中国科学: 信息科学 2023 年 第 53 卷 第 8 期: 1490–1502
SCIENTIA SINICA Informationis
《中国科学》杂志社
SCIENCE CHINA PRESS
国防科技大学建校 70 周年专题 · 评述

群智范式: 软件开发范式的新变革

王怀民[1,2], 余跃[1,2*], 王涛[1,2], 丁博[1,2]

1. 国防科技大学计算机学院, 长沙 410073
2. 复杂关键软件环境全国重点实验室, 长沙 410073
* 通信作者. E-mail: yuyue@nudt.edu.cn

收稿日期: 2023–03–08; 修回日期: 2023–06–05; 接受日期: 2023–06–15; 网络出版日期: 2023–08–14

科技创新 2030 — "新一代人工智能" 重大项目 (批准号: 2021ZD0112904) 和国家自然科学基金 (批准号: 62141209) 资助项目

摘要 软件开发作为人类当代独特的智力活动, 经历了从作坊式的个体创作到工业化群体大生产, 再由工业化群体大生产回归大规模群体创作的历史转变, 产生了工程范式与开源范式两次变革. 工程范式聚焦线性的确定性问题的软件开发, 几乎放弃对不确定性问题的关注. 开源范式全面拥抱不确定性, 但对结果不做确定性承诺. 本文面向软件定义一切的新时代, 系统总结两次软件开发范式变革的核心规律, 结合我们长期的研究与实践, 提出软件开发群智范式, 采用 "群体智能" 为本源的科学观、"宏观演化、微观求精" 的核心理念, 以及 "两个联接、一个转化" 的方法论, 重新认识软件开发活动, 为人机物三元共融的群智软件开发提供指导.

关键词 软件开发, 范式变革, 工程范式, 开源范式, 群智范式

1 引言

在 "软件定义一切" 的时代 [1], 如何定义软件 [2]? 这是一个大问题! 特别是在今天, 软件上升为现代社会信息基础设施, 人们越来越多以软件定义为手段构造复杂世界, 并在此基础上理解和处理复杂世界问题. 在计算技术发展的历史进程中观察软件开发技术的发展, 我们可以看到, 软件开发面对的危机一个接着一个, 这些危机不仅推动了软件开发技术的发展, 而且带来了软件开发理念和方法的深刻变革, 我们称其为软件开发范式 [3] 的变革. 本文从已经发生的两次软件开发范式变革 (即工程范式与开源范式) 出发, 试图认识总结软件开发技术的发展规律, 重点阐述正在形成的群智范式变革, 结合我们的实践经验探讨软件如何定义未来. 值得注意的是, 这 3 种范式的产生和发展虽然有先后的时间顺序, 但在实践中三者不是完全替代关系. 工程范式与开源范式直到今天仍在很多场景下仍然行之有效. 我们需要把握三者的本质, 结合时代特点与应用场景指导软件开发实践活动.

引用格式: 王怀民, 余跃, 王涛, 等. 群智范式: 软件开发范式的新变革. 中国科学: 信息科学, 2023, 53: 1490–1502, doi: 10.1360/SSI-2023-0064
Wang H M, Yu Y, Wang T, et al. Crowd intelligence paradigm: a new paradigm shift in software development (in Chinese). Sci Sin Inform, 2023, 53: 1490–1502, doi: 10.1360/SSI-2023-0064

2 工程范式: 从个体创作到群体化大规模生产

2.1 从个体创作到群体化大规模生产的变革

软件开发的基本活动是程序设计. 从 19 世纪 40 年代人类历史上公认的第一段程序 [4] 的出现, 到 20 世纪 40 年代通用数字电子计算机的发明以及之后软件概念的提出, 在这百年期间, 程序被视为个别天才在其私人作坊中创作的精妙作品. 这时的软硬件规模较小、功能单一且耦合性强, 优秀的程序员仅凭大脑就足够 "跑" 明白程序的每一个步骤和细节. 到 20 世纪 60 年代, 随着第二代晶体管计算机和第三代集成电路数字计算机的巨大成功, 大型计算机系统技术成熟, 应用范围迅速从军事计算领域扩展到商业和工控等经济领域, 社会对于程序的需求量以及自身的复杂程度迅速增长. 与功能单一的小型程序不同, 上规模的软件系统往往需求复杂、代码量大、涉及的利益相关方广泛. 程序代码的复杂度与开发过程的复杂性已远超出程序员个体直接能够把握的程度, 必须依赖规范表达的设计文档帮助程序员群体之间的理解、交流、协作和传承, 出现了 "程序 + 文档" 的软件概念. 此时的软件开发与维护成本极高, 失败的软件项目屡见不鲜, 所谓 "软件危机 (software crisis)" 终于爆发了 [5]. 以 IBM OS/360 通用操作系统 [6] 的研发为例, 从 1963 年到 1966 年, IBM 共投入超过 5000 人年 [7], 涉及程序员、文档编制人员、计算机操作员、项目管理、文秘等多种角色. OS/360 软件项目不仅仅是在操作系统方面延伸出了一系列影响后世的概念、技术和系统 [8], 更重要的是该项目组织实施过程中获得的经验教训, 使软件从业者产生了历史性的觉醒. 软件开发若不产生变革性的指导原则与方法论, 任何看似偶然发生的事件, 如硬件故障、应急任务、人员变动等, 累积叠加后就极有可能导致其开发过程的灾难性偏离 (disastrous schedule slippage) 和项目崩溃.

处于工业化大生产浪潮中的计算机先驱们, 向经典的工业生产管理学习, 于 1968 年正式提出了软件工程 (software engineering) 的概念 [9], 开启了工程范式的探索. 软件开发从个体独立创作走向了大规模群体有组织的生产时代, 造就了影响深远的软件产业.

2.2 关于工程范式

我们将软件工程遵循的软件开发理念和方法称为软件开发的工程范式. 作为脱胎于计算机工业的软件开发范式, 软件开发的工程范式理念潜移默化的继承了机械的世界观或科学观 [10], 即世界是一部确定不变的、可被理解表述的、可被线性分解还原的 "机器". 由此派生出软件开发的基本原则和方法: 自上而下, 逐步求精. 这些原则与方法源自传统工业化大生产的两点启示: 一是把成规模的软件开发者按照标准化的生产流程组织起来, 实现大规模群体协作开发软件; 二是发明通用性较强的生产工具, 让软件开发活动尽可能自动化. 现代软件企业是工程范式的实践产物, 软件企业把现代企业组织管理思想、方法和技术运用在软件开发生命期中, 由此产生了软件开发过程管理模型, 例如瀑布模型、敏捷模型, 以及评价软件企业能力的软件能力成熟度模型 (capability maturity model, CMM) 等 [9]. 遵循工程范式的理念, 涌现出有效的软件开发方法、技术、工具和平台, 其中最具代表性的成果是高级程序设计语言, 例如 ALGOL, Pascal, Ada 等, 以及相关程序设计与软件开发方法和技术, 例如结构化程序设计 (structured programming) [11]、面向对象程序设计 (object-oriented programming) [12] 等方法、工具和支持环境.

软件开发的工程范式的具体实施过程总体上遵循 "还原论" 的思想 [13], 即将软件开发任务自上而下精细化分解, 简化为可管理、易开发的基本功能单元, 完成基本单元开发后, 再通过自底向上组装获得目标软件系统, 期望达到 "整体等于部份之和" 的效果. 这里涉及软件开发的 3 个经典概念. 第一是需求, 这是工程范式的起点和依据. 软件工程强调, 软件需要可以通过一套标准过程与技术, 指导开发

人员系统化地识别、分析和获取需求, 并形成规范一致的需求规格说明, 由此开展 “自上而下、逐步求精” 的开发活动. 第二是质量, 保障质量是工程范式关注的核心目标. 所谓软件质量是指软件代码满足需求规格说明的程度, 其保障手段和工具包括形式化验证、自动化测试等方式. 第三是效率, 提高效率是工程范式配置资源的主要依据. 开发效率是指开发出满足需求规格的软件所需要人力、时间、投资等资源投入产出比. 提高效率需要对软件生命周期进行阶段分解与过程控制, 再通过相应的自动化手段提升优化投入产出比. 软件开发的工程范式伴随着计算机产业的成熟取得了历史性成功 [14], 形成了软件产业, 但在互联网产业蓬勃发展环境中工程范式面临发展瓶颈.

2.3 工程范式的瓶颈

从 20 世纪 90 年代后期起, 软件开发工程范式出现天花板 [15]. 1995 年, 国际权威 IT 研究咨询公司 Standish Group 对美国境内 8000 个软件项目进行调查分析发现 [16]: 83% 的项目无法在既定时间内开发完成, 开发周期平均超时 222%, 存在 52.7% 的项目预算超出原计划的 189%, 31.1% 的项目在执行过程中因为无法控制质量和成本被取消. 这一洞察在 20 年后 (即 2015 年) 的调查中 [17] 仍未有本质的好转, 在 25000 余个软件项目中无法按预算执行的比例占 56%, 无法按期完成的项目占比 60%, 执行不达标的项目比例高达 44%. 这里既有来自软件自身的内在因素, 也有来自软件发展环境的外部因素. 从软件自身特点规律看, 软件开发的工程范式遇到两个瓶颈: 第一, 协同效率瓶颈, 即软件开发过程管理面临群体协同规模的 “人月神话” [18]; 第二, 自动化瓶颈, 即软件自动化工具面临能行可表达的理论极限 [19]. 从软件发展的外部环境看, 工程范式的发展出现两个不适应: 第一, 软件的生产效率与摩尔定律带来的计算基础设施硬件发展速度严重不适应; 第二, 工程范式的经典原则与迅猛到来的网络时代 [20] 严重不适应. 特别是进入到 (移动) 互联网时代, 软件的利益相关者从明确领域相关的小规模需求主导者转变为动态开放的大规模互联网用户群体, 导致了软件开发不再是边界清晰、需求明确的生产活动, 而是如何持续构造 “人在回路” 的复杂软件系统. 这些新的变化动摇了工程范式的基本前提假设 [19], 即用户需求可以被事先完整分析、获取和描述, 软件系统具备可拆分的明确边界和结构, 软件开发过程精确可控等. 正如当今广泛存在于互联网上的各类应用软件, 在开发之初可能根本无法明确最终的用户群体, 自然也不可能冻结所谓用户需求规格并精细拆分. 随着云计算技术发展与云服务的普及, 软件产品越来越多以网络服务呈现, 软件项目的商业成功逻辑不再是能否在确定的预算约束下按时交付用户满意的软件版本 (即 To Business 企业市场模式), 或销售复制成本极低的软件版本拷贝 (即 To Consumer 消费市场模式), 而是能否形成大规模网络用户生态圈.

3 开源范式: 从群体化生产到大规模群体创作

3.1 大规模群体开源创作变革

在软件开发的工程范式面临巨大发展瓶颈之际, 发端于 1980 年代中期自由软件运动的开源软件, 经过 20 多年的蓬勃发展, 取得了出人意料的成就 [21]. 相对于标准化、有组织的工程范式, 开源范式更尊重每个开发者的个体创作意愿, 通过营造开放性、多元化、自组织的创作环境 [22], 充分激发大规模程序员的参与热情与创作灵感, 通过群体智慧涌现, 最终形成高水平的软件制品. 这样的软件开发过程我们称之为软件创作活动, 由此产生的软件制品称之为软件作品. 优秀的开源软件作品通过互联网可以高效聚集数以万计的开发者参与贡献, 其规模远远超过任何单一商业软件公司, 形成了极高质量软件社区, 如 Linux 社区、Apache 社区等.

3.2 关于开源范式

相较于软件开发的工程范式, 业界并没有就开源范式形成共识. 我们将发端于 1980 年代中期自由软件运动之后开源软件 20 年实践所遵循的软件开发理念和方法统称为开源范式. 作为脱胎于互联网环境的软件开发范式, 开源范式看似无序状态背后的逻辑是演化的世界观和科学观, 即遵循自然演化两个基本原则: 一是 (可以传递给后代的) 变化, 二是竞争 (什么样的变化可以在后代继续存在). 传统开源软件的发起往往源于少部分人的创作冲动, 一开始并没有明确的目标用户, 也未必有详细的设计和规划. 发起者将软件源代码公开分享出来, 任何感兴趣的人都有机会参与创作 [23], 根据个性化需求自由地进行修改和完善, 并自己决定是否向源项目提交贡献 [24, 25]. 开源软件的核心维护者对社区大量自发提交的贡献进行选择, 决定吸纳什么样的代码合并到原始分支中, 故被采纳 (即竞争) 的社区贡献 (即变化) 则随着开源软件的版本迭代传承下去. 开源范式这种基于创作兴趣和社区自组织的发展理念可以简单概况为: 自下而上、演化涌现. 开源范式的软件创作方法是 “程序代码开源、开发过程开放、大众自由参与”. 开源范式支持在不确定的网络世界, 通过众多的开发者带来丰富变化和竞争, 自然演化出得到用户欢迎的软件制品. 以 Linux 演化历程为例, 自 2001 年 12 月至 2005 年期间[1)], 仅内核就发行了 94 个版本 (2.5.1 release∼2.6.14 release) , 上千名全球开发者为其开源内核贡献过代码. 因为开源, 每个 Linux 内核的自由 “下载者” 都可以修改内核, 从而产生 “变异” 并 “繁殖” 出新的 Linux 版本, 这个过程带来的多样性整体上大大提高了 Linux 种群基因延续的可能性. Linux 通过开源范式带来的易变性和多样性, 更能应对未来操作系统发展的不确定性, Linux 基因被更多的衍生操作系统继承, 从而家族 “人丁兴旺、儿孙满堂”, 当智能手机、云计算、物联网等新的需求涌现时, Linux 的后代就具有更强大的生存竞争力.

从工程范式的 3 个经典概念 (即需求、质量、效率) 出发观察开源范式, 可以发现开源范式相对于工程范式是颠覆性的. 第一, 关于需求问题. 开源范式不坚持 “需求在先、开发在后” 的原则, 甚至不坚持在开发软件之前有明确的用户, 开发者基于自己的构思进行软件创作. 第二, 关于质量问题. 按照工程范式的原则, 需求是质量的依据和前提, 在开源范式中, 软件质量体现为软件开发社区的规模和口碑. 第三, 关于效率问题. 在开源范式中, 既然开源软件是在开发社区中持续演化的, 当然就不存在工程范式中的最终交付时刻, 也就无所谓开发效率问题. 如果坚持用开发效率的概念理解开源软件项目, 则可以用开源项目的迭代效率理解开源软件的开发效率, 例如开源软件的缺陷响应与修复速率、新版本发布的频率等. 自 21 世纪以来, 越来越多期盼寻找软件开发 “银弹” 的传统软件企业遵循开源范式进行革新. 这种基于演化科学观下的开源范式是软件开发历史上的又一次重要觉醒, 适应了互联网时代软件作为服务的发展趋势.

人们似乎把软件开发的工程范式与开源范式本质区别认定为软件源代码是否 “开源”, 从而把两种范式对立起来. 我们认为, 这两种开发范式也有一个鲜明的共同点, 即共同关注如何发挥 “群体智能” 的效果, 区别在于发挥 “群体智能” 侧重点不同. 工程范式关注群体智能的 “汇聚”, 即如何将软件开发者群体的智慧聚焦在已经明确获得的 “需求” 上, 如何高效开发出满足需求的高质量软件. 因此, 工程范式严格控制出现所谓软件 “分支”, 因为出现 “分支” 就意味着分散注意力, 意味着增加软件开发和维护成本. 因此, 当软件需求明确且可以线性分解的时候, 工程范式往往是有效的. 开源范式关注群体智能的 “激发”, 即如何吸引和激发更多的参与者参与软件开发, 其极端手段就是利用开放源代码吸引参与者, 故出现 “分支” 对于开源范式是常态. 在软件需求极其不确定的发展领域, 例如新型的互联网服务利用, 开源范式往往是有效的.

1) https://kernelnewbies.org/LinuxVersions.

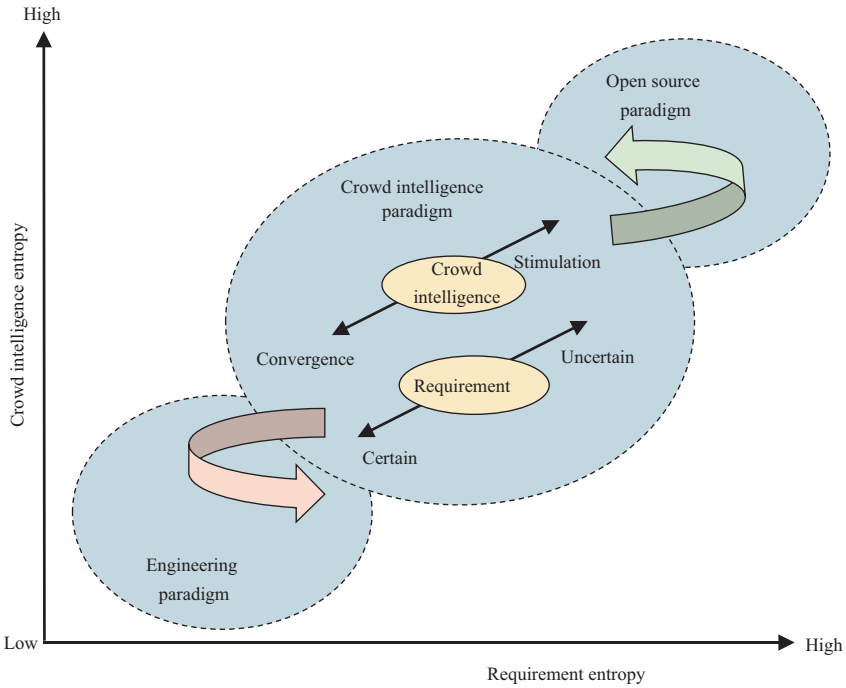

图 1 (网络版彩图) 开源范式与工程范式的对比

Figure 1 (Color online) The comparison between open source paradigm and engineering paradigm

3.3 开源范式的瓶颈

随着开源范式的发展, 各类相对独立的开源软件开发社区逐步演变为多种元素交织依赖、持续成长动态演化的开源软件生态. 置身在自然演化生态中, 开源范式难逃 “物竞天择、适者生存” 的自然规律. 虽然宏观上开源生态发展热火朝天, 但任何一个开源项目是否能够成功是完全不可控的, 开源出来的代码能否得到关注是完全未知的, 甚至任何一个开放的开发任务什么时候能完成以及是否能完成都无法保证. 根据 Feitelson 等 [26] 的统计, 2005 年 5 月之前, 在曾经最大的开源软件发布平台 SourceForge 托管的 122205 个项目中, 66% 的项目一次下载都没有, 真正产生应用价值的开源项目更是可遇不可求. 如果将开源范式和工程范式做一个比较 (如图 1 所示), 工程范式强调面向确定性场景下的问题, 软件需求表达越清晰 (即需求熵越低), 开发团队目标就越明确、资源配置越聚焦, 进而可以通过强组织的方式进行过程管理. 开发过程聚焦软件产品, 强调团队中每个开发人员各司其职 (可以称为群智熵低), 不需要太多的发散思维. 因此, 这一范式可以在相对稳定的环境 (即两类熵都较低) 进行稳定的群智汇聚, 但几乎放弃了对不确定性问题的关注. 而开源范式面向开放式场景问题, 通过自组织的社区群体, 鼓励开展基于兴趣驱动的软件作品自由创作, 以多样性促进创新涌现. 这种对自由创作的极端追求 (即需求熵高) 加剧了群体协同过程的不确定性 (即群智熵高), 能够在开放环境下 (即两类熵都较高) 激发出丰富多样的创新需求, 但软件作品收敛为软件产品的成本极高, 几乎无法对结果做出可预期性承诺. 那么, 如何平衡软件需求的确定性和不确定性之间的矛盾, 进而更有效地、可预期地组织软件开发群体智力? 特别是面对未来 “人机物” 三元融合的万物智联泛在计算时代 [27] 和数字经济新型应用形态 [28], 软件的范畴将从信息空间拓展至物理空间与人类空间三元融合的新场景. “万物均需互联、一切皆可编程” 的新计算特征, 将激发用户对软件的功能、服务与应用模式产生更灵活多样且个性化的期待. 因此这类复杂软件的需求不确定性可能更为显著, 需要通过 “软件定义” 获得

确定性, 至少缓解不确定性. 时代呼唤新的软件开发范式.

4 群智范式: 从大规模群体创作到群智软件开发

4.1 群智范式的基本理念

实际上, 开源范式与工程范式分别适用于不同的软件开发场景, 但本质都是对 “群体智能” 的关注. 从 2007 年起, 我们追本溯源开始研究软件开发的新范式 – 群智范式, 试图化解软件需求的确定性与不确定性之间的矛盾, 在群智激发和汇聚之间寻找到平衡点 (如图 1 所示), 实现规范生产与自由创作的联接与转化, 构建适应泛在计算时代特点的可信软件开发环境 [25, 29].

群智范式的核心理念可以简单概括为: 宏观演化, 微观求精. 在宏观 (长期) 尺度上接受世界的不确定性, 将软件核心开发者、外围软件涉众, 以及软件所处的社区生态视为有机整体, 持续激发各类群体围绕软件项目进行自由创作; 在微观 (短期) 尺度上, 即在软件长期演化进程的具体阶段, 明确阶段性里程碑任务的需求规范 (简称里程碑), 在软件开发核心团队主导下, 采用逐步求精的思路组织任务规划实施. 我们仍然可以从工程范式的 3 个经典概念出发理解群智范式, 与其他两个范式的详细对比如表 1 所示.

(1) 关于需求问题. 群智范式强调, 软件的阶段性可部署执行版本 (简称原型版本) 是其里程碑的精确表达. 通过持续发布原型版本, 吸引和发现潜在用户, 通过引导用户使用体验原型版本, 激发用户以疑修 (issue) 的形式持续产生、表达和丰富需求, 通过不断满足用户当下需求激发新的需求, 形成下一个里程碑, 驱动软件系统的持续迭代演化. 在群智范式中, 需求是一个持续获取凝练的过程, 通常以两种形式呈现, 一种是以自然语言表达的疑修集合, 或称为里程碑, 是需求的非精确表达; 另一种是以源代码呈现的实现里程碑的阶段性可部署执行版本, 或称为原型版本, 是现阶段需求的精确表达.

(2) 关于质量问题. 群智范式中的软件质量不再只是单纯度量软件源代码本身, 而应该是综合考虑软件源代码、围绕软件源代码形成的社区, 以及软件社区所在的生态环境. 因此, 群智范式中的质量问题应该包含传统意义上的软件代码质量、社区规模与口碑, 以及该软件社区在其生态环境中的地位和对生态环境发展的促进作用. 具体而言, 群智范式中的软件质量可以从宏观 (长期) 和微观 (短期) 两个方面理解. 在宏观上, 软件质量主要体现在应对复杂生态演化过程中各类不确定性因素的能力; 在微观上, 软件质量主要体现在持续迭代版本满足里程碑任务规格的程度, 这反映了软件对经过筛选的阶段确定性需求的满足程度.

(3) 关于效率问题. 群智范式中的开发效率同样需要考虑软件本身的开发效率、软件社区形成的效率, 以及软件社区所在生态环境演进的效率. 从群智激发和汇聚两个视角来看, 激发效率是指吸引更多参与者进入社区并做出贡献的效率, 汇聚效率是指吸纳广大参与者贡献的效率. 在宏观上, 软件开发效率主要体现在其软件社区与所在生态环境在应对不确定性因素而发生变异与演化所花费的时间与成本; 在微观上, 软件开发效率主要体现在对于确定的里程碑开发任务与阶段性可部署执行版本的迭代时间与成本.

4.2 群智范式中的软件开发活动

群智范式的软件开发方法可以概括为 “两个联接, 一个转化”, 即联接核心团队与外围群体, 联接自由创作与规范生产, 实现原型作品与原型版本之间的转化.

(1) 联接核心团队与外围群体: “核心团队” 和 “外围群体” 代表了软件开发活动中 “软件涉众” 这一关键要素. 从群智范式视角看, “核心团队” 和 “外围群体” 代表了软件开发社区中两类典型参与

表 1　三次软件范式变革基本理念的对比

Table 1　The comparison of the basic concepts of the three software development paradigms

Element	Crowd intelligence paradigm	Engineering paradigm	Open source paradigm
	Macro evolution and micro refinement	Top-down and step-by-step refinement	Bottom-up and evolution emergence
Requirement	In the process of software evolution, requirements are continuously formed, expressed in the form of milestones and software prototypes. Users are free to propose issues through using prototypes to continuously generate new requirements.	At the beginning of software development, the specification description confirmed by the user is obtained through independent requirement engineering, which is the premise of software development. Once the requirement specification is confirmed, it cannot be modified during the software development process.	There is no need to have a developer-independent software requirement specification. Developers create software based on their own ideas. Source code is the expression of software requirements.
Quality	From both macro and micro aspects, the code quality of series software prototypes, the software community, the status of the software community in the ecological environment and the role of promoting the ecological environment should be considered.	The extent to which the source code meets the requirements specification.	The user size and reputation of the open source community.
Efficiency	From both macro and micro aspects, the development efficiency of software, the efficiency of software community formation, and the efficiency of the evolution of the ecological environment where the software community is located should be considered. These include the efficiency of attracting and motivating more participants to contribute, and the efficiency of gathering and absorbing the talents of a wide range of contributors.	Time and cost of developing software that meets requirements.	Time to respond to open source community issues.

者. “核心团队” 通常是软件项目的创始团队、管理团队和核心参与者, 是初始创新作品的发起者, 是里程碑和原型版本的发布者. 随着软件的迭代演化, 核心团队负责软件演化过程中里程碑规划决策、核心功能开发、吸纳汇聚 “外围群体” 贡献的疑修或代码、发布新的原型版本; “外围群体” 则是参与软件项目的其他大规模利益相关者群体或涉众, 在软件迭代演化过程中贡献需求 (以疑修的形式) 和代码.

(2) 联接自由创作与规范生产: “自由创作” 与 “规范生产” 代表了软件开发活动中 “过程组织” 这一关键要素. 从群智范式视角看, 软件开发是创作与生产相互交织快速迭代的过程. 在需求不清晰、任务不明确时, 核心团队通过发布原型版本吸引并激发外围群体的灵感, 收获并评价外围群体的贡献, 参与软件集体创意; 在阶段性里程碑明确后, “核心团队” 以规范化的组织模式快速推进研发任务, 基于集成部署和测试等机制生成高质量的软件原型版本. 群智范式通过内外交汇的协调机制 “联接” 这两类活动, 充分发挥 “外围群体” 自下而上的创新活力和 “核心团队” 自上而下的组织能力, 协同驱动

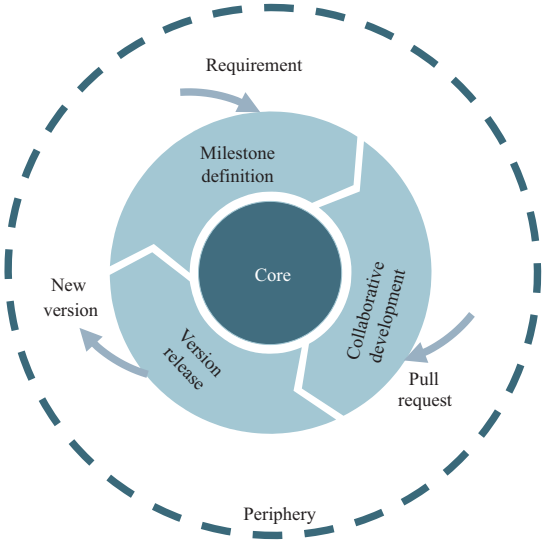

图 2 (网络版彩图) 群智软件开发过程中的 “两个联接、一个转化”

Figure 2 (Color online) Two links and one transformation in the process of crowd intelligent software development

软件项目的快速迭代演化.

(3) 原型作品与原型版本之间的转化: “原型作品” 与 “原型版本” 重点关注软件开发活动中的 “软件制品” 这一关键要素. 从群智范式视角看, 软件是原型作品与原型产品持续交互演化的产物. 原型作品通常是灵感驱动下的创意捕获和表达, 具有不可预期性和多样性; 原型版本则通常是在阶段性里程碑驱动下, 按照工程范式开发产生的软件原型版本, 具有确定性和明确的评判标准. 群智范式关注在联接 “外围群体” 创作活动与 “核心团队” 生产活动的基础上实现这两类软件制品 “转化”, 即对原型作品进行筛选、改进并集成到软件原型版本中, 并持续发布阶段性原型版本, 吸引用户体验, 从而激发产生新的原型作品, 形成迭代演化.

群智范式下的软件开发是大规模软件涉众, 通过 “两个联接、一个转化” 的群智协同, 不断形成满足现阶段性里程碑的原型版本, 然后在大范围应用中获取新需求, 进而持续迭代演进形成良性循环, 如图 2 所示. 这一过程面临的关键挑战是如何通过高效的协作机制、工具链与系统平台保障群智软件全生命期的激发与汇聚效能, 例如支撑大规模群体需求表达的平台环境、核心团队可高效度量大众贡献质量的智能化工具, 以及面向软件制品持续汇聚的里程碑版本管理方法等. 人机混合的持续学习 [24] 将为软件开发中的群智激发与汇聚提供更多可能的支持. 大规模软件涉众的各类开发活动汇聚形成行为大数据, 群智开发平台利用机器学习等智能方法从行为大数据挖掘形成机器智能, 并作用于软件涉众, 从而更精准地引导和激发涉众行为, 软件涉众的反馈和行为变化则进一步提升机器智能的能力, 形成软件涉众与人工智能的持续互学习, 并在互学习过程中实现更好的群智激发与汇聚.

5 Trustie 确实的研究与实践

自 2006 年以来, 我们持续深入地开展群智范式基础研究与系统实践. 以国家 “十一五” 高技术研究发展计划重点项目 “可信的国家软件资源共享与协同生产环境” 为起点, 先后获得了国家自然科学基金重点项目、科技部重点研发计划、科技创新 2030 — “新一代人工智能” 等重点项目支持, 形成了 “Trusite” (中文品牌称为 “确实”) 的核心技术体系与工具平台.

5.1 机理方法与关键技术

群智范式下的软件开发是在开放式环境下, 通过核心团队与外围群体围绕需求定义、代码开发和版本发布等环节开展持续的群智激发与汇聚协作, 推动软件系统向前不断迭代演化的过程. 近年来, 我们围绕群智范式下的新型软件开发机理、方法技术与运行机制, 开展了一系列基础研究与关键技术突破.

在群智软件开发基础理论与方法方面, 聚焦网络时代对软件开发工具、运行环境及服务模式的颠覆性变化, 以大规模开源的成功实践为研究案例, 系统定义了复杂软件系统新形态, 通过提炼总结生命系统、社会系统和经济系统等经典复杂系统的形成和演进规律, 提出了 "成长性构造" 与 "适应性演化" 法则, 以群智范式为视角建立复杂软件成长演化的系统观和方法论, 为此类系统的构建与发展提供新的软件架构与方法学 [19,30,31]. 同时, 深入分析了以众包模式为代表的社区化群体组织形式, 凝练形成了 "大众化协同、开放式共享、持续性评估" 的群智软件开发核心机理, 系统提出了将互联网大规模协同创作机理与工业化可信软件生产相结合的软件开发群体化方法, 并研究构建了一系列群体协同、开放共享与持续评估的支撑机制与服务模式 [25,32,33]. 上述研究为本文提出的群智范式奠定了重要基础.

在核心关键技术方面, 我们形成了可支撑群智范式核心要素全生命期有效运转的技术体系. 针对核心团队与外围群体高效协作问题, 深入研究设计了协作感知 [34,35]、贡献引导 [36,37]、持续激励 [38,39] 等关键机制, 以数据驱动的实证研究建立了影响群智激发与汇聚效能的度量体系 [40,41], 为两类群体间持续协同的效率提升与质量保障提供支持. 针对开放式需求获取与工程化里程碑持续转换问题, 研究提出了基于跨社区关联的需求汇聚方法, 利用信息检索和语义分析技术, 建立起跨项目 [42]、跨社区 [43] 相关需求的关联关系, 并构建了融合软件开发社区与知识分享社区的开源资源汇聚、评估与检索平台 OSSEAN [44], 有效支撑互联网环境下的多源多样化需求反馈的自动获取、聚合与关联. 针对个性化合并请求到确定性功能收敛问题, 我们针对不同粒度、不同类型、不同质量的协作任务, 提出了开发者推荐 [45]、审阅人推荐 [46]、任务规范引导 [47]、贡献质量检测 [48] 等智能化技术, 实现群智软件 "项目 – 开发者"、"任务 – 开发者"、"任务 – 任务" 等异质主体之间的高效联接与适配. 针对群智软件阶段性版本持续性发布问题, 聚焦持续集成、持续部署等 DevOps 关键技术, 开展了软件发布前的初始质量与实际运行态质量细粒度的实证分析 [49], 提出了云原生环境下持续部署工作流的优化策略 [50,51], 可有效支撑群智范式要素中 "原型作品" 的高效筛选与 "原型版本" 的快速上线, 进而促进群智软件转入新一轮需求激发汇聚.

5.2 系统实践与应用

在理论研究与关键技术突破的基础上, 坚持应用牵引的系统实践. 总的来说, 可分为以下 3 个历程:

(1) Trustie 1.0 阶段 (2006~2014 年). 以开源范式视角革新企业级工程范式, 基于 "大众化协同开发、开放式资源共享、持续性可信评估" 为核心的互联网大规模群体协同机理, 提出软件生产线构造方法, 研发构建了可信的国家软件资源共享与协同生产环境 Trustie 1.0, 将软件创作活动与软件生产活动结合起来, 重点解决企业内、企业间的大规模软件协同开发、可信评估、运行监控和持续演化等问题, 初步实现了群智范式 "两个联接、一个转化" 迭代飞轮的基础架构. 相关技术和平台在东软集团、神州数码、航天科技集团、成飞等企业以及各地软件园区得到应用.

(2) Trustie 2.0 阶段 (2014~2020 年). 以大数据与智能技术革新开源范式, 探索协作开发社区、知

识分享社区、应用服务社区等软件涉众与制品的联接, 研发了基于开源大数据的智能化工具, 构建升级了开放创新服务平台 Trustie 2.0, 重点聚焦小规模核心团队与互联网大规模开发者群体 (即群智范式两类群体) 的高效联接与协作, 为开源社区建设提供方法指导和实践指南. 相关方法、技术和平台在启智开源社区[2)]、红山开源社区[3)], 以及头歌实践教学社区[4)]得到应用, 为新一代人工智能、云计算与大数据等行业开源孵化和开源人才培养提供了有力支撑.

(3) Trustie 3.0 阶段 (2020 年至今). 以 "群智" 为核心, 系统性支撑软件开发群智范式, 研发大规模群体的激励、协作与调控工具链, 建立面向 "小核心" 与 "大外围" 协同创新的群智激发汇聚关键机制与开发平台, 全面加速 "两个联接、一个转化" 创新飞轮, 实现面向大规模软件涉众的稳态群智激发与持续汇聚. 以 Trustie 开源内核为基础, 构建并发布了中国计算机学会 (CCF) 开源创新服务平台 "确实开源" GitLink[5)], 探索共建共享的新型开放创新平台以及学术共同体驱动的开源创新新途径.

6 结束语

本文讨论的 3 种软件开发范式的共同点是群体智能. 不同点是由于需求的确定性程度所导致的如何认识群体智能 (即理念层面) 和如何发挥群体智能 (即方法层面) 的不同, 以及由此导致软件开发质量和效率的认知差异 (见表 1) .

工程范式与开源范式的本质差异不在于是否开源上, 而在于需求是否确定以及由此导致的群体智能如何发挥上. 工程范式关注的问题是: 在需求 (规范表达) 确定的前提下, 如何高效汇聚开发团队的智力资源以高质量达成目标. 也就是说, 工程范式专注的焦点是 "汇聚" 群体智能, 理念和方法是 "工程化", 所以被称为 "工程范式". 开源范式关注的问题是: 在需求不确定的情况下, 如何高效激发群体智能, 以获得有意义或有价值的软件需求. 也就是说, 开源范式专注的焦点是 "激发" 群体智能, 理念和方法是 "开源", 期待通过开源的 "祭品效应" 吸引 "信众", 所以被称为 "开源范式". 群智范式关注的问题是: 面对不确定的世界, 如何高效激发和汇聚群体智能, 以实现软件的持续演化, 主动适应不断变化的世界.

什么是有意义或有价值的软件需求? 在大型机时代, 组织 (企业、机构) 投资开发的软件就是有意义或有价值的软件, 此类软件多属于企业级软件 (通常是定制的软件) , 需求由投资方 (也是使用方) 提出, 从开发者角度看, 需求获取有明确的对象, 即投资者 (也是使用方) , 开发者的焦点是如何组织工程师队伍高效开发并交付投资者需要的软件. 这种情况下, 投资者就是使用者, 这个时代产生了工程范式. 到个人计算机时代, 出现了一类新的软件形态, 即面向普通消费者的软件, 即消费级软件. 这种情况下, 消费者出现在产品开发之后, 软件需求被开发者 "代言" 了, 所有软件产品的失败是 "市场" 的失败, 即没有获得足够的消费者. 这个时代工程范式虽然有效, 并且也为了缓解软件开发过程对于前置需求的强依赖, 发展出了增量模型、演化模型和螺旋模型等方法. 但工程范式的生产关系没有发生本质的变革, 在未知的后置消费者市场中失败的风险加大了, 失败的软件增加了. 到了互联网时代, 不确定性持续增大. 面对不确定的世界, 工程范式的失败风险更大, 开源范式因运而生. 对于开源范式, 有意义或有价值的软件需求是形成了良好社区生态的软件需求, 需求直接由可执行的软件原型系统表达. 这个时代应运而生的敏捷开发、DevOps 等模式可有效支撑以软件原型为形态的需求获取与需求确认. 对于群智范式, 有意义或有价值的软件需求更进一步发展为可成长演化的软件需求, 好的软件

2) https://openi.org.cn/.
3) https://osredm.com/.
4) https://www.educoder.net/.
5) https://www.gitlink.org.cn/.

项目是能够很好成长代谢的软件项目.

人类科技发展的历程从一定意义上讲就是在不确定的世界中获得更多确定性的过程, 在这个过程中, 群体智能是获得确定性的锐利武器. 软件开发作为人类当代独特的智力活动, 经历了从作坊式的个体创作到工业化群体大生产, 再由工业化群体大生产回归大规模群体创作, 产生了两次范式变革. 工程范式聚焦线性的确定性问题的软件开发, 几乎放弃对不确定性问题的关注. 开源范式全面拥抱不确定性, 但对结果不做确定性承诺. 这两次范式变革反应了人类对世界的两种科学认知, 即机械论与演化论. 在 "人 – 机 – 物" 日益融合的三元世界中, 计算平台的泛在化必然驱使软件应用的泛在化, 软件定义一切预示着在不久的将来软件必将全面渗透人类社会方方面面. 软件开发活动必将面临更复杂的需求不确定性, 可能孕育新的软件开发范式变革, 产生新的软件开发工具. 例如, 自然语言大模型 [52] 的出现可能帮助软件开发者提升需求诱导迭代的效率, 推动人机物三元融合的软件开发从不确定的必然王国逐渐走向确定性的自由王国. 在这一进程中, 人类的主体性起着主导作用, 工具的进步将不断提升人类群体智能的有效发挥.

参考文献

1 Stroustrup B. Software development for infrastructure. Computer, 2012, 45: 47–58
2 国家自然科学基金委员会, 中国科学院. 软件科学与工程. 北京: 科学出版社, 2021
3 Shapere D. The structure of scientific revolutions. Philos Rev, 1964, 73: 383–394
4 Kim E E, Toole B A. Ada and the first computer. Sci Am, 1999, 280: 76–81
5 Fitzgerald B. Software crisis 2.0. Computer, 2012, 45: 89–91
6 Amdahl G M, Blaauw G A, Brooks F P. Architecture of the IBM System/360. IBM J Res Dev, 1964, 8: 87–101
7 Brooks F P. The Mythical Man-Month. Boston: Addison-Wesley, 1975
8 Morton D. IBM mainframe operating systems: timeline and brief explanation for the IBM system/360 and beyond. Dave Morton Writing Services, 2015. http://vtda.org/docs/computing/IBM/Mainframe/IBM_OS_Timeline_v37.5_Feb2018.pdf
9 Pressman R, Maxim B. Software Engineering: A Practitioner'S Approach. London: Palgrave Macmillan, 2005
10 Newton I. Philosophiæ Naturalis Principia Mathematica. Londini: Jussu Societatis Regiæac Typis Josephi Streater, 1687 [艾萨克 · 牛顿, 著. 赵振江, 译. 自然哲学的数学原理. 北京: 商务印书馆, 2006]
11 Dahl O, Dijkstra E, Hoare C. Structured Programming. Orlando: Academic Press Ltd., 1972
12 Rentsch T. Object oriented programming. SIGPLAN Not, 1982, 17: 51–57
13 Agazzi E. The Problem of Reductionism in Science. Berlin: Springer, 2012
14 Boehm B. A view of 20th and 21st century software engineering. In: Proceedings of the 28th International Conference On Software Engineering, 2006. 12–29
15 Brooks Jr F P. No silver bullet essence and accidents of software engineering. Computer, 1987, 20: 10–19
16 Clancy T. The chaos report. The Standish Group, 1995. https://www.csus.edu/indiv/v/velianitis/161/chaosreport.pdf
17 Group T. Chaos report 2015. https://standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf
18 Brooks Jr F. The Mythical Man-month: Essays on Software Engineering. New York: Pearson Education, 1995
19 Wang H M, Wu W J, Mao X J, et al. Growing construction and adaptive evolution of complex software system. Sci Sin Inform, 2014, 44: 743–761 [王怀民, 吴文峻, 毛新军, 等. 复杂软件系统的成长性构造与适应性演化. 中国科学: 信息科学, 2014, 44: 743–761]
20 Mei H, Huang G, Xie T. Internetware: a software paradigm for internet computing. Computer, 2012, 45: 26–31
21 Bonaccorsi A, Rossi C. Why open source software can succeed. Res Policy, 2003, 32: 1243–1258
22 von Hippel E. Learning from open-source software. MIT Sloan Management Rev, 2001, 42: 4
23 Statista. Number of employees at the Microsoft Corporation from 2005 to 2022. 2022. https://www.statista.com/statistics/273475/number-of-employees-at-the-microsoft-corporation-since-2005
24 Li W, Wu W, Wang H, et al. Crowd intelligence in AI 2.0 era. Front Inf Technol Electron Eng, 2017, 18: 15–43

25 Wang H M, Yin G, Xie B, et al. Research on network-based large-scale collaborative development and evolution of trustworthy software. Sci Sin Inform, 2014, 44: 1–19 [王怀民, 尹刚, 谢冰, 等. 基于网络的可信软件大规模协同开发与演化. 中国科学: 信息科学, 2014, 44: 1–19]

26 Feitelson D, Heller G, Schach S. An empirically-based criterion for determining the success of an open-source project. In: Proceedings of Australian Software Engineering Conference (ASWEC'06), 2006

27 Mei H, Cao D G, Xie T. Ubiquitous operating system: toward the blue ocean of human-cyber-physical ternary ubiquitous computing. Bull Chinese Acad Sci, 2022, 37: 30–37 [梅宏, 曹东刚, 谢涛. 泛在操作系统: 面向人机物融合泛在计算的新蓝海. 中国科学院院刊, 2022, 37: 30–37]

28 中国信息通信研究院. 中国数字经济发展白皮书. 中国信息通信研究院, 2021

29 王怀民, 唐扬斌, 尹刚, 等. 互联网软件的可信机理. 中国科学 E 辑: 信息科学, 2006, 36: 1156–1169

30 Wang H M, Mao X G, Ding B, et al. New insights into system software. J Softw, 2019, 30: 22–32 [王怀民, 毛晓光, 丁博, 等. 系统软件新洞察. 软件学报, 2019, 30: 22–32]

31 Wang H M, Ding B, Shi D X, et al. Auxo: an architecture-centric framework supporting the online tuning of software adaptivity. Sci China Inf Sci, 2015, 58: 092103

32 Wang H W. Harnessing the crowd wisdom for software trustworthiness. ACM SIGSOFT Softw Eng Not, 2018, 43: 1–6

33 Wang T, Yin G, Yu Y, et al. Crowd-intelligence-based software development method and practices. Sci Sin Inform, 2020, 50: 318–334 [王涛, 尹刚, 余跃, 等. 基于群智的软件开发群体化方法与实践. 中国科学: 信息科学, 2020, 50: 318–334]

34 Li Z, Yu Y, Zhou M, et al. Redundancy, context, and preference: an empirical study of duplicate pull requests in OSS projects. IEEE Trans Softw Eng, 2022, 48: 1309–1335

35 Zhang Y, Wang H, Yin G, et al. Social media in GitHub: the role of @-mention in assisting software development. Sci China Inf Sci, 2017, 60: 032102

36 Li Z, Yu Y, Wang T, et al. Opportunities and challenges in repeated revisions to pull-requests: an empirical study. In: Proceedings of the ACM On Human-Computer Interaction, 2022

37 Li Z, Yu Y, Wang T, et al. Are you still working on this? An empirical study on pull request abandonment. IEEE Trans Softw Eng, 2022, 48: 2173–2188

38 Lu Y, Mao X, Zhou M, et al. Motivation under gamification: an empirical study of developers' motivations and contributions in stack overflow. IEEE Trans Softw Eng, 2022, 48: 4947–4963

39 Zhang X, Wang T, Yu Y, et al. Who, what, why and how? Towards the monetary incentive in crowd collaboration: a case study of Github's sponsor mechanism. In: Proceedings of CHI Conference On Human Factors In Computing Systems, 2022

40 Zhang X, Yu Y, Gousios G, et al. Pull request decisions explained: an empirical overview. IEEE Trans Softw Eng, 2023, 49: 849–871

41 Zhang X, Yu Y, Wang T, et al. Pull request latency explained: an empirical overview. Empir Softw Eng, 2022, 27: 126

42 Zhang Y, Wu Y, Wang T, et al. iLinker: a novel approach for issue knowledge acquisition in GitHub projects. World Wide Web, 2020, 23: 1589–1619

43 Wang H M, Wang T, Yin G, et al. Linking issue tracker with Q&A sites for knowledge sharing across communities. IEEE Trans Serv Comput, 2018, 11: 782–795

44 Yin G, Wang T, Wang H, et al. OSSEAN: mining crowd wisdom in open source communities. In: Proceedings of IEEE Symposium On Service-Oriented System Engineering, 2015. 367–371

45 Zhang X, Wang T, Yin G, et al. DevRec: a developer recommendation system for open source repositories. In: Proceedings of the 16th International Conference On Software Reuse, 2017. 3–11

46 Yu Y, Wang H, Yin G, et al. Reviewer recommendation for pull-requests in GitHub: what can we learn from code review and bug assignment? Inf Softw Tech, 2016, 74: 204–218

47 Li Z, Yu Y, Wang T, et al. To follow or not to follow: understanding issue/pull-request templates on GitHub. IEEE Trans Softw Eng, 2023, 49: 2530–2544

48 Zhang T, Yu Y, Mao X, et al. FENSE: a feature-based ensemble modeling approach to cross-project just-in-time defect prediction. Empir Softw Eng, 2022, 27: 162

49 Vasilescu B, Yu Y, Wang H, et al. Quality and productivity outcomes relating to continuous integration in GitHub.

In: Proceedings of the 10th Joint Meeting on Foundations of Software Engineering, 2015. 805–816
50 Wu Y, Zhang Y, Xu K, et al. Understanding and predicting docker build duration: an empirical study of containerized workflow of OSS projects. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, 2022. 1–13
51 Zhang Y, Vasilescu B, Wang H, et al. One size does not fit all: an empirical study of containerized continuous deployment workflows. In: Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), 2018. 295–306
52 Zhao W X, Zhou K, Li J Y, et al. A survey of large language models. 2023. ArXiv:2303.18223

Crowd intelligence paradigm: a new paradigm shift in software development

Huaimin WANG[1,2], Yue YU[1,2*], Tao WANG[1,2] & Bo DING[1,2]

1. *College of Computer, National University of Defense Technology, Changsha* 410073*, China*;
2. *National Key Laboratory of Complex Critical Software Environment, Changsha* 410073*, China*
* Corresponding author. E-mail: yuyue@nudt.edu.cn

Abstract As a unique intellectual activity of human beings, software development has experienced a historical transformation from individual creation models in workshops to well-organized groups of industrialized production models and large-scale group creation models, which represents two types of paradigm shifts (i.e., the shift from the engineering paradigm toward the open source paradigm). The engineering paradigm concentrates on the software development of linear deterministic problems and almost pays no attention to uncertain problems, while the open source paradigm is appropriate for uncertain problems but does not make deterministic commitments regarding development results. This paper summarizes the key ideas of these two software development paradigms and proposes a new paradigm shift in software development, namely the crowd intelligence paradigm. This paradigm follows the scientific view of "crowd intelligence," core concept of "macroevolution and microrefinement," and methodology of "two connections, one transformation," which would guide software development activities in the ternary human–cyber–physical universe.

Keywords software development, paradigm shifting, engineering paradigm, open source paradigm, crowd intelligence paradigm