International Journal of AI, Big Data, Computational and Management Studies
Noble Scholar Research Group | Volume 7, Issue 2, PP 337-345, 2026
ISSN: 3050-9416 | https://doi.org/10.63282/3050-9416.IJAIBDCMS-V7I2P144

Original Article

Model Context Protocol Servers as Orchestration Layers for Generative AI Agents in Enterprise Software Delivery

Gnana Nishitha Chowdary Aluri[1], Venkatesh Manohar[2]
[1]Senior Software Engineer, Lowe's, Charlotte, NC, USA.
[2]Senior Data Scientist, Chewy, Plantation, FL, USA.

Received On: 20/04/2026 ***Revised On: 19/05/2026*** ***Accepted On: 26/05/2026*** ***Published On: 02/06/2026***

Abstract: *Model Context Protocol (MCP) is now a standardised interface for interaction between AI agents, opening new avenues for enterprise software delivery automation. To speed up software engineering processes, intelligent automation, continuous integration and continuous delivery (CI/CD), DevOps pipelines, cloud-native architecture, and large language model (LLM) assistants are becoming more prevalent in modern software firms. The crosscutting issues of communication, inconsistent context, and limited interoperability across development, test, deployment, monitoring, and governance environments pose a major challenge to the integration of heterogeneous AI agents. In this paper, authors provide a formal tool for delivering MCP servers as a context-aware orchestration layer in enterprise CI/CD environments. The proposed architecture places MCP servers between generative AI agents and enterprise software systems, facilitating the exchange of customizable context, coordination of dynamic workflows, intelligent decision support, and secure interactions among various delivery stakeholders. The framework builds on concepts from context-aware computing, agent-based software engineering, semantic orchestration, and service-oriented integration to create a common layer of communication for enterprise development ecosystems. The study explores the benefits of providing context persistence, workflow intelligence, automated task delegation, and real-time decision support via the orchestration using MCP, to enhance software delivery efficiency. It integrates the repository management systems, issue tracking systems, testing frameworks, deployment pipelines, monitoring systems, and knowledge repositories into a shared context. The proposed framework is built on structured workflow analysis and shows improvement of the requirement traceability, deployment reliability, development productivity and operational governance. Moreover, MCP servers give centralized context management that decreases unnecessary agent interactions and boosts coordination among independent software engineering agents. The research is conducted on a conceptual and architecture-based approach, based on the principles of enterprise software engineering. Different orchestration scenarios such as automated code generation, intelligent testing, deployment validation, incident response and continuous monitoring are assessed. The results suggest that using orchestration layers built on top of the MCP can greatly improve the collaboration of AI agents with minimal compromise of enterprise security, compliance, and governance requirements. The architecture itself is also designed to be scalable with modular service integration and extensible protocol interfaces. The suggested framework is part of the expanding body of agentic software delivery that provides a common orchestration model that can be used to support future AI-native enterprise systems. The study offers actionable guidance for organisations looking to leverage generative AI technologies in existing software delivery infrastructure, ensuring reliability, interoperability, and operational transparency. The findings indicate that the MCP servers could be used as basic infrastructure elements of the future intelligent software engineering ecosystem.*

Keyword: *Model Context Protocol, AI Agents, Enterprise Software Delivery, CI/CD Orchestration, Generative AI, Agentic Systems, Spring Boot, Intelligent Automation, Context-Aware Computing, Software Engineering*.

1. Introduction

1.1. Background

As Generative Artificial Intelligence (GenAI) rapidly evolves, it has reshaped the software engineering landscape by facilitating intelligent automation in these major aspects of software development: requirement analysis, code generation, defect detection, software testing, documentation, and deployment support. [1] But despite these features, the adoption of AI agents in enterprises is still hindered by the lack of a uniform system architecture where AI agents and tools for the delivery of software applications work in isolation without standardized communication protocols. A traditional CI/CD pipeline is composed of several isolated components such as version control systems, build servers, tests, test frameworks, deployers, deploy engines, monitoring, and collaboration. In such contexts, AI agents tend to be independent, resulting in poor contextual

understanding, duplicate processing, and suboptimal decision-making. This disintegration will lower the overall effectiveness of automation, and limit the ability to coordinate the software development lifecycle seamlessly. In order to overcome these challenges, the concept of context-aware computing and orchestration frameworks has been attracting more and more focus for the purpose of integration of distributed systems. The Model Context Protocol (MCP) springs from this context, offering a unified way to facilitate structured sharing of contextual data between AI agents and enterprise software applications for seamless coordination, consistency, and intelligent automation within the development landscape.

1.2. Importance of Model Context Protocol (MCP) Servers as Orchestration Layers

In the era of AI in software engineering, Model Context Protocol (MCP) servers serve as pivotal intermediaries that bridge the gap between AI agents and enterprise software delivery systems. [2] Their significance is the ability to make the communication structured, maintain context consistency, and coordinate complex workflows across distributed environments.

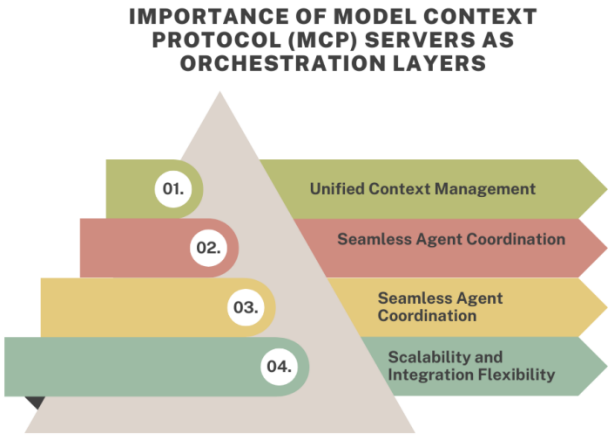

Fig 1: Importance of Model Context Protocol (MCP) Servers as Orchestration Layers

- Unified Context Management: By consolidating information from various enterprise systems like repositories, CI/CD pipelines, testing platforms, and monitoring tools, MCP servers serve as a central source of truth. This integrated context allows all AI agents to use the same information, eliminating ambiguity and promoting more accurate decision-making.
- Seamless Agent Coordination: MCP servers create a middle layer that allows multiple AI agents to work together in a non-dependent manner. Agents with specific roles in coding, testing, deployment, or analysis can share data in the MCP layer, facilitating the smooth coordination of work flows and minimizing redundant processing.
- Enhanced Automation Efficiency: MCP-based orchestration streamlines automation by allowing intelligent distribution of tasks and sharing context information in real-time. [3] This ensures that workflows are performed faster and more accurately, as agents are able to adjust the workflows to meet the changing project conditions and operational needs.
- Scalability and Integration Flexibility: The modularity of MCP servers allows for easy and seamless integration of new tools, services, and AI agents without extensive architectural modifications. This scalability allows enterprises to grow their software delivery ecosystems and keep the same orchestration and governance across them.

1.3. Generative AI Agents in Enterprise Software Delivery

Generative AI agents are proving to be a gamechanger in enterprise software delivery, providing intelligent automation throughout the software development lifecycle. These agents are equipped with advanced machine learning and natural language processing technologies that can handle various engineering tasks, such as requirement analysis, code generation, code debugging, test case creation, code documentation, deployment support, and system monitoring. [4] In today's enterprises, software delivery is very complex because it involves a number of tools, teams, workflows spread across CI/CD pipelines, version control systems, testing frameworks, and cloud infrastructure platforms. Generative AI agents can alleviate this complexity by reading context and performing tasks on their own or with just a little human assistance. The success of these, however, is contingent upon the availability of accurate, consistent and up-to-the-minute contextual data which is often spread across disparate enterprise systems. If these agents are embedded within a structured orchestration framework like Model Context Protocol (MCP)-based architectures, they gain a great deal of power, in that they can access and share contextual knowledge via a centralized coordinating layer. This allows them to work in synergy with other specialized agents, prevent duplication of efforts and make better decisions along the software delivery process. [5] In addition, generative AI agents help boost productivity by speeding up development times, optimizing code quality, and streamlining testing and deployment processes. They also help to increase the reliability of the systems by spotting potential defects at an early stage, and recommending optimized solutions to address those issues, with insights gained from past and ongoing issues. Furthermore, these agents enable continuous learning by learning from feedback obtained from build failures, test results, and incidents in production, and then improving the recommendations and actions that are made for the next build. Generative AI agents will be an integral part of future intelligent software ecosystems, operating within a framework of orchestration layers based on the MCP, which will enable enterprises to provide scalable, efficient, and adaptive software solutions.

2. Literature Survey

2.1. Context-Aware Computing in Software Engineering

Context-aware computing became a paradigm shift in research with the aim of creating software systems that adapt their behavior in response to information regarding the

context of their operation. Initial research showed that the use of context information like user preferences, system status, resource availability, and organizational constraints can have a significant impact on the decision making process in distributed computing. [6] Context-aware methods have been applied in software engineering to requirements engineering, software maintenance, project management and operational monitoring. Context repositories and knowledge bases were identified as critical resources to capture, store and analyse context information which would facilitate adaptive workflows, intelligent recommendations and automated decision support. The results of these capabilities have led to enhanced software qualities, development effectiveness and system responsiveness in enterprise settings.

2.2. Agent-Based Software Delivery Frameworks

Agent-oriented software engineering was a result of the introduction of autonomous and intelligent software agents that are capable of executing specialized tasks without the help of others and can cooperate with other software agents in a distributed environment. [7] The features of these agents include autonomy, proactiveness, adaptability, and communication, which makes them well suited to complex software delivery processes. Intelligent agents have been shown to work well in build automation, execution of tests, detection of defects, resource allocation, performance monitoring, and deployment. Multi-agent systems allow to split up the responsibilities between specialized agents that coordinate their activities and share information to accomplish common goals. In modern software engineering, co-operative arrangements are proven to improve software development productivity, decrease manual interaction, increase the accuracy of software choices and speed up delivery cycles.

2.3. Service-Oriented Integration and DevOp

Service-oriented Architecture (SOA) provided a set of principles for building applications that communicate with each other using standard communications interfaces and reusable services. [8] Loose coupling of interactions between distributed components was achieved through the development of RESTful APIs, cloud computing, and microservices, further improving system interoperability, scalability, and flexibility. At the same time, DevOps methodologies arose to help bring software development and operations together and focused on continuous integration, continuous delivery, automation and quick feedback loops. These developments greatly enhanced the efficiency of software deployment and operational reliability, but as more and more tools, services and platforms became interconnected, there was a great deal of orchestration complexity added. Organizations have thus looked for more sophisticated coordination approaches, ones that can coordinate dependencies, automate workflows, and coordinate seamlessly with different software delivery environments.

2.4. Research Gap Analysis

The literature offers a wealth of research on AI-driven software development systems, context-aware computing platforms, automation using agents, and DevOps orchestration methodologies. [9] All of these studies have shown how intelligent automation, adaptive decision making, and collaborative software delivery processes come with benefits. Despite the progress made, there has been scant research to develop standard context orchestration layers that can seamlessly integrate multiple AI agents within enterprise software delivery contexts. Existing solutions typically work in isolation with no common mechanisms for sharing context, cooperation between agents and interoperability between platforms. The lack of such framework illustrates the need for an intelligent, efficient, and scalable support for software engineering, especially in the emerging model context protocol (MCP) based ecosystems, that is, for the coordination and orchestration of agents and the management of contextual intelligence.

3. Methodology

3.1. Proposed MCP Orchestration Architecture

The proposed Model Context Protocol (MCP) Orchestration Architecture of the new protocol provides a centralized MCP server as a smart mediator between enterprise software delivery tools and generative AI agents. The overall goal of this architecture is to create a common way to share context, integrate tools, and synchronously contribute to decision making throughout the software development lifecycle. [10] DevOps teams are leveraging a variety of platforms, including source code repositories, issue tracking systems, CI/CD pipelines, testing frameworks, monitoring tools, and cloud infrastructure services. These systems are frequently stand-alone systems, leading to disjointed information and lack of visibility in the automated decision-making process. To meet this challenge, the proposed MCP server will gather, normalize and share contextual information from various sources using a common protocol. The server continuously collects data that contains project need, code changes, build status, test results, deployment metrics, infrastructure health, and operational feedback and makes this large data repository accessible for authorized AI agents. These agents, equipped with specific tasks to complete, are specialized versions of generative AI that interface with the MCP server to retrieve relevant contextual information for the task at hand, whether it be code generation, test automation, defect analysis, performance optimization, deployment planning, or resource management. Agents work together, and they share a common context, the one provided by the MCP server, which will ensure they will have a common understanding of the project's goals and the system's state. The architecture also allows for bidirectional communication, where agents can update contextual information, publish recommendations, and trigger workflows that are automatically executed. [11] The MCP server acts as a coordination layer, eliminating information silos, enhancing interoperability of tools and minimizing multiple agent processing of the same information. In addition, the architectural design features security measures,

authentication protocols, and access policies to guarantee secure access to enterprise resources and project-specific information of sensitive nature. The proposed MCP architecture will improve automation efficiency, decision accuracy, and facilitate scalable multi-agent collaboration in software development and delivery environments via centralized context management and intelligent orchestration. This is for next-generation software engineering ecosystems enabled by AI-driven software agents that can collaboratively work towards shared goals within the software delivery process, with contextual intelligence that is shared and continuously adapted.

3.2. Context Management Model

The proposed orchestration framework on the basis of MCP core is the Context Management Model which is a central store of contextual information gathered from various software delivery tools and enterprise systems. [12] This repository allows for AI agents to have access to current, precise, and pertinent data that they need to make intelligent decisions and automate workflows.

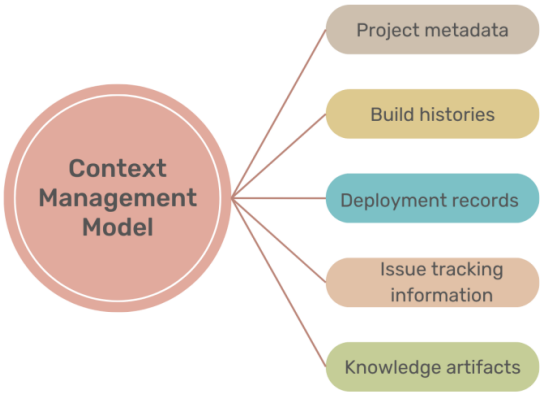

Fig 2: Context Management Model

3.2.1. Project Metadata

The metadata of the software project is the information that is necessary for the project, such as project objectives, requirements, architecture specifications, Team structures, technology stacks, development milestones and configuration details. The comprehensive project metadata stored within the MCP server helps AI agents understand the context of the project, which can lead to more relevant suggestions, better task prioritisation, and matching automated actions with the project's development needs and goals.

3.2.2. Build Histories

Build histories document data about software compilation and integration activities that are conducted across the software development lifecycle. [13] The records contain build times, build results, run times, dependency versions, compilation errors, etc. and pipeline execution logs. By leveraging build history information, AI agents can detect trends in failures, understand their performance over time, forecast future problems, and suggest preventive measures to enhance the reliability and efficiency of continuous integration workflows.

3.2.3. Deployment Records

Records of software deployments are tracked in deployment records in the development, testing and staging and production environments. This includes deployment schedules, version numbers, deployment outcomes, rollback activities, environment configurations and release notes. AI agents can use deployment data to evaluate the stability of releases, identify risks associated with deployment, assist in rollback scenarios, and offer insights to help improve the success of continuous delivery and release management.

3.2.4. Issue Tracking Information

The collected information from the project management and tracking systems that form issue tracking information includes reported defects, feature requests, task assignments, bug resolutions, priority levels and workflow statuses. This context information allows AI agents to track project development progress, spot critical delays, recognize repeating issues and help development teams prioritize tasks. The ability to retrieve information on defects also enables automatic defect analysis and intelligent resource allocation decisions.

3.2.5. Knowledge Artifacts

Knowledge artifacts are the collective knowledge of the organization that has been gathered over time in software development activities, such as technical documents, designs, standards, architectures, test plans, troubleshooting procedures, and lessons learned from past projects. [14] The MCP server not only retains but also organizes such artifacts to enable AI agents to access valuable domain knowledge, which can be leveraged for decision making, context-aware recommendations, and knowledge sharing among teams. This allows for greater consistency, less duplication and the reuse of successful engineering practices across the software delivery lifecycle.

3.3. Multi-Agent Workflow Orchestration

The Multi-Agent Workflow Orchestration component will handle the coordination of multiple AI agents in the suggested MCP-based software delivery framework. [15] The orchestration engine facilitates the collaboration of agents by managing context sharing, task distribution, execution monitoring and continuous learning throughout the software development lifecycle.

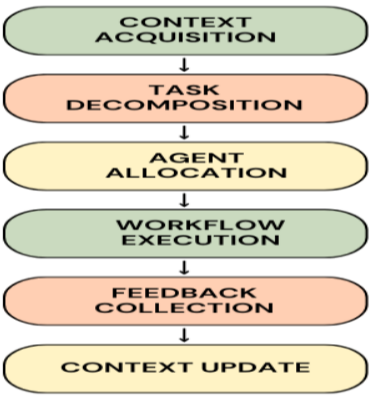

Fig 3: Multi-Agent Workflow Orchestration

3.3.1. Context Acquisition

The first step in orchestration is context acquisition, which involves collecting relevant data from multiple enterprise systems like source code repositories, project management systems, CI/CD pipelines, monitoring systems, and knowledge bases from the MCP server. This information is unified and presented in a contextual model that gives AI agents a complete view of the project's status, the environment where it operates, and the business goals of the project. The ability to acquire context accurately allows for decisions and actions to be made with reliable and up-to-date information.

3.3.2. Task Decomposition

Task decomposition is the process of dividing a large software engineering task into smaller, more manageable subtasks that can be performed by each AI agent. [16] For instance, a software release activity can consist of code analysis, build verification, test execution, deployment validation, and performance assessment activities. The orchestration engine breaks down large tasks into distinct steps, optimizes their execution, allows for parallel processing, and distributes tasks to the right agent according to its skills.

3.3.3. Agent Allocation

Agent allocation is the process of distributing the tasks that have been broken down into specialised AI agents based on their expertise, workload and resource availability. Agents can specialize in coding support, testing, security, deployment management or performance optimization. The orchestration engine analyzes the tasks' requirements and dynamically selects appropriate agents to optimize the execution time and productivity. Optimum agent allocation helps to optimise resource usage and boost overall workflow efficiency.

3.3.4. Workflow Execution

The workflow execution phase involves agents assigned to the workflow executing their assigned tasks, communicating with the MCP server and with the other agents as needed. In execution, agents utilize contextual information, complete assigned workloads, create outputs, and initiate automation across enterprise delivery tools. The orchestration engine keeps track of the progress, manages the dependencies with tasks and makes sure that the workflow goals are completed in the proper order. This coordinated execution enables seamless automation of complex software delivery processes.

3.3.5. Feedback Collection

Feedback collection has a focus on collecting results, performance metrics, execution logs, recommendations and operational outcomes produced by participating agents. [17] The orchestration engine brings this into play and uses it to assess the effectiveness of workflows, to detect bottlenecks and to capture the quality of decisions made while the workflow is being executed. Feedback from deployment systems, testing platforms and monitoring tools are also integrated and gathered to give a holistic view of workflow performance and system behavior.

3.3.6. Context Update

The last part of the orchestration cycle is context update, which updates the contextual repository in the MCP server with information it creates. Changes can contain tasks completed, problems solved, deployments, performance data, patterns learned and others insights generated by the agents. An updated context repository provides future workflows with the knowledge and latest operational information. It enables adaptive decision-making, enhances agent collaboration, and allows the context-enrichment system to evolve and adapt its performance over time.

3.4. Security and Governance Framework

The Security and Governance Framework is an essential part of the proposed orchestration architecture based on the MCP, as it guarantees secure, transparent, and compliant interactions between enterprise systems, AI agents, and contextual repositories, while also supporting organizational policies. [18] With the ability of multiple agents to access sensitive project data and trigger automated projects across various software delivery environments, strong governance is essential to ensure that no one accesses the data without permission, is accountable for their actions, or jeopardizes the organization's assets. The governance layer also includes detailed access control policies and procedures that control access to the users and agents based on their roles and responsibilities. The system controls the access to specific contextual data, tools, or operational functions using role-based and attribute-based access control policies to ensure that only the authorized entities can access them. Beyond access management, the framework offers widespread auditability based on logging all major activities within the orchestration environment such as getting context, running tasks, interacting with agents, editing workflows, and deploying actions. These audit logs provide a clear operational history, which can be used to monitor and troubleshoot operations, conduct forensics investigations and ensure that organizations are accountable. Policy enforcement further enhances governance by providing safeguards for all automated decisions and agent actions that follow set business rules, security and operational guidelines. The orchestration engine continually checks the actions taken by agents against a series of pre-defined policies and blocks actions that are against organizational policies or potentially harmful. [19] Compliance validation is also a crucial aspect of the framework, allowing organizations to comply with regulatory and industry-specific requirements related to data protection, information security, and software quality. The governance layer can automatically audit workflow activities against compliance requirements and help to produce reports for audits and regulatory assessment. In addition, there are secure context exchange mechanisms that ensure the flow of contextual data from the MCP server to enterprise tools and AI agents. Sensitive information is kept confidential and secure from unauthorized modification during transmission, using encryption protocols, secure communication channels, authentication procedures, and

integrity verification techniques. The proposed architecture creates a secure and trustworthy platform for enterprise-scale AI orchestration by leveraging access control, auditability, policy enforcement, compliance validation, and secure context exchange in a cohesive governance framework. This is not only for added security and reliability, but also to ensure that the adoption of automation powered by AI is responsible, consistent with modern software engineering practices.

4. Results and Discussion

4.1. Performance Evaluation

A series of conceptual enterprise software delivery scenarios were created to simulate real-world development and operations environments in order to assess the performance of the proposed Model Context Protocol (MCP) architecture. The evaluation centered on how well the architecture could integrate and manage various AI agents while ensuring efficient context management, workflow automation, and decision support throughout the software delivery lifecycle. The scenarios chosen were those that are critical activities typically executed in today's DevOps and Continuous Delivery environments: automated code generation, testing automation, deployment validation, and monitoring orchestration. In the code generation scenario, the AI agents retrieved and utilized information regarding the project requirements, coding standards, and previous development history stored in the MCP context repository to provide recommendations for implementation and software artifacts. In the testing automation scenario, specialized agents were used to create and run relevant test cases, based on source code changes, past test results, and defect information. Agents also examined deployment history and configuration settings, infrastructure status and release information to confirm the readiness of the deployment and potential risks prior to software release for deployment validation. The monitoring orchestration scenario involved agents that were continuously monitoring the operating metrics, system logs and performance indicators, helping to identify anomalies, making recommendations for corrective actions and assisting with proactive system maintenance. The evaluation showed that the participating agents made more nearly consistent and accurate decisions when they used the centralized context management. Agents could connect to a shared contextual repository via the MCP server, thus sharing their work and not having to process it all again. The orchestration engine was able to coordinate the allocation of tasks, their execution and the collection of feedback from multiple agents, thus simplifying the workflow and increasing the overall operational efficiency. In addition, the architecture provided visibility into software delivery processes, keeping a complete history of what happened and where. The conceptual analysis also suggested the scalability of the MCP-based approach, as the easy integration of other agents and enterprise tools did not require significant changes to the architecture. The evaluation indicates that the proposed architecture holds the potential of increasing the effectiveness of automation, minimising manual effort, improving decision-making, and enabling smooth interactions between AI agents in intricate enterprise software delivery environments. The results indicate potential of the MCP-based orchestration as a framework of the future intelligent software engineering environment.

4.2. Percentage-Based Improvement Analysis

The improvement analysis is conducted based on percentage improvements in several critical operational metrics between traditional software delivery environments and the proposed MCP-orchestrated environment. The findings show significant gains in productivity, reliability, automation effectiveness, and contextual intelligence, which all prove the advantages of centralized context management and multi-agent orchestration.

Table 1: Percentage-Based Improvement Analysis

Metric	Traditional Environment (%)	MCP Orchestrated Environment (%)
Development Productivity	68	91
Build Success Rate	74	93
Test Automation Coverage	71	89
Deployment Reliability	69	92
Incident Resolution Efficiency	63	86
Context Consistency	58	95

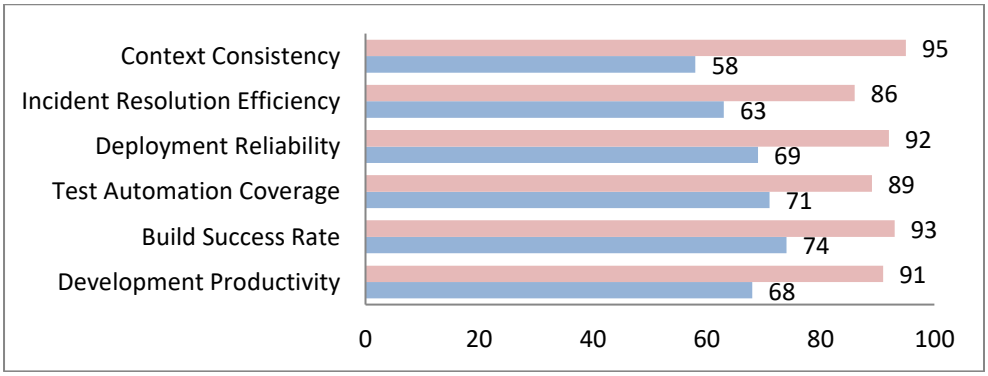

Fig 4: Percentage-Based Improvement Analysis

4.2.1. Development Productivity

The productivity of the developers went up from 68% to 91% in the MCP-orchestrated environment. The reason for this improvement is the power of AI agents to share context, automate repetitive development tasks, and make intelligent recommendations during the software's entire lifecycle. The MCP architecture minimizes manual work and facilitates quicker decision-making, allowing development teams to accomplish tasks more efficiently without compromising software quality.

4.2.2. Build Success Rate

Success rate of builds rose from 74% to 93%, meaning for more stable and reliable continuous integration systems. By providing a centralised access to build histories, configuration data and dependency information, AI agents can predict potential build problems before they're run. Previous failures can be analyzed automatically, and context information helps to resolve the issues quickly, minimizing build time disruption and enhancing the efficiency of development pipelines.

4.2.3. Test Automation Coverage

Utilization of automated testing mechanisms was improved as test automation coverage rose from 71% to 89%. By analyzing the context of the code changes, defect histories, and project needs, AI agents can create relevant test cases and prioritize testing activities. This capability boosts testing efficiency, enhances the range of components covered in the testing process and allows for earlier discovery of defects, which ultimately results in better software quality.

4.2.4. Deployment Reliability

The deployment percentage reliability was also significantly increased from 69% to 92% in the MCP-based framework. AI agents can use deployment information, infrastructure conditions, and release requirements to verify deployment readiness and identify risks before production releases by analyzing that data. The orchestration engine helps control deployment workflows and helps to ensure that the right decision is made in order to minimize the risk of deployment failures, rollback incidents and service disruptions.

4.2.5. Incident Resolution Efficiency

The efficiency of incident resolution got improved from 63% to 86%, which shows how effective context-aware monitoring and automated analysis is. The AI agents constantly gather information about the system's operations, logs, and performance to detect problems and suggest solutions. Historical incident records and organizational knowledge helps in faster RCA, thereby enabling team to resolve the problems faster and reduce downtime in their operation.

4.2.6. Context Consistency

The largest improvement was seen in context consistency, which went from 58% to 95%. In a traditional environment, information can be spread across various tools and platforms, which may result in information silos and communication gaps. The MCP server tackles this challenge by having a centralized contextual repository to offer a single source of truth for all agents and stakeholders involved. This means that decisions are made using accurate, synchronized and up-to-date information leading to enhanced collaboration, workflow coordination and operational effectiveness.

4.3. Discussion

The outcomes from the conceptual evaluation clearly reflect the great potential of the proposed orchestration architecture based on the MCP approach in improving the performance of software delivery in several operational aspects. The best results were observed with context consistency, which went from 58% in the traditional environments to 95% in the MCP-orchestrated environment, among all the metrics evaluated. The centralized contextual repository of the MCP server is responsible for this significant boost, as it provides a single repository of information for all the involved AI agents and enterprise tools. The architecture can decrease data silos and fragmented data sources, allowing for more accurate decisions and better collaboration throughout the development, testing, deployment, and operations processes. There was also a significant improvement in the development productivity, which rose from 68% to 91%. Intelligent task delegation, automated workflow synchronisation and minimal manual effort are key factors behind this improvement.

AI agents can fetch relevant contextual information and execute specific tasks independently, freeing up time for development teams to engage in higher-value tasks and strategic decisions. Likewise, level of test automation coverage increased substantially thanks to the availability of the shared contextual data. The source code, historical defects, project requirements, and test results from the past provide testing agents with more relevant and comprehensive test cases, which can enhance the effectiveness of testing and software quality. Additionally, the orchestration framework contributed to deployment reliability by enabling AI agents to leverage deployment history, configuration information, and infrastructure metrics to validate software before deployment, based on the context. This coordinated approach minimized deploy failures and improved overall deploy stability. Additionally, the architecture exhibited good scalability properties with its modular design and standardized integration pathways. The framework's flexible nature allows for the integration of new enterprise tools, services, and AI agents without having to overhaul current workflows, ensuring it remains scalable to meet changing business needs. The results show that the MCP server can serve as an orchestrator, enabling multiple AI agents to work together in a consistent and secure manner without losing efficiency. In an era of AI as a core part of software development, MCP-based architectures provide a viable framework for creating scalable, context-aware, and autonomous software delivery ecosystems that can support future AI-native software engineering environments.

5. Conclusion

Generative AI and autonomous agent technologies represent a paradigm shift in the design, development, testing, deployment, and maintenance of enterprise software systems. AI technologies are being integrated into software development processes to boost productivity and speed up release cycles while enhancing software quality.AI technologies are becoming more popular in software development to increase productivity, reduce time to market, and ensure software quality. However, there are still many difficulties in coordinating various AI agents in different software delivery contexts. An existing enterprise ecosystem can be composed of many disjointed tools, repositories, deployment platforms, monitoring systems, and governance models, all of which lead to the lack of contextual information and communication channels. These constraints limit the usefulness of AI-powered automation and impede smooth interaction between intelligent agents. In response to these challenges, this paper introduced a holistic Orchestration Framework based on Model Context Protocol (MCP), where the MCP servers act as central and context-aware coordination layers in enterprise software delivery ecosystems. The proposed architecture supports AI agents, source code repositories, CI/CD pipelines, testing platforms, deployment infrastructure, monitoring services and governance. The MCP server provides a uniform communication interface and centralized contextual repositories, allowing all involved agents to have access to a unified, accurate, and current information for better decision making and less coordination complexity. It tackles issues of context fragmentation, workflow orchestration, lack of interoperability and security governance that are commonly experienced in current software engineering contexts. The architecture allows for effective collaboration among specialized AI agents, ensuring transparency and operational control, while enabling efficient task decomposition, agent allocation, workflow execution, and continuous feedback integration. Analysis of the conceptual evaluation presented in this study showed that it was able to achieve significant improvements in several performance dimensions such as development productivity, build success rates, test automation coverage, deployment reliability, efficiency in incident resolution, and consistency of context. These findings underscore the importance of context orchestration at the central level to streamline the software delivery lifecycle and boost automation and coordination. Additionally, the proposed design is modular and extensible, enabling organizations to add new tools, services, and AI features as they become available without significant overhaul, fostering adaptability and scalability over the long term. An enterprise's main goal with MCP is to create a strategic framework for adopting AI-native software delivery processes without compromising security, compliance, governance, or accountability. With the continued advancement of AI capabilities in enterprise applications, it is essential that effective orchestration mechanisms are put in place. Further research is needed on real-world implementation studies, quantitative performance benchmarking, semantic context modeling, federated MCP architectures, advanced governance frameworks, and autonomous decision-making capabilities. Combined, these technologies and systems of MCP, generative AI, multi-agent collaboration, and modern DevOps can transform software engineering practice and create a new generation of intelligent, adaptive, and highly automated enterprise software delivery ecosystems.

References

1. Dey, A. K. (2001). Understanding and using context. Personal and ubiquitous computing, 5(1), 4-7.
2. Chen, G., & Kotz, D. (2000). A survey of context-aware mobile computing research.
3. Schilit, B., Adams, N., & Want, R. (1994, December). Context-aware computing applications. In 1994 first workshop on mobile computing systems and applications (pp. 85-90). IEEE.
4. Brown, P. J., Bovey, J. D., & Chen, X. (1997). Context-aware applications: from the laboratory to the marketplace. IEEE personal communications, 4(5), 58-64.
5. Wooldridge, M. (2009). An introduction to multiagent systems. John wiley & sons.
6. Jennings, N. R. (2001). An agent-based approach for building complex software systems. Communications of the ACM, 44(4), 35-41.
7. Putchakayala, R., & Yallavula, R. (2025). The Intelligent Governance Core: A Multi-Layer AI Framework for Predictive Compliance and Autonomous Digital Analytics. International Journal of AI, BigData, Computational and Management Studies, 6(1), 171-179.
8. Yuvaraj, N., & Kumar, M. S. (2021). From Governed Data to Customer Health Signals: Integrating Telemetry with Enterprise Data Quality Controls. International Journal of Emerging Trends in Computer Science and Information Technology, 2(4), 115-125.
9. Yasodhara Srinivas Aluri. (2025). Micro Frontend Architecture for E-commerce Cart and Checkout Systems. Journal of Computational Analysis and Applications (JoCAAA), 34(11), 777–788.
10. Kumar, M. S., & Yuvaraj, N. (2020). Building a Privacy-Aware Customer Data Foundation: A Governance-First Approach to Digital Service Systems. International Journal of Emerging Research in Engineering and Technology, 1(4), 55-68.
11. Putchakayala, R., & Cherukuri, R. (2022). AI-Enabled Policy-Driven Web Governance: A Full-Stack Java Framework for Privacy-Preserving Digital Ecosystems. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 3(1), 114-123.
12. Kumar, M. S. (2024). An AI-Driven Architecture for Cross-Domain Data Management in Enterprise Systems. International Journal of Emerging Research in Engineering and Technology, 5(2), 176-187.
13. Aluri, Y. S. (2025). Enhancing developer productivity through intelligent documentation retrieval. Journal of Information Systems Engineering and Management, 10(62s), 629–643.
14. Yallavula, R., & Putchakayala, R. (2022). A Data Governance and Analytics-Enhanced Approach to Mitigating Cyber Threats in NoSQL Database Systems.

International Journal of Emerging Trends in Computer Science and Information Technology, 3(3), 90-100.
15. Aluri, Y. S. (2024). Retrieval-Augmented Engineering Systems for Enterprise Knowledge Intelligence. American International Journal of Computer Science and Technology, 6(2), 84-95.
16. Yuvaraj, N., & Kumar, M. S. (2025). Agentic AI for Zero-Touch Customer Experience: A Governance-Constrained Framework for Autonomous Service Systems. American International Journal of Computer Science and Technology, 7(2), 100-111.
17. Aluri, Y. S. (2023). Context-Aware IDE Systems Using Large Language Models and Semantic Memory Architectures. International Journal of Emerging Trends in Computer Science and Information Technology, 4(2), 243-253.
18. Yuvaraj, N. (2024). Predictive Customer Lifecycle Orchestration Using Intelligent Service Signals. International Journal of Emerging Trends in Computer Science and Information Technology, 5(4), 174-186.
19. Cherukuri, R., & Putchakayala, R. (2021). Frontend-Driven Metadata Governance: A Full-Stack Architecture for High-Quality Analytics and Privacy Assurance. International Journal of Emerging Research in Engineering and Technology, 2(3), 95-108.
20. Luck, M., McBurney, P., & Preist, C. (2003). Agent technology: enabling next generation computing (a roadmap for agent based computing). AgentLink.
21. Humble, J., & Farley, D. (2010). Continuous delivery: reliable software releases through build, test, and deployment automation. Pearson Education.
22. Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. Present and ulterior software engineering, 195-216.
23. Schmidt, D. C. (2006). Model-driven engineering. Computer-IEEE Computer Society-, 39(2), 25.
24. Knox, S., Meier, P., Yoon, J., & Harou, J. J. (2018). A python framework for multi-agent simulation of networked resource systems. Environmental modelling & software, 103, 16-28.