

MAS-XP: A Conceptual Multi-Agent System Model for Representing and Coordinating Extreme Programming Practices

Khamisa Yousef^{1*} , Huweida Darbi² 

¹Computer Department, Faculty of Science, University of Tobruk, Tobruk, Libya.

²Computer Department, Faculty of Science, University of Derna, Derna, Libya.

Email: khamisa.yousef@tu.edu.ly

Abstract

Extreme Programming (XP) is a prominent agile methodology emphasising technical discipline, continuous feedback, pair programming, test-driven development, and refactoring. However, managing human coordination overhead, synchronising continuous integration, and maintaining architectural health across distributed teams present significant challenges. This paper proposes a comprehensive Multi-Agent System (MAS) as a high-level conceptual model for extreme programming (MAS-XP). By mapping XP roles, artefacts, and engineering practices into a high-level conceptual and architectural model, smart blackboards, and distributed profiles, the proposed model operationalises decentralised software engineering governance. The framework defines specialised agents for planning, pair programming, testing, refactoring, and feedback, interconnected via intelligent blackboard architectures and formal protocol sequences. This conceptual model provides an architecture for representing and coordinating XP software development activities.

Keywords. Multi-Agent Systems (MAS), Conceptual Model, Extreme Programming (XP), Agile Software Engineering.

Introduction

The modern software industry operates in an environment characterised by rapid technological shifts, volatile user requirements, and intense market competition. Historically, traditional heavyweight software development methodologies, such as the Waterfall model, relied heavily on exhaustive upfront planning, strict sequential phases, and rigid documentation. While structured, these methodologies often proved inflexible when user needs evolved, leading to delayed deliveries, misaligned features, and high project failure rates [1]. To overcome these structural limitations, agile software development methodologies emerged, prioritising adaptive planning, evolutionary development, early incremental delivery, and continuous stakeholder feedback [2]. Among the various agile frameworks, Extreme Programming (XP), pioneered by Kent Beck in the late 1990s, stands out as a uniquely disciplined and rigorous methodology [3]. Unlike management-centric Agile frameworks (such as Scrum), which primarily focus on sprint organisation and backlog prioritisation, XP targets the technical core of software engineering. XP is grounded in five core values—Communication, Simplicity, Feedback, Courage, and Respect—and translates them into a cohesive set of interdependent engineering practices designed to elevate software quality and maintainability to extreme levels [4].

Lifecycle and Phases of Extreme Programming (XP)

The XP lifecycle operates as a continuous, iterative feedback loop rather than a linear sequence. It revolves around four primary developmental activities: Coding, Testing, planning, and designing. The Extreme Programming method consists of four stages, as shown in (Figure 1) [5].

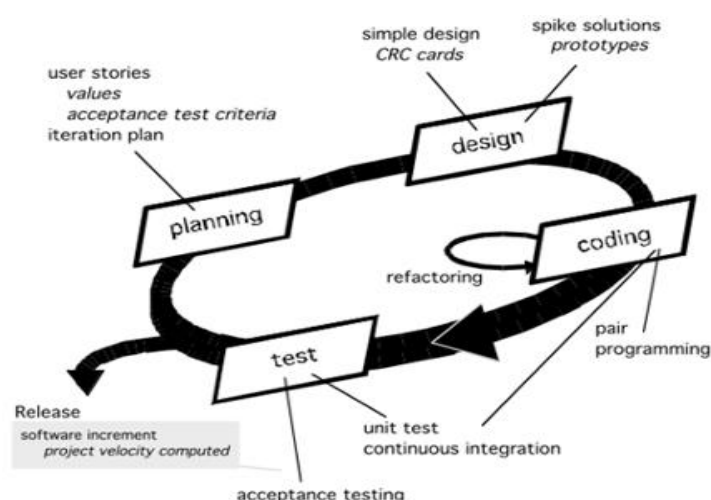


Figure 1. Extreme Programming (XP) Method.

Planning phase (the planning game & small releases)

Development begins with the customer defining business requirements as user stories. Through the "planning game," technical staff and customers negotiate scope and schedule for upcoming iterations, culminating in frequent, small production releases [5].

Designing Phase (Simple Design & Metaphor)

Teams maintain the simplest possible design that works for current requirements, avoiding premature over-engineering. Establishing a shared system metaphor to guide architectural naming and understanding across the team [5].

Coding Phase (Pair Programming and TDD)

This phase is implemented through rigorous technical practices. Pair programming requires two developers (driver and navigator) to write all production code at a single workstation (driver and navigator). Concurrently, TDD enforces the writing of automated unit tests before functional code is written [5].

Testing and Continuous Integration (CI) Phase

Automated unit and acceptance tests continuously verify system behaviour. Code increments are integrated into the main repository multiple times a day via automated continuous integration pipelines to instantly detect merge conflicts and regressions [5].

Related Work

The convergence of MAS and Agile software engineering methodologies has gained considerable momentum in software engineering research [6]. Researchers have increasingly utilised Agent-Based Modelling (ABM) to simulate, analyse, and optimise decentralised development processes, recognising that software teams naturally exhibit multi-agent characteristics such as autonomy, collaborative negotiation, and decentralised coordination [7]. To position the proposed MAS-XP framework, this review examines three primary streams of literature focusing on agent-based modelling in Agile and Extreme Programming environments:

Agent-Based Simulation of Agile and Scrum Management Processes

Early research at the intersection of MAS and Agile methodologies primarily focused on macro-level project management and Scrum sprint organisation. For instance, Yousef and Fazzani [8] proposed a conceptual multi-agent model to manage Scrum processes, utilising smart blackboard architectures and specialised agent roles to handle user stories, sprint planning, and backlog profiles. Similarly, studies examining agent-based simulation for agile management demonstrated how autonomous agents could model estimation, task allocation, and iteration workflows [9]. International studies have explored the application of agent-based simulation to improve agile software management, highlighting how distributed decision-making agents can dynamically respond to shifting project constraints [10].

Multi-Agent Modelling of Team Dynamics and Agile Strategies

In addition to project management, researchers have used agent simulations to investigate the micro-behavioural dynamics of software development teams using agent simulations. A notable study by Wang [11] examined the team-working strategies of agile teams through multi-agent simulation, evaluating how different agent behaviours and communication patterns affect team productivity and task completion times. Expanding on team dynamics, research into multi-agent simulation of agile team strategies provided deep insights into how peer interaction and collaboration protocols affect overall project performance under varying workloads [12]. These studies confirm that agent-based models effectively capture the emergent properties of human collaboration that traditional analytical models fail to represent.

Agent-Based Approaches for Technical Practices (Pair Programming and TDD)

While much of the existing Agile-MAS literature focuses on management frameworks such as Scrum, specialised research has begun addressing engineering-intensive practices such as XP. For instance, academic investigations into simulating the impact of pairing on agile software development projects utilised multi-agent architectures to model pair programming dynamics, knowledge transfer, and defect reduction trade-offs [13]. Such frameworks illustrate how agent attributes—such as skill heterogeneity, fatigue, and peer-to-peer code review—can be mathematically operationalised. The reviewed studies show that MAS has been applied to Agile management [8]–[10], team dynamics [11], [12], pair programming [13], and DevOps automation [14]. However, these studies generally address one management activity, one team behaviour, or one technical practice at a time. In addition, the previous MAS model developed for Feature-Driven Development by Yousef and Darbi [16] is concerned with feature-oriented development and does not specifically represent the technical practices and feedback cycle of XP. Although there is an increasing amount of research that links Multi-Agent Systems (MAS) with agile frameworks, a significant research gap remains. The majority of existing MAS-based models primarily focus on high-level management processes such as the organisation of Scrum sprints or other methodologies like Feature-Driven Development, largely neglecting the rigorous technical foundation and ongoing feedback mechanisms that define Extreme Programming (XP). In particular, earlier studies have not offered a cohesive, decentralised architectural model capable of simultaneously managing XP's essential engineering practices—specifically, pair programming, test-driven development,

continuous integration, and continuous refactoring. This paper aims to fill this gap by introducing MAS-XP, a thorough conceptual multi-agent model intended to operationalise and oversee decentralised XP software development processes with the use of specialised agents, intelligent blackboards, and adaptive project profiles.

The Proposed MAS-XP Conceptual Model

The proposed MAS-XP conceptual model establishes a decentralised architecture where software engineering tasks, testing cycles, and code integration are managed through interacting agent roles, smart blackboards, and dynamic profiles. The proposed MAS-XP conceptual model is illustrated in Figure 2.

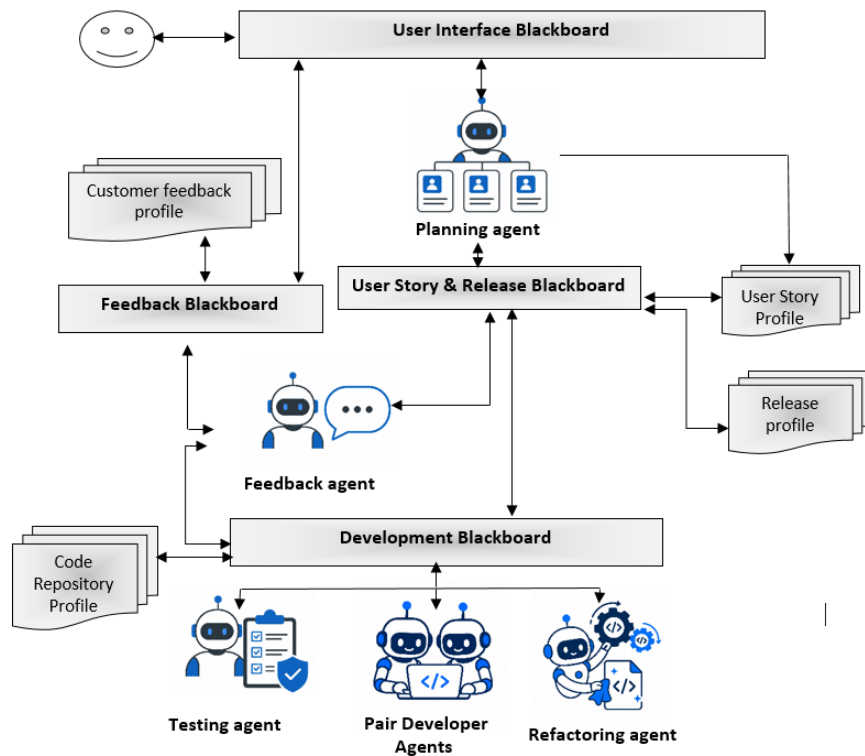


Figure 2. A proposed model for the XP development process based on MAS

Before listing the details of our work, we must first clarify the important structures on which our proposed model is based:

Multi-Agent System Roles

This proposed model contains five main agents with various responsibilities, which are succinctly described in (Table 1).

Table1. Proposed MAS-XP Agents

Agent role	Responsibility
Planning agent	Manages user story elicitation, backlog prioritisation, and iteration planning.
Pair Developer Agents	Collaborate on a single workstation (Driver/Navigator) to implement features and write code.
Testing Agent	Manages automated test execution, verifies acceptance criteria, and reports test results, while the Pair Developer Agents perform the Red–Green–Refactor development cycle.
Refactoring agent	Monitors code complexity, ensures a simple design, and refactors continuously.
Feedback Agent	Gathers customer feedback, manages acceptance testing, and tracks release satisfaction.

XP-Smart Blackboards

A blackboard is modelled as a shared coordination space through which authorised agents publish, retrieve, and update project-related information. It is not considered an autonomous agent; rather, it is an environmental artefact that supports indirect communication and shared project memory. This proposed model contains four main smart boards with their various features, which are succinctly described below in (Table 2).

Table 2. Proposed XP Smart Blackboards

Blackboard Name	Description
User Interface Blackboard	Gathers customer feedback, manages acceptance, serves as the primary gateway for User/Customer interaction with the system to submit initial requirements, review increments, and provide acceptance feedback, and tracks release satisfaction.
User Story & Release Blackboard	Stores and organises user stories, iteration backlogs, and release milestones.
Development Blackboard	Coordinates technical execution between developers, testers, and designers.
Feedback Blackboard	Facilitates communication regarding customer evaluations, bug reports, and acceptance status.

XP-profiles

There are four main profiles that make up proposed model: the user story profile, the release profile, feedback profile, and the code repository profile. Which are succinctly described below in (Table 3).

Table 3. Proposed XP- profiles

Profile name	Description
User Story Profile	Maintains detailed user stories, acceptance criteria, and estimation points.
Release Profile	Tracks release schedules, sprint increments, and delivery milestones.
Feedback Profile	Records customer reviews, defect logs, and retrospective improvement items.
Profile Code Repository	The repository in which the code is saved and managed

To represent and describe the interaction and communication between the roles of agents, we suggested using the serial model in this work, as it helps in following up the temporal sequence between communications [15].

Planning Agent

The Planning Agent analyses incoming requirements, refines user stories, and organises them according to their priorities and release objectives. It communicates with the User Story and Release Blackboard to update acceptance criteria and release information. Once a story is ready, the agent publishes the corresponding iteration task to the Development Blackboard. The interaction sequence of the Planning Agent is illustrated in Figure 3.

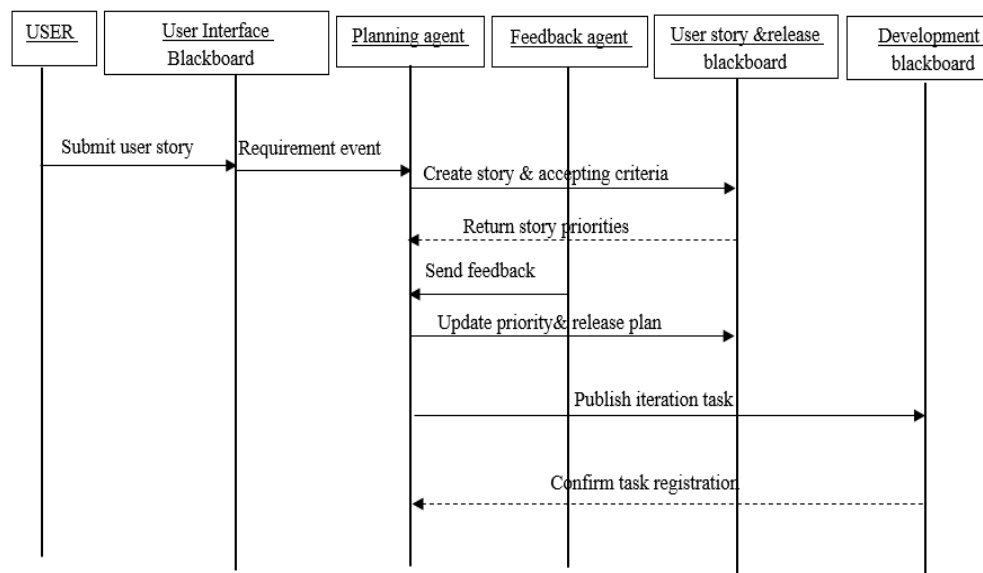


Figure 3. Sequence diagram for High-Level Communication Protocol for Requirements Analysis and Iteration Planning

Feedback Agent

The Feedback Agent captures customer responses to a delivered increment and records them in the Feedback Blackboard. It classifies each response as an acceptance, change request, or defect, and associates it with the affected user story. The agent then notifies the Planning Agent so that priorities and future release plans can be revised. The interaction sequence of the Feedback Agent is illustrated in Figure 4.

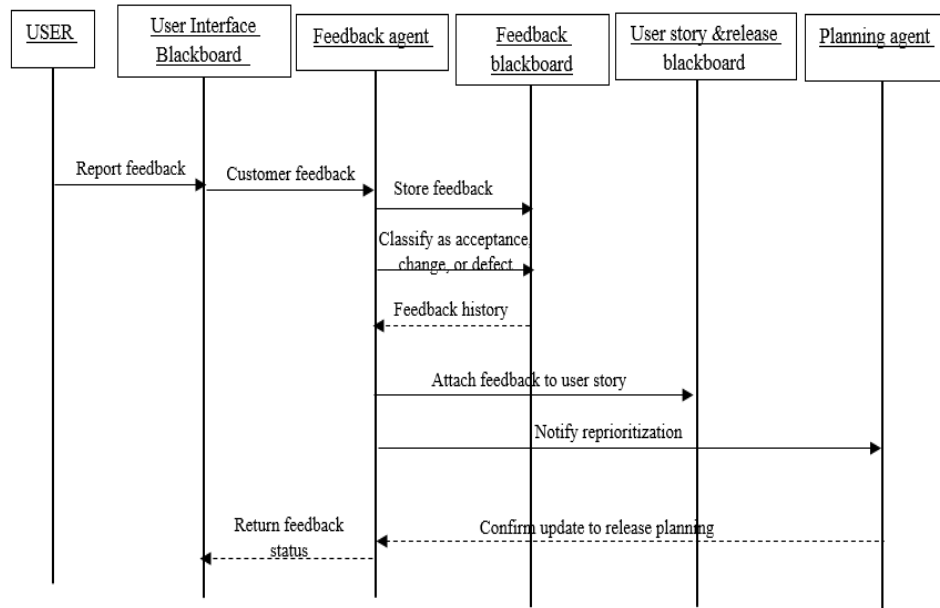


Figure 4. Sequence diagram for High-Level Communication Protocol for Customer Feedback Classification and Prioritisation

Pair Developer Agents

The pair developer agents collaboratively implement the selected user story by adopting temporary driver and navigator roles. They share the development context, review the implementation continuously, and rotate their roles during the coding activity. After completing the increment, the pair submits the code to the repository and continuous integration environment for verification. The interaction sequence of the Pair Developer Agents is illustrated in Figure 5.

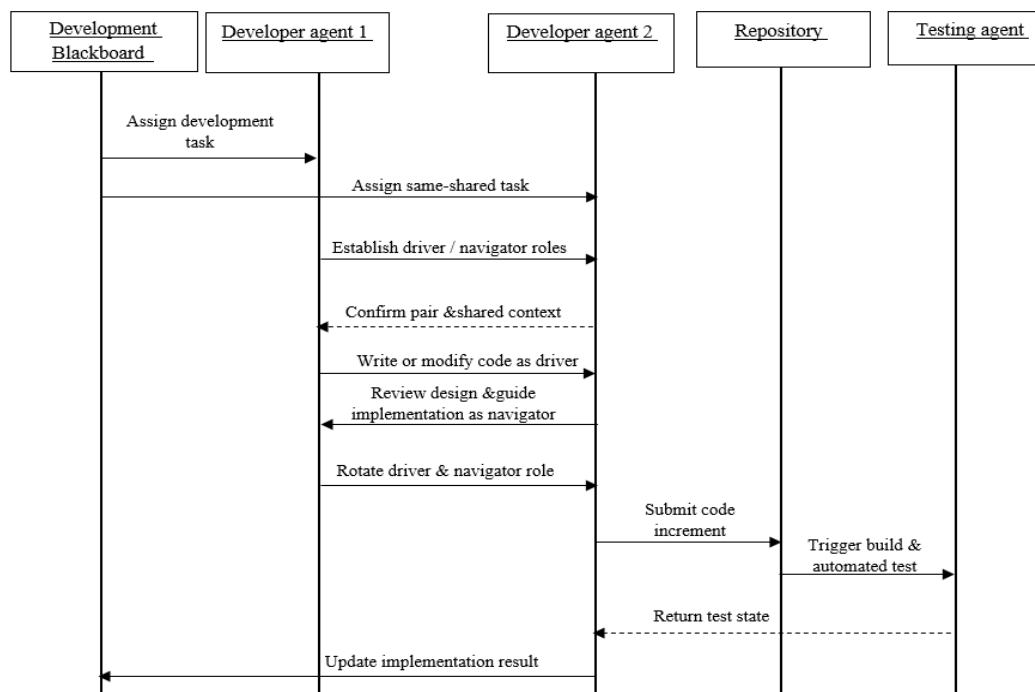


Figure 5. Sequence diagram for High-Level Communication Protocol for Pair Programming and Collaborative Implementation

Testing Agent

The Testing Agent verifies the submitted code by executing unit, integration, and acceptance tests against the defined acceptance criteria. When a defect is detected, it publishes a failure report and notifies the development and refactoring agents. If all tests pass, the agent records the verification status and makes the increment eligible for the release decision. The interaction sequence of the Testing Agent is illustrated in Figure 6.

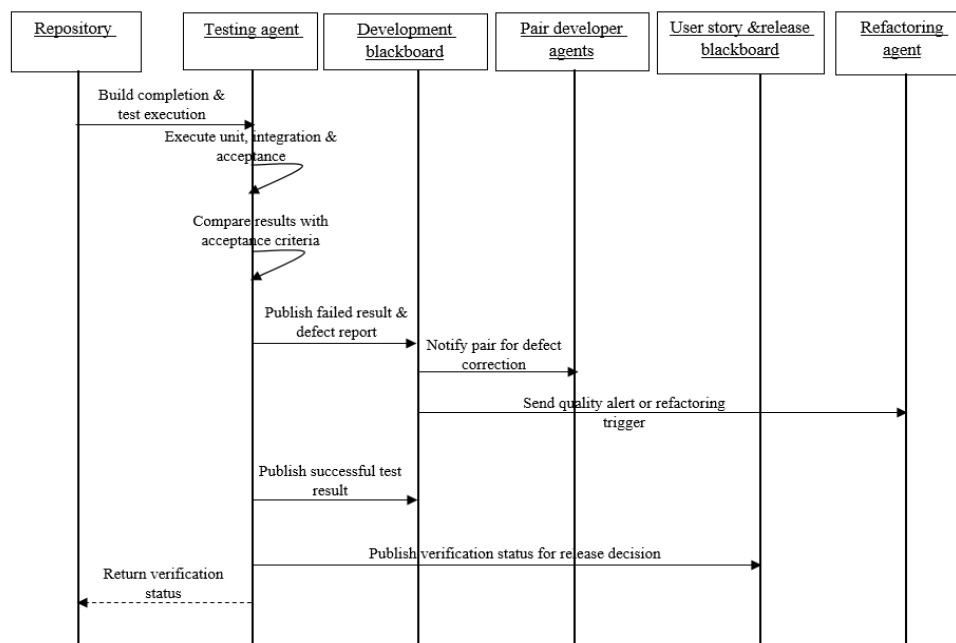


Figure 6. Sequence diagram for High-Level Communication Protocol for Automated Testing and Defect Reporting

Refactoring Agent

The Refactoring Agent is activated when code-quality problems, excessive complexity, duplication, or maintainability risks are identified. It collaborates with the pair developer agents to select and apply a structural improvement without changing the expected external behaviour. The refactored increment is then submitted for regression testing, and the quality status is updated according to the test results. The interaction sequence of the Refactoring Agent is illustrated in Figure 7.

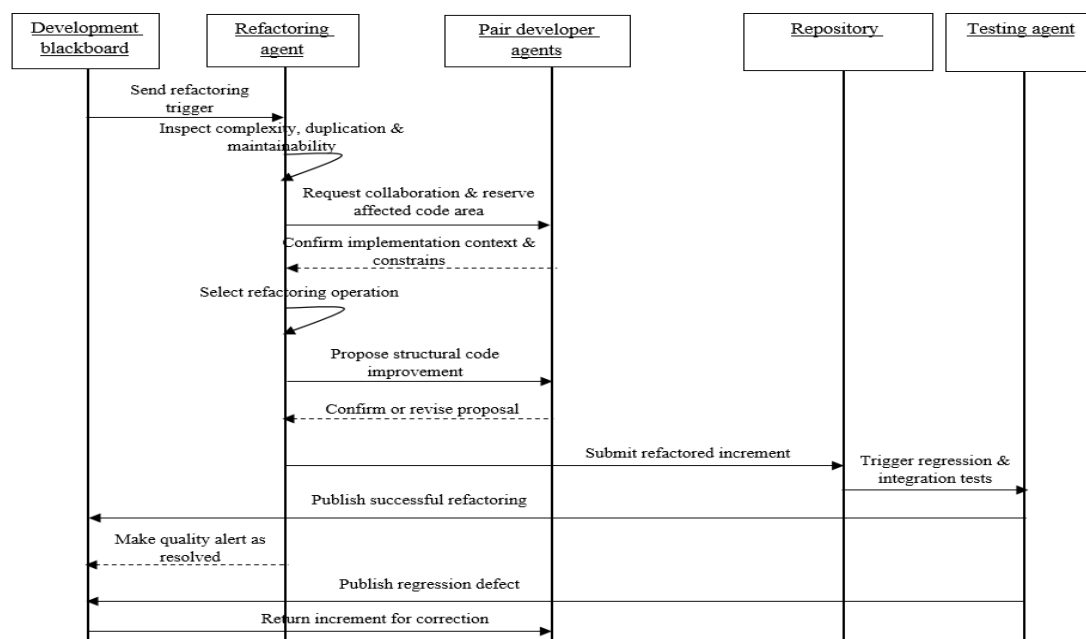


Figure 7. Sequence diagram for High-Level Communication Protocol for Code Quality Improvement and Refactoring

Conclusion

Extreme programming, when combined with multi-agent systems, produces an architecture that improves the software development lifecycle. Previously, we produced a study that was conducted on the multi-agent system as a conceptual model for an adaptation of feature-driven development methodology [16]. This paper presented MAS-XP, a conceptual model for supporting the Extreme Programming process through a multi-agent system. The proposed model was motivated by the need for a structured approach capable of representing the collaborative, iterative, and technically intensive nature of XP. Unlike conventional process-support models, MAS-XP integrates the main XP practices within a unified agent-oriented architecture. The model assigns specific responsibilities to specialised agents. The planning agent manages customer requirements and transforms them into user stories and development features. Development agents support implementation and pair programming, while the testing agent coordinates test-driven development and verifies the

implemented functionality. The pair developer and refactoring agent maintain simple and adaptable software design, and the feedback agent communicates customer evaluations to the development environment. The proposed blackboard architecture provides a shared communication environment through which agents exchange requirements, tasks, testing results, refactoring decisions, and feedback. The associated profiles preserve project information and support traceability throughout the development cycle. This organisation improves coordination among agents and ensures that the information required by each agent remains available during the different stages of the XP process. MAS-XP is also characterised by modularity, scalability, and flexibility. Agents can be added, removed, or reorganised according to changes in project requirements and team structure without modifying the overall architecture. Thus, the model can accommodate different XP development contexts while maintaining decentralised cooperation among its components. In conclusion, MAS-XP provides a strong conceptual foundation for representing XP practices as explicit, cooperating agent responsibilities. It connects agile software engineering with agent-oriented system design and offers an extensible architecture for future intelligent development environments. Future work will focus on formalising agent behavior and communication protocols, implementing the model on a MAS platform, and evaluating its effectiveness in terms of coordination, traceability, software quality, and responsiveness to changing requirements.

References

1. Sommerville I. Software engineering. 10th ed. Boston (MA): Pearson; 2015.
2. Beck K. Extreme programming explained: embrace change. 2nd ed. Boston (MA): Addison-Wesley; 2004.
3. Fowler M, Beck K, Brant J, Opdyke W, Roberts D. Refactoring: improving the design of existing code. Boston (MA): Addison-Wesley; 1999.
4. Cockburn A. Agile software development: the cooperative game. 2nd ed. Boston (MA): Addison-Wesley; 2006.
5. Carolina I, Supriyatna A. Penerapan metode Extreme Programming dalam perancangan aplikasi perhitungan kuota SKS mengajar dosen. J IKRA-ITH Inform. 2019 Mar;3(1):106-13.
6. Wooldridge M. An introduction to multiagent systems. 2nd ed. Chichester (UK): Wiley; 2009.
7. North MJ, Macal CM. Managing business complexity: discovering strategic solutions with agent-based modeling and simulation. New York (NY): Oxford University Press; 2007.
8. Yousef KA, Fazzani FH. Multi agent system as conceptual model for managing Scrum process. J Bani Waleed Univ Humanit Appl Sci. 2023;8(2):524-39. Available from: <https://jhas-bwu.com/index.php/bwjhas/article/view/478>
9. Wang Z. Estimating productivity in a Scrum team: a multi-agent simulation. In: Proceedings of the 2019 3rd International Conference on Computer Science and Artificial Intelligence (CSAI 2019); 2019. p. 239-45. doi: 10.1145/3307363.3310985.
10. Hurtado N, Ruiz M, Capitas C, Orta E. Applying agent-based simulation to the improvement of agile software management. In: Mas A, Mesquida A, O'Connor RV, Rout T, Dorling A, editors. Software process improvement and capability determination: SPICE 2017. Cham (Switzerland): Springer; 2017. p. 173-86. doi: 10.1007/978-3-319-67383-7_13.
11. Wang Z. Teamworking strategies of Scrum team: a multi-agent based simulation. In: Proceedings of the 2nd International Conference on Computer Science and Artificial Intelligence (CSAI 2018); 2018. p. 404-8. doi: 10.1145/3297156.3297179.
12. Wang Z. Multi-agent simulation of agile team dynamics: an investigation on team strategies perspective. In: Software engineering and algorithms; 2021. p. 1-36. doi: 10.1007/978-3-030-77442-4_1.
13. Wang Z. Simulating the impact of pairing on agile software development projects: a multi-agent approach [dissertation]. Christchurch (New Zealand): Lincoln University; 2020. Available from: <https://hdl.handle.net/10182/13060>
14. Yasar AUH, Tahir M, Asim M. Agent-based DevOps pipeline automation: a simulation-based study. J Syst Softw. 2020 Apr;162:110489. doi: 10.1016/j.jss.2019.110489.
15. Odell JJ, Van Dyke Parunak H, Bauer B. Representing agent interaction protocols in UML. In: Ciancarini P, Wooldridge MJ, editors. Agent-oriented software engineering: AOSE 2000. Lecture Notes in Computer Science, vol. 1957. Berlin (Germany): Springer; 2001. p. 121-40. doi: 10.1007/3-540-44564-1_8.
16. Yousef KA, Darbi HA. Multi-agent system adaptation for feature driven development methodology. Afr J Adv Pure Appl Sci. 2025;4(3):71-8. Available from: <https://www.aaasjournals.com/index.php/ajapas/article/download/1316/1226>