Annales Mathematicae et Informaticae
63 (2026) pp. 73–87
DOI: 10.33039/ami.2026.06.005
URL: https://ami.uni-eszterhazy.hu

Introducing TAIPO, an AI-based product owner assistant for vibe coding*†

Gábor Kusper[a], Judit Szabó[a], Márk Szabó[a], Ádám Kovács[a], Tibor Tajti[a], Csaba Szabó[b], Ján Perháč[b], Marek Horváth[b], Branislav Sobota[b]

[a]Eszterházy Károly Catholic University
{kusper.gabor,szabo.judit,szabo.mark,kovacs2.adam,tajti.tibor}@uni-eszterhazy.hu

[b]Technical University of Košice
{csaba.szabo,jan.perhac,marek.horvath,branislav.sobota}@tuke.sk

Abstract. Vibe coding promises rapid progress by letting developers rely heavily on generative AI during implementation. In collaborative and educational settings, however, this speed also creates pressure on requirement traceability, prioritization, and change management. We introduce TAIPO (The AI-based Product Owner), an AI-supported product owner assistant that embeds generative AI into a Kanban workflow instead of treating it as a separate chat tool. TAIPO stores AI-generated clarifications directly on cards, generates prioritized backlog items from requirement text, decomposes stories into implementable tasks, supports refinement and acceptance-related feedback, and can proactively simulate product-owner interventions through contextual comments and unexpected change requests. The prototype is implemented as a web-based system with a PHP backend, a Vue 3 frontend, and Gemini-based AI services. It is grounded in a cleaned version of the TAWOS issue-tracking dataset in order to support more realistic project dynamics. We position TAIPO as an environment for structured vibe coding rather than as a pure code generator, with a particular focus on software engineering education where one instructor may need to support multiple student teams simultaneously.

*This project, i.e., Project no. 2024-1.2.5-TÉT-2024-00072 has been implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed under the 2024-1.2.5-TÉT funding scheme.

†This work was also supported by the Slovak Research and Development Agency under the Contract no. SK-HU-24-0037.

Submitted: March 31, 2026
Accepted: June 30, 2026
Published online: July 17, 2026

Keywords: vibe coding, generative AI, product owner assistant, Kanban, software engineering education

2020 *Mathematics Subject Classification:* Primary 68T05, 68U35; Secondary 68M99

1. Introduction

Vibe coding is an emerging approach to programming that emphasizes creativity, flow, and collaboration with AI assistants. AI-assisted programming began to spread rapidly in 2022, as many developers started using Large Language Models (LLMs) to generate, explain, debug, and refactor code. However, the specific concept of *vibe coding* emerged later as a distinct way of describing a more fluid, AI-mediated style of software development.

The term "vibe coding" was introduced to broad public attention by Andrej Karpathy in an X post on February 2, 2025. After this post, the expression spread rapidly and became widely used to describe a style of development in which programmers rely heavily on generative AI to move quickly from idea to implementation while staying in a creative flow. Karpathy described his experience as follows [7]:

> There's a new kind of coding I call "vibe coding", where you fully give in to the vibes, embrace exponentials, and forget that the code even exists. It's possible because the LLMs (e.g. Cursor Composer w Sonnet) are getting too good. Also I just talk to Composer with SuperWhisper so I barely even touch the keyboard. I ask for the dumbest things like "decrease the padding on the sidebar by half" because I'm too lazy to find it. I "Accept All" always, I don't read the diffs anymore. When I get error messages I just copy paste them in with no comment, usually that fixes it. The code grows beyond my usual comprehension, I'd have to really read through it for a while. Sometimes the LLMs can't fix a bug so I just work around it or ask for random changes until it goes away. It's not too bad for throwaway weekend projects, but still quite amusing. I'm building a project or webapp, but it's not really coding - I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works.

So, vibe coding is a development style in which programmers rely heavily on generative AI in order to maintain creative flow and move quickly from idea to implementation. For exploratory solo work this can be highly productive.

In team settings, however, the same speed introduces a different problem: the faster ideas and code are produced, the easier it becomes to lose track of why a decision was made, which requirement it satisfies, and how later changes should be propagated across the project. This problem becomes even sharper in education, where students need room for experimentation but instructors still need visibility, traceability, and process discipline.

The risks are not only organizational. Empirical studies show that AI coding assistants may suggest insecure solutions [11] and that code models may reproduce memorized training fragments, potentially raising intellectual-property and confidentiality concerns [22]. These observations do not imply that generative AI should be avoided in software engineering. They indicate that AI-based programming support needs stronger process integration than ad hoc prompting in isolated chat windows.

In agile development, the product owner (PO) is the role that most directly mediates between stakeholders, requirements, prioritization, and the development team [6]. Kanban, in turn, offers a lightweight but disciplined coordination structure through visualized work items, explicit workflow states, and work-in-progress (WIP) limits [1]. Our starting point is that these two ideas can be combined with generative AI: instead of treating AI as a detached coding companion, it can be embedded into the very artifacts through which agile coordination happens.

This paper introduces **TAIPO** (**T**he **AI**-based **P**roduct **O**wner), pronounced like *"typo"*. TAIPO is an AI-supported PO assistant that operates directly on a Kanban board. Developers can ask prompt-based questions on concrete cards, and the answers are stored on those cards as persistent project knowledge. TAIPO can generate and prioritize user stories from requirement text, decompose stories into implementable tasks, refine descriptions, support acceptance-related feedback, and proactively simulate PO activity by adding contextual comments or injecting new requirements and change requests.

The present work builds on an earlier line of research on the artificial implementation of the Product Owner (PO) role in educational agile project simulation. In that earlier work, the focus was on a simplified educational prototype that generated Minimum Viable Product (MVP) requirements and backlog items for predefined project topics [18]. TAIPO extends this direction toward a Kanban board centric environment in which AI interactions are persistent, card-bound, and embedded into an explicit workflow structure.

The work is situated in a Slovak–Hungarian joint research project on realistic project simulation and intelligent PO support for software engineering education. A major resource behind the project is the TAWOS dataset, a relational corpus of 508,963 issues mined from 44 Jira-based open-source agile projects [20]. TAWOS does not only contain issue titles and descriptions, but also linked comments, users, releases, sprints, versions, and change histories. In our associated data-preparation work, the processed TAWOS issue corpus underwent cleaning and validity scoring to remove noisy or low-value records; the selected filtering configuration flagged 16.2% of issues for removal [19]. This cleaned corpus provides a stronger empirical basis for simulating realistic project dynamics.

The contribution of the paper is threefold. First, it defines the design rationale of a card-centered AI product owner assistant for structured vibe coding. Second, it presents a publicly accessible prototype implemented as a PHP backend and a Vue 3 frontend with Gemini-based AI services. Third, it shows how cleaned issue-tracking data can support proactive simulation of PO interventions in project-based

education. Although TAIPO can also generate code, our main claim is not that it replaces software developers or product owners. Rather, TAIPO demonstrates that vibe coding can be embedded into an auditable agile workflow in which AI outputs remain linked to requirements, tasks, and later changes.

The remainder of the paper first positions TAIPO in the literature, then presents its design concept, and finally describes the first prototype.

2. Related work

TAIPO sits at the intersection of several research threads, but the combination is unusual: most prior work addresses only one piece of the problem, such as the interaction style of vibe coding, the educational use of AI programming assistants, backlog generation from requirements, or the quality of issue-tracking data. The closest literature can therefore be read as a set of adjacent building blocks rather than as direct precedents for a Kanban board-centered AI product owner assistant.

The public starting point of vibe coding already captures both its attraction and its danger: one can "forget that the code even exists" while an AI tool takes over more of the concrete implementation work [7]. Later conceptual work formalized this as a shift in intent mediation, where natural-language dialogue and probabilistic inference take over part of the path from human intent to executable artifacts [10]. Empirical observation of extended vibe-coding sessions refined this picture by describing iterative cycles of prompting, rapid inspection, application testing, and occasional manual editing rather than a simple one-shot generation model [16].

Current research also makes clear that vibe coding should not be understood as fully autonomous software production. The central distinction is that human judgment remains responsible for goal setting, trust calibration, and final acceptance even when a large share of the concrete code is AI-generated [15]. Qualitative studies further show that co-creation, communication, and flow are experienced as real benefits, but they coexist with recurring difficulties around specification, incomplete solutions, large review effort, debugging, and trust [12]. This combination of attraction and fragility is directly addressed to TAIPO. We suggest that better project memory and workflow structure may be at least as important as better code generation.

The need for such structure is reinforced by adjacent evidence on AI-generated code quality. Security-oriented evaluation of AI code assistants has shown that generated suggestions can reproduce insecure patterns in realistic scenarios [11]. Separate work on code-model memorization has shown that generated outputs may contain memorized training fragments, which brings confidentiality, licensing, and provenance concerns into the development process [22]. These results do not argue against AI support, they argue against leaving AI use disconnected from auditable project artifacts.

A second research stream concerns education. Vibe coding has already been framed as an educational setting in which AI can act as more than a code-emitting

utility and may be experienced as a teammate-like collaborator inside project work [9]. Observational evidence from student sessions shows that much of the interaction effort shifts toward testing and debugging, while direct code reading becomes relatively rare; the same work also indicates that more advanced learners tend to supply richer contextual prompts [4]. For us this is important because a PO-oriented assistant should not only accelerate work, but also help keep intent explicit when students do not continuously read every generated line.

A direct predecessor of the present work can also be found in earlier research on the artificial implementation of the product owner role in educational agile project simulation [18]. That work introduced a simplified educational prototype in which generative AI was used to produce MVP requirements and backlog items for predefined project topics, while student progress was tracked through a mobile application interface. The main contribution of that earlier approach was to show that AI-supported PO simulation can reduce teacher workload in project-based software engineering education. TAIPO builds on this foundation, but moves beyond it in several ways: it shifts from a mobile requirement-delivery tool toward a web-based Kanban environment, it makes AI interaction persistent at card level, and it frames the whole system explicitly as support for structured vibe coding rather than only as a requirement-generation assistant.

The broader computer science education literature points to the same shift from low-level syntax production toward problem formulation and evaluation. Prompt Problems were proposed as a new exercise type in which students solve programming tasks by crafting sufficiently effective natural-language prompts for code-generating models [3]. Experience reports from introductory courses suggest that prompt-based activities may tap a partly different skill profile than traditional programming assessments and may soften some syntax-related barriers [8]. At the same time, benefits for novices come with risks of over-reliance, weakened understanding, and uneven learning effects [13]. This literature supports the view that code generation AI tools should support specification, review, and reflection, not only faster code production.

A third body of related work comes from agile coordination. The product-owner role has been analyzed as a central mechanism for linking stakeholder value, backlog decisions, and day-to-day development work, rather than as a purely administrative role [6]. Kanban research emphasizes visualization of work, explicit workflow stages, and work-in-progress control as practical devices for improving flow and reducing overload [1]. TAIPO inherits both ideas by locating AI support inside the Kanban board that already governs the project and by storing the results on cards instead of in detached conversations.

A fourth research direction moves generative AI upward from source code toward backlog and design artifacts. Requirement documents have already been used as inputs for automated user-story generation, showing that LLMs can populate project-management artifacts directly rather than only emit code [14]. From a model-driven perspective, vibe modeling argues that AI-supported development needs stronger support for higher-level abstractions if reliability and system com-

plexity are to remain manageable [2]. TAIPO fits this direction, but with a more operational emphasis: the central artifact is the Kanban card, which carries prioritization, clarification, decomposition, and change over time.

The final direction is repository mining and issue-quality research, which matters because TAIPO's simulation layer is grounded in issue-tracking data rather than in hand-written scripts. TAWOS was introduced as a versatile relational dataset spanning 44 Jira-based agile open-source projects and more than half a million issues, explicitly intended to support multiple downstream analyses such as effort estimation, prioritization, and assignment [20]. This makes it a natural basis for realistic PO-style simulation, but only if the data are clean enough for reuse.

The importance of issue quality has long been recognized. Classical bug-report research found a persistent mismatch between what developers consider most useful and what reporters actually provide, with reproduction steps, stack traces, and test cases repeatedly treated as high-value information [24]. Later work showed that different bug-report elements do not contribute equally to debugging effectiveness and that their significance can be studied empirically across repositories [17]. At a broader level, bug-report analysis has been surveyed as a large family of mining tasks rather than a single classification problem [23]. More focused work has examined automated discrimination between bug and non-bug issues [5], while dedicated metrics have also been proposed for assessing the quality of issue-tracking data itself [21].

This background directly supports the data-preparation step behind TAIPO. Our associated TAWOS cleaning work treated issue validity scoring as a prerequisite for later reuse and found that 16.2% of the processed issue corpus should be filtered out before simulation or educational use [19]. This is why TAIPO is grounded in cleaned repository traces instead of raw issue text.

Our survey suggests a gap. Existing vibe coding work explains the interaction style and its tensions. Educational work explains why prompt-mediated development needs scaffolding. Agile work explains why product-owner decisions and board discipline matter. Requirements-oriented AI work shows that backlog artifacts themselves can be generated, and repository-mining research explains why realistic issue corpora require quality control. TAIPO addresses this gap by combining these directions in a single Kanban-board-centered solution.

3. Design concept

The immediate target environment of TAIPO is project-based software engineering education. In such courses, a single instructor often needs to act as product owner (PO) for several student teams at the same time. The main bottleneck is not only answering questions, but doing so quickly enough and consistently enough that each team can continue working without long waiting periods. Purely manual PO support scales poorly, while generic AI chat tools fail to preserve decisions as stable project artifacts.

Therefore, TAIPO should be designed around four main requirements. First,

persistence: AI-generated clarifications should remain attached to the relevant work item rather than disappearing in a detached chat history. Second, *controllability*: the system should respect workflow states, priorities, and WIP limits, and it should remain robust even when human users edit cards or reorganize work manually. Third, *realism*: the assistant should not only react to direct prompts, but also introduce the kind of asynchronous feedback and requirement volatility that real projects exhibit. Fourth, *scalability*: the overall interaction model should reduce instructor workload without turning the course into a fully automated black box.

The TAWOS resource is central to the realism requirement. The original dataset provides large-scale traces of agile issue management, see Section 2. TAIPO should use TAWOS as inspiration for simulation patterns, comment styles, and change-request (CR) dynamics, thereby linking repository mining with educational process support.

These design goals position TAIPO somewhere between a project management tool and an AI teaching assistant. It should remain closer to the former in the sense that the Kanban board should serve as the primary source of truth. At the same time, it should behave like the latter by continuously supplying explanations, suggestions, and project perturbations, for example by occasionally introducing a new CR that keeps teams active even when a human PO is not immediately available.

TAIPO should treat Kanban cards as the operational context of AI interactions. Each card should store a user story or task together with its description, priority, workflow state, and associated discussion. When a user invokes TAIPO from a concrete card, the prompt should be interpreted in the context of that card and of the surrounding board. The answer should then be written back to the board, typically as a comment or as a structured modification of the card fields. In this way, requirement interpretation should become part of the project state itself.

The Kanban board should support multiple workflow columns with explicit WIP limits, with the exception of the Backlog and Done columns. The Backlog should function as the source of user stories, whereas Done should serve as the sink for completed tasks.

Cards should be movable through drag-and-drop interactions so that users can reorganize work naturally while the system still preserves process constraints. These workflow restrictions should not suppress creativity; rather, they should provide a lightweight process envelope within which AI-supported backlog refinement and clarification can remain traceable and manageable.

TAIPO should include backlog population as a capability compatible with recent LLM-assisted user-story generation approaches, see Section 2, but it should treat generation as only the first step of a longer workflow. Because cards should persist as editable project artifacts, external human edits should not break the AI context, instead, they should become part of the next reasoning step.

TAIPO should support two complementary PO interaction modes: (1) a reactive mode and (2) a proactive mode.

The reactive mode should be initiated by users. In this mode, the assistant

should transform requirement text into prioritized user stories, refine existing cards, decompose user stories into tasks, and answer clarification questions attached to a concrete card.

The proactive mode should simulate the asynchronous behavior of a human PO. Its intervention frequencies should be configurable. For example, TAIPO should be able to add a contextual comment to a card every few hours during working days (between 8AM and 4PM). It should also be able to prompt users to update project progress when the board state suggests that the recorded status may no longer reflect the actual state of work. Less frequently, it should be able to introduce a new requirement, user story, or CR.

When a team reaches a milestone, or at the end of a sprint, which should be represented as an event generated by a configurable time-based service, TAIPO should be able to (1) accept or reject completed tasks, (2) provide *adaptive coaching*, i.e., explain the decisions of the PO, and (3) extend the requirement set with more specific follow-up expectations, thereby making the simulated project evolve in a less static and more realistic way.

TAIPO should also be able to ask the user proactively, through *notifications prompting*, whether the connected GitHub repository and the Kanban board are significantly out of sync.

The content of these interventions should be generated from the cleaned TAWOS-derived knowledge base together with the current board context. In classroom settings, this mechanism should help recreate the fact that real requirements rarely remain unchanged throughout a project.

The first prototype described in the next section implements a substantial subset of these design ideas.

4. The first prototype of TAIPO

The first prototype of TAIPO implements the core board-centered interaction model. It operationalizes the main ideas of the design concept. It should be understood as a first implementation, not as a fully functional release.

The prototype has been implemented as a web-based system that combines a PHP backend API with a Vue 3 single-page frontend. Board state is stored in a relational database. By default, the prototype uses SQLite, but the backend is configurable to work with other PDO-supported database systems as well. The frontend provides responsive drag-and-drop interaction, priority marking, and card editing, while the backend is responsible for persistence, workflow enforcement, configuration, and communication with external AI services. The current prototype uses the Gemini API for AI-supported content generation. The source code of the system is publicly available at `https://github.com/szabojuci/AIKanban.git`.

The interface is organized around a five-stage Kanban workflow that reflects the basic project lifecycle: Sprint Backlog, Implementation, Testing, Review, and Done. The in-progress columns use explicit WIP limits, so the board does not merely display tasks but also constrains parallel work. This is important for our

interpretation of TAIPO: the system is not intended to replace agile discipline with unrestricted AI automation, but to place AI-supported decisions inside a controlled process.

Figure 1 illustrates the current interface. The interface consists of five workflow columns, namely Sprint Backlog, Implementation, Testing, Review, and Done, with visible WIP counters where relevant. Cards contain titles, priorities, and action buttons. Opening a card exposes the "Ask TAIPO" dialog, where the user can enter a free prompt or select built-in Quick Prompts such as explanation, implementation advice, test-case generation, security-oriented checking, and code-snippet support. The generated response is stored with the selected card, so the AI interaction remains part of the board history.

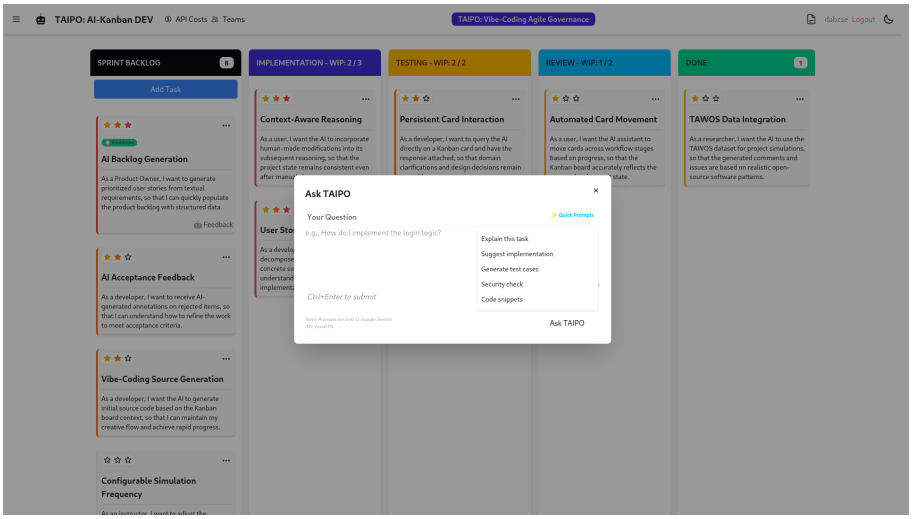

Figure 1. The TAIPO prototype with the "Ask TAIPO" dialog and built-in Quick Prompts.

The core interaction principle is that AI support is attached to concrete cards. When a user opens the "Ask TAIPO" dialog from a selected card, the prompt is interpreted in the context of that card and of the surrounding board state. To reduce prompting overhead in typical product-owner situations, the prototype also offers a small set of built-in Quick Prompts, including task explanation, implementation suggestions, test-case generation, security-oriented checking, and code-snippet support. The resulting answers are stored as persistent card-related artifacts rather than disappearing in an isolated chat session. If a card already contains earlier TAIPO feedback, the dialog can notify the user and offer regeneration of the answer. In this way, requirement clarification, design rationale, and implementation guidance remain inspectable later by both students and instructors.

The present prototype already supports the main interactions needed for a PO-oriented vibe-coding workflow. It can populate the backlog from textual re-

quirement descriptions, generate prioritized user stories, refine and reprioritize existing cards, and decompose a selected story into more concrete tasks. It can also support acceptance-related feedback by annotating non-accepted items and returning them to the backlog for further refinement. Optional code assistance is also available. This is auxiliary to the main idea: TAIPO is primarily a structured project-governance environment for AI-assisted development, not a stand-alone code generator. Repository integration is not yet implemented.

Table 1 summarizes the main capabilities currently available in the prototype.

Table 1. Main capabilities of the TAIPO prototype.

Capability	**Board-level effect**
Backlog population	Generates prioritized user stories from textual requirements and inserts them into the backlog.
Story decomposition	Breaks a selected story into concrete, implementable tasks.
Card-level clarification	Answers prompt-based questions and stores the resulting explanation on the relevant card.
Refinement and prioritization	Updates descriptions, priorities, and other card metadata to support backlog grooming.
PO simulation	Adds contextual comments and injects new requirements, user stories, or change requests at configurable intervals.
Acceptance handling	Supports returning rejected items to the backlog; richer AI-generated rejection feedback is ongoing work.
Optional code assistance	Can generate implementation snippets, but there is no repository integration.

Beyond direct user interaction, the prototype also supports proactive PO simulation. The intervention schedule is configurable. In the current default setup, TAIPO adds a contextual comment to a card every two hours during working days between 8AM and 4PM, based on a weekday-sensitive time-based mechanism. At present, this mechanism does not include calendar integration or public-holiday awareness. Less frequently, on average once every three days, it introduces a new requirement, a user story, or a change request. These interventions are generated from the current board context together with patterns derived from the cleaned TAWOS-based knowledge base. In educational settings, this makes the board behave less like a static assignment list and more like a small but evolving agile project.

To demonstrate the prototype, we use a weather-forecast web application containing a zoomable map view and a time-slider component, followed later by unforeseen requirement changes. This scenario is deliberately compact, yet rich enough to exercise the central functions of TAIPO: backlog creation from a short description, story decomposition, card-level clarification, reprioritization, and reaction to injected changes.

The example also makes the main design decision of TAIPO visible: project

memory is stored on the board itself. Clarifications about map behavior or time handling remain attached to the relevant cards, and later changes appear as new board artifacts instead of getting lost in side conversations.

A public demonstration instance is available at `https://dabzse.net/TAIPO/`. At the time of writing, it can be accessed with the demo credentials `testuser` / `password123`. This makes the prototype directly inspectable for reviewers and readers. Together with the public repository, the demo supports the practical reproducibility of the system and shows that TAIPO already exists as an operational prototype rather than as a purely conceptual proposal.

4.1. Exploratory classroom scenario

In addition to the weather-forecast example, we conducted a small exploratory classroom walkthrough using a compact *Mini-University* scenario. The log of the walkthrough can be accessed at: `https://docs.google.com/document/d/1d7S1YnEKmRrVrZbVZUDvb8k5ZMLROAdl/`.

The product brief described a single-file Streamlit application with JSON-based persistence and three roles: administrator, teacher, and student. The required functionality included role-based login, user and course administration, teacher-side grade entry, student-side course enrolment and drop, transcript and GPA views, and persistence after each data-changing operation. The purpose of the scenario was not to evaluate the Mini-University application itself, but to observe whether an initially unstructured AI-development brief could be transformed into a persistent, card-centered TAIPO workflow.

The walkthrough deliberately avoided a one-shot prompting style. During Sprint 0, the product brief was given to TAIPO as a requirement seed and decomposed into twelve backlog cards, including project skeleton, JSON persistence, authentication, administrator-side user management, course management, teacher-side course roster and grade entry, student-side enrolment, transcript calculation, role-based access-control tests, and later change-request handling. Four short mini sprints were then used to implement and review the cards: first the application skeleton, persistence, and login; then administrator functions and security constraints; then teacher and student workflows; and finally change management, review, and hardening.

At card level, TAIPO was used before implementation to generate acceptance criteria in Given/When/Then form, implementation notes, and smoke-test checklists. This made several rule-like requirements explicit before coding started. For example, administrators must not delete their own account or the last remaining administrator; teachers may only modify grades in their own courses; students may only enrol in or drop courses; and deletion operations must not leave inconsistent enrolment or grade records behind. These constraints are typical examples of product-owner knowledge that can easily disappear in a detached chat transcript, but can remain inspectable when attached to Kanban cards.

The most useful part of the walkthrough was the change-management step. In the last mini sprint, the team introduced a small change request: students should

not be able to enrol in full courses, and the user interface should explicitly mark such courses as full. TAIPO identified the affected earlier card, updated the relevant acceptance criteria, and proposed regression tests for the modified enrolment workflow. The resulting decision history remained attached to the change-request card and the affected feature card, illustrating how TAIPO can support traceability when requirements evolve after implementation has already started.

The walkthrough log recorded 28 card-level TAIPO interactions, 36 acceptance criteria, and 22 test items, including nine role-based negative tests. One change request was handled, and two defects or missing edge cases were identified during the process: self-deletion of an administrator account and stale course-related data after deletion operations. The final implementation was checked against the board history, and the reported features were linked back to TAIPO cards. These numbers should be interpreted as preliminary process observations, not as statistically valid evidence of productivity or learning gains.

This walkthrough is therefore not a controlled classroom evaluation. It does not measure learning outcomes, usability, developer productivity, or code quality against a baseline tool. Its value is narrower but still relevant: it shows that TAIPO can turn a long natural-language development brief into smaller work items, preserve acceptance criteria and tests as card-level artifacts, and make later changes auditable. A controlled study with student teams remains future work.

The proactive simulation layer of TAIPO should be interpreted in the same cautious way. It is derived from the cleaned TAWOS-based knowledge base in a pattern-based rather than replay-based manner. After filtering low-value issues, we use issue metadata and text traces, such as issue type, priority, comment density, status changes, and change-request-like comments, to derive intervention categories and plausible timing patterns. At runtime, the scheduler selects an intervention type, for example clarification, progress reminder, acceptance feedback, or a new change request, and conditions the generated text on the current board state. Thus, the realism claim means empirically inspired PO-like behavior grounded in issue-tracking traces, not a validated statistical reproduction of a specific Jira project.

Educational deployments also require explicit operational safeguards. The Mini-University walkthrough used synthetic project data, and the same principle should be followed in classroom use: students should not submit real personal data, secrets, private repository content, assessment-sensitive material, or confidential stakeholder information to an external model provider. TAIPO's code-oriented prompts are auxiliary; generated snippets should be treated as suggestions that require human review, testing, and security inspection before acceptance. The repository should not contain API keys, and deployments should provide keys through local configuration or environment variables. Instructors can also disable code-oriented prompts or restrict tasks to synthetic project descriptions when confidentiality or intellectual-property concerns are important.

Finally, the current prototype uses the Gemini API, which introduces practical dependencies such as cost, availability, data-transfer policies, and rate limits. This dependency is not central to the TAIPO concept: the board model, persistence

layer, and prompt construction can be separated from the concrete model backend. For classroom use, deployments should therefore include quota-aware settings, caching of repeated responses where appropriate, and a fallback mode in which proactive comments can be disabled or delayed. Future versions should make the model interface explicitly pluggable so that cloud models, institutional endpoints, or local models can be compared under the same board-centered workflow.

5. Conclusion and future work

Vibe coding is attractive because it accelerates exploration, but without process support it can weaken traceability and requirement control. TAIPO addresses this problem by embedding an AI-based product owner assistant directly into a Kanban workflow.

Instead of placing the model outside the process and leaving teams to negotiate how to remember or trust its output, TAIPO makes the Kanban board the shared memory of human and AI activity. This shift has several important consequences: (1) It improves traceability because AI-generated answers remain linked to cards. (2) It improves auditability because instructors or team leads can review not only code, but also the requirement interpretations and decisions that led to it. (3) It also improves resilience to change, since later prompts can reuse the updated board state rather than depend on vague chat context.

The educational value of this approach is particularly important. In many project-based software engineering courses, the scarcest resource is timely PO attention. TAIPO cannot replace a human instructor, but it can absorb a substantial part of repetitive clarification and backlog-maintenance work, while also making projects more realistic through controlled requirement changes. Because the assistant is grounded in cleaned issue-tracking data, its proactive behavior is closer to plausible project dynamics than a purely hand-written script would be.

At the same time, the current work has clear limitations. The Mini-University walkthrough is only a small exploratory scenario, not a controlled classroom study. We do not yet report quantitative learning outcomes, developer-productivity measurements, code-quality comparisons, or a formal comparison against alternative AI-assisted project-management tools. The present prototype relies on an external generative-model API, and some functions, especially acceptance-related rejection feedback and repository synchronization, are still evolving.

The next step is therefore to move beyond the first prototype and evaluate it in real educational settings at both the Technical University of Košice and Eszterházy Károly Catholic University. Based on these practical experiences, we plan to refine both the underlying concept and the implementation of TAIPO.

Since the source code is openly available, anyone can also try the system and experiment with it. Naturally, the repository does not include an API key, so users must provide their own. The authors warmly welcome feedback from all users, as such experience can directly support the further development of both the prototype and the broader idea of structured vibe coding.

References

[1] M. O. Ahmad, J. Markkula, M. Oivo: *Kanban in Software Development: A Systematic Literature Review*, in: Proceedings of the 2013 39th Euromicro Conference on Software Engineering and Advanced Applications, 2013, pp. 9–16, DOI: 10.1109/SEAA.2013.28.

[2] J. Cabot: *Vibe Modeling: Challenges and Opportunities*, in: Advances in Conceptual Modeling, 2025, pp. 105–118, DOI: 10.1007/978-3-032-08620-4_7.

[3] P. Denny, J. Leinonen, J. Prather, A. Luxton-Reilly, T. Amarouche, B. A. Becker, B. N. Reeves: *Prompt Problems: A New Programming Exercise for the Generative AI Era*, in: Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE 2024), 2024, pp. 296–302, DOI: 10.1145/3626252.3630909.

[4] F. Geng, A. Shah, H. Li, N. Mulla, S. Swanson, G. Soosai Raj, D. Zingaro, L. Porter: *Exploring Student-AI Interactions in Vibe Coding*, in: Proceedings of the 28th Australasian Computing Education Conference, 2026, pp. 45–54, DOI: 10.1145/3786228.3786236.

[5] S. Herbold, A. Trautsch, F. Trautsch: *On the Feasibility of Automated Prediction of Bug and Non-Bug Issues*, Empirical Software Engineering 25.6 (2020), pp. 5333–5369, DOI: 10.1007/s10664-020-09885-w.

[6] M. D. Kadenic, D. A. de Jesus Pacheco, K. Koumaditis, G. Tjørnehøj, T. Tambo: *Investigating the role of Product Owner in Scrum teams: Differentiation between organisational and individual impacts and opportunities*, Journal of Systems and Software 206 (2023), pp. 1–17, DOI: 10.1016/j.jss.2023.111841.

[7] A. Karpathy: *Vibe Coding*, X (formerly Twitter) post, Feb. 2025, URL: https://x.com/karpathy/status/1886192184808149383 (visited on 01/14/2026).

[8] C. Kerslake, P. Denny, D. H. Smith IV, J. Prather, J. Leinonen, A. Luxton-Reilly, S. MacNeil: *Integrating Natural Language Prompting Tasks in Introductory Programming Courses*, in: Proceedings of the 2024 ACM Virtual Global Computing Education Conference, 2024, pp. 88–94, DOI: 10.1145/3649165.3690125.

[9] G. Kusper, C. Szabó: *Vibe Coding in Education*, in: Proceedings of the 2025 International Conference on Emerging eLearning Technologies and Applications (ICETA), 2025, pp. 506–511, DOI: 10.1109/ICETA67772.2025.11280285.

[10] C. Meske, T. Hermanns, E. von der Weiden, K.-U. Loser, T. Berger: *Vibe Coding as a Reconfiguration of Intent Mediation in Software Development: Definition, Implications, and Research Agenda*, IEEE Access 13 (2025), pp. 213242–213259, DOI: 10.1109/ACCESS.2025.3645466.

[11] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, R. Karri: *Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions*, in: 2022 IEEE Symposium on Security and Privacy (SP), 2022, pp. 754–768, DOI: 10.1109/SP46214.2022.9833571.

[12] V. Pimenova, S. Fakhoury, C. Bird, M.-A. Storey, M. Endres: *Good Vibrations? A Qualitative Study of Co-Creation, Communication, Flow, and Trust in Vibe Coding*, 2025, DOI: 10.48550/arXiv.2509.12491, arXiv: 2509.12491.

[13] J. Prather, B. N. Reeves, J. Leinonen, S. MacNeil, A. S. Randrianasolo, B. A. Becker, B. Kimmel, J. Wright, B. Briggs: *The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers*, in: Proceedings of the ACM Conference on International Computing Education Research (ICER '24 Vol. 1), 2024, DOI: 10.1145/3632620.3671116.

[14] T. Rahman, Y. Zhu: *Automated User Story Generation with Test Case Specification Using Large Language Model*, 2024, DOI: 10.48550/arXiv.2404.01558, arXiv: 2404.01558 [cs.SE].

[15] R. Sapkota, K. I. Roumeliotis, M. Karkee: *Vibe Coding vs. Agentic Coding: Fundamentals and Practical Implications of Agentic AI*, 2025, DOI: 10.48550/arXiv.2505.19443, arXiv: 2505.19443.

[16] A. Sarkar, I. Drosos: *Vibe Coding: Programming through Conversation with Artificial Intelligence*, in: Proceedings of the 36th Annual Conference of the Psychology of Programming Interest Group (PPIG 2025), 2025, pp. 87–118.

[17] M. Soltani, F. Hermans, T. Bäck: *The Significance of Bug Report Elements*, Empirical Software Engineering 25.6 (2020), pp. 5255–5294, doi: 10.1007/s10664-020-09882-z.

[18] C. Szabó, K. Lengyel: *Artificial Intelligence Based Implementation of the Product Owner Role in Educational Agile Project Simulation*, in: ICERI2025 Proceedings, 18th annual International Conference of Education, Research and Innovation, Seville, Spain: IATED, 2025, pp. 5910–5915, isbn: 978-84-09-78706-7, doi: 10.21125/iceri.2025.1622.

[19] M. Szabó, Á. Kovács, G. Kusper: *Issue Validity Scoring for Cleaning the TAWOS Database: OwnMetrics versus Local Small Language Models*, Manuscript for the Proceedings of the 13th International Conference on Applied Informatics (ICAI 2026), 2026.

[20] V. Tawosi, A. Al-Subaihin, R. Moussa, F. Sarro: *A Versatile Dataset of Agile Open Source Software Projects*, in: Proceedings of the 19th International Conference on Mining Software Repositories (MSR '22), 2022, pp. 707–711, doi: 10.1145/3524842.3528029.

[21] F. Tu, F. Zhang: *Measuring the Quality of Issue Tracking Data*, in: Proceedings of the 6th Asia-Pacific Symposium on Internetware, 2014, pp. 76–79, doi: 10.1145/2677832.2677844.

[22] Z. Yang, Z. Zhao, C. Wang, J. Shi, D. Kim, D. Han, D. Lo: *Unveiling Memorization in Code Models*, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, 2024, pp. 1–13, doi: 10.1145/3597503.3639074.

[23] J. Zhang, X. Wang, D. Hao, B. Xie, L. Zhang, H. Mei: *A Survey on Bug-Report Analysis*, Science China Information Sciences 58.2 (2015), pp. 1–24, doi: 10.1007/s11432-014-5241-2.

[24] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, C. Weiss: *What Makes a Good Bug Report?*, IEEE Transactions on Software Engineering 36.5 (2010), pp. 618–643, doi: 10.1109/TSE.2010.63.