Model-Execute-Update (MEU) - A Framework for Robust and Steerable AI-Assisted Development

Nils Kakoseos Nyström

April 6, 2025

1 Intent Specification

This document describes a framework for AI-assisted development workflows, intended to establish the foundation for robust, transparent, and comprehensibly steerable processes for implementing software projects involving both human and AI-driven software engineering. The goal of the proposed framework is to facilitate the design and implementation of workflows for efficiently human-steerable AI-driven code generation which:

1. Enforces transparent, **modular structure** in the repository throughout the development roadmap.
2. Facilitates robust **continual automated testing** of local software elements and their global integration accounting for all interdependencies.
3. Aids and enhances **code comprehension** to both human developers and AI coding assistants of the global code structure and interdependencies across all local testable software components.

These properties gives the potential to make software development both radically faster and more robust by enabling extensive automation of code generation while maintaining architectural principles and human understanding. In particular the proposed framework aims to facilitate coding workflow designs that both aids and incentivizes human comprehension and steering of the AI-enhanced software development process.

2 Implementation Specification

Our main strategy for enhancing AI-driven development while making it more comprehensible and steerable is to utilise accurate and comprehensibly structured embedding maps of the codebase in terms of relational graphs across text-embeddings of the project repository source code. This embedding map should ideally be completely humanly comprehensible while accurately and robustly maintaining a *local* $\rightarrow$ *global* indexing structure encoding the sourcecode and all relevant interdependency relations across the project repository throughout the project implementation roadmap.

To achieve the aim of optimal balance between robustly automizable- and human steerable- software development processes, the proposed framework moreover builds on concepts from test-driven development (TDD), particularly as instantiated within behaviour-driven development (BDD).

These conceptual frameworks for software design and development often refers to processes which can be described as logical triplets, formulated in the language of Hoare-logic as precondition-command-postcondition, or in the analogous syntax of Gherkin (a domain specific language for software behaviour specification) Given-When-Then.

As a foundation for our proposed framework we will thus reformulate and generalise these principles abstractly as follows [1].

Definition (MEU-Triplet). A MEU-triplet (M, E, U) consists of three components, respectively defining:

1. **Model (M):** The state of a model encoding current assumptions for operating and performing computation on data in the environment to obtain system goals.
2. **Execution Environment (E):** The computation environment in which the system operates and interacts, including channels for sensing measurable feedback data.
3. **Update Domain (U):** The domain of evaluator states providing mechanism by which the system evaluates or verifies feedback and updates its model state (M) accordingly.

The above defined three domains comes with six arrows **a**, **b**, **c**, **d**, **e**, **f**, respectively specifying the following mechanims for mapping data across the domains:

a: $M \to E$; Implementation/instantiation of M for execution in E.

b: $E \to U$; Logging of feedback from E and structuring for U.

c: $U \to M$; Update logic to M based on U's evaluation of feedback.

d: $M \to U$; Updates to U's verification criteria based on M's updates.

e: $U \to E$; Updates to E's logging/execution based on U's update mechanisms.

f: $E \to M$; Influence of E's feedback on M's input features/input data structure/orchestration.

Each of the three components of a MEU-triplet can be recursively subdivided into similar lower-level triplets, forming a vertically nested hierarchy of MEU-triplets.

Concretely, in our proposed development framework this abstract logical triple is instantiated as follows. Firstly, as starting point for the implementation of a **project** is a **project intention statement** defined in three parts, respectively specifying:

1. **(M):** Initial assumptions and goals of the project.
2. **(E):** Specification of the environment where the result of the project will operate (e.g., a software execution and/or deployment environment).
3. **(U):** Process for continual updates to the specification and operation, based on feedback from the environment.

Next, the initial project intent specification is broken down into one or more parallel sequences (**roadmap tracks**) of implementation tickets. In each such roadmap track, each individual ticket is associated with its own unique MEU-triplet, and for each of its subtask-tickets a nested roadmap

[1] While this framework is logically similar to the triplets in Hoare logic as used in the foundation of BDD, it is more geared towards statistical inference and reinforcement learning agents.

track of MEU-triplets is defined, and so on recursively for a finite number of times, until a "leaf" MEU-triple is reached, whose roadmap track and subcomponents are all "identity". In the context of a software development project, each ticket-associated MEU-triplet **must** implement the following:

1. **(M):** An implementation or specification of the current state of the testable software element described in the ticket.
2. **(E):** The execution environment for the associated software components including logging mechanisms for recording execution feedback data.
3. **(U):** The mechanisms for updating the state definied in (M) based on the logged data recorded from execution/test in (E).

Each ticket can be subdivided into subtasks along a vertical hierarchy, such that each associated MEU-triplet is similarly nested according to the definition above.

The framework thus arranges MEU-triplets along two principal axes:

1. **Horizontal Axis:** Represents a linear workflow of implementation tickets across one or multiple parallel project roadmap tracks.
2. **Vertical Axis:** Represents the nested hierarchical structure of MEU-triplets. This extends from the top-level project specification down to the lowest-level source code elements (each treated as a single testable software component).

Additional dependencies and relationships can exist among triplets or their subcomponents, potentially spanning different roadmap tracks and tickets. Consequently, the entire project can be encoded as a graph whose nodes are embedded MEU-triplets, with edges representing dependencies and relationships among tickets and components.

In the project repository, each testable software component associated with a ticket (in the hierarchy of nested subtasks tickets along the horizontal parallel roadmap tracks) is indexed precisely according to its position in the project graph. Each such component is also via text embedding stored in a vector database, with metadata specifying its placement within the project graph. [2]

This vector embedding graph is then used to *navigate* context data provided to large language model (LLM) agents that assist with the software project implementation.

[2]This global graph across the project source code in particular contains the following three subgraphs with complementing edge-sets:

1. The tree-graph with edge set branching *"vertically"* down from the top level MEU-triplet (representing the project description) downwards across all nested MEU-triplets.
2. The graph with edge set connecting all MEU-triplets dependent *"horizontally"* across a *roadmap track* (possibly nested within a MEU-triplet or any of its components).
3. The graph whose edge set (complementing the two above) represents all other interdependencies across MEU-triplets.

It may be useful to embedd these three graphs separately to encode different aspects of the source code. It can moreover be noted that the hierarchically nested roadmap tracks of MEU-triplets have the structure of sieves (in the category theory -sense), and hence that the thus defined project abstractly can be viewed as a Grothendieck topos across the three domains, viewed as "sites". In this view, the mappings a, b, c, d, e, f defined above across domains are functors defining three adjunctions (with corresponding monads) across the domains, and the global U-part hence instantiates these as sheaf morphisms adapting the structure of the thus interlinked topoi interactively with the environment dynamics. Conjecturally, this highly stringent mathematical structure facilitates the mathematical optimization necessary for learning local-to-global structures in dynamically interactive online environments beyond what has been possible with prior machine learning techniques.

3 Update Specification: Proof of concept (POC)- implementation and evaluation framework for MEU

To verify the consistency and usefulness of the MEU-framework, we propose to utilise this specification of it as its M-domain; a blueprint prompt to illicit the relevant knowledge from large language models (LLMs) in order to utilise the MEU-framework to implement and evaluate real world software projects with specified M-, E-, and U-components, hosted in a github organisation as a curated software project available to selected contributors. The environemnt E thus consists of a github repository organisation containing repositories which have been implemented using the MEU-framework. The U is specified by a neural model of "interestingness" together with the collection of user-specified evaluation-criterions in the sub-repositories of the organisation. The mappings are specified by;

a: $M \rightarrow E$; Specifies the instantiation of the MEU-blueprint in guiding the development of the project repositories.

b: $E \rightarrow U$; Logging of feedback data from users project implementation based on their verification and evaluation mechanisms.

c: $U \rightarrow M$; Update logic to M based on U's evaluation of feedback, as a pull request on the MEU-specification which is hosted in the repository.

d: $M \rightarrow U$; Updates to U's verification criteria based on M's updates.

e: $U \rightarrow E$; Updates to E's logging/execution based on U's update mechanisms.

f: $E \rightarrow M$; Specifies an update to the mechanisms instantiating the MEU-specification in project implementations, based on feedback from project implementations.

Influence of E's feedback on M's input features/input data structure/orchestration.