International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

AGENTIC AI-BASED AUTOMATED SOFTWARE DEVELOPMENT AND TESTING SYSTEM

[1] Siddamma CM, [2] Syed Ruhan, [3] K Gopi Raju, [4] R Nikhil Sai
[1]AssistantProfessor, [234]Students
Department of IOT
Siddhartha Institute of Technology & Sciences, Narapally
siddammacm@siddhartha.org.in, 23tq1a6902@siddhartha.co.in, 23tq1a6956@siddhartha.co.in, 23tq1a6936@siddhartha.co.in

Abstract

Software development is a complex process that involves requirement analysis, system design, coding, testing, debugging, documentation, and deployment. Traditional software development requires developers and testing teams to perform many repetitive activities manually, which can increase development time and effort. As software projects become larger and more complex, there is a growing need for intelligent tools that can assist developers throughout the software development lifecycle. The Agentic AI-Based Automated Software Development and Testing System is proposed as an intelligent platform that uses Agentic AI to automate and coordinate multiple stages of software development.

The proposed system uses an AI agent capable of understanding software requirements provided in natural language and converting them into structured development tasks. The agent can analyze requirements, generate suitable project structures, create source-code modules, and produce configuration files according to predefined project requirements. It can also use approved development tools, code repositories, testing frameworks, and development environments to perform multi-step software engineering tasks.

An important feature of the system is automated software testing. After generating or modifying code, the AI agent can automatically create test cases, execute unit and integration tests, analyze test results, identify potential defects, and suggest or apply controlled code corrections. The agent can repeat the development and testing cycle until predefined quality conditions are satisfied or the task requires human review. This creates a continuous workflow of requirement analysis → code generation → testing → debugging → verification.

The system can provide a centralized development dashboard containing project requirements, generated code, test cases, test results, detected issues, code-quality information, and development progress. Developers can review the changes produced by the AI agent and approve, modify, or reject them. High-impact operations such as production deployment, destructive database changes, or security-sensitive modifications can require explicit human approval.

I. Introduction

Software development involves a sequence of activities that transform user requirements into reliable and maintainable software applications. Developers must understand requirements, design system components, write code, perform testing,

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

identify defects, debug problems, document the application, and prepare it for deployment. Each stage requires technical knowledge and careful coordination. As projects increase in size, manually performing all these activities can require significant time and resources.

Traditional software development tools provide developers with environments for coding, debugging, version control, and testing. However, developers still need to manually interpret requirements, create implementation plans, write many parts of the code, design test cases, analyze failures, and determine suitable fixes. Automated testing frameworks can execute predefined tests, but they generally do not independently understand requirements or plan comprehensive development workflows.

Artificial Intelligence has introduced new capabilities for assisting software engineers. Large Language Models can understand natural-language requirements, generate code, explain existing programs, create test cases, and suggest fixes for common programming errors. AI-powered development assistants can therefore reduce the effort involved in several individual development activities. However, many conventional AI coding tools primarily respond to individual prompts and require developers to coordinate the overall workflow themselves.

Agentic AI provides a more autonomous approach by enabling AI systems to reason about objectives, create plans, use external tools, observe results, and determine subsequent actions. In software engineering, an AI agent can potentially analyze a requirement, divide it into tasks, generate code, run tests, inspect failures, modify the implementation, and repeat the process. Such autonomy must be controlled through permissions, validation, testing, version control, and human approval.

The proposed Agentic AI-Based Automated Software Development and Testing System integrates these capabilities into a unified development environment. Users provide project requirements in natural language, after which the agent analyzes the requirements and creates an implementation plan. The system can generate code, execute approved development tools, create and run tests, analyze failures, and suggest or perform controlled corrections.

The primary objective of the proposed system is to reduce repetitive software-development effort while improving testing coverage and development efficiency. Rather than replacing software engineers, the system acts as an AI development assistant capable of handling well-defined tasks and providing developers with faster feedback. Human developers remain responsible for reviewing important code changes, validating requirements, and approving sensitive or production-level operations.

II. Literature Survey

Traditional software-development environments have relied on integrated development environments, version-control systems, build tools, debugging utilities, and separate testing frameworks. Developers manually move between these tools

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

while implementing requirements and validating software. These tools provide strong support for individual development activities but generally do not independently coordinate the complete software-development lifecycle.

Automated software-testing frameworks have significantly improved the efficiency of testing activities. Unit-testing, integration-testing, regression-testing, and continuous-integration tools can automatically execute predefined test cases whenever code changes are introduced. However, conventional testing systems generally require developers or testers to manually create and maintain test cases and interpret failed tests before deciding how to correct the underlying implementation.

Machine-learning techniques have been applied to areas such as defect prediction, code classification, test-case prioritization, and anomaly detection. These approaches can identify patterns in historical software-engineering data and help teams focus on potentially problematic areas. However, many such systems provide predictions or recommendations rather than independently performing the complete development and debugging workflow.

Large Language Models have expanded AI capabilities in software engineering. Modern language models can generate code from natural-language descriptions, explain code, convert code between programming languages, create documentation, and generate test cases. AI coding assistants can therefore reduce the effort required for repetitive programming activities. Nevertheless, generated code can contain logical errors, security weaknesses, incorrect assumptions, or incomplete implementations, making testing and human review essential.

Agentic AI extends language-model capabilities by introducing planning, tool use, memory, observation, and iterative execution. An agent can divide a complex software task into smaller objectives, interact with development tools, inspect execution results, and adjust its actions based on feedback. In software development, this can enable workflows such as requirement analysis, code generation, automated testing, debugging, and documentation within a single process.

Recent research and practical systems increasingly explore AI-assisted software engineering, automated code repair, repository understanding, and AI-driven testing. However, reliable autonomous development requires strong safeguards because incorrect changes can introduce defects or security risks. The proposed system addresses these concerns through controlled tool access, automated testing, version-controlled changes, validation, audit logging, and human approval for sensitive operations.

III. System Analysis

The proposed system provides an intelligent platform for automating selected software-development and testing activities. The process begins when a developer submits software requirements through a web-based development interface. The requirement-analysis module interprets the description and identifies functional requirements, technical constraints, expected inputs, outputs, and development tasks.

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

The Agentic AI planner converts these requirements into an ordered implementation plan. A code-generation module creates project structures, source-code files, configuration files, and initial documentation according to the approved plan. The system can interact with authorized development tools, repositories, compilers, interpreters, and testing frameworks. An automated testing module generates suitable test cases and executes unit, integration, or other configured tests. Test results are analyzed by the AI agent to identify failures, inconsistencies, and potential defects. The debugging component determines possible causes and proposes or applies controlled code modifications. The updated code is tested again to verify whether the detected issue has been resolved. Version-control mechanisms maintain records of generated and modified files so changes can be reviewed or reverted. A developer dashboard displays requirements, code changes, test results, detected issues, and agent activities. Sensitive operations such as production deployment or destructive modifications can require explicit human approval. The system therefore creates an iterative development cycle while maintaining testing, traceability, and human oversight.

Existing System

Existing software-development environments generally require developers to manually interpret requirements, design solutions, write code, create tests, execute testing tools, analyze failures, and debug defects. Integrated Development Environments provide useful coding and debugging features, but developers remain responsible for coordinating most development activities. Automated testing frameworks can execute predefined test cases, but test creation and maintenance are often performed manually. Continuous Integration and Continuous Deployment tools automate build and deployment pipelines but typically follow predefined workflows rather than independently reasoning about software requirements. Conventional code-generation tools can generate code snippets based on prompts, but developers must integrate and validate the generated code. Static-analysis tools identify certain code-quality or security problems, but developers must determine appropriate fixes. Debugging often requires manual examination of logs, stack traces, and source code. Different development tools may operate independently, requiring developers to switch between multiple environments. Large projects can therefore involve significant repetitive work. Existing systems generally provide limited autonomous planning across the complete software-development lifecycle. They may also lack continuous reasoning between code generation, testing, failure analysis, and correction. These limitations create a need for an intelligent system capable of coordinating multiple development activities while keeping developers in control of important decisions.

Disadvantages of Existing System

- Heavy dependence on manual development activities.
- Requirement analysis often requires significant developer effort.
- Code generation and integration are frequently manual.
- Test-case creation can be time-consuming.
- Developers must manually analyze test failures.

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

- Debugging complex issues requires significant effort.
- Development tools may operate as separate systems.
- Limited autonomous task planning.
- Limited automated iteration between coding and testing.
- Repetitive development tasks consume valuable time.
- Manual documentation can be inconsistent.
- Difficult to scale development activities for large projects.
- Limited automated root-cause analysis.
- Deployment and high-impact operations require careful manual coordination.

Proposed System

The proposed system introduces an Agentic AI platform that can coordinate multiple stages of software development and testing. Developers provide project requirements through a natural-language interface, and the AI agent analyzes the requirements to identify tasks, dependencies, expected behavior, and technical constraints. An intelligent planning module creates a structured development workflow and determines the tools required for each task. The code-generation module creates project files, source-code components, configuration files, and initial documentation. The agent can interact with approved repositories, development environments, compilers, interpreters, and testing frameworks through controlled integrations. After generating code, the testing module automatically creates and executes appropriate test cases. Test results are analyzed to identify failed cases, potential defects, and unexpected behavior. The agent uses the available evidence to determine possible root causes and generate corrective changes. Modified code is automatically tested again to verify whether the correction improves the result. Version control maintains a history of AI-generated changes so developers can review or revert modifications. A dashboard provides visibility into requirements, development tasks, code changes, test coverage, failures, and agent activities. Sensitive operations can require explicit human approval before execution. The system therefore provides an iterative AI-assisted development workflow while preserving developer control and software-quality safeguards.

Advantages of Proposed System

- Automates repetitive software-development activities.
- Converts natural-language requirements into development tasks.
- Generates code and project structures automatically.
- Automatically creates relevant test cases.
- Executes unit and integration tests.
- Analyzes test failures using AI reasoning.
- Supports automated debugging and code correction.
- Enables iterative code-test-fix cycles.
- Reduces manual development effort.
- Provides centralized development monitoring.
- Maintains version-controlled AI-generated changes.
- Improves development feedback speed.
- Provides better traceability of development activities.

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

- Supports controlled tool execution.
- Allows human approval for sensitive operations.
- Can improve development productivity and scalability.

IV. Methodology

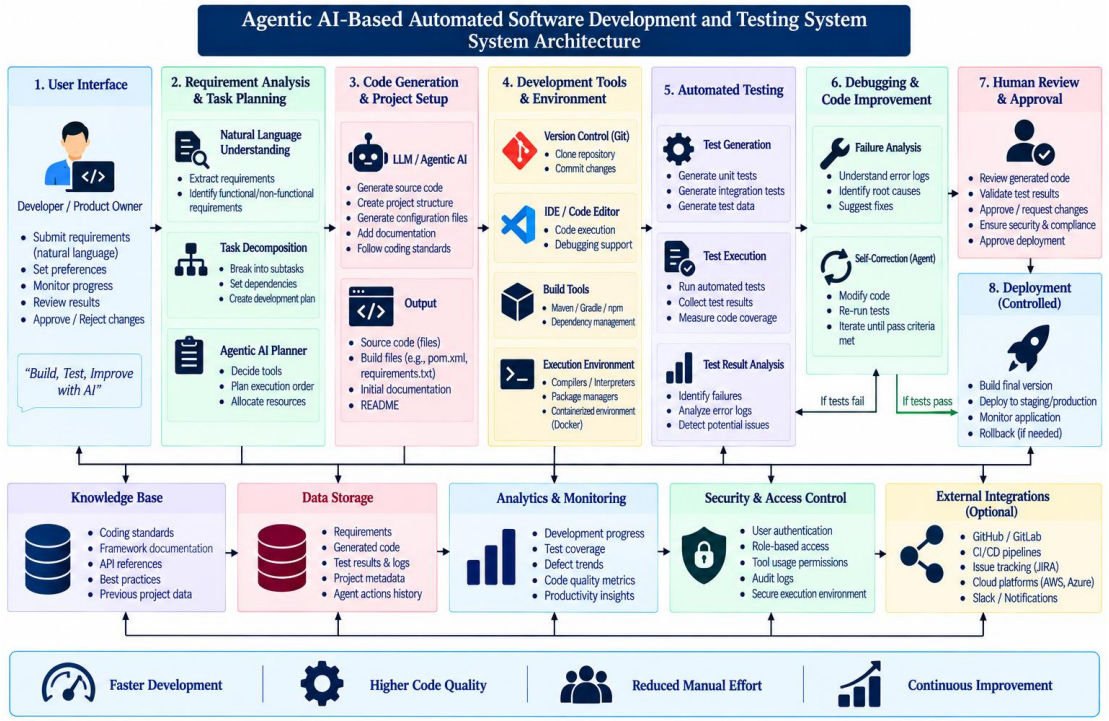
The methodology of the proposed system consists of multiple stages beginning with requirement submission and ending with validated software or human-assisted review. The first stage is **requirement collection**, where developers provide application requirements, expected features, constraints, and technical preferences through the system interface. The requirement-analysis module processes the input and converts it into structured functional and technical requirements. The Agentic AI planner then divides the requirements into smaller development tasks and establishes their execution order. The code-generation module creates the required project structure, source files, configuration files, and documentation. Generated code is checked using syntax validation and static-analysis techniques before testing. The automated testing module generates suitable test cases based on requirements and executes unit, integration, and other configured tests. Test results, logs, and error messages are provided to the AI reasoning engine for analysis. When failures occur, the debugging agent identifies possible causes and proposes controlled modifications to the affected code. Approved modifications are applied within a version-controlled development environment and the relevant tests are executed again. This iterative process continues until the configured quality criteria are achieved or the system determines that human intervention is required. The final software, test results, documentation, and change history are presented through the development dashboard for developer review. Finally, system performance can be evaluated using metrics such as test-pass rate, defect-detection rate, code-quality measures, task completion time, generated-code acceptance rate, and human-review effort.

System Architecture

The system architecture consists of several interconnected components including the Developer Interface, Requirement Analysis Module, Agentic AI Planner, Code Generation Engine, Development Tool Layer, Automated Testing Engine, Debugging and Code-Repair Module, Version Control System, Secure Database, Analytics Engine, and Development Dashboard. The process begins when developers submit software requirements through the web-based interface. The requirement-analysis module interprets the input and extracts functional requirements, constraints, expected behavior, and development tasks. The Agentic AI planner creates a structured implementation plan and determines the sequence of actions required to complete the project. The code-generation engine produces source code, project structures, configuration files, and documentation according to the approved plan. The development-tool layer provides controlled access to repositories, compilers, interpreters, package managers, development environments, and other authorized tools. Generated code is passed to the automated-testing engine, which creates and executes unit, integration, regression, or project-specific tests. Test results and execution logs are sent to the AI reasoning and debugging module for failure analysis. When appropriate, the agent proposes or applies controlled code corrections and sends

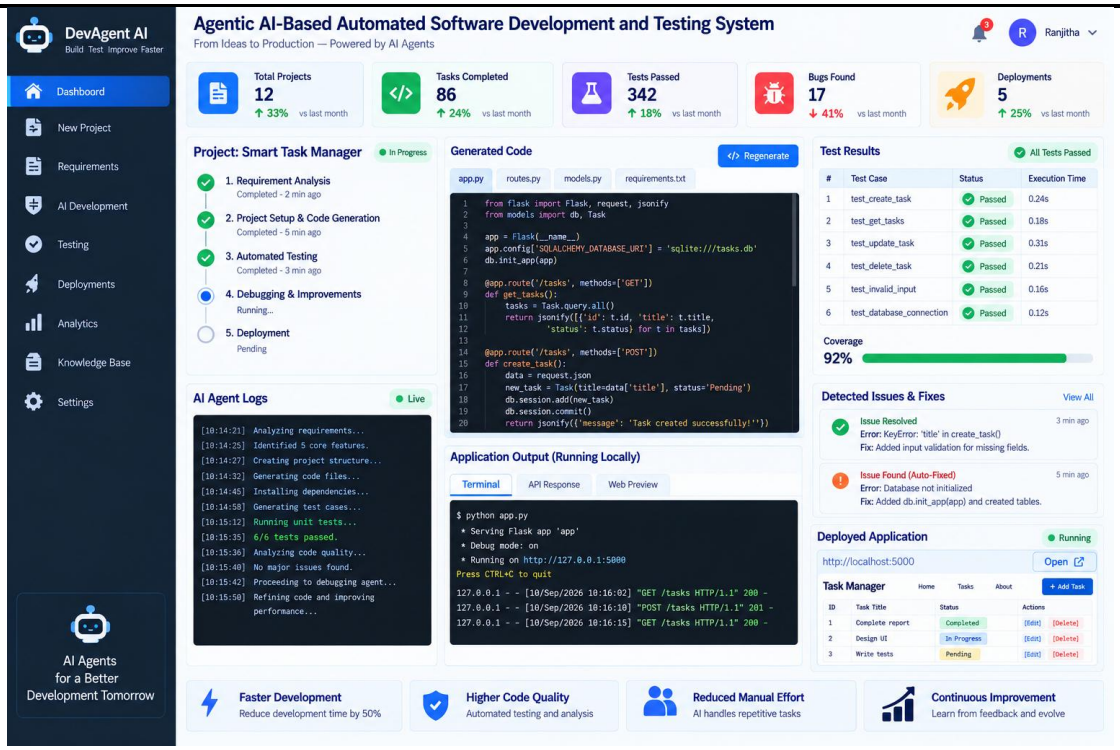
International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

the modified code back through the testing cycle. Version control records every significant change so developers can review, compare, approve, or revert AI-generated modifications. A secure database stores project requirements, code metadata, test results, logs, and development history. The analytics module calculates development metrics, test performance, defect trends, and task-completion statistics. The dashboard provides developers with a centralized view of project progress, generated code, test results, detected issues, and agent actions. The complete architecture follows the flow **Developer → Requirement Input → Requirement Analysis → Agentic AI Planning → Code Generation → Development Tools → Automated Testing → Failure Analysis → AI Debugging & Code Repair → Re-Testing → Version Control → Human Review → Validated Software**.



V. Result and Output

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper



VI. Conclusion

The Agentic AI-Based Automated Software Development and Testing System provides an intelligent approach to automating major activities in the software development lifecycle. By combining Agentic AI, Large Language Models, automated code generation, testing, debugging, and development tools, the system can transform natural-language requirements into development tasks and assist in producing functional software.

The proposed system can automatically generate code, create test cases, execute tests, analyze failures, identify possible causes, and perform controlled code improvements. The iterative code → test → debug → re-test workflow helps reduce repetitive development effort and provides faster feedback to developers.

The system also improves visibility through a centralized dashboard that displays project progress, generated code, test results, bugs, code-quality information, agent activities, and deployment status. Version control, security policies, and human approval mechanisms help maintain control over AI-generated changes and sensitive operations.

Overall, the project demonstrates how Agentic AI can support faster, smarter, and more automated software development while keeping human developers involved in critical decisions. Future enhancements can include multi-agent software development, automated vulnerability detection, intelligent test optimization, autonomous DevOps pipelines, repository-wide code understanding, predictive bug detection, and advanced production monitoring.

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

References

[1] Kumar, R. D., Prudhviraj, G., Vijay, K., Kumar, P. S., & Plugmann, P. (2024). Exploring COVID-19 through intensive investigation with supervised machine learning algorithm. In Handbook of Artificial Intelligence and Wearables (pp. 145-158). CRC Press.

[2] Swathi, B., Vijay, K., Sushanth Babu, M., & Dinesh Kumar, R. (2024, November). Machine Learning Techniques in Cloud Based Intrusion Detection. In The International Conference on Artificial Intelligence and Smart Environment (pp. 557-564). Cham: Springer Nature Switzerland.

[3] Sv satyakrishna, shirisha rangu ,bhargavi nalacheruve.(2024) Prospective investigation on colorectal cancer with SMOTE on machine learning Algorithm

[4] Dr.G.Vishnu Murthy, BhargaviNalacheruve 1Professor, Department of computer Science & engineering, Anurag University, TS, India. 2Student, Department of computer Science & engineering, Anurag University, TS, India.

[5] V. N. S. Manaswini, K. K, C. Nigam, S. S. Ali, R. Niranjana, and Suman, "Real-Time Object Detection in Drone Surveillance Using YOLOv5," in Proc. 2025 3rd Int. Conf. IoT, Communication and Automation Technology (ICICAT), Gorakhpur, India, 2025, pp. 1–6, doi: 10.1109/ICICAT68430.2025.11414670.

[6] B. Soundarya, V. N. S. Manaswini, M. Ayyakrishnan, R. D. Kumar, "Contextual Analysis of Big Data Analytics in Intelligent Transportation Frameworks," in Intersection of Artificial Intelligence, Data Science, and Cutting-Edge Technologies: From Concepts to Applications in Smart Environment, Lecture Notes in Networks and Systems, vol. 1353, Cham: Springer, 2025, doi: 10.1007/978-3-031-88304-0_79.

[7] R. D. Kumar, V. N. S.Manaswini, "Applications of blockchain in smart cities: detecting fake documents from land records using blockchain technology," in Blockchain for Smart Cities, Elsevier, 2021, pp. 105–117, doi: 10.1016/B978-0-12-824446-3.00017-X.

[8] Tejavath Veeramma, Badarla Anil, Guguloth Ravinder, "An advanced movie recommender using collaborative filtering and sentiment analysis," International Research Journal of Modernization in Engineering Technology and Science, vol. 7, no. 7, July 2025, doi: 10.56726/IRJMETS81618.

[9] Ravi Kumar Banoth, Ramana Murthy B V, "Automatic crop recommendation system using LightGBM and decision tree machine learning models," Journal of Machine and Computing, vol. 5, no. 1, pp. 343, Jan. 2025, doi: 10.53759/7669/jmc202505026.

[10] Ravi Kumar Banoth, Dr. B.V. Ramana Murthy, "Smart agriculture through IoT and machine learning for analyzing carbon footprints," in Proc. Int. Conf. Computer Science and Communication Engineering (ICCSCE), Apr. 2025.

[11] Ravi Kumar Banoth, B. V. Ramana Murthy, "Soil image classification using transfer learning approach: MobileNetV2 with CNN," SN Computer Science, vol. 5, art. no. 199, 2024, doi: 10.1007/s42979-023-02500-x.