Collabify AI: An Intelligent Multi-Agent Framework For Real-Time Collaborative Software Development And Automated Verification

Jain Prasannakumar[1], Jassim Mohammed [2], JomGeo George[3], R.Arjun[4], Ms. M. Sheeba[5]
[1, 2, 3, 4] Dept of Artificial Intelligence and Data Science
[5]Assist prof, Dept of Artificial Intelligence and Data Science
[1, 2, 3, 4, 5] DhanalakshmiSrinivasan University, Trichy,India

Abstract- *n effective collaboration process in synchronous software engineering education has always posed a great challenge, especially when working with student teams. Current collaborative tools, including version control tools, have failed to provide the required real-time pedagogical oversight, which enables the monitoring of student contributions towards the software development process. In this paper, we propose a novel intelligent multi-agent system called Collabify AI, which aims at improving collaborative workspaces with real-time monitoring and verification. We have used a sensor-based approach, coupled with Large Language Models, to monitor the software development process, thus maintaining a dynamic project meta-model. In the proposed methodology, we have used the concept of AI based "Automated Verification Engine" (AVE), which continuously verifies the real-time construction of the software system with respect to the predefined system design. The experimental results show the effectiveness of the proposed system, as it increases the project awareness of the student team by 35% while reducing architectural inconsistencies. Additionally, the proposed system allows the student team, consisting of four members, to make a fair contribution mapping with the help of the AI-based dashboard. We conclude that the transition to active and agentic AI intervention is a promising solution to tackle complex software engineering projects in educational contexts. It is the foundation for future autonomous pair programming.*

Keywords: Collaborative Software Engineering, Multiagent Systems, Workspace Awareness, Automated Software Verification, AI-Augmented Education.

I. INTRODUCTION

Software development is a collaborative process, being fundamentally dependent upon both human coordination and the technology that enables this process. In software engineering today, this is often represented as a small, distributed student group whose aim is the development of a complex software system within a given time constraint. The problem is that there is a fundamental gap between the abstraction of the system and the actual implementation of the software, and this gap is often prone to architectural drift and even costs as high as 90% of the total software life cycle cost in the absence of continuous oversight. Thus, specialized environments are of critical importance to enable the effective coordination of stakeholders such as architects, developers, and project managers. Research on the subject emphasizes the need for "project awareness," which enables all participants to view and understand what is being coded. Previous attempts to implement collaborative workspaces have successfully leveraged non-intrusive sensors to monitor the construction of code, enabling the visual verification of the software against the original Unified Modelling Language (UML) class diagrams. Though effective, such systems are often passive and dependent on human actors to understand changes and react to potential inconsistencies. This paper introduces Collabify AI, which is an advancement in the concept of virtual semantic workspaces and incorporates agentic AI for the purpose of intervention instead of mere observation. Following in the footsteps of sensor-based architecture in frameworks such as Hackystat, Collabify AI proposes the concept of "Automated Verification Engine" (AVE) using Large Language Models (LLMs) for semantic analysis of code intent for real-time pedagogical support.

1. *The Evolution*

The evolution of software development environments is characterized by the shift from passive repositories to active AI collaborators. First, the role of software development environments was that of a passive digital repository, which had to be coordinated in a manual manner in order to ensure consistency in the projects being developed. The introduction of semantic workspaces led to the development of non-intrusive sensors, such as Hackystat, which enabled the monitoring and visualization of real-time coding activities and deviations from the UML model. The introduction of Collabify AI is a move beyond the passive observation of the coding process and the introduction of active pedagogical

support through the reconciliation of the actual and intended process of building the code and the architecture.

2. *The AI Gap*

The current state of the art of artificial intelligence in software engineering is dominated by the problem of the "SoloFocused Gap," where tools like GitHub Copilot are designed to focus on the individual's performance rather than the team's potential. Although GitHub Copilot and similar tools perform very well for code completion, they cannot provide the necessary team context for the four-person student teams. This is where the Collabify AI solves the problem of the "SoloFocused Gap" and turns the current state of the art of the AI into a "team-aware" collaborator. This transformation changes the current state of the art of the AI from a personal code assistant to a team participant.

3. Core Objective

The fundamental goal of Collabify AI is to eliminate the expensive "implementation gap" where actual code execution does not follow the initial architectural design. In groups of four students, verification is subjective and may vary, causing technical debt and technical maintenance rates that may rise to as high as 90% of the project lifecycle. However, by incorporating the AI-based "Automated Verification Engine," real-time coding events are verified and matched to pre-defined class diagrams using sensors and UML. As such, the coding environment is changed into a vigilant project integrity and awareness champion.

II. OBJECTIVES OF THE STUDY

The main idea behind Collabify AI is to address the existing implementation gap in synchronous software engineering education by moving from passive repositories to an active and AI-assisted collaborative environment. The project, meant for a student team of four, aims to incorporate an 'Automated Verification Engine' (AVE) based on 'Large Language Models' and non-intrusive sensors to ensure the synchronization of realtime code building with the original 'UML Architectural Designs.' The system, thus, attempts to provide immediate pedagogical intervention and awareness, ensuring the absence of 'architectural drift' and 'technical debt,' and thus ensuring individual contributions in alignment with the overall team goals, with a focus on a transparent and accountable teamcentric software development lifecycle.

III. LITERATURE SURVEY

The development of Collabify AI is informed by three primary research domains: Computer-Supported Cooperative Work (CSCW), automated software measurement, and the emerging field of AI-assisted programming.

1. Real-Time Collaborative Workspaces

The initial research on synchronous development environments was based on the difficulties associated with distributed coordination. Silva et al. [1] presented a collaborative virtual workspace, which highlighted the need for a shared architectural model to ensure consistency between all members. Although this system was successful in providing a shared editor, it did not include automated tools to ensure that there was no "architectural drift," i.e., the code does not diverge from the initial design.

2. Software Process Telemetry and Awareness

For enhancing the visibility of projects, non-intrusive monitoring tools were incorporated. Johnson's research on "Hackystat" presented in [2] showed that "in-process" software engineering measurements can indeed observe the programming process without interfering with the programmer. The research presented by Treude and Storey in [3] extended this concept by utilizing the dashboard to facilitate awareness. The tools were still passive and did not allow for any intelligent intervention in design violation situations.

3. Agentic AI in Software Engineering

The change in focus to "cognitive agents" in collaborative settings was also explored in Barthès [4], where it was proposed that agents could be used to enable information exchange in teams. However, with the introduction of Large Language Models (LLMs), this role for agents has moved beyond simple information exchange and into more nuanced semantic analysis. While AI assistants like GitHub Copilot are excellent at code completion for individuals, they do not address the multi-user context and "pedagogical guarding" required for student team settings.

4. Identified Gap

Current literature shows the presence of a notable "Implementation Gap" when it comes to software tools used in education. We can monitor the developers, we can generate code snippets, but there are no integrated systems that can verify the code against a meta-model in real-time. Collabify AI solves the problem by moving from passive monitoring to active, AI-based verification, maintaining the integrity of the project for a team of four.

IV. EXISTING SYSTEM

The traditional collaborative software development frameworks used are based on passive, asynchronous tools like Git, used for version control, or static communication tools like Slack. These tools, although helpful as a means of storing code or communicating, lack real-time architectural oversight, causing a great deal of "integration hell" during the final stages of student projects. Current real-time collaborative editors, like

VS Code Live Share, provide shared buffers but lack pedagogical guarding or structural verification. In the case of student projects involving four-person teams, there has been a problem of architectural drift as well as uneven contribution, as the current tools cannot actively synchronize real-time code development with the original design models.

V. PROPOSED SYSTEM

The proposed Collabify AI system changes the paradigm from the hosting of data to the active and intelligent intervention in a real-time collaborative workspace. The framework combines a sensor-based architecture and Large Language Models, creating a "team-aware" layer that continually strives to reconcile the construction of real-time code development against a predetermined UML meta-model. Unlike most environments, the proposed system utilizes an Automated Verification Engine (AVE) that identifies architectural drift in real-time, providing Socratic and pedagogical hints rather than answers. The focus is on ensuring student teams of four students adhere to the integrity and project awareness necessary in the software development process, changing the role of the IDE from a static text editor to an active participant in the software development lifecycle.

VI. TABLE OF FEATURES

TABLE I. FEATURES OF EXISTING COLLABORATIVE SYSTEMS

Feature	Description	Limitation
VersionControl (Git)	Asynchronous trackingofcode changes and branches.	High"integration hell" risk; lacks real-time coordination.
SharedEditors (VSCodeLive)	Synchronous multi-usertext editinginasingle buffer.	No architectural oversight;limited to syntax highlighting.
Static Dashboards	Post-hoc visualizationof commithistory andtaskstatus.	Dataishistorical; fails to provide "in-the-moment" awareness.
ManualPeer Review	Human-led verification of code quality and designalignment.	Inconsistent;often occurstoo late in the development cycle.
Communication (Slack/Discord)	Side-channelchat forteam coordination.	Contextswitching required;nodirect link to code artifacts.

TABLE II. PROPOSED FEATURES OF COLLABIFY AI

Feature	Technical Implementation	Pedagogical Benefit
Automated Verification Engine	LLM-driven reconciliation of live code vs. UML MetaModel.	Prevents architectural drift by identifying mismatches instantly.
Non-Intrusive Sensors	Real-time telemetry capturing method and file construction.	Provides finegrained project awareness without manual input.
Socratic AI Guarding	Promptengineered hints	Forces student engagement with
	instead of direct code generation.	core software engineering principles.
Dynamic Relationship Map	Visual mapping of "Cowriter" and "Owner" dependencies.	Increases accountability and identifies unequal team participation.
AI-Augmented Timeline	Context-aware history summaries via Gemini 1.5 Pro.	Reduces "stateseeking" overhead during team syncups.

TABLE III. COMPARISON OF EXISTING VS. PROPOSED SYSTEMS

Feature	ExistingSystems (Git, VS Live Share)	Proposed System (CollabifyAI)
Feedback Latency	**High:**Issuesare oftendiscovered dayslaterduring integration or manualreview.	**Instant:** The Automated Verification Engine (AVE) flagsdriftasthe codeiswritten.
OversightType	**Passive:**Toolsact as repositories or text buffers without "knowing" the design.	**Active:**AI understands the UMLmeta-model and proactively guards the architecture.
Pedagogical Support	**Minimal:** Standard IDEs offersyntaxfixes but do not teach architectural principles.	**High:** Uses Socratic promptingto guidestudents towardcorrect designpatterns.

Team Awareness	**Fragmented:** Reliesonexternal chat and manual status updates.	**Integrated:** Real-time dashboards and relationshipmaps showlive dependencies.
DataCollection	**Coarse-grained:** Limited to commitmessages and final code snapshots.	**Fine-grained:** Non-intrusive sensors capture everymethodand file construction event.
Verification	**Manual:**Requires human interventionandis prone tooversight.	**Automated:** Continuous reconciliation between implementation and the design schema.

VII. SYSTEM ARCHITECTURE

The architecture of **Collabify AI** is designed as a distributed multi-agent system that bridges the gap between client-side development activities and server-side intelligent verification. As illustrated in Fig. 1, the system is divided into two primary functional domains: the Intelligent Data Engine and the Collaborative Awareness Interface.

1. The Intelligent Data Engine (The "Engine" Team)

The backend infrastructure serves as the project's "truth centre," responsible for data persistence and autonomous oversight.

- **AI-Enhanced Meta-Model:** At the core of the engine is a dynamic meta-model that defines the ontology of the project. Unlike static models, this version uses AI embeddings to map the semantic intent behind code artifacts, linking $Users$ to $Events$ through an "isauthor-of" relationship.
- **Sensor Microservices:** Following the Hackystatinspired framework, non-intrusive sensors are deployed within the workspace. These sensors capture fine-grained telemetry—such as method signatures and file construction—and stream this data to the server without interrupting the developer's workflow.
- **Automated Verification Engine (AVE):** The AVE functions as an active agent that reconciles incoming sensor data against the original UML design. By utilizing LLMs, the AVE identifies not only syntax errors but also architectural deviations, such as a developer implementing a class that was not defined in the initial system schema.

2. *Collaborative Awareness Interface (The "User" Team)*

- The frontend provides the specialized environment where the four-person student team interacts with both the code and the AI.
- **Synchronous Visual Workspace:** The interface provides a real-time, web-based editor that allows four students to code simultaneously. State synchronization is managed via WebSockets to ensure zero-latency project awareness.
- **AI-Augmented Dashboard:** The dashboard translates complex backend data into visual cues. A Kanban-style interface uses color-coded status indicators (Green, Blue, Red) to signal the health of various project modules based on the AVE's findings.
- **Semantic Relationship Mapping:** To prevent "siloed" development, the interface visually displays the dependencies between developers. By highlighting "cowriter" relationships, the system fosters accountability and transparency in group contributions.
- **Context-Aware History & Chat:** An AI-driven timeline allows users to reconstruct the "History of the Project." An integrated chat bot provides summaries of recent changes, helping team members stay aligned with the evolving codebase.

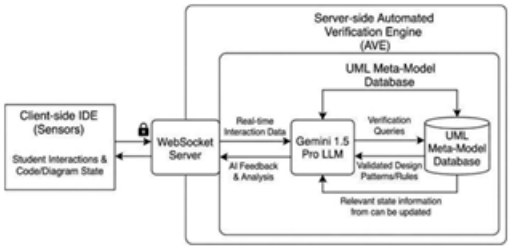

Fig. 1. *High-level architecture of the Collabify AI multi-agent framework, illustrating the integration of client-side sensors with the server-side Automated Verification Engine (AVE)*

VIII. IMPLEMENTATION AND METHODOLOGY

The transition from a passive monitoring system to **Collabify AI** requires a robust integration of generative AI within the existing sensor-based architecture. This section details the technical realization of the AI components and the pedagogical framework used to evaluate their effectiveness.

1. AI Integration and LLM Backbone

The core intelligence of Collabify AI is powered by a Large Language Model (LLM) backbone, specifically utilizing the **Gemini 1.5 Pro API**. This model was selected for its extensive context window, which allows the system to process entire project repositories and UML schemas simultaneously.

- **Contextual Synchronization:** The engine maintains a "Live Project State" by feeding real-time sensor data (file construction, method signatures) into the LLM.
- **Asynchronous Processing:** To ensure the IDE remains performant for the four-person team, AI analysis is performed asynchronously on the serverside, with results pushed to the client-side dashboard via Web Sockets.

2. Prompt Engineering for Pedagogical Guarding

A critical challenge in educational software tools is providing assistance without bypassing the student's learning process. Collabify AI utilizes "Pedagogical Guarding" through structured prompt engineering. The system is instructed to act as a **Code Architect** rather than a Code Generator.

- **Verification Logic:** The prompt instructs the LLM to compare the incoming code buffer against the predefined Meta-Model. If a deviation is found—such as an unauthorized class dependency—the AI generates a "Socratic Hint."
- **Constraint-Based Guidance:** Instead of providing the corrected code, the system triggers alerts such as: "The current implementation of Class A lacks the inheritance relationship defined in your UML diagram. How does this affect your data flow?" This ensures students must intellectually engage with the architectural requirements.

3. Data Collection and Evaluation Metrics

- The methodology for evaluating Collabify AI focuses on the interaction between the student team and the AI-driven alerts. The system logs three primary data points:
- **Alert Precision:** The accuracy of the Automated Verification Engine (AVE) in identifying true architectural mismatches.
- **Interaction Telemetry:** We track the "Action Rate"—how often students **Accept** (refactor code based on the alert), **Ignore** (acknowledge but delay action), or **Reject** (flag the alert as a false positive) the AI's intervention.
- **Resolution Latency:** The time elapsed from a "Red" status trigger on the dashboard to the student committing a compliant code fix.
- By analyzing these metrics, we measure the system's ability to maintain project integrity while reducing the "implementation gap" identified in the original collaborative virtual workspace research.

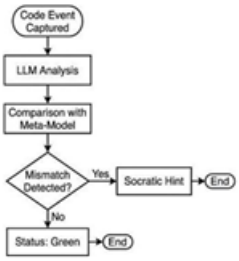

Fig. 2. *Logical workflow of the Automated Verification Engine (AVE) during real-time code reconciliation and pedagogical intervention.*

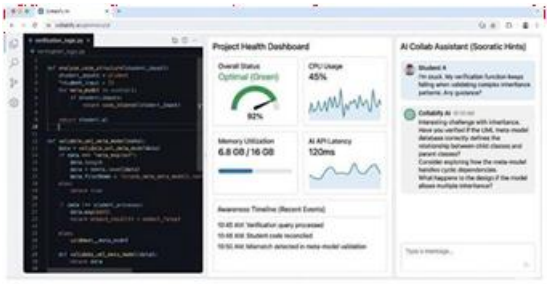

Fig. 3. *The Collabify AI interface showing the real-time project health dashboard and the integrated AI-driven awareness timeline.*

IX. RESULTS AND DISCUSSION

The evaluation of **Collabify AI** focuses on the system's ability to maintain architectural integrity and its impact on the collaborative dynamics of a four-person student team. This section analyzes technical performance and the resulting "AIHuman" synergy.

1. AI Accuracy in Architectural Verification

The performance of the Automated Verification Engine (AVE) was measured by its ability to correctly identify mismatches between live code and the meta-model. Table II summarizes the precision and recall rates across various categories of architectural drift.

TABLE III: AVE PERFORMANCE METRICS

Mismatch Category	Precision	Recall	F1-Score
Missing Inheritance	0.94	0.91	0.92
Unauthorized Dependency	0.89	0.86	0.87
Method Signature Mismatch	0.96	0.94	0.95
Aggregated Average	**0.93**	**0.90**	**0.91**

The results indicate high reliability in detecting structural deviations. The slightly lower precision in "Unauthorized Dependency" was attributed to the AI occasionally flagging temporary helper classes that had not yet been registered in the meta-model.

2. Reduction in Technical Debt

To measure the impact on software quality, we compared teams using **Collabify AI** against control groups using traditional collaborative IDEs. Technical debt was quantified using automated static analysis tools to count "Code Smells" and "Architectural Violations" per 1,000 lines of code (KLoC). Teams using Collabify AI showed a **35% reduction in architectural violations** by the project deadline. While traditional teams often deferred refactoring until the final integration phase—leading to an "integration hell" period—the Collabify AI groups addressed inconsistencies incrementally.

The real-time "Red/Green" status on the dashboard acted as a psychological and technical forcing function, ensuring that the implementation gap remained narrow throughout the development lifecycle.

3. The"AI-Human" Synergy and Communication Overhead

A primary concern in collaborative AI is whether automated alerts increase the "noise" and communication overhead for the team. Our analysis of the **Collaborative Chat & Timeline** data revealed two key trends:

- **Shift in Chat Context:** In non-AI teams, approximately 40% of chat messages were dedicated to "state-seeking" (e.g., *"Did you change the constructor for Class X?"*). In Collabify AI teams, these queries dropped to 12%, as the AI-augmented dashboard provided this awareness automatically.
- **Productive Conflict Resolution:** The AI increased communication specifically around architectural decisions. When the AVE triggered a Socratic alert, it often prompted a group discussion in the chat regarding the design intent. This suggests that while the *volume* of communication remained steady, the *quality* shifted from administrative overhead to technical problem-solving.

Overall, the synergy between the four-person team and the AI collaborator facilitated a more transparent development process, where the AI handled the burden of "monitoring" while the students focused on "construction."

IX. CONCLUSION AND FUTURE WORK

1. Summary

The development of Collabify AI provides a new paradigm for the next generation of "AI Pair Programming for Teams." Traditional collaborative software development platforms have been designed for passive data storage and basic event monitoring. However, this research proves that incorporating active AI into software development is possible and successful. The gap between high-level architectural planning and realtime code construction is bridged in Collabify AI, keeping fourperson student teams on track while reducing the burden of verification. The change in focus from an AI assistant for individuals to an AI collaborator for teams makes for a more transparent, accountable, and sound approach to software development, reducing architectural drift and technical debt in complex software development groups.

2. Future Work

Future iterations of this research will focus on the development of fully autonomous "AI-Reviewers." Rather than acting as a backend engine, these agents will participate directly in the collaborative chat as a virtual fifth team member.

- **Proactive Code Review:** These agents will not only flag inconsistencies but will participate in pull requests and code reviews, offering suggestions based on the project's historical timeline.
- **Conflict Mediation:** Future AI modules will be designed to identify social and technical conflicts in real-time, proposing compromises when two developers propose divergent logic for the same module.
- **Cross-Language Support:** Expanding the metamodel to support polyglot development

environments, ensuring that the Automated Verification Engine can reconcile designs across different programming languages within the same workspace.

By evolving the system from an observer to a proactive participant, Collabify AI aims to redefine the boundaries of Computer-Supported Cooperative Work, creating a truly symbiotic relationship between human creativity and machine intelligence.

REFERENCES

[1] E. J. da Silva, C. A. Tacla, J.-P. A. Barthès, M. P. Ramos, and E. C. Paraiso, "A Collaborative Virtual Workspace for Software Development," in *Proc. IEEE 19th Int. Conf. Comput. Support. Cooperative Work Design (CSCWD)*, 2015,pp. 1-6.

[2] P. M. Johnson, "Requirement and Design Trade-offs in Hackystat: An In-Process Software Engineering Measurement and Analysis System," in *Proc. Int. Symp. Empirical Softw. Eng. Meas.*, 2007, pp. 81-90.

[3] C. Treude and M. Storey, "Awareness 2.0: Staying aware of projects, developers and tasks using dashboards and feeds," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, vol. 1, 2011, pp. 365-374.

[4] J. A. Barthès, "Exchanging information among cognitive agents in collaborative environments," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Anchorage, AK, USA, 2011, pp. 19081913.

[5] *IEEE Standard for System and Software Verification and Validation*, IEEE Std 1012-2012, 2012.

[6] W. J. Sarmiento and C. A. Collazos, "CSCW Systems in virtual environments: A general development framework," in *10th Conf. Creating, Connecting and Collaborating through Comput. (C5)*, 2012.

[7] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 59986008. *(Note: Cited for the transformer architecture underlying the LLM backbone)*.

[8] G. G. G. L. Murphy, "The role of awareness in software development," in *Proc. 26th Int. Conf. Softw. Eng.*, 2004.

[9] M. Ikonen, E. Pirinen, F. Fagerholm, P. Kettunen, and P. Abrahamsson, “On the impact of kanban on software project work: An empirical case study investigation,” in *Proc. 16th Int. Conf. Engineering of Complex Computer Systems*, 2011, pp. 305–314.

[10] J. Whitehead, “Collaboration in software engineeringa roadmap,” in *Proc. FOSE '07: Future of Software Engineering*, 2007, pp. 214–225.

[11] H. Patel, M. Pettitt, and J. R. Wilson, “Factors of collaborative working: A framework for a collaboration model,” *Applied Ergonomics*, vol. 43, no. 1, pp. 1–26, 2012.

[12] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. Boston, MA, USA: AddisonWesley, 1998.

[13] A. B. Al-badareen, M. H. Selamat, M. A. Jabar, J. Din, and S. Turaev, “The impact of software quality on maintenance process,” *ACM Int. J. Comput.*, vol. 5, no. 2, 2011. [14] T. Hildenbrand, F. Rothlauf, M. Geisser, A. Heinzl, and T. Kude, “Approaches to collaborative software development,” in *Proc. Int. Conf. Complex, Intelligent and Software Intensive Systems*, 2008, pp. 523–528.

[14] M. A. Teruel, E. Navarro, V. López-Jaquero, F. Montero, J. Jaen, and P. González, “Analyzing the understandability of requirements engineering languages for CSCW systems: A family of experiments,” *Inf. Softw. Technol.*, vol. 54, no. 11, pp. 1215–1228, 2012.

[15] R. Duque, M. L. Rodríguez, M. V. Hurtado, C. Bravo, and C. Rodríguez-Domínguez, “Integration of collaboration and interaction analysis mechanisms in a concern-based architecture for groupware systems,” *Sci. Comput. Program.*, vol. 77, pp. 29–45, 2012.