Sarcouncil Journal of Engineering and Computer Sciences

ISSN(Online): 2945-3585

Volume- 04| Issue- 11| 2025

Review Article **Received:** 20-10-2025| **Accepted:** 14-11-2025 | **Published:** 27-11-2025

Multi-agent Orchestration for Software Development: A Comprehensive Review

Siddhant Sonkar

University of California, Irvine, USA

Abstract: Multi-agent orchestration is a revolutionary approach to contemporary software development, solving the spiraling complexity of enterprise applications cutting across several services, frameworks, and platforms. This systematic review delves into how independent computational agents work together, negotiate, and coordinate tasks throughout the entire software development process. The smart agents have advanced capabilities such as requirements analysis, architecture synthesis, code generation, test execution, defect localization, and deployment orchestration. The agents interact through advanced coordination mechanisms that control task allocation, conflict resolution, resource sharing, and synchronization across distributed development teams. The orchestration framework provides hierarchical, peer-to-peer, and hybrid architectural styles with their unique strengths for scalability, fault tolerance, and coordination efficiency. Mechanisms of communication, such as shell agent communication languages, Contract Net Protocol, commitment-based coordination, and consensus algorithms, facilitate unimpeded information exchange and collective decision-making. Usage extends to requirements engineering, architecture design, code implementation, testing and quality assurance, continuous integration, deployment automation, and maintenance activities. Even with considerable improvements in productivity, defect reduction, and deployment frequencies, there are still important challenges to address, such as scalability issues with coordination overhead, interoperability hurdles with heterogeneous tool chains, trust and transparency issues impacting developer adoption, security risks from compromised agents, and ethical aspects of responsibility attribution. Future directions encompass advanced learning algorithms, human-agent collaboration frameworks, and emerging technologies, including quantum computing and edge computing architectures that promise to fundamentally transform multi-agent orchestration capabilities.

Keywords: Multi-Agent Orchestration, Software Development Lifecycle, Autonomous Agents, Distributed Coordination, Intelligent Automation.

INTRODUCTION

The complexity of modern software development has escalated dramatically over the past decade, with enterprise applications often comprising millions of lines of code distributed across multiple services, frameworks, and platforms. Modern software systems are often larger than five million lines of code for mid-scale commercial applications, with large-scale cloud-native systems over twenty million lines spread across microservices architectures with hundreds of independent deployable components. Older software engineering practices, based on strong centralized control and sequential methods, struggle ever more to cope with the requirements of high-frequency deployment cycles, continuous integration, and dynamic system structures. Organizations indicate that traditional waterfall approaches take six to twelve months for big release cycles, whereas contemporary agile environments expect deployment rates of multiple releases weekly or even daily. Multi-agent orchestration has also been identified as a promising paradigm to support these needs based on independent computational entities that can cooperate, bargain, and coordinate their behaviors to pursue intricate software development goals (Kumar, R. 2025). This method takes its inspiration from distributed artificial intelligence and uses the paradigms of agent-based systems in different stages of the software development cycle, ranging from requirements engineering to design, implementation, testing, deployment, and maintenance.

Multi-agent systems in software development applications are composed of numerous intelligent agents that have certain capabilities, knowledge areas, and decision-making autonomy, allowing them to act independently but be a part of the collective goals (Sarangi, L. & Panda, C. 2011). These agents work in a common world where they can sense changes through sensors tracking code repositories, build systems, test outputs, and deployment status, exchange messages with other agents via standard protocols, and take actions to realize individual and collective objectives. Each agent can be specialized to perform specific roles like requirements analysis, architecture generation, code generation, testing, defect location, performance tuning, security scanning, or deployment management. The coordination of these agents includes advanced coordination mechanisms that control task distribution through auction-based or contract-net protocols, conflict resolution through negotiation and consensus algorithms, resource sharing across distributed computing infrastructure, and synchronization of

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

***Corresponding Author:** Siddhant Sonkar

DOI- https://doi.org/10.5281/zenodo.17741178

activities across distributed development teams and automated processes across diverse geographic locations and time zones.

Studies in this area have shown that multi-agent coordination can cause substantial productivity improvements across software development organizations. Empirical data from business deployments have reported development time savings of between twenty-five and forty percent for applications with over fifty developers, with the size of improvement being positively correlated to project complexity and team spread. Defect density reductions of around thirty to fifty-five percent per thousand lines of code are evidence of improved code quality, quantified by lower post-deployment defect rates and less technical debt being accumulated. Metrics for test coverage grow by two to thirty-five percentage points with automated test generation and smart test case prioritization, while regression test run times are reduced by fifty to seventy percent with parallel distributed testing commanded by expert agents. Multi-agent testing environments exhibit special utility in verifying intricate applications via collaborative artificial intelligence methods that allocate test burden among a number of specialized agents, each bearing responsibilities on detailed quality aspects such as functional accuracy, performance properties, security weaknesses, or usability issues.

The scalability of multi-agent orchestration platforms offers opportunities as well as challenges, because coordination mechanisms may increase with the number of participating agents. Research shows that coordination overhead consumes thirty to fifty percent of computations in systems with more than one hundred agents when implementing naive communication patterns. But hierarchical coordination techniques and locality-oriented strategies can bring the overhead down to ten to twenty percent without diminishing the level of collaboration. The present review offers a comprehensive investigation into the condition of multi-agent orchestration in software engineering; it considers the theoretical perspective, implementations in the real world, and empirical evidence from the industrial context to inform researchers and practitioners of the potential, limitations, and possibilities of this emerging paradigm.

Foundational Concepts and Architectural Frameworks

Software development multi-agent orchestration is based on distributed artificial intelligence, software engineering, and organisational theory principles developed over the past few decades of research and practice. In essence, an agent can be described as an independent computational actor that receives its perception from sensors in its environment and sends its actions to the environment through effectors to meet a particular goal. Agents in software development applications can be implemented as automated tools carrying out activities like continuous integration monitoring, code quality checking, or deployment orchestration, human developers enhanced with smart assistants offering real-time suggestions, or hybrid agents mixing automated reasoning with human management for key decision-making nodes (Mahmud, S. *et al.*, 2025). Defining features of agents are autonomy of decision-making, reactivity to environmental changes to respond quickly to code commits or build failures, proactivity in chasing goals with anticipatory planning, and social ability to communicate with other agents using structured communication protocols to support hundreds to thousands of messages per minute for active development use cases.

The multi-agent orchestration architectural models differ based on centralization level, interaction between agents, and the type of development activity being facilitated. Hierarchical models use a centralized orchestrator that coordinates task assignment, tracks progress through regular status polling, and resolves conflicts between the underlying agents. This strategy offers high consistency guarantees with consistency levels usually over ninety-eight percent correctness, but has the potential to cause bottlenecks upon scaling to large agent counts. Hierarchical orchestration has been demonstrated to reduce test run time by three to eightfold when parallel agent pools are distributed, depending on test suite behavior and infrastructure power.

Peer-to-peer architectures are another solution where agents communicate directly without centralized control, negotiating duties via distributed protocols. These architectures provide higher fault tolerance, with system availability frequently being higher than ninety-nine point five percent, even if ten to twenty percent of the agents fail. Market-based coordination mechanisms, in which the agents bid for work according to their ability and existing workloads, have proven efficient solutions with load balancing

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher

improvements of about fifteen to thirty percent over static task assignment schemes (Tricon Infotech, 2024).

Hybrid systems are blends of hierarchical and peer-to-peer structures to take advantage of the benefits of both architectures. Agents are grouped into clusters, normally with five to fifteen agents per cluster, with peer-to-peer within teams and hierarchical between team leaders. Experience from case studies of enterprise software projects suggests that hybrid orchestration can decrease coordination overhead by about twenty to forty percent when dealing with over fifty agents. It takes meticulous attention in the design of knowledge representation schemes, capturing code structure, design patterns, quality measures, and constraints of the project. Ontology-based solutions facilitate semantic interoperability, decreasing integration effort by two to four times in the addition of new agent types. Agent reasoning skills integrate rule-based systems for transparency with machine learning methods for adaptability, with reinforcement learning methods demonstrating built success rate improvements of five to fifteen percentage points through adaptive parameter optimization (Sun, L. *et al.*, 2025; Gaur, S. *et al.*, 2025)

Architecture Type	Key Characteristics	Primary Applications
Hierarchical	Centralized orchestrator with subordinate agents, strong consistency guarantees, periodic status polling	Automated testing environments, parallel test execution, build coordination
Peer-to-Peer	Direct agent interaction, distributed protocols, market-based coordination mechanisms	Distributed task allocation, fault-tolerant systems, dynamic agent collaboration
Hybrid	Combined hierarchical and peer-to-peer elements, cluster-based organization, multi-level coordination	Agile development teams, enterprise software projects, scalable coordination
Knowledge Representation	Ontology-based approaches, semantic interoperability, domain-specific vocabularies	Agent integration, cross-platform communication, artifact description
Reasoning Systems	Rule-based and machine learning integration, reinforcement learning, adaptive parameter tuning	Continuous integration pipelines, decision-making automation, process optimization

Figure 1: Architectural Frameworks for Multi-agent Orchestration (Mahmud, S. *et al.*, 2025; Tricon Infotech, 2024)

Communication and Coordination Mechanisms

Successful multi-agent orchestration intrinsically relies on stable communication infrastructures allowing agents to share knowledge, negotiate obligations, and coordinate actions between dispersed software development environments. Agent communication languages offer standardized syntactic and semantic structures for message formation among agents, assuring interoperability between heterogeneous agent implementations created with various programming languages, frameworks, and platforms (Sonkar, S. 2025). FIPA-ACL has become a commonly used standard that specifies message structures having header fields for sender addressing, receiver addressing, and conversation threading, performatives to state communicative acts such as requests, proposals, acceptances, and rejections, and ontologies to represent content allowing semantic interpretation of domain-specific concepts. In applications for software development, FIPA-ACL messages are used for coordinating activities like requesting a code review with defined levels of urgency, assigning implementation tasks with precise specifications, sending reports of bug findings with severity

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher

rankings, and negotiating computational infrastructure resource allocations in terms of CPU cores, memory allocation, and storage space. Performance measurements have indicated message processing latencies generally between five and twenty milliseconds for typical message types, with latency increasing roughly proportionally with message content size to practical limits.

Aside from standardized communication languages, coordination protocols specify structured interaction patterns that regulate how agents cooperate to realize common goals while dealing with dependencies, resource limitations, and temporal constraints. Contract Net Protocol is a basic coordination mechanism where manager agents post tasks with descriptions specifying the required capabilities, quality constraints, and deadlines for task completion, and the contractor agents bid in terms of their capability and their availability at the moment (Siddhant, S. 2025). This protocol has been widely used in distributed software development situations for assigning coding tasks among development agents with heterogeneous expertise, sharing testing tasks among specialized test agents, and jointly coordinating documentation tasks among technical writing agents. Extensions to the simple Contract Net Protocol include temporal constraints specifying deadlines and sequencing dependencies, quality requirements listing acceptance criteria, and trust metrics quantifying agent reliability based on past performance data. Empirical comparisons show that contract net-based coordination can produce task allocation efficiency gains of about twenty to thirty-five percent over random or round-robin allocation schemes, especially when agent abilities are diverse with performance differences of two to five times within agents.

Commitment-based coordination mechanisms focus on social responsibility and accountability by way of open contracts prescribing behavior and performance targets. Agents make explicit promises to do something or produce some artifact by given timeframes, forming webs of dependencies that organize cooperative endeavors and facilitate traceability of dependencies. Commitment protocols define how agents engage in commitment negotiation via proposal exchanges, check for commitment fulfillment via progress monitoring, manage violations upon deadline failure, and reconcile commitments when conditions change as a result of requirement changes or resource availability changes. Research in software process modeling has shown that commitment-based coordination can decrease project delays by factors of around one point three to one point eight through enhanced visibility into agent obligations and active management of conflict of commitments.

Mechanism Type	Core Functions	Application Scenarios
Agent Communication Languages	Message structuring, sender identification, performative expressions	Code review requests, task delegation, bug reporting, resource negotiation
Contract Net Protocol	Task announcement, bid submission, contractor selection	Coding task allocation, testing distribution, documentation coordination
Commitment-Based Coordination	Explicit commitments, obligation monitoring, violation handling	Long-running projects, deadline management, dependency tracking
Consensus Mechanisms	Collective decision-making, Byzantine fault tolerance, agreement protocols	Version control systems, deployment pipelines, collaborative editing
Synchronization Protocols	Timing coordination, resource locking, conflict detection	Concurrent code modifications, shared artifact access, distributed builds

Figure 2: Communication and Coordination Protocols (Sonkar, S. 2025; Siddhant, S. 2025)

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher

Applications in Software Development Lifecycle
Multi-agent orchestration has been utilized in every phase of the software development lifecycle, reshaping conventional development methods by enabling improved automation, better coordination, and adaptive decision-making features that react dynamically to evolving project circumstances. During requirements engineering, multi-agent systems drive software requirement elicitation, analysis, and validation from heterogeneous stakeholder communities such as business analysts, end users, domain specialists, and regulatory compliance managers. Requirements gathering agents engage stakeholders by means of natural language interfaces, abstracting functional and non-functional requirements from dialogue, texts, and system specifications present before them and presenting these requirements in organized forms like use case diagrams, user stories, and formal specification languages. Collaborative filtering agents examine needs for conflicts in which stakeholders possess conflicting expectations, ambiguities in which requirements do not contain enough detail to implement, and incompleteness in which essential system behaviors are still unspecified, highlighting issues that need stakeholder explanation or negotiation through formal review sessions. Case studies on enterprise requirements engineering projects have indicated twenty to forty percent decreases in requirements defects when using multi-agent orchestration over more conventional manual requirements processes, with defect types being inconsistencies, missing requirements, and incorrect interpretation (Sonkar, S. 2025).

In architecture and design stages, expert agents interact to produce system architectures that meet functional requirements as well as optimize non-functional quality characteristics like response time performance in milliseconds, reliability in terms of availability percentages, and maintainability in terms of coupling and cohesion ratios. Architecture synthesis agents search design space by producing candidate architectures in the hundreds or thousands, analyzing their properties via simulation or analytical models, and iteratively improving designs from feedback of constraint verification and performance prediction. Pattern-matching agents detect where to use pre-proven design patterns like Model-View-Controller, Observer, Strategy, and Factory patterns that solve common design issues, enhancing consistency among system parts and taking advantage of battle-tested solutions extensively used in the industry. Empirical studies have shown increases in measures of architectural quality with reductions in architectural technical debt by around fifteen to thirty percent over hand-designed architectures, according to studies.

Code generation and implementation tasks are greatly aided by multi-agent orchestration, especially in large development projects where more than one programming language is involved such as Java, Python, JavaScript, and C++, varied frameworks that include web development, mobile apps, and enterprise software, and several deployment environments that cover cloud infrastructure, on-premises data centers, and edge devices (Sonkar, S. 2025). Code generation agents convert design specifications to executable code, using templates for typical code patterns, libraries offering reusable functionality, and domain-specific knowledge encoding best practices for specific application domains. Industry deployments are said to have achieved productivity gains from twenty-five to sixty percent for typical coding tasks such as data access layers, user interface components, and API endpoints.

Testing and quality assurance are very mature application fields where automated agents run enormous numbers of test cases, examine results by comparing with anticipated output, and modify testing techniques according to experience gained from previous test runs. Detailed research has reported defect detection rate increases of about thirty to seventy percent over manual testing techniques, with increases being highest for regression testing and boundary condition verification.

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher

Development Phase	Agent Specializations	Transformation Outcomes
Requirements Engineering	Requirements gathering, collaborative filtering, negotiation agents	Defect reduction, conflict resolution, stakeholder consensus building
Architecture and Design	Architecture synthesis, pattern-matching, constraint verification	Design space exploration, technical debt reduction, quality optimization
Code Implementation	Code generation, code review, refactoring, integration management	Productivity enhancement, defect detection, modularity improvement
Testing and Quality Assurance	Test generation, test execution, test oracle, fault localization	Defect detection improvement, coverage expansion, execution acceleration
Deployment and Operations	Continuous integration, deployment orchestration, monitoring, self-healing	Deployment frequency increase, recovery time reduction, failure rate decrease

Figure 3: Software Development Lifecycle Applications (Sonkar, S. 2025; Sonkar, S. 2025).

Challenges, Open Issues, and Future Research Directions

In spite of major advances in multi-agent orchestration for software development, there are still a lot of technical, organizational, and research challenges that need to be tackled before this technology can find broad industrial usage. Scalability issues come into play when agents increases in number and coordination mechanisms become more complex in large-scale development settings that comprise hundreds or thousands of autonomous agents that run simultaneously over a distributed infrastructure. Coordination overhead can increase super-linearly with the number of agents since each agent will have to communicate with numerous other agents via message passing protocols, keep shared state distributed over network nodes and data stores consistent, and engage in group decision-making processes that involve consensus algorithms or voting protocols to achieve agreement among distributed parties (Sonkar, S. 2025). It has been found in research that coordination costs can account for as much as thirty to fifty percent of the computational workloads in systems with over a hundred agents when employing simple coordination protocols that utilize all-to-all communication patterns that lack hierarchical structure, regional clustering, or locality-aware message routing optimizations.

Interoperability issues make it difficult to integrate multi-agent orchestration systems with the installed base of development tools, platforms, and workflows already established in software organizations that have heavily invested in their current tool chains. Software development organizations have generally used heterogeneous tool chains across version control systems, issue trackers, continuous integration servers, code review platforms, and communication tools. Facilitating agents to communicate with such various systems in an integrated manner needs a huge integration effort in the form of API mapping, protocol conversion, and data format transformation across various system interfaces, adapter creation, making bridges between agent communication languages and tool-specific interfaces, and constant maintenance as tools change over time through version upgrades and feature enhancements (Sonkar, S. 2025). Standardization initiatives to establish standard APIs, data formats, and communication protocols for development tools would simplify integration by offering standardized interfaces, but broad industry-wide adoption of such standards is still difficult because of competitive issues, legacy system limitations, and variable organizational needs (Hill, B. A. *et al.*, 2025).

Trust and transparency issues impinge on the uptake of autonomous agent systems within

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher

software development, especially where agents are used to make important decisions that have implications for code quality indicators, system security stance, or project schedules and resource allocations. Programmers can be unwilling to accept recommendations from agents if they do not see the rationale behind the recommendations, particularly where recommendations disagree with their own professional expertise or existing organizational traditions. Studies have found that trust from developers in autonomous agents is boosted by about thirty to fifty percent when agents give rich explanations for their suggestions, versus black-box systems that do not even offer insight into decision-making. Security vulnerabilities are risky, as hacked agents can inject malicious code, manipulate development pipelines, or exfiltrate valuable intellectual property. Directions for future research include next-generation learning algorithms that allow agents to learn development skills by watching and analyzing historical data, human-agent collaboration paradigms that maximize the division of labor between automated and manual tasks, and next-generation technologies such as quantum computing to optimize hard optimization problems and edge computing architectures for lower latency in distributed, geographically diverse development teams.

Challenge Domain	Critical Issues	Potential Solutions
Scalability	Coordination overhead growth, resource consumption, communication complexity	Hierarchical decomposition, locality-aware coordination, optimized routing
Interoperability	Heterogeneous tool chains, integration effort, adapter maintenance	Standardized APIs, common protocols, uniform interfaces
Trust and Transparency	Developer reluctance, recommendation acceptance, reasoning opacity	Explainable AI techniques, detailed explanations, audit mechanisms
Security	Agent compromise, malicious code injection, intellectual property theft	Authentication systems, communication encryption, sandboxing techniques
Future Technologies	Learning algorithm advancement, collaboration optimization, paradigm shifts	Meta-learning, transfer learning, quantum computing, edge computing

Figure 4: Challenges and Future Research Directions (Sonkar, S. 2025; Sonkar, S. 2025)

CONCLUSION

Multi-agent orchestration has become a potent paradigm revolutionizing modern-day software development practices with accelerated automation, greater coordination, and adaptive decision-making features. Autonomous agents have shown to be very helpful throughout the development process - from requirements elicitation to deployment and maintenance. They have several benefits, including a shorter development cycle, improved code quality, better testing coverage, and quicker deployment cycles. Architectural patterns supporting hierarchical, peer-to-peer, and hybrid structures provide flexible solutions that are flexible in many organizational contexts, as well as across many project sizes. Communication and coordination infrastructures, such as standardized agent communication languages, contract-net protocols, commitment-based approaches, and consensus algorithms, support rich collaborative activities that reflect and enhance human development team dynamics. Still, actualizing the full potential of multi-agent orchestration involves solving key challenges such as scalability issues with increasing agent populations, interoperability issues across heterogeneous development tool stacks, trust and transparency issues impacting developer adoption of agent suggestions, security threats exposing the system to malicious meddling, and ethical issues

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher

regarding responsibility assignment when agents make consequential actions. Future breakthroughs in machine learning algorithms, human-agent collaboration frameworks, and new computing paradigms hold the potential for vastly improved agent capabilities. The effective incorporation of multi-agent orchestration into mainstream software development methodologies is contingent upon further innovation in coordination protocols, standardization of integration interfaces, creation of explainable artificial intelligence methods for establishing developer trust, deployment of solid security mechanisms, and the creation of proper governance frameworks for addressing ethical issues. Organizations adopting multi-agent orchestration must carefully balance automation benefits against integration costs, coordination overhead, and the ongoing need for human expertise in creative problem-solving, domain knowledge application, and ethical judgment.

REFERENCES

1. Kumar, R. "Multi-Agent Testing Systems: How Cooperative AI Agents Validate Complex Applications," *VirtuosoQA*, (2025).
2. Sarangi, L. & Panda, C. "A Review on Intelligent Agent Systems," *ResearchGate*, (2011).
3. Mahmud, S., Goldfajn, D. B., & Zilberstein, S. "Distributed Multi-Agent Coordination Using Multi-Modal Foundation Models." *arXiv preprint arXiv:2501.14189* (2025).
4. Tricon Infotech, "Scaling Multi-Agent Architectures for Enterprise Use Cases," *Tricon Infotech*, (2024).
5. Sonkar, S. "Fine-tuning AI Models for code generation: Advances and applications," *World Journal of Advanced Research and Reviews*, (2025).
6. Siddhant, S. "Recent Innovations in AI Privacy: Protecting Data in the Age of Machine Learning." *INTERNATIONAL JOURNAL* 11.2 (2025): 611-619.
7. Sonkar, S. "Reducing Seller Friction through Generative AI: An Analysis of E-commerce Innovation." *IJSAT-International Journal on Science and Technology* 16.1 (2025).
8. Sonkar, S. "Boosting Developer Efficiency with AI: A Technical Review." *Journal Of Engineering And Computer Sciences* 4.9 (2025): 500-508.
9. Sonkar, S. "Fine-tuning AI Models for code generation: Advances and applications," *World Journal of Advanced Research and Reviews*, (2025).
10. Sonkar, S. "Transfer Learning: Leveraging Pretrained Models For Limited Datasets Through Fine-Tuning," *International Journal of Information Technology and Management Information Systems (IJITMIS)*, (2025).
11. Sun, L., Yang, Y., Duan, Q., Shi, Y., Lyu, C., Chang, Y. C., ... & Shen, Y. "Multi-agent coordination across diverse applications: A survey." *arXiv preprint arXiv:2502.14743* (2025).
12. Hill, B. A., Wei, M. K. E., & Jishnuanandh, T. "Communicating Plans, Not Percepts: Scalable Multi-Agent Coordination with Embodied World Models." *arXiv preprint arXiv:2508.02912* (2025).
13. Gaur, S., Sonkar, S., & Roy, P. P. "Generation of synthetic training data for handwritten Indic script recognition." *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, (2015).

Source of support: Nil; **Conflict of interest:** Nil.

Cite this article as:
Sonkar, S. "Multi-agent Orchestration for Software Development: A Comprehensive Review." *Sarcouncil Journal of Engineering and Computer Sciences* 4.11 (2025): pp 190-197.

Copyright © 2022 The Author(s): This work is licensed under a Creative Commons Attribution- NonCommercial-NoDerivatives 4.0 (CC BY-NC-ND 4.0) International License

Publisher: SARC Publisher