SICHUAN UNIVERSITY

题　　目 CMMI 成熟度模型和项目评估

学　　院 软件学院

学生姓名 朱铄

专　　业 软件工程

学　　号 2022141461135

指导教师 母攀良

关键词：代数式代码；智能体；函数式编程；范畴论；类型系统；可验证人工智能

摘要：随着大语言模型（LLM）驱动的智能体系统在各类场景中的广泛应用，如何提升其结构清晰性、可验证性与系统可靠性，成为当前智能体开发的重要挑战。本文提出将代数式代码范式引入智能体架

构中，通过函数组合、代数变体类型与类型系统的约束，实现感知 - 推理 - 行动过程的结构化表达。我们系统介绍了代数式代码的数学基础，包括一般代数与范畴论的映射关系，并分析了 Swift、Scala、F# 与 Rust 等语言在代数式建模方面的特性差异。此外，本文构建了一个基于 Azure Function 和 GPT-4o mini 的最小可运行智能体，并通过组合式错误恢复与逻辑单元测试验证其实用性。最后，本文探讨了在多智能体协同、类型级推理等方向上代数式代码的潜力与未来研究空间。该方法为构建结构透明、可组合与可验证的智能体系统提供了全新路径。

一、CMMI 的层次成熟度模型

在我们平时的软件开发过程中，经常会遇到项目管理混乱、需求变来变去、最后产品质量也难以保证的问题。这时候，一个科学、系统的过程管理框架就显得非常重要了。CMMI，也就是“能力成熟度模型集成”（Capability Maturity Model Integration），就是专门用来帮助组织规范和优化软件开发流程的一个模型。它最早是由美国卡内基梅隆大学的软件工程研究所提出的，现在已经成为衡量软件团队开发流程是否规范、是否成熟的重要标准，很多软件公司和大型项目都会参考这个模型来提升管理水平和产品质量。

CMMI 把软件开发的过程成熟度分为五个等级，从最低的第 1 级到最高的第 5 级。每一级都代表了一个组织在项目管理、流程规范、质量控制等方面的不同水平，也为组织提供了明确的改进方向。换句话说，CMMI 就像是一张“成长路线图”，一步一步告诉你怎么从“写代码靠感觉”变成“有标准、有计划、有度量、能持续改进”的专业开发流程。

CMMIC 层次成熟度模型分为五个等级：

1. 初始级（Initial）

这是最基础的一个等级，也就是说团队几乎没有明确的开发流程，一切都靠经验和临场发挥。项目能不能成功主要看人，如果团队里某个人特别厉害，项目可能就顺利；但一旦遇到人员变动或需求变更，项目很容易出问题。

2. 可管理级（Managed）

到了第二级，团队开始有一些基本的管理意识，比如会做项目计划、控制进度、记录需求变化等。虽然流程还不完善，但至少项目的执行开始变得可控，出了问题也能及时发现和调整。

3. 已定义级（Defined）

这个等级的团队已经有了相对成熟的开发流程，并且会在整个组织中推广统一的标准，比如统一的编码规范、文档模板、评审流程等。这意味着每个项目虽然由不同人做，但遵循的是同一套流程，效率和质量都有了保障。

4. 量化管理级（Quantitatively Managed）

第四级更进一步，不只是“有流程”，而是开始“用数据说话”。比如会定期统计缺陷数量、测试覆盖率、开发进度偏差等，用这些量化指标来帮助做决策和优化流程。这个阶段的管理就像是“有仪表盘”的管理，不靠感觉，而是靠度量。

5. 优化级（Optimizing）

最后一级是最成熟的状态，不仅流程规范、数据完整，而且还会主动做过程改进，比如分析项目中反复出现的问题、推广新的开发工具或实践，形成持续优化的机制。这个阶段的团队具备强大的自我提升能力，即使外部环境变化，也能灵活应对。

总的来说，CMMI 是一个帮助开发团队从“混乱无序”走向“高效可控”的指南，尤其适合用于评估我们在课程项目或科研开发中的过程管理水平，看看哪些地方做得好，哪些地方还可以改进。

二、评估过往开发软件成熟度

我去年参加新加坡国立大学暑期课程期间，独立完成了一个软硬件结合的图像识别项目。项目的核心目标是通过摄像头实时获取视频流，识别镜头前出现的猫的品种。整个系统由三部分组成：一是基于 TensorFlow Keras 训练得到的图像分类模型；二是部署在云端的识别服务器和远程控制界面；三是由树莓派和 Arduino 控制的机器人小车，负责视频采集与执行控制命令。小车通过 Flask 构建的接口与云端进行通信，实现视频上传和指令下发。该项目是一次完整的软硬件协同开发过程，具有较强的实践意义。

过程成熟度评估（基于 CMMI 框架）

结合 CMMI 的五个层次，我认为我这次项目整体上处于 Level 2（可管理级） 与 Level 3（已定义级） 的过渡阶段，具体如下：

1. 项目规划与控制（Level 2）

(1) 项目在初期制定了明确的目标：从模型训练到实时识别、从数据采集到远程控制，整体技术路线清晰。
(2) 虽然没有严格遵守甘特图，但完成度较高。
(3) 开发过程中的重要阶段如模型训练、云端部署和硬件测试，均进行了阶段性验证和记录，具备基本的可追踪性。
(4) 风险应对措施有限。例如中间遇到模型识别误差高的问题，主要通过人工调参和重采样数据集来应对，但缺乏系统的质量控制机制。

2. 开发流程规范（部分达到 Level 3）

(1) 编码过程中使用了基本的代码风格规范（如 PEP8），并通过 Git 进行版本控制，保证了代码的可追溯性。

(2)对 Flask 接口和模型推理逻辑编写了部分注释和接口文档，但整体文档化程度偏弱，未形成统一的技术文档。
(3)虽然模型训练过程有记录使用的数据集、参数配置和训练日志，但未建立“过程资产库”，未对模型版本及实验过程做标准归档。
(4)基本没有代码评审和单元测试

3. 流程量化与持续改进（未达到 Level 4/5）

(1)没有系统性地采集开发过程中的度量数据（如开发周期、识别准确率变化、Bug 数量等）。
(2)没有使用测试覆盖率工具、性能分析工具等进行数据驱动的过程优化。
(3)缺乏对整个流程（特别是数据预处理、模型部署、接口测试等）的优化反馈闭环，无法实现基于数据的持续改进。

改进建议与计划

针对目前处于 Level 2~3 的状态，我制定了一些切实可行的改进计划，以逐步向更高成熟度演进。

1. 项目流程标准化（迈向 Level 3）

(1)文档体系补全：补充并统一包括架构设计说明、接口文档、模型训练过程说明、部署指南等文档，确保他人也能复现该系统。
(2)开发模板和代码规范：在未来类似项目中建立统一的代码模板（如 Flask 接口模板、训练脚本模板），并使用 `pre-commit` 工具自动检查代码风格。
(3)模型管理工具引入：考虑引入 MLFlow 或 Weights & Biases 管理训练过程与实验版本，以提升模型管理与追踪能力。

2. 测试与质量控制（迈向 Level 4）

(1)引入自动化测试：为 Flask 接口编写单元测试和 API 测试脚本，使用 `pytest` 和 `Postman` 测试模型服务的稳定性与正确性
(2)部署端监控指标：部署端增加模型推理时间、请求响应率、准确率的实时监控，并设置性能报警机制。
(3)数据版本与异常管理：将训练数据版本化（如通过 DVC 工具），保证模型版本与数据的一致性；同时建立错误日志分析机制，提升问题定位效率。

3. 量化过程管理（迈向 Level 4~5）

(1)建立过程度量体系：记录项目中各阶段所用时间、问题数量与类型、识别精度变化等，作为未来优化的依据。

(2) 定期回顾机制：每次项目完成后进行回顾总结，记录经验教训与流程痛点，为后续项目提供参考。
(3) 持续优化机制：引入 GitHub Actions 等自动化工具，在每次代码提交时执行测试、检查模型依赖和代码变更，构建自动化 DevOps 流水线。