



OPEN AGENTIC INDUSTRY CONSORTIUM
开放智能体产业联盟

OAIC WHITE PAPER

智能体软件工程 产业白皮书

Agentic Software Engineering

Release 1: 从“人写代码”到
“人定义需求、Agent 实现代码”

智能体软件工程的定义、框架与三项核心工程能力

2026年10月

开放智能体产业联盟 (OAIC)

www.oaic.ai



版权与知识产权声明

© 2026 开放智能体产业联盟 (OAIC)。保留所有权利。

本白皮书由 OAIC 发布。版权声明适用于 OAIC 对本出版物享有权利的内容及其编排；第三方材料、软件、商标以及作者依法或依约保留的权利，仍归相应权利人所有。

除适用法律允许的使用或权利人另行授予的许可外，复制、改编、翻译、转载或再分发本白皮书，应取得相应权利人的许可，并保留完整来源与版权信息。对本白皮书的引用，应注明标题、版本、发布机构及引用位置。

知识产权与专利

本白皮书所述方法、技术或实现可能涉及第三方知识产权，包括专利权。本白皮书的发布不构成专利、商标或其他知识产权许可，也不意味着 OAIC 已完成专利检索、权利清理或侵权风险评估。OAIC 不就相关专利的存在、有效性或权利范围作出认定。实施者应依据具体用途核查相关权利，并在必要时向权利人取得许可。

本白皮书不承诺相关技术可免费实施，也不构成开放源代码许可或公平、合理、无歧视的专利许可承诺。引用的软件、代码示例和开源项目，遵循各自适用的许可；本白皮书不改变这些许可条件。

商标与使用说明

OAIC 名称及标识用于标明本出版物的发布机构。本文涉及的其他名称、商标和标识归各自权利人所有；引用不表示该权利人对本白皮书的认可或背书。

本白皮书用于产业交流与工程实践参考，不是强制性标准、认证依据或法律意见。文中观点不必然代表各作者所在单位的正式立场。信息与技术实践可能随时间变化，读者应结合具体场景、风险及适用要求作独立判断。

在适用法律允许的范围内，本出版物按现状提供，OAIC 不对其适用于特定用途、无误或不侵权作出保证。使用者应自行开展必要的验证与专业评估；本声明不排除依法不得排除或限制的责任。

授权、勘误与合作信息：www.oaic.ai

作者与编委

姓名	单位
茹炳晟	腾讯研究院
孙栋	Futurewei
林云	上海交通大学
李建忠	奇点智能研究院
杨子江	中国科学技术大学

发布机构：开放智能体产业联盟（OAIC）

版本：Release 1 2026年10月

目标读者：企业业务负责人、技术负责人、咨询与研究机构、软件与人工智能企业、高等院校及科研单位

官网：www.oaic.ai



扫码了解 OAIC

目录

执行摘要	6
A.1 核心判断	6
A.2 ASE 的一句话定义	6
A.3 三项核心结论	6
A.4 主链路	7
第一篇 WHY：为什么软件工程必须改变	8
第 1 章 从"人写代码"到"人定义、Agent 实现"	8
1.1 能力跃迁：从代码补全到可持续执行工程任务	8
1.2 旧范式：以人为主要生产与控制中心	8
1.3 新范式：以人的意图和责任为边界，以 AGENT 为主要执行者	9
1.4 四个结构性变化	9
1.5 人的角色与不可转移的责任	10
第 2 章 新范式为什么需要新的工程体系	11
2.1 模型能力不等于生产系统能力	11
2.2 概率性认知与确定性工程责任之间的矛盾	12
2.3 新的失败模式和工程债务	12
2.4 传统软件工程需要扩展，而不是被抛弃	13
2.5 本篇小结	13
第二篇 WHAT：Agentic Software Engineering 是什么	14
第 3 章 ASE 的定义、对象与边界	14
3.1 正式定义	14
3.2 两类工程对象	14
3.3 与相邻概念的边界	15
3.4 首版范围与非目标	15
第 4 章 ASE 总体框架、能力要素与核心能力	16
4.1 从需求到证据的总体闭环	16
4.2 四层参考框架	16
4.3 六类能力域及关键要素	16
4.4 首版重点建设的三项核心能力	17
4.5 五项工程原则	17
4.6 人、AGENT 与平台的责任边界	18
第 5 章 全生命周期、关键工件与质量闭环	19
5.1 六阶段全生命周期	19

5.2 各阶段的最小工件与阶段门	19
5.3 贯穿生命周期的证据链.....	20
5.4 质量目标	20
5.5 生命周期中的人机责任门	21
第三篇 HOW：三项核心能力如何构建.....	22
第 6 章 规范定义：把人的意图转化为 Agent 可执行的工程契约.....	22
6.1 规范从说明文档升级为运行时控制对象	22
6.2 四类最小规范	22
6.3 "需求编译"过程.....	23
6.4 规范质量与冲突处理	24
6.5 规范变更与影响分析	24
6.6 最小实践模板	24
第 7 章 测试与评测：同时检查结果、过程和生产能力.....	25
7.1 明确测试对象	25
7.2 分层测试与评测方法	26
7.3 判定基准（TEST ORACLE）：谁来判断"正确"	26
7.4 统计纪律与可重复性	27
7.5 证据驱动的发布门禁	27
7.6 最小实践模板	28
第 8 章 验证与可信保证：用分层证据约束高风险行为.....	29
8.1 区分"做对的事"与"把事做对"	29
8.2 风险分级决定保证强度.....	29
8.3 分层验证机制	30
8.4 优先形式化的对象	30
8.5 可采用的形式化方法及渐进路径	31
8.6 总体保证模型	31
8.7 形式化验证的边界与诚实声明.....	32
8.8 最小实践模板	33
结语：从可观察、可验证的最小闭环开始.....	34
参考资料.....	35
附录 A ASE 最小闭环检查清单.....	37
附录 B 统一术语表	38

执行摘要

A.1 核心判断

以大语言模型（Large Language Model, LLM）为核心的系统，已从代码生成工具演进为能够利用上下文、制定计划、调用工具、修改软件工件，并根据反馈迭代修复的智能体（Agent）。这一变化正在改变软件生产的基本组织方式。

软件生产的主要执行者和控制方式正在改变，影响不止“编码更快”。过去，人主导执行与控制；如今，越来越多的执行工作由 Agent 承担，控制必须从“人盯过程”转向“人定义边界、系统提供证据、人依风险验收”。由此形成基于智能体的“需求编译”式软件开发方法 [18]（详见 6.3 节）。

据此，本白皮书提出新的范式概括：

人定义意图与责任边界，Agent 执行工程任务，系统形成可审计证据，人依据风险进行验收和问责。

在这一概括中，人并未退出。人的工作重心由“直接编写代码”转向“定义需求、设计系统、验证结果、承担责任”。

本白皮书主张：人设定目标并承担责任，**Agent 负责理解与编排，人、Agent 与平台共同建设和使用验证与回滚能力。**人设定目标、边界与验收标准，确认关键需求解释并承担最终责任；Agent 理解上下文与任务，编排工作与工具，完成实现、测试和修复；验证与回滚（含补偿与止损）是跨任务、跨主体的系统能力。

A.2 ASE 的一句话定义

本白皮书将 **Agentic Software Engineering (ASE, 智能体软件工程)** 定义为：

在人类设定的目标、规范、约束与验收标准之下，由智能体承担规划、实现、测试与修复等主要工程执行任务，并由工程系统全过程形成可审计证据，使人类能够基于风险进行验收与问责的软件工程范式。

该定义包含四个缺一不可的要素：**人的意图与责任边界**（谁定义、谁负责）、**Agent 的工程执行**（谁来做）、**可审计证据**（凭什么判断），以及**基于风险的验收与问责**（谁授权或预授权、谁批准）。缺少任何一项，都不构成本白皮书所讨论的 ASE。

A.3 三项核心结论

结论一：规范成为新的控制中心。执行主体由人转向 Agent 后，需求文档必须从供人阅读的说明材料升级为 Agent 可理解、可执行、可检查、可追踪的工程契约。规范从软件开发的前置文档转变为贯穿开发、测试、发布与运行全过程的运行控制对象。

结论二：测试与验证成为主要瓶颈。实现成本下降时，判断“是否做对、是否可交付、是否可追责”的成本与重要性同步上升。公开基准与工业任务在任务分布、环境约束和验收口径上不同，基准分数不能直接代表可交付能力。需求表达、上下文质量和测试、评测与验证，共同制约 ASE 的实际产能。

结论三：生产能力来自系统，而非模型本身。生产结果由“模型 + 上下文 + 运行控制框架 + 工具 + 权限 + 执行循环 + 测试验证 + 治理”共同决定。模型提供基础能力，系统决定这些能力能否转化为稳定的工程产出。模型能力的提升须与规范、测试、验证和治理形成闭环，才能转化为可持续的生产能力。

A.4 主链路

ASE 的主链路形成从意图到证据、再由证据回到意图的闭环：



给读者的三点建议：第一，以“可交付工件的比例”和“证据完备度”衡量 ASE 成效，而非“代码生成量”。第二，从规范与验收标准入手，反向约束工具与流程，不要先建平台再补规范。第三，不要追求全域自主，应先在风险可控、可观察、可回滚的场景中建立最小闭环，再逐步扩大自主范围。

第一篇 WHY：为什么软件工程必须改变

第 1 章 从"人写代码"到"人定义、Agent 实现"

本章回答：LLM/Agent 带来的变化，为什么是范式变化，而不只是效率工具的升级？

1.1 能力跃迁：从代码补全到可持续执行工程任务

智能体在软件工程中的能力演进可概括为五种逐步扩展的形态。这些形态并非严格的行业成熟度分级，也不表示所有系统均已达到最后一项：

- 1. 代码补全与问答：**在人类编写过程中提供片段级建议，人始终是唯一执行者与决策者。
- 2. 跨文件修改：**能够理解局部工程结构，在人的逐步指令下完成多文件一致性修改。
- 3. 工具调用：**能够调用编译器、测试框架、包管理器、命令行等外部工具，从"生成文本"走向"改变环境"。
- 4. 任务规划与迭代：**能够将目标分解为子任务，依据执行结果调整计划，形成"观察—决策—行动—验证"的循环。
- 5. 长时程、自主化工程执行：**在受限环境中连续工作较长时间，自主处理异常、回溯失败路径并在必要时中止。

第三种形态是关键分水岭。工具调用使 Agent 从“文本生成器”转为“环境改变者”；第四、第五种形态使其能够持续执行过程性任务，而非仅产出一次性结果。执行任务不等于承担法律或组织责任。

公开基准既反映能力进步，也揭示评测边界。SWE-bench 以真实仓库问题及测试评估修复结果；SWE-Bench Pro 进一步关注长时程任务和数据污染风险。[2][3] 不同基准及其子集在任务难度、样本筛选、运行控制框架和计算预算上存在差异，因此不能将跨基准分数拼接为同口径的增长曲线，更不能据此推断企业中的可合并率或可交付率。组织仍需在本地任务、权限与验收条件下验证生产能力。

1.2 旧范式：以人为主要生产与控制中心

传统软件工程（含敏捷与 DevOps 实践）的隐含前提是：软件生产的主要执行者是人，主要控制中心也是人。其特征包括：

- **知识载体以人为中心：**需求、设计意图、架构约束分布于人员、文档与工程工件中，部分隐含知识依赖协作传递。
- **控制结合人和自动化：**通过代码评审、持续集成、测试与发布门禁，在过程中发现并纠正偏差。

- **质量依靠分层保证**：测试、静态分析、评审和必要的形式化方法共同提供证据，任何单一环节的覆盖均有限。
- **责任依靠制度追溯**：版本记录可标识贡献者，但产品责任仍须通过角色、审批与治理安排明确。
- **实现是重要成本项**：编码与调试耗费人力，同时需求、集成、验证和组织协调也可能成为瓶颈。

这种模式依靠人的专业判断和沟通处理隐含知识与例外，但并不意味着传统工程可以接受粗略规范或不完备测试。ASE 延续这些工程纪律，并将更多隐含判断显式化、结构化，以适应 Agent 的执行方式。

1.3 新范式：以人的意图和责任为边界，以 Agent 为主要执行者

Agent 成为主要执行者后，上述前提需要扩展。Agent 不会自动获得项目历史、团队惯例和业务语境中的隐含知识；约束不足时，它可能以统计上合理而非工程上正确的方式补全模糊要求；其高速产生的大量工件也会增加逐项人工审查的负担。

新范式因此必须重塑控制方式：

- **将隐含知识写入规范**：隐含知识必须显式写入可执行规范，使规范成为人与 Agent 之间的主要接口。
- **通过边界约束与证据验收实施控制**：在保留必要代码评审的基础上，约束 Agent 的行动边界，并要求其产出可供机器人和人类共同检查的证据。
- **以证据链保证质量**：测试、静态分析、形式化验证和运行观测共同提供证据，覆盖范围由软件结果扩展到 Agent 行为轨迹和生产过程；质量结论由证据综合得出，而不依赖对每项产出的人工判断。
- **责任须显式设计**：本白皮书不将 Agent 视为法律、伦理或组织责任主体，责任须落实到组织及其获授权人员。
- **瓶颈可能转向验收**：随着实现效率提升，需求可表达性、上下文质量和结果可验证性可能成为新的制约因素。

1.4 四个结构性变化

变化	表现	直接后果
执行主体变化	Agent 承担主要执行工作，人主导目标、边界与治理	实现效率可能提升，产出不再仅取决于人工编码投入
工作流变化	从以阶段交接为主，转向持续规划、执行、反馈与修复	规范、行动与证据形成可检查的反馈闭环
交付物变化	从代码中心扩展为规范、上下文、约束、代码与证据的生产基线	证据的工程化生产与管理成为新增工作量

变化	表现	直接后果
瓶颈变化	从编码速度扩展到需求表达、上下文质量、结果验证与责任追溯	需要系统性建设规范、上下文、测试验证与治理能力

这四个变化彼此关联：执行主体变化推动 workflow 持续闭环；闭环要求规范、上下文、工件和证据同步管理；由此，工程瓶颈从局部实现效率扩展到整个生产系统的质量与可控性。

1.5 人的角色与不可转移的责任

新范式中，人的主要价值由直接实现转向以下四种职能；这不排除关键模块的人工编码、代码审查或人工接管：

- 需求定义者**：明确目标、边界与验收标准，将模糊意图转化为可判定要求。
- 系统设计者**：确定架构、接口、约束与非功能要求，划定 Agent 的可行动范围。
- 验证者**：设计独立判定机制，依据证据判断结果是否可接受，并按风险审查关键实现。
- 责任承担者**：依适用法律与组织制度承担相应产品、合规、安全与伦理责任。

传统软件工程	Agentic 软件工程
人主导生产与过程控制	人定义意图与责任，Agent 主导执行
需求：文档、契约与工程工件	需求：可检查的工程规范
执行：人主导，自动化工具辅助	执行：Agent 规划、实现、测试、修复
控制：人工评审与自动化门禁	控制：边界约束与风险分级验收
质量：分层测试、分析与评审	质量：产品、轨迹与生产系统证据
交付：代码、文档与验证记录	交付：规范、代码、轨迹与验收依据
瓶颈：实现及组织协作	瓶颈：表达、上下文与验证等

图 1 传统软件工程与 Agentic 软件工程范式对比

在组织中，这四种职能通常由不同角色承担：业务负责人负责价值目标、风险偏好和可接受结果；产品与架构人员把意图转化为可执行边界和系统设计；工程与质量人员负责生产系统、测试机制、验证策略和阶段门；组织管理者负责授权范围、责任归属、审计要求和例外处置。一人可兼任多种职能，但每项职能都须有明确的责任人。

问责不能因采用 Agent 而悬空，应落实到适用法律和组织制度确定的组织及人员。欧盟《人工智能法案》针对其适用范围内的高风险 AI 系统规定人工监督等要求；ISO/IEC 42001 提供 AI 管理体系要求；NIST AI 风险管理框架将治理作为横切职能。[8]—[10] 三者的法律性质、适用对象和实施安排不同，既不能据此推导全球统一义务，也不能认为所有采用 Agent 的软件都自动属于高风险 AI 系统。自主执行可以授权，但责任归属仍须明确。

第 2 章 新范式为什么需要新的工程体系

本章回答：为什么更强的模型，或既有的软件流程本身，都不足以支撑生产级的 Agent 参与式软件生产？

2.1 模型能力不等于生产系统能力

模型足够强，并不意味着工程问题会自动消失。生产结果由以下要素共同决定，任一项缺失都会显著削弱整体表现：

生产结果 = f (模型, 上下文, 运行控制框架, 工具, 权限与沙箱, 执行循环, 测试与验证, 治理机制)

其中，运行控制框架（Harness）组织并落实围绕模型、依托底层运行环境构建的智能体系统的运行行为与控制策略，包括组织模型与工具调用，按任务和策略装配上下文，管理会话与任务状态，强制执行权限、隔离、预算和停止策略，提供观测、验证、检查点及异常恢复机制。它无须改变模型参数，而是通过上下文、工具、状态和反馈，将模型能力转化为受控、可验证、可持续运行的系统能力与工程产出。

开发框架（Framework）提供构建系统所需的组件与抽象；运行时（Runtime）提供进程、容器、队列、网络、存储、资源限制和调度等执行机制。Harness 可以利用开发框架构建，并依托运行时执行，但其核心职责是组织系统行为和落实运行策略，不能直接等同于开发框架或底层运行环境。

上下文（Context）是模型单次调用可见的完整输入，由运行控制框架按任务边界和策略，从指令、规范、任务状态、会话历史、记忆、知识、工具结果和验证结果中选择并装配。它是当前调用的信息视图，不等于系统的全部状态、会话历史或记忆。

执行循环（Loop）是围绕任务目标反复迭代的反馈控制过程，可概括为“观察—决策—行动—验证”：观察上下文与环境状态，决定下一步计划或行动，在权限和策略约束下执行，将结果和验证反馈纳入后续决策。每个循环都应明确目的、进度度量、迭代次数、时间与成本预算、重复状态检测、成功与失败的终止条件、升级阈值及暂停与恢复规则。

执行图（Graph）以节点、边和共享状态表达执行依赖、路由、并行与汇合关系。节点可以是模型调用、工具操作、确定性检查、人工审核或有界子循环。Loop 关注获得结果后如何继续、修正、停止或升级；

Graph 关注下一步执行哪个节点，以及状态和证据如何沿依赖关系传递。执行图可以包含循环，也可以是无环流程，因此本白皮书不将 Loop 与 Graph 作为同义词。复杂任务可按需要引入执行图，简单任务不必增加图式编排。

上述要素是相互协作的工程职责，不是彼此隔离的产品层或严格的集合包含关系。Harness 通常实现执行循环，并可承载执行图的运行控制机制；公式中单列上下文、工具、权限与沙箱、执行循环，是为了强调各自需要独立的工程投入。相关术语及边界见附录 B。[17]

同一模型在不同运行控制框架、上下文质量与测试约束下，产出质量可能明显不同；差异须在相同任务与预算下度量。因此，**ASE 的工程重心在于把上述要素组织为闭环，而非单纯等待模型能力提升。**

已有工程实践通过任务拆分、会话交接工件、增量检查和端到端测试支持长时程执行。[16] 本白皮书将其归入 Harness / Loop 工程：Harness 提供工具、上下文、状态和控制设施，并强制执行策略；Loop 组织观察、决策、行动与验证的迭代，管理预算与终止条件。两者是 ASE 的执行基础，仍需结合规范、独立验收和全生命周期治理。Harness 中的权限、证据、验证与停止等控制设施应独立于具体模型长期保留；用于弥补当期模型弱点的临时脚手架，应在模型能力提升后及时退役，避免形成技术债。

2.2 概率性认知与确定性工程责任之间的矛盾

LLM 输出可能具有非确定性：相同输入在不同运行中可能得到不同结果，模型也可能自信地给出错误结论。工程责任则要求明确的判定标准、风险阈值和责任主体；部分功能可按确定性条件验收，可靠性等性质还需统计证据。并非所有工程性质都能二值化，关键是用可检查的机制管理不确定性。

这种矛盾无法消除，只能通过工程手段管理。基本思路有三：

1. **以确定性机制约束概率性输出**：用类型系统、模式（Schema）校验、契约检查、静态分析与形式化方法，在概率性输出之外建立确定性边界。
2. **以重复与统计获得稳定性**：对非确定性过程进行多次运行与统计聚合，以分布而非单点结果作为判断依据。
3. **以证据替代信任**：不要求 Agent 自证正确，而要求其提供可独立复核的证据。

2.3 新的失败模式和工程债务

Agent 进入生产后，出现了一些传统软件工程中少见或程度显著不同的失败模式：

- **偏差放大**：需求理解或计划偏差在长时程任务中逐步放大，直到终态才暴露，返工成本随之上升。
- **表面合规**：实现通过了测试，但并未满足真实需求；测试本身被修改或被绕过，使“通过”失去意义。
- **隐性回归**：修改通过了全部现有测试，但在未被覆盖的路径上引入缺陷。
- **幻觉依赖**：引用了不存在的接口、库函数或配置项，且在本地环境中未被执行到。

- **架构漂移**：多次局部最优修改累积，使系统逐渐偏离原定架构，且无人整体把控。
- **理解与证据债务**：代码可用但人员不能解释关键设计或重建变更依据，后续维护、审计与故障处置受到影响。
- **治理债务**：权限边界模糊、审计日志不全、回滚路径未经演练，导致事故发生时无法快速止损。
- **交互与执行风险**：包括提示注入、上下文污染、状态漂移、越权或不可逆副作用、无界重试与预算耗尽，以及多智能体之间的依赖失配或冲突。控制应覆盖输入来源、状态、工具和停止条件。

这些失败在单次检查中难以发现，累积后会显著增加系统的长期维护成本，是新范式下工程债务的主要来源。

2.4 传统软件工程需要扩展，而不是被抛弃

ASE 不替代传统软件工程，而是在其基础上扩展。版本管理、代码评审、持续集成、测试分层、缺陷管理和发布管理等实践依然有效，但实现方式需要调整：

传统实践	在 ASE 中的扩展
版本管理	除代码外，规范、提示词、评测集、Agent 配置均须版本化
代码评审	人的评审重点由实现细节转向规范符合性、架构一致性与证据完备性
持续集成	门禁由"构建与测试通过"扩展为"规范校验、测试、评测、安全与合规检查共同通过"
测试分层	新增面向 Agent 行为与生产过程的评测层
发布管理	发布决策依据由人工判断转为证据驱动的智能门禁加人工按风险放行

传统软件工程关注“如何可靠地构建软件”；ASE 还须解决“如何可靠判断机器构建的软件是否可信”。

2.5 本篇小结

第一，软件生产范式已发生结构性变化，根源是执行主体和控制方式改变，而非单点效率提升。第二，模型能力是必要条件而非充分条件，生产能力来自系统工程。第三，风险贯穿需求、执行、验收和运行，不能留到最终验收再处理。第四，既有工程体系需要扩展，版本、评审、测试与发布的核心纪律仍须坚持。

据此，本白皮书在以下两篇中依次回答：ASE 是什么（WHAT），以及首版应当优先建设哪三项核心能力（HOW）。

第二篇 WHAT: Agentic Software Engineering 是什么

第 3 章 ASE 的定义、对象与边界

本章回答：ASE 研究和建设的对象是什么，又不是什么？

3.1 正式定义

在第 A.2 节一句话定义的基础上，本白皮书给出 ASE 的正式定义：

ASE 是围绕“人定义、Agent 执行”建立的软件工程方法与生产体系：在人类设定的目标、规范、约束与验收标准下，由智能体承担软件工程任务的主要执行工作；工程系统以可执行规范、受控执行、测试评测和验证保证管理上下文、过程与工件，形成可审计证据；组织及获授权人员基于风险对结果进行验收、放行与问责。

该定义强调三点：第一，**Agent 的执行**须受人类授权边界约束，越界属于需要防控的失效；第二，**证据与代码同等重要**；第三，**人类问责**是必要条件，但不要求每次执行都由人点击批准。低风险任务可按事先批准的策略与门禁自动放行，高风险和例外须由人明确决策。预授权也是人事先作出的验收决策：获授权人员确定验收标准、放行策略、抽检与升级条件，并对结果负责。因此，自动放行不改变“人最终验收”的原则。

3.2 两类工程对象

ASE 涉及两类相互关联的工程对象：

对象一：利用 Agent 开展软件工程（Engineering with Agents）。Agent 参与需求、设计、实现、测试和修复，交付普通软件或包含 AI 的软件产品；产品本身不必是智能体系统。

对象二：智能体系统本身的工程化（Engineering Agentic Systems）。研究对象包括模型、运行控制框架、工具、上下文、状态、记忆、执行循环、协作与运行治理。用于生产软件的 Agentic 系统属于其中一种。

两类对象共享规范、测试、验证、证据和治理能力，但不能相互替代。采用 Agent 生产软件的组织，仍须同时保证所交付软件的质量与 Agentic 生产系统的可控性；这两个保证对象贯穿后续章节。

3.3 与相邻概念的边界

概念	核心焦点	与 ASE 的关系
AI4SE / AI 辅助软件工程	以 AI 辅助需求、编码、测试等软件工程活动	与 ASE 有交集；本白皮书强调 Agent 持续执行与端到端治理闭环
智能体编码（Agentic Coding）	Agent 完成局部编码任务	与 ASE 在编码活动上重叠；仅完成局部编码不足以覆盖全生命周期
ASE（本白皮书）	全生命周期的工程范式，含规范、执行、验证、治理与问责	—
通用 AI 智能体	跨领域任务自动化	与 ASE 在工具调用与执行循环上共享技术，但不以软件工程工件与质量为对象
MLOps / LLMops	模型及 LLM 应用的开发、评测、部署与监控	与 ASE 在模型、评测与运行治理上交叉；ASE 还覆盖软件生产的规范、工件与责任闭环
DevOps / 平台工程	交付流程自动化与平台能力建设	ASE 落地的承载基础；ASE 强化面向 Agent 的规范、执行控制与证据要求

研究文献还区分面向人的软件工程（SE for Humans）与面向 Agent 的软件工程（SE for Agents），并讨论人类指挥环境和 Agent 执行环境。[1] 这一划分关注工程环境服务的主体，不等同于本节的两类工程对象，但其“人和 Agent 均需专门工程支持”的思想可供借鉴。

3.4 首版范围与非目标

本白皮书覆盖的范围：ASE 的定义、总体框架、能力体系、全生命周期，以及三项首版核心能力（规范定义、测试与评测、验证与可信保证）。首版以软件生产（Coding）为切入场景；方法论向经营管理系统等其他业务领域的推广留待后续版本讨论。

本白皮书不覆盖的内容：具体算法与模型架构；工具与供应商选型；完整参考实现；多智能体协作协议；部署与运行时的详细架构；标准与认证体系的具体条款。这些专题在总体框架中保留位置，技术细节留待后续版本展开。

非目标声明：本白皮书不主张任何特定产品或技术方案的优越性，不提供模型能力排名，不对市场规模作出预测。所有引用的外部数据仅用于说明趋势与机制，不构成采购或投资决策依据。

第 4 章 ASE 总体框架、能力要素与核心能力

本章回答：新范式由哪些层次、能力和控制机制组成？

4.1 从需求到证据的总体闭环

ASE 的总体运行逻辑形成闭环：**人的业务意图被编译为可执行规范，规范驱动 Agent 在受控环境中规划与执行；执行产生工件与过程轨迹，测试、评测与验证检查工件和过程并生成证据；证据支持验收决策，验收结果与运行反馈再回流至规范和知识，驱动下一轮演进。**

该闭环的关键不在于自动化程度，而在于**每一步都有明确的输入、输出与判定标准。**

4.2 四层参考框架

本白皮书提出的四层参考框架用于描述责任与能力，不代表四套独立技术栈，也不是单向串行流水线：

第一层：意图与规范层。 承载业务目标、价值判断、责任边界及其可执行表达，产出需求、任务契约、约束和完成定义（Definition of Done）。人主导确认意图，Agent 可辅助澄清与结构化。

第二层：Agent 执行层。 在模型、Harness、工具、上下文和权限约束下完成观察、决策、行动与验证的迭代，要求执行有预算与终止条件、可停止、异常可恢复、关键轨迹可追溯。

第三层：软件生命周期层。 组织定义、设计、构建、验证、发布与运行、演进与退役各阶段的工件、活动和阶段门，保证跨阶段的一致性和可交接性。

第四层：保证与治理层。 通过贯穿各层的测试、评测、验证、风险控制和审计，形成质量声明、证据与放行决策，落实组织和人员的责任。

规范、行动、工件、证据与反馈将四层连接起来：意图约束执行，执行和生命周期活动产生工件，保证与治理层独立检查各阶段，并将结果反馈到规范和生产系统。证据用于支持其适用范围内的结论，并不自动构成数学证明。

4.3 六类能力域及关键要素

表 1 六类能力域、关键要素与主要输出

能力域	关键要素	主要输出
1. 需求与规范工程	目标、约束、验收标准、非功能要求、策略、追踪关系	可执行规范、任务契约、完成定义

能力域	关键要素	主要输出
2. 上下文与知识工程	上下文、会话、状态、记忆、知识、版本与来源标注	可用且可追溯的任务上下文
3. Agent 执行工程	模型、运行控制框架、工具、权限、沙箱、执行循环、预算与终止、异常恢复	受控、可停止、关键轨迹可追溯的执行过程
4. 软件工件工程	代码、配置、数据、测试、文档、依赖与供应链	一致、可构建、可维护的版本化工件
5. 测试验证与证据	测试、评测、判定基准、静态/动态/形式化验证、证据链	可复核的质量结论与验收依据
6. 运行治理与演进	发布门禁、观测、事件响应、安全治理、反馈与退役	可运营、可审计、可持续改进的系统

六类能力域横跨四层，不能视为六个互不重叠的串行步骤。组织可按风险确定建设深度，但必须明确每类能力由谁提供，用哪些工件和证据验收。

4.4 首版重点建设的三项核心能力

本白皮书在六类能力域之上聚焦三项贯穿性核心能力；其中，测试与评测、验证与可信保证共同支撑第五类能力域：

能力一：规范定义。 解决“Agent 知道做什么、不能做什么、做到什么程度算完成”的问题，是贯穿闭环的控制中心。

能力二：测试与评测。 解决“如何可靠判断软件与生产过程是否达到要求”的问题，是可交付产能的重要约束。

能力三：验证与可信保证。 解决“如何证明或持续确认关键边界未被突破”的问题。它决定 ASE 能够进入何种风险等级的应用场景。

三项能力形成“规范—执行—测试与验证—证据验收—规范修订”的最小质量闭环。必要的上下文管理、受控执行和运行治理也应同步具备；多智能体等复杂能力可按需逐步扩展。

4.5 五项工程原则

- 1. 规范驱动原则：**任务执行前明确目标、边界与验收条件；未澄清的关键要求不得由 Agent 自行猜测。
- 2. 风险约束原则：**通过最小权限、隔离、预算、停止条件及回滚或补偿措施限制影响；不可逆的高风险动作应前置审批，必要时禁止自主执行。

3. **证据闭环原则**：对关键要求建立可追溯的工件、独立判定、质量声明与批准记录；验收不得只依赖执行者自评。
4. **人类问责原则**：明确组织及获授权人员的授权、风险接受与例外处置职责，自动化不免除问责。
5. **持续演进原则**：把失败、运行事件与评测结果回流到规范、测试和治理策略，并按证据调整自主范围。

4.6 人、Agent 与平台的责任边界

三方分工如下：人确定目标并承担责任，Agent 负责理解任务与编排工作，验证和回滚由人、Agent 与平台共同建设和使用。任何一方都不能单独宣告“完成”。具体边界如下：

主体	应当承担	不应承担
人	定义目标与验收标准；划定约束与权限；基于风险验收与放行；对外承担责任	以人工检查替代全部自动化检查，或不分风险地逐行审查所有产出；关键实现仍须按需人工复核
Agent	在给定边界内规划与执行；产出工件与过程证据；在越界或不确定时中止并请求澄清	自行设定价值目标；自行放宽验收标准；自行决定风险接受
平台	强制落实权限与门禁；执行预先批准的自动放行策略；记录证据；提供停止、恢复及受限回放能力	自行决定目标、放宽风险标准、执行未经批准的自动放行策略，或代替组织及人员承担责任



图 2 ASE 四层参考框架与六类能力域

第 5 章 全生命周期、关键工件与质量闭环

本章回答：ASE 如何覆盖从业务意图到运行演进的全过程？

5.1 六阶段全生命周期

本白皮书将 ASE 的全生命周期划分为六个阶段：

- 1. 定义：**明确业务目标、范围、责任与风险；将意图编译为规范、任务契约与可判定验收条件。
- 2. 设计：**确定架构、接口、上下文与权限边界，制定验证策略、发布策略及回滚或补偿方案。
- 3. 构建：**Agent 在受控环境中规划、实现、测试与修复，产出版本化工件、依赖清单与变更轨迹。
- 4. 验证：**独立检查工件、轨迹与生产能力，汇总测试、评测、分析及必要的形式化证据，处理残余风险。
- 5. 发布与运行：**按授权策略与风险等级放行，执行灰度、监控、事件响应及回滚或补偿。
- 6. 演进与退役：**将失败与运行反馈回流至规范和测试，管理知识更新、系统迁移、权限撤销、数据处置与服务退役。

5.2 各阶段的最小工件与阶段门

表 2 六阶段生命周期的最小工件与阶段门

阶段	最小工件	阶段门 (Gate)
1. 定义	目标与风险说明、规范、任务契约、验收条件	范围与责任明确；关键需求可判定
2. 设计	架构与接口、权限边界、验证及回滚/补偿方案	边界可执行；方案可验证；不可逆风险已审批
3. 构建	版本化工件、依赖清单、执行轨迹、变更记录	可构建；变更受控；关键轨迹可追溯
4. 验证	测试评测报告、验证证据、残余风险清单	关键标准达标；独立复核完成；残余风险获授权处置
5. 发布与运行	放行记录、运行策略、观测与事件记录	依风险授权发布；监控和止损措施可用
6. 演进与退役	失败样本、规范与知识更新、迁移及退役记录	反馈已闭环；迁移与数据/权限处置经责任人确认

“最小工件”指每个阶段应具备的可复核依据，不要求每项都形成独立文档。组织可合并工件；不适用项应说明理由、风险和批准人，不能静默省略关键控制。

5.3 贯穿生命周期的证据链

证据链（Evidence Chain）连接需求与结论，核心要求是**双向可追溯**：

- **正向追踪**：从任一需求，可找到其验收标准、对应测试用例、执行结果、相关代码变更与验收人。
- **反向追踪**：从任一代码变更，可回溯其触发的任务、对应需求、所依据的规范条款与验证证据。

证据链可表示为 Intent（意图）→ Specification（规范）→ Action（行动）→ Artifact（工件）→ Verification（验证）→ Claim（质量声明）→ Approval（批准）→ Audit（审计）。每项证据至少应标明来源、版本、时间、责任人、适用范围和关联要求，并注明质量声明的局限及残余风险。关键轨迹包括工具调用、权限使用及结果，无须暴露模型内部推理。证据应有完整性保护和访问控制，按法规及组织政策确定留存期限，并随工件版本更新。



5.4 质量目标

ASE 的质量目标应覆盖正确性、安全性、可靠性、可控性、可追溯性和可演进性，并落实到产品与生产过程两个维度：

产品维度：功能正确性、性能与资源效率、可靠性与可用性、安全性、可维护性、合规性。

过程维度：

- **可复现与可回放性**：固定可控版本与输入，留存足以复核关键动作和判定的记录，并对非确定性结论报告统计稳定性。
- **可解释与可控性**：记录关键动作的授权依据和来源，并具备停止、限权、接管与异常处置机制。

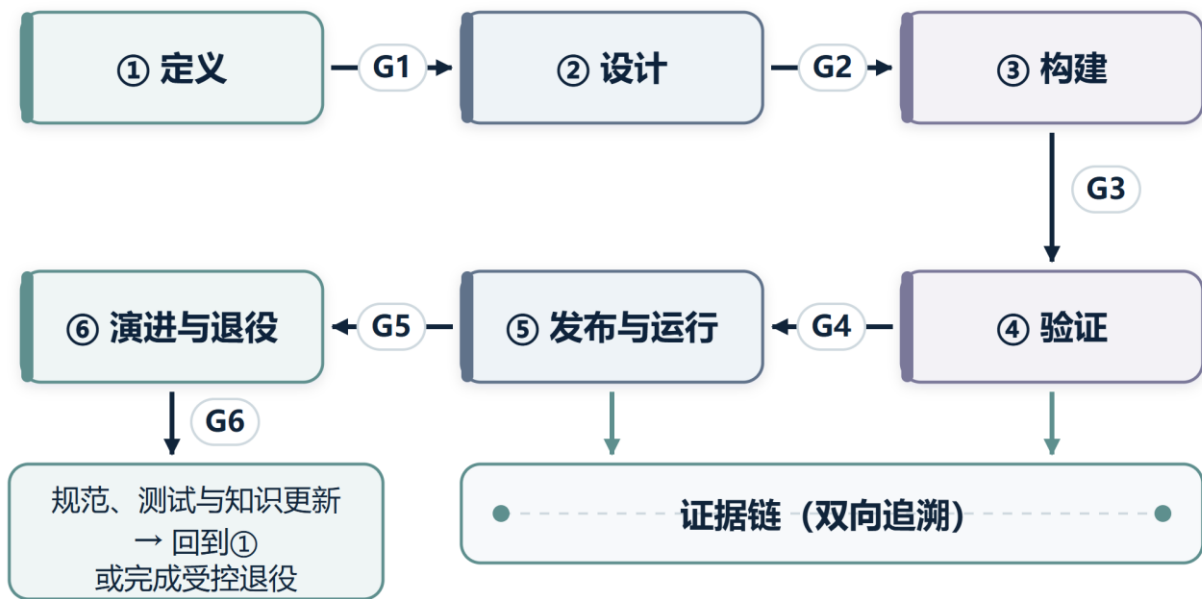
- **可审计性**：获授权的第三方可依据证据重建关键行动与批准链；证据不足之处必须显式披露。
- **可演进与可恢复性**：已发布变更具备与风险相称的回滚、补偿、迁移或止损方案，并维护到退役阶段。

在非确定性系统中，可复现性并非逐字节相同，而是可控条件下的统计稳定性和对可观察过程的受限回放；不能重演的外部依赖须明确说明。

5.5 生命周期中的人机责任门

人机责任门用于落实授权、风险接受与人工接管。**风险越高、可逆性越低、影响面越广**，关卡越应前置。责任门至少覆盖规范确认、发布放行和运行异常处置：低风险事项可按已批准策略自动判定并接受抽检；高风险和例外须由获授权人员明确批准。持续监控可触发人工复核，无须在每次低风险变更后人工签字。

责任门可采用策略授权记录、审批或强制确认等形式，但必须明确责任人、适用范围、版本和时间戳；风险等级、权限或验证证据发生实质变化时，应重新评估授权。



阶段门释义

G1 规范与责任明确

G2 方案及权限可验证

G3 工件与轨迹可复核

G4 证据及风险获处置

G5 运行风险受控

G6 演进/退役确认

图 3 六阶段生命周期、阶段门与证据闭环

第三篇 HOW：三项核心能力如何构建

第 6 章 规范定义：把人的意图转化为 Agent 可执行的工程契约

本章回答：怎样让 Agent 知道“做什么、不能做什么、做到什么程度才算完成”？

6.1 规范从说明文档升级为运行时控制对象

传统软件工程通过文档、契约和人员沟通表达需求。在 ASE 中，Agent 未必能主动发现歧义或正确追问，缺少明确的澄清机制时可能自行补全模糊要求。规范因此须成为**运行时控制对象**，在开发、测试、发布与运行中持续被引用、检查和追踪。

规范因此要让人能理解、机器能读取、结果能检查、版本能追溯。并非所有自然语言要求都必须完全形式化，但安全红线、接口和权限等关键条款应以结构化契约、策略或断言表达；开放性要求应给出明确的人工或组合判定程序。

6.2 四类最小规范

本白皮书建议，任何交由 Agent 执行的任务，至少应提供以下四类规范：

第一类：目标与验收规范。 说明要达成什么、如何验收，包括业务目标、范围、功能要求、优先级及成功判据。

第二类：行为与质量规范。 说明必须满足的行为和质量边界，包括安全、隐私、性能、可靠性、资源预算与可维护性要求。

第三类：接口与工具规范。 说明允许的输入输出、接口前置/后置条件、工具参数、授权范围、外部副作用、幂等性及错误处理。

第四类：保证与责任规范。 说明测试验证方法、完成定义、证据要求、判定人、审批人，以及停止、澄清、接管和回滚或补偿条件。

四类规范共同构成任务契约。完成定义不能只写“任务完成”，而应列明可检查的工件、证据和批准条件。契约不完整时，应先澄清再执行。

6.3 "需求编译"过程

需求编译（Requirements Compilation）将自然语言表达的业务意图、规则、约束和示例，逐步转化为可执行、可验证、可追踪的软件工件 [18]。它借鉴传统编译器把高级语言编译为机器程序的思路，建立从人的意图到软件系统的系统化转换链路，而不只是整理结构化需求文档。

需求编译可以分为以下几个阶段：

- 意图获取与交互式澄清。**从自然语言描述、业务文档、会议记录、现有系统、用户操作示例和原型等来源获取需求。对于歧义、缺失、冲突和隐含假设，系统应通过提问、原型反馈和示例确认等方式主动与用户交互，逐步明确真实意图。
- 需求建模与中间表示。**将自然语言需求编译为机器可处理的需求模型，涵盖实体与数据关系、业务规则、状态迁移、用户场景、接口契约、质量约束、权限及跨需求依赖。这种表示类似于传统编译器的中间表示（IR），是连接自然语言意图与后续软件生成的核心载体。
- 需求分解与约束传播。**将高层业务目标逐步分解为场景、任务、接口、数据、行为和质量约束，并维护各层级之间的依赖关系。上层需求变化时，相应约束能够传播到受影响的软件工件。
- 软件工件生成。**从统一的需求表示进一步编译出或驱动生成数据库模式、API、用户界面、测试用例、配置、权限规则乃至实现代码。需求由此从开发活动的参考文档转为软件构造过程的直接输入。
- 可执行验证与需求调试。**将需求同时编译为可执行的验收条件、测试、断言、约束和检查程序，用于验证生成的软件是否符合用户意图。测试失败或行为出现偏差时，应能反向定位相应需求、约束或需求冲突，从而调试需求本身。
- 增量编译与双向演化。**用户修改需求时，不应重新生成整个系统，而应分析变更的影响范围，仅重新编译受影响的设计、接口、代码和测试。软件实现和运行反馈也应能反向映射到需求模型，形成“需求—软件—验证结果”持续演化的闭环。

需求编译输出连接用户意图与软件实现的中间表示、约束和可执行工件，而非仅一份更规范的需求文档。目标是建立以下链路：



这构成一条完整的编译链路。

需求编译试图解决一个更基础的软件工程问题：如何在把需求写清楚的同时，让人的需求成为可直接驱动软件生成、验证和持续演化的计算对象。

6.4 规范质量与冲突处理

可用以下指标检查规范质量：

- **覆盖率**：每一条需求是否都有对应的验收条件？
- **可判定性**：每条验收条件是否都有预先规定、可复核的自动、人工或组合判定程序？
- **一致性**：不同规范条目之间是否存在冲突？
- **可追溯率**：每一条规范是否都能追溯到业务来源，并能追踪到实现与证据？
- **可行性与优先级**：要求在给定预算、技术和权限下是否可实现？优先级与冲突裁决权是否明确？

冲突处理遵循三条规则。第一，**约束优先于目标**：安全、合规、隐私等红线条款不得因功能目标而放宽。第二，**显式优于隐式**：必须标出冲突条款，不得由 Agent 自行取舍。第三，**升级优于猜测**：无法自动消解的冲突必须交由人裁决，Agent 不得自行决策。

6.5 规范变更与影响分析

在 ASE 中，规范变更的影响远大于传统模式：一次措辞调整可能改变 Agent 的整体执行路径。因此须建立以下变更纪律：

- 规范变更必须版本化，记录变更人、时间、原因与影响范围。
- 规范变更须分析其对架构、代码、测试、证明假设、授权及运行策略的影响，标记失效证据，并重新执行受影响的验证；复用未受影响的证据须有依据。
- 对高风险的规范条款，建议设置变更审批，避免在执行过程中被就地修改。
- Agent 在执行中提出的规范澄清请求应记录并统计，作为规范缺陷的信号，用于反向改进规范质量。

6.6 最小实践模板

组织可从以下最小集开始，无需等待完整平台建成：

1. **任务契约模板**：每个任务列明目标与范围、输入、行为与质量约束、接口、工具和权限、完成定义和证据、责任人与升级条件，以及回滚或补偿方式。
2. **验收条件清单**：每条需求对应一行，包含唯一标识、判定方式、当前状态、证据位置。
3. **规范校验脚本**：在任务启动前自动检查规范完整性，缺失必填项则拒绝启动。

4. 规范变更日志：记录每次变更的影响范围与重验结论。

5. 澄清请求台账：汇总 Agent 提出的澄清请求，作为规范质量的反向指标。

最小任务契约示例（示意，非通用阈值）：R-01 要求为现有订单查询接口增加分页，并保持旧调用兼容。输入包括接口规范 v1、脱敏数据和测试基线。只允许修改查询模块及经评审的测试，禁止生产凭据、生产数据写入和数据库结构变更。完成定义包括兼容性与分页契约测试通过、指定负载下的时延满足已批准阈值，以及独立复核通过。证据包括提交标识、测试报告、环境版本和授权轨迹。发生需求冲突、越权请求或预算超限时，停止执行并交由接口负责人裁决。

追踪示例：R-01“无分页参数时保持兼容”→ T-01 旧客户端契约测试 → V-01 带提交与环境版本的测试报告 → C-01“在声明接口范围内兼容”→ A-01 责任人批准或预授权门禁记录。失败应回到规范或实现修订，不得自行放宽契约。

本能力小结：规范定义针对意图被误解，以及目标和边界在执行中漂移的风险。输入包括业务意图、约束、组织政策与既有工件；输出包括可执行规范、任务契约、验收条件与追踪关系。需求责任人确认关键解释，规范变更经授权审批。

第 7 章 测试与评测：同时检查结果、过程和生产能力

本章回答：面对非确定性的 Agent，怎样可靠判断软件和生产过程是否达到要求？

7.1 明确测试对象

ASE 应同时检查软件结果、执行过程和生产能力，具体分为以下五类测试对象：

对象一：模型与 Agent。通过代表性任务检查其规划、工具使用、指令遵循和拒绝越界的能力，并记录运行条件与适用范围。

对象二：工具与组件。通过单元、契约、权限和故障测试检查接口、沙箱、状态存储及依赖组件的行为。

对象三：执行轨迹。检查任务推进、授权使用、状态转换、停止条件及异常恢复，识别未经授权的测试修改或副作用。

对象四：软件工件与环境终态。检查代码、配置、数据和部署后的可观测状态，而非只看 Agent 的结束消息。

对象五：业务结果与生产能力。检查用户目标是否达成，并度量成功率、返工、人工介入、成本、时延及严重失败，据此判断系统能否持续交付。

五类对象分别提供不同证据：工件正确不能抵消越权行为，Agent 自报完成不等于环境达到预期终态，单次成功也不能证明持续生产能力。[16]

7.2 分层测试与评测方法

建议组合使用以下四类方法，不将其视为严格互斥的测试层级：

- 1. 单元与契约测试：**验证函数、模块和接口契约。Agent 可大量生成这类测试，但必须由规范驱动，且需检查测试本身的有效性。
- 2. 集成与系统测试：**验证模块间协作和端到端行为。这类测试是发现隐性回归的主要手段。
- 3. 性质与不变量测试：**在生成或选取的输入和状态下检查幂等性、守恒性、权限等性质，并用故障注入、对抗样本和长时程场景寻找反例。测试通过只支持已覆盖范围，不能证明性质始终成立。
- 4. 行为与能力评测：**在代表性任务集上多次运行，并结合回归回放、影子运行和灰度试验评估系统表现。预先规定的安全与能力阈值可作为版本发布门禁；趋势分析用于长期改进，但不能替代硬性门禁。

性质测试能系统生成边界情形，仍须与明确用例、集成测试和端到端测试互补。方法应按风险、可判定性和覆盖目标选择；代码由概率模型生成，并不意味着传统测试的价值必然下降。

7.3 判定基准（Test Oracle）：谁来判断“正确”

判定基准（Test Oracle）是判断被测对象行为是否正确的依据。在 ASE 中，必须明确设计其来源。以下五类来源各有局限：

判定基准	适用场景	主要局限
规范导出的断言	功能与契约验证	依赖规范质量；规范模糊则断言无效
既有实现作为参照	重构与迁移	既有实现可能有缺陷，会被一并继承
性质与不变量	并发、分布式、状态机	需要抽象能力，性质本身可能不完整
人工标注与专家评审	主观质量、体验、合规	成本高、不可规模化、存在主观差异
统计与分布比较	非确定性输出、生成质量	只能给出置信度，不能给出确定性结论

核心纪律：不得将 Agent 自评作为唯一放行依据。应优先使用编译器、Schema、规则、测试和性质断言等确定性判定。开放性任务可综合参照实现、差分结果、独立模型评分和专家抽检，并校准评分器。生产者与判定者应在权限、数据和评价流程上隔离；仅更换模型并不构成独立性。缺乏可靠判定基准时，应缩小任务范围、降低自主级别或升级至人工判断。[16]

7.4 统计纪律与可重复性

非确定性系统的能力与可靠性不能由一次成功推断。固定条件试验用于复现故障，重复抽样试验用于估计波动和失效率，两者应予区分。统计方案须说明估计对象、任务抽样和重复运行方式。

- **多次运行**：对非确定性能力，按任务分布开展重复试验，报告运行次数、成功率、波动、置信区间和尾部失败，并说明样本相关性、重试次数和预算。区间估计按任务等相关单元处理，不能把同一任务的重复尝试视为新的独立任务。
- **固定可控条件**：记录模型、Harness、提示词、工具、数据、环境、参数、规范及评测集版本；可控制时**固定随机源**，但随机种子不保证跨平台或服务版本完全复现。
- **防止污染**：隔离开发集、回归集和保留评测集，保护隐藏用例，记录公开任务暴露情况并定期更新。无法核实闭源模型训练数据时，应披露污染不确定性，不宣称已实现完全隔离。
- **隔离与保护**：离线评测与开发、生产环境保持隔离，并保护隐藏测试与判定器。影子、灰度等受控生产试验须另行授权，并具备数据边界、监控和止损措施。
- **区分趋势与红线**：能力均值用于趋势判断，严重安全失败、越权或数据泄露应立即触发阻断和调查。平均成功率不能代替严重失效风险的上限评估。

评测结论应说明**环境、任务分布和局限**。[3][16] 同时记录质量、成本、时延、人工介入、回滚和严重失败，避免只优化平均成功率。有限样本未观察到严重失败，不代表真实风险为零。

7.5 证据驱动的发布门禁

发布门禁（Release Gate）应由一系列可自动检查的条件组成，建议包含：

门禁项	判定内容
规范符合性	关键要求达到阈值；非关键未满足项有显式、获授权的残余风险处置
测试通过性	适用测试通过；非确定性能力达到预先规定的统计门槛
覆盖率	变更部分达到组织设定的覆盖要求
安全与合规	适用安全、依赖、许可证及敏感数据规则通过，无未处置的严重问题
过程合规性	关键轨迹完整，无越权、隐藏评测泄露或未经授权削弱测试的行为
恢复与止损	回滚、补偿或止损措施与风险匹配，必要演练已完成
授权放行	低风险按预授权策略放行；高风险及例外由指定人员批准

门禁默认拒绝。例外须显式授权，并记录原因、责任人、残余风险、有效期和补偿措施；法律禁止事项及不可豁免的安全红线不得例外放行。门禁失败时，应返回规范、实现或验证环节修正，禁止通过无限重选筛选偶然成功。

7.6 最小实践模板

1. **测试基线**：按任务风险选取单元、契约、集成、端到端与性质测试，说明不适用项并纳入门禁。
2. **行为评测集**：先选择一组具有代表性的正常、边界和失败任务开展试点，再按风险覆盖和统计要求扩展。少量任务可用于起步，但不足以证明系统在低概率严重失效方面的安全性。
3. **防篡改检查**：识别断言弱化、错误静默和隐藏评测访问；允许经独立评审批准的测试维护，禁止 Agent 以改动判定标准掩盖缺陷。
4. **多次运行报告模板**：记录版本、任务与试验次数、成功率及置信区间、失败分类、成本、时延、人工介入与严重失效。
5. **门禁清单**：可勾选、可留痕、可审计的发布前检查表。

最小评测矩阵示例（对象 / 方法 / 判定基准 / 证据）：模型与 Agent / 代表任务重复运行 / 批准的评分规则 / 试验分布报告；工具组件 / 契约与故障注入 / 接口断言 / 组件测试记录；轨迹 / 对抗场景与回放 / 权限及停止策略 / 行动审计；工件与终态 / 集成及端到端测试 / 验收断言 / 带版本结果；业务结果 / 影子或灰度试验 / 业务目标与专家确认 / 运行指标和批准记录。

门禁示例：对于指定候选版本、环境和任务范围，要求关键契约全部通过，不存在未处置的严重安全失败，且回归指标达到预先批准的阈值。独立判定人核对证据完整性，风险责任人批准可接受的残余风险。证据、适用范围、责任人以及通过/失败状态均随放行记录保存。

本能力小结：测试与评测针对结果不正确、过程越界或生产能力不可持续，却被判为可交付的风险。输入包括规范与验收条件、工件、执行轨迹、评测集和判定基准；输出包括测试与评测报告、统计报告和门禁记录。独立判定人核对证据，风险责任人批准残余风险并对放行负责。

第 8 章 验证与可信保证：用分层证据约束高风险行为

本章回答：除了测试之外，怎样证明或持续确认 Agent 和软件没有突破关键边界？

8.1 区分“做对的事”与“把事做对”

质量保证包含两类不同的问题：

- **“把事做对” (Verification)**：实现是否满足规范？即“是否按规范实现了”。
- **“做对的事” (Validation)**：规范与最终软件是否满足真实用户需求及预期用途？不仅确认规范值得实现，也检查交付结果是否解决业务问题。

测试、评审、用户试验和分析可分别为 Verification 与 Validation 提供证据。即使错误规范得到实现和验证，交付的产品仍可能错误。因此，须持续确认规范与业务意图一致，并确认运行结果适合预期用途。Agent 既可能误解规范，也可能执行错误规范，不能预设它会忠实且正确地实现要求。

8.2 风险分级决定保证强度

保证强度应当与风险等级匹配，既不应对低风险变更施加过度成本，也不应对高风险变更仅依赖测试。

表 3 风险等级、验证手段、证据要求与人工关卡

风险等级	典型场景	主要验证手段	证据要求	人工关卡
R1 低	内部工具、文档、样式调整	单元测试、自动门禁	适用检查、策略授权与关键动作记录	预授权自动放行，配合抽检与异常升级
R2 中	内部业务功能、非核心服务	单元 + 集成 + 性质测试、静态分析	测试、独立复核、故障注入及轨迹证据	指定责任人复核；按风险灰度及回滚
R3 高	面向客户的核心功能、数据处理、权限变更	契约、策略/数据流分析、运行监控；关键性质按需形式化	完整证据链 + 独立复核	强制人工放行
R4 极高	资金与支付、人身安全相关、关键基础设施、不可逆操作	关键性质建模或证明、独立验证与确认、受限执行	适用形式化证据、假设、独立复核及演练记录	双人/多级批准；灰度与止损；必要时禁止全自主

表 3 是示例分级，不是法定分类，场景名称也不自动决定等级。组织应综合影响范围、数据敏感度、权限、可逆性、失败成本和适用法规划级，并固化权限与门禁。Agent 可按批准规则识别升级信号，但不得自行降级或放宽边界。

8.3 分层验证机制

建议组合以下四类互补检查，并由独立人员或专家全程复核：

- 1. 类型与模式层：**通过类型系统、Schema 和接口契约检查排除结构性错误。应优先覆盖关键接口，并区分结构有效与语义正确。
- 2. 静态分析层：**静态分析、符号执行和抽象解释可用于发现缺陷，或在工具能力和假设范围内证明特定性质。这些方法可能存在误报、不完备和路径限制，报告须区分已证实、反例和未决结果。
- 3. 动态验证层：**通过测试、模糊测试、性质测试和运行时断言检查执行行为。对于经受控工具通道且可观测、可判定的动作，结合沙箱、权限、状态和预算监控，尽量在副作用发生前拦截，并处理监控盲区和失效情形。
- 4. 形式化验证层：**对明确模型与假设中的关键性质给出机器可检查证明，或在声明的状态/搜索范围内进行模型检测。

这些机制的成本和保证强度并非严格递增，静态分析和运行时验证本身也可采用形式化方法。应根据风险、覆盖范围和可执行性组合使用，而不是用某一种方法替代其他证据。

8.4 优先形式化的对象

形式化方法的成本因对象、性质和工具而异，不宜要求整个 Agentic 系统一次性完全形式化。建议优先覆盖：

- **并发与分布式协议：**以状态机和时序性质检查死锁、一致性及活性；工业实践已展示形式化建模发现复杂交错缺陷的价值，但不替代实现测试。[4][15]
- **安全与权限边界：**越权、信息流、授权前置条件，以及“批准之后才能执行”的时序要求。
- **Agent 工作流与状态机：**检查执行循环或多智能体协作是否必然到达允许的终态，是否可能死锁、失控或绕过必经关卡，以及关键行动是否必然生成可关联、不可遗漏的审计记录。
- **资源、终止与副作用：**预算上限、停止条件、资金守恒、幂等性、补偿和不可逆操作触发条件。
- **编译器、运行时与关键基础设施：**按影响面选择关键模块验证，参考seL4与CompCert的有界保证实践。[5][6]
- **规范、证据与实现的一致性：**关键条款是否得到实现，批准所需证据是否齐备且版本匹配。

用户体验和真实需求仍需用户验证与专家判断。对于其中可形式化的接口、状态或可访问性约束，也可采用形式化方法，不宜因其属于“界面”而一概排除。

8.5 可采用的形式化方法及渐进路径

可用的形式化方法包括：

- **模式与契约校验** (Schema / Contract)：作为渐进起点检查结构和局部约束；通过校验不等于已证明整体正确。
- **模型检测** (Model Checking)：对有限状态模型穷尽检查时序性质，适用于协议与状态机。
- **定理证明** (Theorem Proving)：在形式系统中构造可检查证明；保证范围取决于性质、假设、实现对应关系及可信计算基础，成本依具体任务而定。
- **抽象解释与静态证明**：自动推导程序性质，可扩展性较好。
- **运行时验证** (Runtime Verification)：将关键性质编译为监控器，在运行中持续检查并在违反时触发处置。

建议的渐进路径为：



组织不必从完整证明起步。模式校验与运行时监控可显著提高保证水平；待规范成熟、团队具备相应能力后，再逐步引入模型检测和关键性质证明。

8.6 总体保证模型

本白皮书提出 ASE 的总体保证模型：**形式化内核 + 统计外壳 + 运行证据**。

- **形式化内核**：在形式化模型和假设范围内证明或检查关键性质，并通过实现对应检查和强制执行机制落实边界；不构成对整个真实系统的无条件保证。
- **统计外壳**：对可观测行为与剩余不确定性进行代表性评测，给出带置信度的判断，包括与安全相关的经验性证据。
- **运行证据**：持续观测真实条件下的偏差，并结合事前拦截、预警、灰度和事件处置，检验和修订已有结论。

三者共同支撑有边界的质量声明。形式化证明不能替代真实需求确认；统计结果不能保证未覆盖情形；运行观测也不能覆盖未被监控或尚未出现的风险。因此，应结合互补证据和残余风险管理建立保证。

运行证据层

生产观测 · 事件记录 · 灰度与熔断 · 反馈回流

统计外壳层

多次运行 · 评测集 · 覆盖率 · 置信区间

形式化内核层

契约 · 不变量 · 模型检测
关键性质证明（限定假设）

内核 = 假设内的关键性质

外壳 = 统计评估

运行 = 监控、处置与再确认

图 4 “形式化内核 + 统计外壳 + 运行证据”可信保证模型

8.7 形式化验证的边界与诚实声明

引入形式化验证时，必须明确说明其局限：

- 结论只覆盖明确性质和假设。形式化分析可能发现规范冲突，但不能自动保证规范完整或忠实于自然语言业务意图。
- **证明对象通常是模型而非实现。**除非建立从模型到实现的严格对应关系，否则证明结论不能直接迁移到实际系统。
- **成本**取决于对象与目标。轻量契约、模型检测与完整机器证明的投入差异很大，不应以未经限定的倍数概括所有形式化实践。[7]

· **工具链本身也可能出错。**证明辅助工具和求解器的可信性，是证明结论可信性的前提。

质量声明必须绑定规范、模型和实现版本，并列出已验证性质、工具与配置、假设、未覆盖项和残余风险。声明还须说明模型到实现的对应关系。规范、模型、工具或环境发生变化时，应评估原有结论是否失效；“已通过形式化验证”不能脱离这些边界，被用作整体安全承诺。

8.8 最小实践模板

1. **关键性质清单：**依据风险识别少量优先验证的性质，标识责任人、形式化对象、假设与强制执行点，不设通用条数门槛。
2. **契约与模式校验：**优先覆盖关键跨服务接口，作为结构化检查的起点；结构检查通过不等同于整体功能正确性证明。
3. **运行时监控器：**将可执行的关键性质转化为监控或拦截规则。对于不可逆副作用，优先在发生前阻断，并配套超时、接管和失效处置。
4. **风险分级表：**按表 3 建立本组织的风险分级与对应保证要求，并固化到发布流程。
5. **声明模板：**规范对外质量声明的措辞，强制包含证明范围与未覆盖部分。

高风险示例（支付系统变更）：以“未批准不得部署、重复请求不得重复扣款”为关键性质；对授权状态机和幂等性进行建模或证明，结合并发、重试、故障注入测试检查实现。生产凭据隔离，双人批准后才允许部署，执行前由策略拦截未授权动作，运行中监控金额与状态异常并暂停后续操作。已发生的扣款不能假定可回滚，须有对账、补偿及人工处置。证据包绑定规范、代码、模型、测试、批准和运行版本，并明确未建模的支付网络与环境假设。

本能力小结：验证与可信保证针对关键边界被突破，以及局部证明被外推为整体安全承诺的风险。输入包括关键性质清单、风险分级、形式化模型与假设、运行策略；输出包括验证证据、监控与拦截规则，以及注明假设和未覆盖项的质量声明。获授权人员按风险等级放行，对外质量声明由责任人签署。

结语：从可观察、可验证的最小闭环开始

C.1 对产业的共同主张

本白皮书提出三项共同主张：

其一，关注执行主体的变化。重点不仅是 Agent 能否写代码，还在于产出的软件是否可验证、可运行、可维护，并由明确主体负责。ASE 的目标是保留人的参与，重新划分人、Agent 与工程平台的职责：人确定目标并承担责任，Agent 负责理解与编排，验证与回滚则建成共同能力。

其二，把投资从单点模型能力扩展到工程闭环。模型能力能转化为多少可交付产能，取决于规范、上下文、执行控制、测试、验证和治理的协同建设。

其三，从最小闭环而非全面自主起步。建议先选择风险可控、边界清晰、可回滚的场景，建立包含“规范—执行—证据—验收”四要素的最小闭环，使其可观察、可度量、可改进，再逐步扩大自主范围与场景复杂度。

自主执行可以授权，但责任不能因此悬空。应明确需求确认、风险接受和例外处置的责任人，并由组织及获授权人员依适用法律和组织制度承担相应责任。

C.2 后续版本建议展开的专题

限于首版篇幅与定位，以下专题仅在框架中保留位置，后续技术版或专题报告将进一步展开：

- 规范语言与需求编译、上下文/状态/记忆/知识的工程方法
- Agent 运行控制框架（Harness）的参考架构与能力要求
- 执行循环（Loop）与执行图（Graph）的设计模式及异常恢复机制
- 多智能体协作的角色划分、通信协议与冲突消解
- 行业场景；AI 原生软件研发成熟度模型（5×5×5）与 ASE 能力域的映射；按业务视角划分的 L1—L5 自动化程度；Testbed 和评测基准（Benchmark）建设
- 部署运行、灰度发布与实时熔断的工程实践
- 接口互操作、标准、认证与合规体系的可能对接路径

其中，标准与认证相关的设想属于长期方向，尚不构成产业共识，本白皮书不将其表述为既定结论。

参考资料

1. Hassan A E, Li H, Lin D, et al. Agentic Software Engineering: Foundational Pillars and a Research Roadmap. arXiv:2509.06216, 2025, v3 更新于2026-06-24 (预印本) 。 <https://arxiv.org/abs/2509.06216v3>
2. Jimenez C E, Yang J, Wettig A, et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? ICLR, 2024. <https://arxiv.org/abs/2310.06770>
3. Deng X, Da J, Pan E, et al. SWE-Bench Pro: Can AI Agents Solve Long-Horizon Software Engineering Tasks? arXiv:2509.16941v2, 2025 (预印本; 其抗污染设计不等于已排除全部污染风险) 。 <https://arxiv.org/abs/2509.16941v2>
4. Newcombe C, Rath T, Zhang F, et al. How Amazon Web Services Uses Formal Methods. Communications of the ACM, 2015. <https://www.amazon.science/publications/how-amazon-web-services-uses-formal-methods>
5. Klein G, Elphinstone K, Heiser G, et al. seL4: Formal Verification of an OS Kernel. SOSP, 2009. <https://sel4.systems/Research/pdfs/seL4-formal-verification-operating-system-kernel.pdf>
6. Leroy X. Formal Verification of a Realistic Compiler. Communications of the ACM, 2009, 52(7): 107–115. <https://xavierleroy.org/publi/compcert-CACM.pdf>
7. Bernardi G, Francalanza A, Peressotti M, Mousavi M R. Software is infrastructure: failures, successes, costs, and the case for formal verification. arXiv:2506.13821, 2025, v4 更新于2026-01-29 (预印本) 。 <https://arxiv.org/abs/2506.13821v4>
8. 欧洲议会与欧盟理事会. Regulation (EU) 2024/1689 (人工智能法案) , 2024。须区分原始生效日与各条款适用日期, 并核对官方现行合并文本。 <https://eur-lex.europa.eu/eli/reg/2024/1689/oj>
9. ISO/IEC. ISO/IEC 42001:2023, Information technology — Artificial intelligence — Management system, 2023。 <https://www.iso.org/standard/42001>
10. NIST. Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1, 2023。 <https://doi.org/10.6028/NIST.AI.100-1>
11. NIST. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile, NIST AI 600-1, 2024。 <https://doi.org/10.6028/NIST.AI.600-1>
12. ISO/IEC/IEEE. 12207 《系统与软件工程—软件生命周期过程》及29119 《软件测试》系列。(相关标准族背景资料; 项目引用应明确适用年份、部分号及条款, 不据此直接声明符合性。)
13. RTCA DO-178C / EUROCAE ED-12C; ISO 26262; IEC 61508; EN 50128。(高保证领域背景资料; 各标准的现行版本、替代关系、适用行业及审定要求须逐项核对, 不能直接平移为ASE合规依据。)
14. 国家互联网信息办公室等七部门. 《生成式人工智能服务管理暂行办法》, 2023年8月15日起施行。(是否适用须结合服务对象、提供方式和适用地区判断, 不能泛化为所有内部ASE活动的统一义务。)
15. Lamport L. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley, 2002. <https://lamport.azurewebsites.net/tla/book.html>

16. Anthropic Engineering. Effective harnesses for long-running agents, 2025-11-26; Demystifying evals for AI agents, 2026-01-09。（厂商工程实践，非跨行业标准。） <https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents> ; <https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents>
17. Agentic AI Harness Engineering and Loop Engineering: Why, What, and How to Build High-Performance, Reliable, Consistent, and Trusted Agent Systems. Deep Research Report, Version 4.1, 2026年7月。附件研究文档，文件名：Agentic_AI_Harness_and_Loop_Engineering_Deep_Dive_v4.1.md。术语参见第7.5—7.11、8.4及9.6节。
18. Kong, W., Lin, Y., Teoh, X., Nguyen, D.M., Ren, R., Chang, J., Hu, H. and Chen, H., 2026. Compiling Large Multi-Modal Requirement Documents into Runnable Software Systems. International Symposium on Software Testing and Analysis (ISSTA 2026).

引用说明：这些文献用于说明研究、机制和工程实践，不代表本白皮书采纳其全部结论，也不构成产品推荐。正文中的框架、分级和示例是本白皮书的建议，不表述为经广泛验证的行业标准。标准和法规应核对适用版本、范围和实施状态；无法充分核验的断言已删除、限定或在审阅批注中标出。

附录 A ASE 最小闭环检查清单

本附录用于自查首个 ASE 闭环。核心项（●）不满足时应阻断；确属不适用的条目，须记录理由、风险和批准人，不得以“不适用”为由绕过法律禁止事项或安全红线。

规范（●）

- 任务具备目标与验收、行为与质量、接口与工具、保证与责任四类规范
- 每条需求均有预先规定、可复核的自动、人工或组合判定程序
- 规范已纳入版本管理，变更可追溯
- 规范启动前通过自动完整性校验

执行（●）

- Agent 在沙箱中运行，仅授予最小必要权限，且权限可随时收回
- 关键行动、工具调用、权限使用与判定记录可追溯，回放范围及限制明确
- Agent 具备明确的中止与澄清请求机制

测试与评测（●）

- 按风险具备适用的测试基线，并纳入独立发布门禁
- 检查未经授权的测试弱化和隐藏评测访问，合法测试维护须独立批准
- 对涉及非确定性能力或比例估计的关键结论，开展重复试验，并提供与抽样结构相匹配的统计报告
- 开发、回归与保留评测集隔离，记录污染风险及训练数据不可核实的限制

验证与保证（关键控制为●，强度按风险确定）

- 按风险识别关键性质，明确假设、验证范围、责任人及强制执行点
- 跨服务接口启用模式与契约校验
- 对可监控的关键性质设置拦截、告警和处置；不可逆副作用优先事前阻断
- 已建立本组织的风险分级表并固化到发布流程

证据与验收（●）

- 意图、规范、行动、工件、验证、声明、批准与审计形成版本化证据链
- 门禁默认拒绝，例外须授权留痕；不可豁免红线不得例外放行
- 低风险依预授权策略放行，高风险及例外由指定人员批准
- 按风险具备经过必要演练的回滚、补偿或止损方案

治理（●）

- 已明确人、Agent、平台三方的责任边界并形成书面规定
- 对外质量声明包含证明范围、所用工具、假设与未覆盖部分
- 度量成功率、返工、成本、时延、人工介入、回滚及严重失效
- 发布后的事件、失败样本和评测反馈回流到规范和测试；迁移与退役过程包含数据处置和权限撤销

附录 B 统一术语表

本表列出本白皮书在智能体执行工程语境中使用的术语及其概念边界，供第2.1节和后续专题统一使用。本白皮书不将这些中文译名表述为既定行业标准。[17]

术语与约定名称	定义	边界与使用说明
Framework 开发框架	为开发者提供模型适配、工具封装、流程、状态访问等构建组件与抽象。	支持构建并暴露设计选择；Harness 可用它实现，但二者不作为同义词。
Runtime 运行时	提供进程、容器、队列、网络、存储、资源限制和调度等执行机制。	侧重执行机制；Harness 利用这些机制组织行为并落实运行策略。
Harness 运行控制框架	组织并落实围绕模型、依托底层运行环境构建的智能体系统的运行行为与控制策略。	组织上下文、工具、状态、权限、预算及恢复；不等于开发框架或底层运行环境，不以改变模型参数为前提。
Context 上下文	为单次模型调用选择、装配的完整可见输入信息。	是当前调用的信息视图，不等于全部任务状态、会话历史、记忆或知识。
State 任务状态	用于记录任务当前进展、工件、待处理事项和控制条件的状态对象。	关键状态应持久化并受更新规则约束；只有当前决策所需部分进入上下文。
Session 会话	记录一次持续任务的事件历史、交互及关联工件，支持交接、检查点与恢复。	不只是聊天记录，也不等于工作进程或上下文；持久记录可跨模型调用与进程保存。
Memory 记忆	从历史观察与经验中保留、可供后续任务复用的信息。	不是全部会话历史；应区分候选与已验证记忆，并管理来源、范围、时效及失效条件。
Knowledge 知识	经治理并获准在更广范围复用、具有来源与适用边界的信息。	记忆可经验证与治理转化为知识；知识仍须按权限与相关性筛选后进入上下文。
Loop 执行循环	根据观察、验证和进度反馈，控制继续、修正、暂停、恢复、升级或终止的迭代过程。	描述随时间推进的反馈控制，须设置预算并遵守停止纪律；不表示节点关系或执行拓扑。
Graph 执行图	以节点、边和共享状态表达执行依赖、路由、并行、汇合及状态传递的图结构。	可包含有界循环，也可无环；节点不必是 Agent。本文运行编排语境使用此名称。
Relationship Graph 关系图	显式表达参与者、目标、任务、工件、证据与权限等实体之间的关系。	用于澄清 Graph 的广义含义；关系图不自动构成可执行流程，图结构也不等同于图数据库。

使用约定：Harness 统一译为“运行控制框架”，Framework 译为“开发框架”，Runtime 译为“运行时”。Loop 与 Graph 分别使用“执行循环”和“执行图”，不写成同义术语“Loop / Graph”。在非执行编排语境下，Graph 可泛指关系图，应按上下文明确其范围。

文档结束

本白皮书是首版产业共识讨论稿，框架和方法将根据实践持续修订。欢迎研究机构、企业、开源社区和标准化组织贡献可脱敏、可复核的场景、任务契约、失败样本、评测方案和验证案例，共同检验框架的适用边界。



OPEN AGENTIC INDUSTRY CONSORTIUM
开放智能体产业联盟

构建智能体 AI 时代的中立基础设施 与全球协作生态

开放智能体产业联盟 (OAIC)

www.oaic.ai

