Using Process Mining to Generate AI Agents from Software Engineering Process Records

Saimir Bala[1], Fabiana Fournier[2,3], Lior Limonad[2,3], and Andreas Metzger[4]

[1] Hasso Plattner Institute (HPI), University of Potsdam, Germany
saimir.bala@hpi.de
[2] IBM Software Innovation Lab (SIL), Haifa
fabiana@il.ibm.com,liorli@il.ibm.com
[3] Dept. of Information Systems, Fac. of Comp. & Inform. Science, University of Haifa
[4] paluno Institute, University of Duisburg Essen, Essen Germany
andreas.metzger@paluno.uni-due.de

Abstract. Integrating AI agents into Software Engineering (SE) raises an important challenge: how can we specify and realize AI agents that work effectively alongside humans in hybrid SE teams? Determining the right granularity and separation of concerns for such agents is non-trivial. Coarse-grained agents may introduce unmanageable complexity, whereas micro-agents may create severe coordination overhead. Moreover, existing multi-agent SE frameworks typically rely on predefined role structures and do not account for project-specific characteristics or process adaptations. We address this by combining object-centric, imperative, and declarative process mining. Using event logs extracted from software repositories, our approach discovers project-specific agent roles and generates matching agent specifications and implementations. As proof-of-concept, we applied our approach to a well-established open-source project. We performed functional tests of the resulting AI agents and an exploratory user study to determine how well the generated AI agent specifications are aligned with human expectations.

Key words: Process Mining, AI Agents, Software Engineering

1 Introduction

Process mining turns event data into insights about how work is performed, who performs it, and under which constraints [7]. Software engineering (SE) processes are no exception: the activities of developers, reviewers, and testers leave rich event traces in version control and issue management systems, forming a largely untapped source of process knowledge. Today, SE teams are undergoing a fundamental transformation as AI agents (autonomous systems capable of decomposing goals, executing tasks, and collaborating with humans) are increasingly embedded into development workflows [21, 2, 19]. The SE community has already acknowledged the rise of *AI teammates*, and the evolution towards *hybrid human-AI SE teams* raises an important new engineering challenge: *How to specify and realize AI agents that can effectively work alongside humans?*

Addressing this question has several challenges. Assigning a classical SE role (such as *team member* in agile processes) to a single agent results in a monolithic and complex teammate that is difficult to build, test, maintain, and hand tasks over to. Conversely, defining highly fine-grained, single-task agents increases coordination overhead among agents and humans. Also, not all SE tasks may be relevant in every project, and those that are may require concrete adaptations. Formal verification may be mandatory in safety-critical systems yet irrelevant in a simple utility tool, while complex business logic may demand property-based tests beyond standard unit testing. A predominant expectation of human software engineers is that agents strictly adhere to established standards and project-specific processes [10]. Current frameworks rely on rigid, predefined canonical roles that do not adapt to the actual workflows of a given project [13, 21, 11, 14].

We address these challenges with PM$_4$AA, a generative pipeline that mines project-specific SE agent specifications directly from repository event data. PM$_4$AA treats SE processes as business processes and the activity history recorded in version control and issue management systems as event logs. It applies object-centric process mining [23] to capture the multi-object structure of SE activities. This allows the discovery of the task scope and object interactions of each role, and declarative process mining [8] to elicit behavioral constraints that serve as guardrails for agent execution. Together, these two complementary views provide both the procedural blueprint and the normative boundaries needed to specify a well-scoped agent. PM$_4$AA then leverages LLMs for process model analysis [15] to synthesize executable agent specifications. As a proof of concept, we implement PM$_4$AA using the LangGraph framework and evaluate it through a case study on the open-source *Commitizen* project, which yields an event log of more than 21,000 events spanning eight years of development history, functional testing of the generated agents, and a user study involving ten human participants, who assessed how well process-mined, project-specific agents align with human expectations in hybrid SE teams.

Below, Sec. 2.1 introduces relevant background and related work. Sec. 3 explains PM$_4$AA. Sec. 4 presents the case study and reports our PM$_4$AA evaluation. Sec. 5 concludes the paper and outlines future work.

2 Preliminaries

2.1 Background

Process Mining: Process mining extracts insights about processes from event data traces [7]. Object-Centric Process Mining (OCPM) [1, 23] extends traditional process mining by capturing processes that involve multiple interacting objects (such as orders, invoices, and shipments) rather than a single predefined case, enabling accurate representation of synchronization, interaction, and dependency relations. A central OCPM artifact is the Object-Centric Directly-Follows Graph (OC-DFG), which extends the classical Directly-Follows Graph (DFG) to multiple object types simultaneously.

Declarative process mining elicits process behavior through constraints that govern allowable behavior, rather than prescribing a fixed sequence of activities. Constraints express temporal or logical relations such as precedence, response, or mutual exclusion [8, 20, 18, 17]. Declarative approaches are well-suited to flexible, knowledge-intensive processes where strict control-flow specifications are difficult to define or maintain. Languages such as DECLARE have become widely adopted due to their formal semantics and expressive power, and have been applied in healthcare, customer service, and human-centric workflows.

Event Logs from Software Repositories: Software development is a process: developers open issues, write code, commit changes, review pull requests, run tests, and release software. Like a business process, it generates event data as a natural by-product of execution. These events are captured in version control and issue management systems such as Git, GitHub, and Jira, which serve as the primary source of ground truth about what happened, when, and by whom.

Repositories provide a rich set of data. A single commit records the actor, a timestamp, a descriptive message, and references to modified files, related issues, and pull/merge requests. An issue has its own lifecycle: opened, labeled, assigned, commented upon, and resolved. A pull/merge request links one or more commits to one or more issues and involves multiple actors across review, approval, and merge events. A single SE action, such as fixing a bug, thus spans multiple objects simultaneously, which is precisely what OCPM is designed to capture [23].

2.2 Related Work

AI agents possess their own thread of control and make autonomous decisions about which actions to perform and when [25, 16]. They decompose high-level goals into tasks and execute them in unstructured environments. Agents can form multi-agent systems in which they collaborate with humans and other agents to carry out tasks [25, 22, 4]. Their tasks are realized through AI algorithms and models, including generative AI such as LLMs [12]. In SE, examples include autonomous coding agents that initiate, review, and evolve code at scale [16].

Recent systematic literature reviews document a rapid paradigm shift from single-agent coding assistants to complex multi-agent SE ecosystems [19, 13]. These reviews emphasize that relying on a single, monolithic AI agent is insufficient due to the multifaceted complexity of real-world software projects. Consequently, state-of-the-art frameworks (such as ChatDev and MetaGPT) divide the development lifecycle into specialized roles—such as product managers, programmers, and testers—to improve task execution, collaborative reasoning, and fault tolerance. However, most multi-agent SE ecosystems rely on fixed architectures based on canonical, predefined roles such as Programmer, Reviewer, and Tester [13]. Rigid role structures and informal communication protocols may cause severe inter-agent coordination bottlenecks when scaling to complex projects [19]. Both [19] and [13] call for adaptation of agent roles based on evolving project demands, yet operational and data-driven techniques to achieve this are still missing. Our PM$_4$AA approach directly addresses this challenge by

dynamically extracting, scoping, and generating project-specific SE agents from project-specific event logs.

The BPM field has developed organizational mining and Agent System Mining (ASM) to derive role structures and behavioral models from event logs [24]. ASM tools like *Agent Miner* extract complete, multi-agent behavioral and interaction models directly from event logs. Still, these approaches are retrospective and simulative: they mine structures to generate synthetic traces or build digital twins of existing processes. Our work takes a more prescriptive and operational stance. We argue that event logs contain role-specific behavior and focus on mining (OC-)DFGs and declarative constraints to engineer actual AI agents rather than simulate human actors.

While PM$_4$AA shares the foundational premise of Shen et al. – that historical event logs contain vital, role-specific behavioral blueprints – our objective is to build a *prescriptive* and *operational* approach in which mined process models and declarative constraints are not there to simulate human actors, but to engineer actual AI agents and deploy them within an executable LangGraph architecture.

3 The PM$_4$AA Approach

PM$_4$AA (Process Mining for Agentic AI) is a generic, multi-step pipeline that transforms SE repository data into operational, role-specific AI agents. Fig. 1 provides an overview of the steps of the PM$_4$AA pipeline, which we elaborate below. A repository containing the full pipeline implementation and the accompanying application is available at: https://github.com/liorlimonad/pmaa.

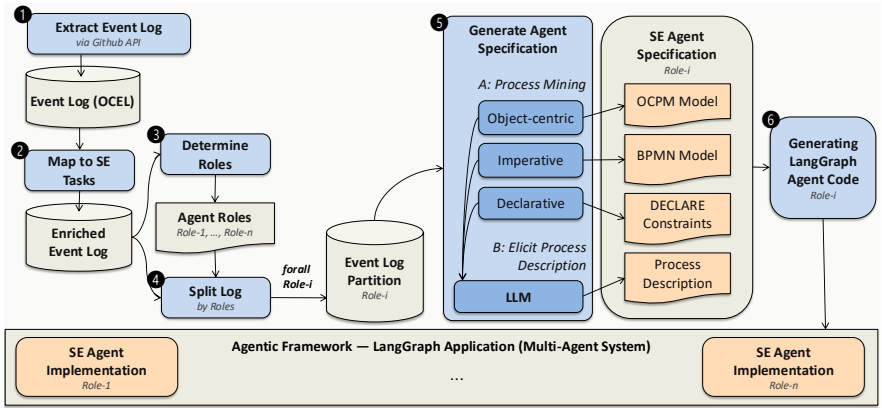


Fig. 1: The PM$_4$AA pipeline.

Step ❶ Event Log Extraction: The first step is to extract a raw Object-Centric Event Log (OCEL) from a GitHub repository using *PyStack't* [3], a Python library designed to produce OCEL 2.0 logs via the GitHub REST API. PyStack't maps GitHub entities to OCEL object types (i.e., user account, issues and commits), and repository interactions to timestamped events, yielding a log that natively

captures the multi-object nature of software development activity. To derive roles, we first identify the tasks recorded in the event log.

Step ❷ Mapping to SE Tasks: The raw OCEL does not represent SE *tasks* as first-class objects. PyStack't also does not parse pull requests (often associated with tasks). Therefore, we need to derive the tasks from the commit messages.

A key enabler for this step is the commit message. Many open-source projects enforce *Conventional Commits* [6], a lightweight standard that structures commit messages as `type(scope): description`. The type prefix classifies the intent of each change: `feat` signals a new feature, `fix` a bug repair, `docs` a documentation update, and `test` a testing contribution. This allows assigning each event a corresponding type of SE task without manual annotation [21]. Many projects (e.g., Angular, Vue.js, ESLint, and Commitizen) follow this standard.

We thus enrich the event log by parsing each commit message against the Conventional Commits pattern *type(scope): description* [6] using a regular-expression classifier. Each successfully parsed commit is annotated with a `task_type` (e.g., `feat`, `fix`, `docs`) and a `task_semantic_class` drawn from a fixed ten-class classification scheme derived from the Conventional Commits specification (e.g., `feature_development`, `defect_resolution`, `quality_assurance`). A corresponding `task` object is created for each matching commit and linked to its commit and to all events that reference that commit.

Step ❸ Determine Roles: Given the tasks, the goal of this step is to derive a set of process roles from the contributor behavior recorded in the OCEL (cf. organizational mining [7]). All persons who have contributed to the repository (e.g., by committing a change) are available as user objects (collected in Step ❶). For every user object, a *resource profile* is computed by aggregating three behavioral dimensions: (i) the distribution of activities the user participates in, (ii) the number of commits attributed to the user, and (iii) the distribution of task semantic classes (e.g., feature development, defect resolution, documentation) obtained from commit-level attributes in the log.

A *priority-ordered rule-based* classifier then maps each profile to exactly one *fixed* and *prescribed* role. The rules discriminate along two axes: *platform identity* and *behavioural dominance*. Automated accounts are identified first through their platform metadata. Contributors with no commit activity are classified as issue reporters, capturing actors whose participation is limited to issue creation and discussion. Among committing contributors, those exceeding a configurable commit-count threshold and exhibiting diversity across at least three task classes are classified as maintainers. The remaining contributors are assigned to specialist roles (such as quality engineer, DevOps engineer, technical writer, or feature developer) based on the task-class group that accounts for the plurality of their work, with a generic *contributor* role as fallback. The assigned role is persisted as a new attribute on the corresponding user object in the enriched OCEL log, enabling the downstream partitioning in the next step.

Step ❹ Split Log: Having the roles, we isolate the behavior of each user in the event log, and use it for our goal of specifying agents. Using the role-annotated

OCEL, a per-role sub-log is constructed for each discovered role. For a given role, each event in which at least one user object carrying that role participates is selected. All objects referenced by those events (including non-user objects such as issues, commits, and tasks) are retained, thereby preserving the object-centric structure of the original log. The result is one OCEL sub-log per role, capturing the multi-typed object context of the activities performed by that role.

Step ❺-***A****: Process Mining.* The per-role event log partitions serve as input for three complementary process mining steps:

Object-centric: An OC-DFG is discovered using the `pm4py` library. To filter out infrequent behavior while still capturing sub-logs of varying size, a dynamic frequency threshold is applied: both the activity and edge thresholds are set to approximately 2% of the sub-log's event count, with a minimum of one. The resulting OC-DFG captures the typical activity sequences of each role, annotated with the object types co-involved in each transition. The OC-DFG serves a dual purpose in the pipeline. First, it delineates the *task scope* of the agent: the set of activities retained after threshold filtering defines which operations the agent is responsible for. Second, the object types appearing in the graph specify the *entities the agent interacts with* (e.g., issues, commits, pull requests), grounding the agent's interface in empirical process data rather than manual specification.

Imperative: Complementing the OC-DFG, a BPMN model is derived via Inductive Miner. The log is first flattened onto a designated primary object type (selected as the case notion), converted to a traditional event log, and the resulting process tree is then transformed into a BPMN diagram. Together, with the OC-DFG this provides a per-role view of *how* work is carried out.

Declarative: Declarative process mining is applied to the role-specific logs to elicit normative behavioral constraints, complementing the imperative models produced in the previous steps. Using the `Declare4Py` [9] library's `DeclareMiner`, each log was first flattened onto the `issue` object type to yield a case-centric representation, and constraint discovery was executed with a minimum support threshold of 0.5 and a maximum constraint cardinality of two. Roles exhibiting fewer than two distinct activities after flattening were excluded from discovery, as the absence of activity diversity precludes meaningful constraint inference. For the remaining eligible roles, the `DeclareMiner` was executed, and the discovered constraint sets were serialised as JSON artifacts for downstream consumption.

Step ❺-***B****: Elicit Process Description.* For each role, a process profile with activity frequencies, object-type distributions, directly-follows patterns, and event-object interaction counts is provided to GPT-4-mini using the following prompt.

Prompt to LLM: Analyze the process profile for `{role_name}` and generate a process description covering: (i) process overview and responsibilities, (ii) main activities grouped by function, (iii) object interactions and semantics, (iv) flow patterns and common sequences, (v) key behavioral characteristics, and (vi) business interpretation of the role's contribution to the software lifecycle. Base the analysis on an extracted process profile including total events/objects, unique activities/object types, activity frequencies, object type distributions, top directly-follows flows, and top event-object interactions. Write in professional language for business stakeholders, emphasizing object-centric behavior.

The markdown output provides a structured business-oriented description of the role-specific object-centric process, including behavioral, interactional, and statistical perspectives for each of the roles.

Step ❻ Generate LangGraph Agent Code: With the specification in place, we move to the implementation of the agents. We make use of LangGraph as the target Agentic AI technology to instantiate the roles generated in the previous steps. The LangGraph application is realized through a prompt-driven synthesis process in which IBM BOB[1] interpreted the user request and turned the mined SE Agent specifications into an executable application. IBM BOB employs a mixture of frontier and specialized language models for interpretation, drawing on Anthropic Claude, Mistral open-source models, IBM Granite, and additional specialized fine-tuned models. The BOB prompt is as follows:

⟨**Context**⟩ This workspace is applied to an event log harvested from a GitHub repository, which was then partitioned into 8 separate sub-logs corresponding to 8 SE roles invoked in the handling of the GitHub issues. We derived the Object-Centric Directly-Follows Graphs (OCDFG), BPMN graphs, DECLARE constraints and a process description markdown, for each of the logs.
⟨**Task**⟩ Project this onto an agentic application that will facilitate and automate, where possible, online issue handling on top of the GitHub repository. Inspect all the attached artifacts and propose a langgraph-based agentic application architecture matching these artifacts.

The resulting agent implementations are embedded into a LangGraph workflow with shared typed state, prompt templates, and routing logic. The population process combines mined process semantics with software architecture constraints: the mined artifacts determine what kinds of role behavior are meaningful. The LangGraph implementation turns those behaviors into executable agent nodes with explicit interfaces, routable responsibilities, and bounded actions.

4 Evaluation

We evaluated PM$_4$AA through (1) a case study, (2) functional testing of agent implementations, (3) a user study assessing the generated agent specifications.

4.1 Case Study: Commitizen

We applied PM$_4$AA to a repository of a real-world software project to demonstrate the technical feasibility of our approach. We chose *Commitizen*[2], a well-established open-source project active since 2017 and having 3,400+ stars and 339 forks.

Step ❶ Event Log Extraction. The raw OCEL extracted for *Commitizen* covers the period November 2017 – November 2025 (approximately eight years of project history) and contains 21,488 events and 4,813 objects across three object types: 1,459 issues, 2,765 commits, and 589 user accounts.

[1] https://bob.ibm.com/
[2] https://github.com/commitizen-tools/commitizen

Step ❷ Mapping to SE Tasks. Of the 2,765 commits in the log, 1,721 (62.2%) carry a well-formed Conventional Commit message and were successfully enriched, yielding 1,721 task objects and a final OCEL of 6,534 objects across four types (commits, tasks, issues, users).

Step ❸ Determine Roles. To partition the enriched OCEL into role-specific sub-logs, PM4AA automatically assigns each of the 589 user objects to one of eight roles using a rule-based classifier applied in priority order, see Table 1. The classifier first separates GitHub bot accounts (`bot`), then distinguishes users who participate in issue discussions but never commit (`issue_reporter`) from those who do. Among committing users, `maintainer` is assigned to those with at least 20 commits spanning at least three distinct task classes – reflecting broad, sustained involvement. The remaining committing users are classified by their dominant task with `contributor` being a "catch-all" for low-volume or single-contribution users. The four main, distinct roles (with ≥ 20 users) that remain are highlighted in bold.

Table 1: Role distribution in the Commitizen event log.

Role	Users	Events	Objects
issue_reporter	425	16,037	1,863
contributor	66	236	495
quality_engineer	37	194	395
technical_writer	23	71	194
feature_developer	20	100	212
maintainer	8	2,149	4,386
bot	6	3,394	1,055
devops_engineer	4	15	39
Total	589	21,488	6,534

Step ❹ Split Log. Given the above roles, the annotated OCEL was split accordingly, resulting in role-specific sub-logs, one per role.

Step ❺ Generate Agent Specification. Except `bot` and `issue_reporter`, all roles only performed a single activity type, and thus DECLARE constraints are only informative for the above two roles. The following is a summary of the markdown generated for the `issue_reporter` role.

PROCESS DESCRIPTION: ISSUE_REPORTER
Overview: The `issue_reporter` role initiates, coordinates, and follows issue lifecycles, linking issue and user objects through creation, discussion, review, subscription, assignment, closure, and reopening.
Main Activities: Issue initiation (`created`, `labeled`, `assigned`), communication (`commented`, `mentioned`, `subscribed`), review handling (`review_requested`, `reviewed`, `ready_for_review`), lifecycle management (`closed`, `reopened`), and structural updates (`referenced`, `cross_referenced`, `renamed`).
Object Interactions: The process centers on `issue` and `user` objects. Issues are updated through creation, discussion, and resolution, while users participate as actors, reviewers, and collaborators. Interactions coordinate visibility, accountability, and traceability.
Flow Patterns: Typical flows include `created` → `commented`, repeated communication (`commented` → `commented`, `commented` → `mentioned`), review paths (`reviewed` → `merged` → `closed`), and non-linear behaviors such as reopening or review removal.
Characteristics: Communication-driven, highly collaborative, iterative, issue-centric, lifecycle-spanning, and coordination-oriented.
Business Interpretation: The role supports software delivery by ensuring issues are captured, discussed, coordinated, and resolved while maintaining visibility and traceability.

For the remaining roles, the OC-DFG is the primary process model represented as shown in the following example, expressing that `Existence1[closed]` requires the activity `closed` occuring *at least once*.

```
{  "raw": "Existence1[closed] | |",
  "template": "Existence1",  "activity_a": "closed", "activity_b": ""  }
```

Step ❻ Generate LangGraph Agent Code. This yielded an issue-centric agentic application in which the `issue` object coordinates requests across all lifecycle stages. BPMN artifacts provided the issue management structure, DECLARE artifacts contributed policy checks and approval gating, and process descriptions populated the role-specific agents. The example below shows a human-readable agent specification, also used in our user study.

SE AGENT: issue_reporter

Goal: You focus on intake, clarification, communication, and sustained issue coordination.

Tasks (DO): • summarizing issue intent and missing context • asking clarifying questions • detecting duplicates • suggesting labels, assignees, milestones • preparing issues for downstream work

Avoid (DON'T): • make deep code changes directly • close issues without strong resolution evidence

Actions: • "summarize_issue", "ask_for_clarification" • Input: question: "Please provide expected behavior and actual behavior." • Output: issue_id, prepared documentation work items

4.2 Functional Testing

We tested the generated LangGraph application for *Commitizen* using actual GitHub issues. We report three representative test cases that checked whether the populated role-specific agents and the surrounding LangGraph control flow behave coherently on realistic issue inputs. In all cases, the app ingested the issue, loaded its GitHub context, interpreted the text through LLM-based classification, routed the issue to a role-specific agent, and derived a corresponding next action.

Issue #1: "Readme.md requires more content to describe this project." This was interpreted as a documentation-oriented request. The routing logic correctly selected the `technical_writer_agent`. In turn this agent then used its LLM-based analysis to derive a suitable documentation task focused on updating or preparing explanatory material.

Issue #2: "Add a first basic hello-world function in Python with no input arguments that prints 'hello world'." This issue was routed correctly to the `implementation_agent`. The agent translated the request into a development-oriented action plan, containing a concrete implementation workflow comprising repository inspection, minimal code changes, traceability links to the originating issue, and preparation of a pull request for review.

Issue #3: "Application crashes when uploading large CSV files." This issue was routed correctly to the `bot_agent`. The agent translated the issue into a workflow-oriented coordination plan, containing low-risk repository maintenance actions comprising issue triage through the application of `triage` and `needs-review` labels, routing to the appropriate review stage, and automatic reviewer assignment to the `maintainer` for further technical assessment.

4.3 Exploratory User Study

To empirically evaluate PM$_4$AA, we performed an exploratory, qualitative user study focusing on the four main, distinct roles identified in Table 1. The aim was to assess how well process-mined, project-specific agents address the problem of *Human-AI alignment*, which is a concept from HCI research referring to how well an AI agent meets the goal of producing the desired outcomes, without undesirable side effects [5, 11]. Due to the project-specific separation of concerns and agent modularity, PM$_4$AA's agent specifications should exhibit a relatively high degree of human-AI alignment.

We involved ten participants recruited from our pool of PhD students, teaching assistants, and Master students.

Study Flow: The study entailed the following steps. *1. Motivation and Introduction:* Participants received a briefing on the problem of modeling and realizing SE agents, the ensuing challenges of human-AI alignment, and the PM$_4$AA approach. *2. Case Study & User Study Goals:* The participants were given an overview of the case study, the generated SE Agent roles and the evaluation dimensions (explained below). We shared all identified roles except `bot`, as this is already automated by Github. *3. The Survey:* Human participants were informed about the study goals, protection of personal data and the voluntary nature of the study. Each participant received a set of agent specifications together with a questionnaire to be answered as detailed below.

Questionnaire Structure: We evaluate Human-AI alignment from a human-computer-interaction (HCI) perspective. We address the following alignment concerns introduced in [11], thereby contributing to the challenge of effective AI-human collaboration [21]. Assessments for these concerns are given on a five-point Likert scale: (1) strongly disagree, ..., (5) strongly agree.

Knowledge Schema: The information needed by an agent for a successful transaction must be clear to prevent back-and-forth communication. Herewith, we assess what level of contextual clarity role-scoped agents may provide.

> C1: The SE Agent specification clearly communicates what specific input the agent needs to successfully complete its task.

Autonomy: Humans must understand the boundaries within which an AI agent can operate autonomously and when it must consult a human for decisions.

> C2: The SE Agent specification makes it clear when the agent is allowed to act autonomously and when it must wait for human approval or interaction.

Operational: Humans and agents must actively identify and align on which tasks to execute, which requires that humans are aware of the agent skills.

> C3: The tasks and responsibilities of the SE Agent are clearly identifiable, making it easy to understand what specific SE work I can assign to it.

Accountability: Humans might worry that an agent misbehaves, which could negatively impact the humans' personal reputation. Consequently, for the sake of accountability, it must be possible to efficiently debug the AI agents.

C4: If the SE Agent makes a mistake (e.g., merging bad code), the agent specification(s) allow me to isolate, understand, and debug the specific problem.

Human Engagement: Because humans may have varying preferences for control and oversight, humans and SE agents should align on when, how, and why the human should be interrupted or engaged.

C5: Coordinating hand-offs, monitoring outputs, and intervening with the SE Agent would require minimal cognitive effort and cause low friction.

To strengthen the findings and compensate for the relatively small sample size, we asked additional open-ended, cross-cutting questions.

Q1: *Granularity* – Based on the agent specifications you reviewed, would you merge some agent roles, split them further, or keep them as generated?

Q2: *Collaboration* – Imagine you are a software engineer working alongside the AI agents on a real project. Are there any concrete moments in your workflow where you anticipate a handoff or coordination problem? If so, explain what you think which agent(s) and which reason(s) would cause it.

Q3: *Trust* – The role boundaries and SE agent behavior were automatically derived rather than designed by a human. Are there potential reasons that would undermine your trust or acceptance of such generated SE agents?

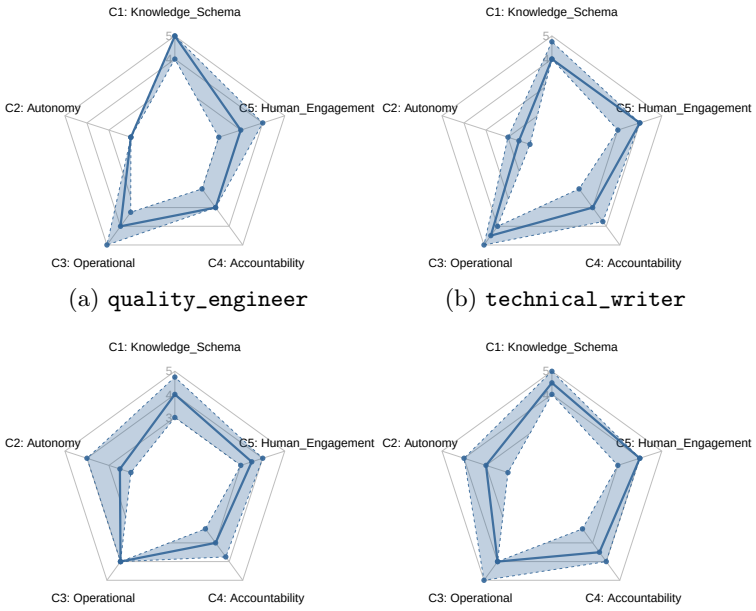
Data Quality and Threat Mitigation: To ensure the integrity of the survey data and mitigate the risk of careless responding (e.g., straight-lining), we implemented several established quality control mechanisms: (1) To mitigate fatigue and ordering effects when answering four roles × eight questions (= 32 questions in total), the presentation order of the agent specifications was *randomized* for each participant; (2) Statement C5 was *negatively framed* to detect straight-lining; (3) Responses that are *logically inconsistent* (e.g., answering "Strongly Agree" to both C1 and the reverse-coded C5) were flagged.

Results: Fig. 2 shows the medians of the participants' assessment for each of the four SE agents.

Overall Achievement of Alignment Criteria: Analyzing the medians across all four evaluated agent roles reveals distinct strengths and weaknesses in how process-mined specifications address human-AI alignment:

C1: Knowledge Schema (Med: 4): This was the highest-rated dimension, indicating strong agreement that the specifications clearly communicate the specific inputs the agents need to successfully complete their tasks. The object-centric process models successfully delineate task scope and required entities.

C3: Operational Clarity (Med: 4): Participants found the tasks and responsibilities of the agents to be highly identifiable, suggesting that deriving role boundaries from event logs yields practical and understandable work assignments.



(a) `quality_engineer` (b) `technical_writer`

(c) `issue_reporter` (d) `feature_developer`

Fig. 2: Radar Charts of the Qualitative User Study Results (showing Median as thick line and IQR as shaded area)

C5: Human Engagement (Med: 4): A moderate-to-high score implies that humans feel coordinating hand-offs, monitoring outputs, and intervening with the agents would require relatively low cognitive effort.

C4: Accountability (Med: 3): The ability to isolate, understand, and debug an agent's mistakes scored neutrally. While process boundaries are clear, the underlying execution may still appear a "black box" for debugging.

C2: Autonomy (Med: 2): This dimension scored notably low, highlighting a severe challenge. Humans struggled to understand the boundaries dictating when an agent is allowed to act autonomously versus when it must wait for human approval. This reflects a marked human-AI divergence on autonomy boundaries.

Role-Specific Assessment of C1-C5: A detailed breakdown of the five criteria across the four specific agent roles uncovers variations in how alignment is perceived depending on the nature of the software engineering tasks.

`feature_developer`: This role exhibited the highest overall human alignment. Participants found its operational scope (4) and required knowledge schema (4.5) exceptionally clear. It also achieved the highest accountability score (3.5), likely because code implementations provide concrete, verifiable artifacts (e.g., a pull request) that are native to human review processes.

`quality_engineer`: While the inputs (5) and tasks (4) for testing were well understood, accountability (3) and autonomy (2) dropped significantly, suggesting users are uncertain about how much leeway a testing agent has to automatically reject code or define its own test parameters.

`technical_writer`: This role exposed the most extreme disparity in the study. While participants clearly understood its operational purpose (4.50) and the inputs it requires (4.00), it received an exceptionally poor autonomy score (1.5). Users seem highly uncertain about whether documentation agents publish changes autonomously or strictly draft content for human approval.

`issue_reporter`: The issue reporter role yielded the most balanced, albeit slightly lower, assessment. Because this role centers on communication, coordination, and repeated review cycles, its autonomy boundaries (2.5) and human engagement expectations (3.5) may be slightly more apparent to human counterparts compared to the testing and writing roles.

Summary of Q1-Q3: Below, we summarize the main positive (+) and negative (−) comments to the open-text questions.

Q1: (+) Roles map to distinct SE responsibilities, avoiding monolithic complexity and provide good coordination balance without excessive friction. (−) `feature_developer` and `quality_engineer` roles seem too broad. Handoffs lack detailed boundaries, and there is perceived redundancy between `technical_writer` and `issue_reporter`.

Q2: (+) Agents successfully handle tedious tasks, such as technical documentation and systematic test coverage improvements. (−) Ambiguous validation criteria pose a risk of endless loops between `feature_developer` and `quality_engineer`. Furthermore, the rapid generation of agent output risks turning the human developer into a vetting bottleneck.

Q3: (+) There is strong support for agents handling unappealing peripheral work and preparatory phases, allowing humans to focus on major architectural decisions. (−) Echoing Q2, risks include endless validation loops, premature actions (e.g., reporters interrupting drafts), and severe human review bottlenecks due to high-volume outputs lacking prioritization guardrails.

4.4 Threats to Validity & Limitations

Our user study measures *perceived* human-AI alignment based on reading agent specifications, not actual task performance or agent behavior. A specification may appear well-scoped and comprehensible while producing misaligned behavior at runtime. We partly mitigate this through functional testing in Sec. 4.2.

The absolute assessments of our user study limit causal claims. Yet, determining a fair baseline for a comparative assessment is rather challenging in our case. Monolithic or fine-grained agent baselines are obviously unfair. Using canonical SE roles – as done by current frameworks [13] – while presenting realistic and feasible baselines, would significantly overlap with the generated roles. This makes it difficult to use them as hidden baselines, which are required to not compromise the study's validity due to psychological effects that skew perceived alignment.

The participant sample comprised Master's students, teaching assistants, and PhD researchers from a single university, recruited as a convenience sample. This may introduce social desirability bias. Also, as non-practicing software engineers, participants may not truly reflect the perspectives of developers deploying hybrid human-AI SE teams in production.

PM$_4$AA was evaluated on a single open-source project, *Commitizen*, which follows the Conventional Commits specification and exhibits a relatively clean event log. Generalization to projects lacking structured commit conventions, or to proprietary or enterprise repositories, remains an open question.

SE event logs capture coarse-grained developer activities (e.g., commits or issue updates), whereas AI coding agents operate at a finer granularity (e.g., writing code or running tests). Mined role specifications thus may not fully capture deployed agent behavior. OCPM partly mitigates this by linking commits, issues, and pull requests across object types.

5 Conclusion

We presented PM$_4$AA, a pipeline that derives project-specific SE agent specifications and implementations from SE repository event logs by mining roles, behavioral patterns, and normative constraints. As proof-of-concept we realized the pipeline and applied it to the *Commitizen* open-source project. Initial functional testing showed coherent issue routing, and an exploratory user study suggested that the generated specifications appear to be well-aligned with human expectations.

Future work will evaluate PM$_4$AA on more diverse repositories, including projects without structured commits and proprietary code bases, and involve practicing engineers. Enriching event logs with IDE telemetry and CI/CD logs will support finer-grained agent scoping and specifications.

References

1. van der Aalst, W.M.P.: Object-centric process mining: Dealing with divergence and convergence in event data. In: SEFM (2019)
2. Amalfitano, D., Metzger, A., Autili, M., Fulcini, T., Hey, T., Keim, J., Pelliccione, P., Scotti, V., Koziolek, A., Mirandola, R., Vogelsang, A.: A research roadmap for augmenting software engineering processes and software products with generative ai. ACM Trans. Softw. Eng. Methodol. (Jan 2026)
3. Bosmans, L., Peeperkorn, J., De Smedt, J.: Pystack't: Real-life data for object-centric process mining. In: Proceedings of the BPM 2025 Demos and Resources Forum. pp. 208–215 (2025)
4. Calvanese et al.: Agentic business process management: A research manifesto. Information Systems **140**, 102738 (2026)
5. Christian, B.: The alignment problem: Machine learning and human values. WW Norton & Company (2020)
6. Conventional Commits Authors: Conventional Commits 1.0.0 (2019), accessed: 2026-05-07
7. van Der Aalst, W.: Data science in action. In: Process mining: Data science in action, pp. 3–23. Springer (2016)
8. Di Ciccio, C., Montali, M.: Declarative process specifications: Reasoning, discovery, monitoring. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook, LNBIP, vol. 448 (2022)

9. Donadello, I., Riva, F., Maggi, F.M., Shikhizada, A.: Declare4py: A python library for declarative process mining. In: BPM (PhD/Demos). CEUR Workshop Proceedings, vol. 3216, pp. 117–121. CEUR-WS.org (2022)
10. Dong, T., Shi, S.Y., Sampath, H., Macvean, A.: Towards AI as a collaborative partner: A taxonomy of AI agent behavior in software engineering. In: 3rd ACM Intl Conf on AI-powered Software (AIware 2026)
11. Goyal, N., Chang, M., Terry, M.: Designing for human-agent alignment: Understanding what humans want from their agents. In: CHI Extended Abstracts. pp. 106:1–106:6. ACM (2024)
12. Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N.V., Wiest, O., Zhang, X.: Large language model based multi-agents: A survey of progress and challenges. In: IJCAI. pp. 8048–8057. ijcai.org (2024)
13. He, J., Treude, C., Lo, D.: Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. ACM Trans. Softw. Eng. Methodol. **34**(5), 124:1–124:30 (2025)
14. Hemmer, P., Schemmer, M., Kühl, N., Vössing, M., Satzger, G.: Complementarity in human-ai collaboration: concept, sources, and evidence. Eur. J. Inf. Syst. **34**(6), 979–1002 (2025)
15. Kubrak, K., Botchorishvili, L., Milani, F., Nolte, A., Dumas, M.: Explanatory Capabilities of Large Language Models in Prescriptive Process Monitoring (Extended Abstract). In: IJCAI. pp. 10901–10905. ijcai.org (2025)
16. Li, H., Zhang, H., Hassan, A.E.: The Rise of AI Teammates in Software Engineering (SE) 3.0: How Autonomous Coding Agents Are Reshaping Software Engineering. CoRR **abs/2507.15003** (2025)
17. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: CAiSE (2012)
18. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: An approach based on colored automata. In: BPM (2011)
19. Otoum, N.A., El-Khalili, N.: Methods and techniques of agentic software engineering: A systematic literature review. IEEE Access **14**, 7443–7465 (2026)
20. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Eder, J., Dustdar, S. (eds.) Business Process Management Workshops. pp. 169–180. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
21. Roychoudhury, A., Pasareanu, C.S., Pradel, M., Ray, B.: Agentic AI software engineers: Programming with trust. Commun. ACM **69**(5), 56–58 (2026)
22. Sapkota, R., Roumeliotis, K.I., Karkee, M.: AI agents vs. agentic AI: A conceptual taxonomy, applications and challenges. CoRR **abs/2505.10468** (2025)
23. Seidel et al.: Object-centric process management: A research manifesto. Information Systems **141**, 102728 (2026)
24. Shen, Q., Polyvyanyy, A., Lipovetzky, N., Kampik, T.: Mining role-based behavioral patterns from event data for effective process simulation. In: CAiSE (2026)
25. Wooldridge, M.: An introduction to multiagent systems. John wiley & sons, second edn. (2009)