的具体经验判断，例如某项工程措施在给定条件下是否改善二者的匹配并提高交付可靠性（R），则仍需通过任务轨迹和交付结果检验。本文不主张搜索空间（S）与覆盖能力（C）已经可以被压缩为跨项目通用的单一数值，其操作化和代理指标属于后续经验研究。

SCR 关系由此给出两个基本干预方向：扩展覆盖能力（C），或者调整搜索空间（S）。模型、Agent 与工具的进步主要作用于前一方向；任务边界、工程结构和验证条件能够直接改变后一方向；部分机制可能同时作用于二者。本文主要讨论对搜索空间（S）的干预。

3. 搜索空间治理

3.1 干预对象的选择

在 AI 开发系统既定或短期变化有限时，工程实践最直接的控制变量是搜索空间（S）。通过调整任务边界、工程结构和验证条件，可以改善搜索空间（S）与覆盖能力（C）的匹配，进而提高交付可靠性（R）。

搜索空间治理，是持续调整任务边界与工程环境，使搜索空间（S）进入当前覆盖能力（C）可稳定处理范围的工程过程。

搜索空间治理不仅处理生成结果，也处理生成所处的任务条件和工程条件。明确目标与非目标、限制相关范围、固化稳定约束、隔离状态与依赖，以及建立能够区分路径的验证反馈，都可以改变搜索空间（S）。工程干预的目标由此从帮助 AI 生成当前结果，推进到持续建立支持可靠交付的任务条件和工程环境。

任务拆分只有在原始目标、关键约束和子任务之间的集成关系得到保留时，才构成搜索空间治理。拆分后，子任务可以形成各自的任务会话，但父任务层面的交付要求仍然存在；被转移到接口和集成阶段的复杂性，也仍属于整体任务需要处理的搜索空间（S）。

搜索空间治理不以最小化候选数量为目标，而以改善搜索空间（S）与覆盖能力（C）的匹配为目标。错误排除必要信息或有效实现路径，即使减少了表面候选数量，也可能降低交付可靠性（R）；增加必要的验证路径，虽然扩大了部分显式处理范围，却可能提高候选可区分性并改善整体匹配。因此，治理既包括减少无效搜索负担，也包括保留必要路径、改善空间结构和建立有效反馈。

3.2 任务治理与项目治理

搜索空间治理同时作用于当前任务和未来任务：

治理尺度	直接对象	典型作用	时间范围
任务治理	当前任务会话	明确目标、选择必要上下文、拆分任务、建立验收反馈	当前交付
项目治理	持续演化的工程环境	隔离状态与依赖、固化接口和类型、保留回归验证	后续多个任务

任务治理控制本次交付涉及的解释、状态、路径与影响范围；项目治理则把已经确认的知识转化为后续任务能够直接依赖的结构。二者使用的工具可能相同，区别在于作用范围和时间尺度：任务治理调整当前任务面对的搜索空间（S），项目治理塑造未来一系列任务所依赖的工程环境。

一次任务中确认的知识，只有被工程环境持续承载，才能供后续任务直接利用。例如，任务规范可以限定本次交付。规范中经过验证且长期稳定的约束，可进一步固化为项目规则、接口或类型；临时验收测试被保留为回归测试后，也会持续排除后续修改中的错误路径。

3.3 从一次干预到持续治理

生成结果具有双重性质：它既是当前任务的产出，也是后续任务的工作环境。一次修改引入或确认的边界、依赖、接口、类型和测试，会决定未来任务需要重新理解什么、可以排除哪些路径，以及必须验证多大范围。如果关键关系只存在于当前会话的推理中，后续任务就必须重新发现；如果这些关系被转化为稳定结构，后续任务便可直接利用已经确认的约束和关系。

新增需求和代码修改会不断重写这一环境，原本有效的边界与验证机制也会随之变化。当前生成改变工程环境，工程环境继续塑造未来任务的搜索空间（S），因此一次性干预不能保证后续任务仍与覆盖能力（C）保持匹配。

因此，搜索空间治理必须形成“生成—验证—固化—继续生成”的循环，依据后续生成与验证暴露的新信息持续修正工程边界。它不是单次任务优化，而是伴随项目演化持续进行的工程过程。

4. 如何判断搜索空间的变化并积累证据

4.1 判断搜索空间的可处理性如何变化

搜索空间（S）由需求解释、相关状态、隐含约束、候选路径和待验证影响共同构成。虽然难以对它进行绝对度量，但可以面向具体任务比较工程变化对处理范围、空间结构、候选可区分性和验证条件的综合影响。

搜索空间治理可以从一个基本判断开始：某项工程变化是否改善当前或未来任务的搜索空间（S）与覆盖能力（C）的匹配。

这一判断应围绕项目中高频、关键或高风险的典型任务展开。判断时，应比较工程变化前后系统需要定位和理解的代码、状态与依赖，需要发现的隐含约束，需要选择的候选路径，以及需要验证的影响范围，同时考察关键关系是否更清晰、候选路径是否更容易区分、反馈是否更能暴露决定性错误。处理范围整体减少通常会降低搜索负担，但范围增加并不必然恶化匹配；如果新增内容显式化了必要约束或建立了有效验证，搜索空间（S）的整体可处理性仍可能提高。

4.2 既有工程实践的统一解释

工程实践	主要空间作用
任务规范与上下文选择	减少对需求的解释空间，提供必要约束，限制相关范围
任务拆分	减少单次任务需要同时覆盖的状态、依赖和影响范围
模块化与重构	使状态和变更影响局部化，删除重复或失效路径
接口与类型	将交互和约束显式化，自动排除部分非法路径
测试与运行反馈	提高正确与错误路径的可区分性，明确回归边界

SCR 关系为这些实践提供统一的评价结构：一方面比较变化前后典型任务的搜索空间（S），另一方面考察当前交付中获得的知识是否转化为未来任务可以利用的边界和反馈。传统工程手段与 AI 开发规范由此可以依据其对搜索空间（S）、覆盖能力（C）和交付可靠性（R）的实际作用进行统一评估。

4.3 用工程证据持续校准判断

后续研究可以先判断某项变化会如何改变特定任务的处理范围、空间结构、候选可区分性和验证条件，再记录变化前后的工程轨迹和交付结果，在相近的项目、任务类型与系统条件下校准可观察的代理指标，并用实际结果持续修正判断。随着证据积累，可以逐步形成适用于特定项目和任务类型的经验规律。

可积累的证据包括：典型任务涉及的实体和模块数量，跨越的依赖边界与状态范围，需要重新发现的隐含约束，验证覆盖的执行路径与影响范围，多轮生成或审计中新增问题的数量、类型与收敛特征，以及项目增长后同类任务的交付成本和失败率变化。

交付可靠性（R）是最终评价对象，搜索空间（S）的变化、任务执行轨迹、问题收敛、验证负担和历史失败率则是不同层次的代理证据。这些证据不能与交付可靠性（R）直接等同；只有在任务类型、覆盖能力（C）和风险条件相近时，才适合比较同一项目中相近任务的轨迹与结果。持续积累此类证据，可以提高交付风险评估与治理效果评价的可比较性和可验证性，并逐步提高对特定项目和任务类型的判断准确度。

5. 长期工程进度不能只按功能数量衡量

5.1 功能生成与工程进度的分离

关于 Vibe coding 的灰色文献综述发现，实践者常以速度和可及性为动机，快速获得“即时成功”，但多数又认为生成代码“快但有缺陷”，且测试等质量保障活动经常被忽略[11]。从工程进度看，这构成一种典型错觉：功能能够运行，只能证明某条路径在当前观测条件下可行；它既不证明关键条件已被覆盖，也不证明结果已经达到可被信赖的交付状态。

传统开发把功能实现视为重要进度，是因为实现通常需要大量人工投入，开发者也会在此过程中处理相当一部分需求、结构与验证问题。在一项受控实验中，使用 GitHub Copilot 的参与者完成特定 JavaScript 编程任务的速度提高了 55.8%[5]，说明生成式 AI 能显著压缩部分实现活动的时间成本；但确认边界、固化约束、隔离依赖和建立有效验证的成本未必同步下降。AI 的低成本生成进一步放大了功能生成与工程治理之间原已存在的差异，已生成功能的数量也更难近似代表长期工程进度。

5.2 可运行代码不等于可持续维护的工程成果

能运行的代码并不一定能够被后续任务持续利用。只有当边界、依赖与关键约束足够明确，结果得到有效验证，后续任务能够在受控范围内理解和修改时，当前实现才会成为可以长期维护和利用的工程成果。

未经良好治理的实现难以形成可累积的工程进度，还可能增加状态、依赖与验证负担，扩大未来任务的搜索空间（S），甚至使返工成本超过初次生成成本。因此，功能生成只能衡量项目产生了多少直接结果；这些结果能否被后续工作稳定继承，才影响它们是否真正构成长期工程进展。

5.3 长期工程治理影响成果能否持续累积

对于大型、长期演化的 AI 软件项目，功能生成体现项目产生了多少直接结果，长期工程治理则影响这些结果能否被后续任务持续理解、修改和验证。

功能仍是项目需要完成的交付对象，但功能数量无法说明当前实现是否可以被长期依赖。对于小型原型或一次性工具，当前可用可以直接作为完成标准；对于需要持续演化的项目，工程治理影响当前结果是支持后续开发，还是把复杂度和风险转移给未来任务。

长期工程治理的目标，是在项目复杂度持续增长时，仍将典型新任务的搜索空间（S）控制在当前覆盖能力（C）可稳定处理的范围内。这一条件越能持续成立，交付可靠性（R）越不容易随着项目增长而系统性下降。因此，大型 AI 软件项目的工程进度不能只按功能数量衡量，还应考察当前成果能否继续支持后续任务的可靠交付。

6. 相关工作与理论定位

6.1 问题空间与搜索式软件工程

将问题求解理解为在问题空间中选择操作与路径，是人工智能和认知科学中的经典视角[1]。Simon 对非良构问题的分析进一步指出，问题表征和任务结构可能在求解过程中逐步形成与修正[2]。搜索式软件工程通常把既定软件工程问题形式化为具有表示、目标函数和搜索算子的优化问题[3]。本文不设计搜索算法，也不假定任务已经形式化，而是研究实际 AI 开发任务中搜索空间（S）、系统覆盖能力（C）与交付可靠性（R）之间的基础关系，以及工程治理如何改变这一关系。

6.2 仓库级 Agent 与上下文工程

SWE-bench 表明，真实仓库任务往往需要跨函数、类和文件协调修改，并要求系统处理长上下文、复杂推理与执行环境[6]。CodePlan 与 SWE-agent 进一步表明，依赖分析、规划、上下文组织和 Agent—计算机接口会显著影响仓库级任务表现[7-8]。长上下文研究表明，增加上下文长度并不保证相关信息被稳定利用[9]；仓库级代码上下文工程研究则揭示了不同策略在效果、任务复杂度与计算成本之间的权衡[10]。本文把模型与 Agent 机制主要理解为覆盖能力（C）的来源，同时把检索、规划、上下文选择和动态拆分视为可能同时改变搜索空间（S）与覆盖能力（C）的系统机制。

6.3 AI 工程治理的统一分析

信息隐藏和模块化是传统软件工程控制理解范围与变更传播的经典路径[4]；接口、类型、测试和变化隔离等机制也承担着显式化约束、局部化影响的作用。AI 软件工程又引入任务规范、上下文选择、检索增强、Agent 规划和执行反馈等机制[6-10]。这些实践分布在需求、知识、上下文、代码结构与运行验

证等不同层次。本文所要补充的不是又一种局部机制，而是将需求规范、持久工程状态、上下文机制、系统能力与交付结果纳入同一基础关系的统一分析框架。

SCR 关系提供了这一基础结构。任务规范（Spec）固定需求解释、关键约束和验收边界，主要缩小并结构化搜索空间（S）。代码仓库（Code Repo）不仅是 Agent 的信息来源，还持久承载代码、文档、配置、依赖与历史治理结果；其模块、接口、类型、测试和依赖结构共同塑造当前及未来任务的搜索空间（S）。检索增强生成（RAG）、代码检索与上下文选择把任务相关信息提供给系统，主要增强覆盖能力（C）；精准选择也可以减少系统需要辨别的无关状态和路径。模型、Agent 与工具决定系统能够稳定处理的范围，测试与执行反馈则帮助区分和排除错误路径，并将已经确认的判断转化为后续任务可复用的验证边界。这些机制虽然处于不同技术层次，却都可以沿搜索空间（S）与覆盖能力（C）的匹配关系，解释其如何影响交付可靠性（R）。

搜索空间（S）与覆盖能力（C）是分析维度，不是要求把每项工程机制唯一归类的组件清单。对机制的归类取决于具体分析层次，同一机制可以同时作用于不同维度。这一分析也表明，工程机制不能仅依据是否采用某种机制或其投入规模来评价。更详细的 Spec 可能固化必要约束，也可能加入与交付无关的实现限制，增加系统需要同时处理的条件与验证路径；更多检索内容或仓库上下文可能补充关键知识，也可能引入噪声、冲突和额外判断负担；更大的代码仓库可能积累可复用的工程能力，也可能因边界失效和依赖扩散而持续扩大未来任务的搜索空间（S）。因此，同一种机制可能同时作用于搜索空间（S）与覆盖能力（C）。其工程价值取决于是否改善二者的匹配并最终提高交付可靠性（R），而不能由机制名称、上下文规模或工具数量预先决定。

搜索空间治理进一步把这一关系转化为持续的工程过程。传统软件工程中的模块、接口、类型、测试和变化隔离等机制，不再只是彼此分散的最佳实践，也可以被统一理解为控制当前及未来搜索空间（S）、支撑有限覆盖能力（C）并维持交付可靠性（R）的工程基础。AI 软件工程由此延续了传统软件工程对复杂性的控制，同时也发生了关键变化：当生成系统能够在关键约束尚未充分处理时产出高度完整的结果，工程治理就必须直接识别和降低交付不确定性。SCR 关系因而为传统软件工程结构与 AI 开发机制提供共同的解释，搜索空间治理则说明可以从哪些工程条件入手改善交付结果。

7. 适用范围与外推

SCR 关系可能具有超出软件工程的解释力。在一般 AI 任务中，任务要求、环境状态与交互方式同样会塑造搜索空间（S），模型、Agent 与反馈条件共同形成覆盖能力（C）；二者失配时，可能表现为幻觉、约束遗漏、路径选择错误或验证失败。软件工程具有状态可持久化、约束可显式化、变化可追踪、结果相对可验证等条件，适合对搜索空间（S）进行外部治理；研究、战略、创作和社会判断等开放任务，则更难充分外部化目标、状态、约束与评价标准。本文仅讨论软件工程，SCR 关系在其他领域的解释力及其搜索空间能否得到有效治理，仍需独立检验。

8. 结论

本文依次定义搜索空间 (S)、覆盖能力 (C) 和交付可靠性 (R)，并提出 SCR 关系这一基础分析框架：在该框架中，覆盖能力 (C) 与搜索空间 (S) 的匹配构成交付可靠性 (R) 的基础约束；在其他关键条件相近时，匹配越充分，具体交付通常越可能达到可被信赖的状态。其中，交付可靠性 (R) 描述一项具体结果能否被作为完成交付加以信赖，既不是正确性的改名，也不是统计成功率；覆盖能力 (C) 描述系统处理任务的过程能力，交付可靠性 (R) 描述本次交付达到的结果状态。

本文提出的不是一组孤立的 AI 开发实践，也不是已经完成操作化的定量公式，而是一套组织这些实践的基础分析结构。SCR 关系使任务规范 (Spec)、代码仓库 (Code Repo)、RAG、模型与 Agent、测试与执行反馈，以及传统软件工程中的模块化、接口、类型和重构，都能依据其对搜索空间 (S) 与覆盖能力 (C) 匹配关系的实际作用被统一理解。搜索空间治理进一步说明，可以通过持续调整任务边界与工程环境改善这种匹配；其目标不是单纯最小化搜索空间 (S)，而是减少无效搜索负担、保留必要路径、改善空间结构和建立有效反馈。对于长期项目，工程进展还取决于廉价生成的功能能否被后续任务持续理解、修改和验证。

这一分析重新说明了传统软件工程与 AI 软件工程的连续性和变化。AI 软件工程仍依赖模块、接口、类型和测试等工程结构控制复杂性；但生成能力与可靠交付之间的分离，使工程治理不能只保证这些结构存在，还必须直接处理当前覆盖能力 (C) 与任务搜索空间 (S) 的失配。搜索空间治理由此将传统工程结构、AI 开发机制与长期交付结果连接起来，为 AI 时代的工程化理论、系统设计和证据积累提供共同起点。

未来研究可以先判断局部工程变化如何改变搜索空间 (S) 的处理范围、结构、候选可区分性与验证条件，再积累任务轨迹、依赖范围、隐含约束、验证负担与交付结果等证据，持续校准适用于特定项目和任务类型的经验判断。框架本身用于建立共同分析语言，由框架形成的具体预测、代理指标与治理效果则应持续接受经验检验；SCR 关系在其他智能任务中的解释力及其搜索空间能否得到有效治理，也需独立研究。

参考文献

- [1] NEWELL A, SIMON H A. Computer science as empirical inquiry: Symbols and search[J]. *Communications of the ACM*, 1976, 19(3): 113-126. DOI: [10.1145/360018.360022](https://doi.org/10.1145/360018.360022).
- [2] SIMON H A. The structure of ill structured problems[J]. *Artificial Intelligence*, 1973, 4(3-4): 181-201. DOI: [10.1016/0004-3702\(73\)90011-8](https://doi.org/10.1016/0004-3702(73)90011-8).
- [3] HARMAN M, JONES B F. Search-based software engineering[J]. *Information and Software Technology*, 2001, 43(14): 833-839. DOI: [10.1016/S0950-5849\(01\)00189-6](https://doi.org/10.1016/S0950-5849(01)00189-6).
- [4] PARNAS D L. On the criteria to be used in decomposing systems into modules[J]. *Communications of the ACM*, 1972, 15(12): 1053-1058. DOI: [10.1145/361598.361623](https://doi.org/10.1145/361598.361623).
- [5] PENG S, KALLIAMVAKOU E, CIHON P, et al. The impact of AI on developer productivity: Evidence from GitHub Copilot[EB/OL]. arXiv:2302.06590, 2023. DOI: [10.48550/arXiv.2302.06590](https://doi.org/10.48550/arXiv.2302.06590).

- [6] JIMENEZ C E, YANG J, WETTIG A, et al. SWE-bench: Can language models resolve real-world GitHub issues?[C]//*The Twelfth International Conference on Learning Representations*. 2024. <https://openreview.net/forum?id=VTF8yNQM66>.
- [7] BAIRI R, SONWANE A, KANADE A, et al. CodePlan: Repository-level coding using LLMs and planning[J]. *Proceedings of the ACM on Software Engineering*, 2024, 1(FSE): 675-698. DOI: [10.1145/3643757](https://doi.org/10.1145/3643757).
- [8] YANG J, JIMENEZ C E, WETTIG A, et al. SWE-agent: Agent-computer interfaces enable automated software engineering[C]//*Advances in Neural Information Processing Systems* 37. 2024: 50528-50652. DOI: [10.52202/079017-1601](https://doi.org/10.52202/079017-1601).
- [9] LIU N F, LIN K, HEWITT J, et al. Lost in the middle: How language models use long contexts[J]. *Transactions of the Association for Computational Linguistics*, 2024, 12: 157-173. DOI: [10.1162/tac1_a_00638](https://doi.org/10.1162/tac1_a_00638).
- [10] LI Y, LIN Q, PENG Y, et al. One size does not fit all: Revisiting code context engineering for repository-level code generation[J]. *Proceedings of the ACM on Software Engineering*, 2026, 3(FSE): 2952-2975. DOI: [10.1145/3808138](https://doi.org/10.1145/3808138).
- [11] FAWZY A, TAHIR A, BLINCOE K. Vibe coding in practice: Motivations, challenges, and a future outlook—A grey literature review[C]//*2026 IEEE/ACM 48th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 2026: 212-223. DOI: [10.1145/3786583.3786866](https://doi.org/10.1145/3786583.3786866).