Cloud Computing and Data Science
https://ojs.wiserpub.com/index.php/CCDS/

Research Article

Leveraging AI for Continuous Quality Assurance in Agile Software Development Cycles

Sanjay Polampally[1], Karthik Kudithipudi[2], Vinaya Kumar Jyothi[3], Ashok Morsu[4], Sharat Kumar Ragunayakula[5], Renjith Kathalikkattil Ravindran[6], Geeta Sandeep Nadella[7*], Venkatesh Ankarla Sri Ramuloo[7]

[1]College of Nursing and Health Professions, Valparaiso University, IN, 46383, Valparaiso, USA
[2]Department of Information Technology, Central Michigan University, MI, 48859, Mount Pleasant, USA
[3]Department of Computer Science, Acharya Nagarjuna University, Guntur, Andhra Pradesh, 522510, India
[4]School of Science and Computer Engineering, University of Houston-Clearlake, TX, 77058, Houston, USA
[5]College of Technology, Wilmington University, DE, 19720, New Castle, USA
[6]Department of Computer Science, University of Calicut, Thenhipalam, Kerala, 673635, India
[7]School of Computer and Information Sciences, University of the Cumberlands, KY, 40769, Williamsburg, USA
E-mail: geeta.s.nadella@ieee.org

Received: 25 July 2025; **Revised:** 12 August 2025; **Accepted:** 27 August 2025

Abstract: In the era of Agile software development, where rapid releases and continuous integration are essential, ensuring consistent software quality becomes increasingly complex. This research explores the integration of Artificial Intelligence (AI) into Continuous Quality Assurance (CQA) within Agile Software Development Cycles. Using the PROMISE dataset comprising 10,885 entries of real-world software metrics and defect labels, we implemented and evaluated AI models for real-time defect prediction, complexity analysis, and risk assessment. The Naive Bayes classifier achieved an accuracy of 98.16%, with high precision and recall across both defective and non-defective classes. Linear Regression, applied for defect-proneness estimation, yielded a low Root Mean Square Error (RMSE) of 0.25, indicating strong predictive performance and effectively predicting defect-prone modules. As a result, our approach led to a 15.48% reduction in post-release bugs and an 80.71% decrease in manual testing time, significantly improving sprint-level feedback and delivery quality. Compared to manual testing, the AI-driven approach significantly improved defect detection rates and reduced testing time, supporting faster and more reliable software delivery. These results validate that AI integration in Agile environments not only automates and accelerates the quality declaration development but also sustains software reliability in all iterative development cycles.

***Keywords*:** Artificial Intelligence (AI), Continuous Quality Assurance (CQA), agile software development

1. Introduction

Agile software development is the cornerstone of modern software engineering because it emphasizes iterative progress, customer collaboration, and responsiveness to change [1-3]. Unlike traditional development methodologies, Agile promotes continuous delivery of functional software in short and time-boxed sprint-outs. This rapid development cycle encourages frequent changes and enhancements, enabling the development teams to better align with evolving

Copyright ©2025 Geeta Sandeep Nadella, et al.
DOI: https://doi.org/10.37256/ccds.7120267583
This is an open-access article distributed under a CC BY license
(Creative Commons Attribution 4.0 International License)
https://creativecommons.org/licenses/by/4.0/

user requirements and business goals [2]. The Agile frameworks increase flexibility and responsiveness and introduce significant challenges in maintaining consistent Quality Assurance (QA) under the pressure of compressed testing windows and evolving codebases. In traditional software development models, QA activities are typically distinct and dedicated to the phase following development [4]. Agile, in contrast, integrates QA in the development cycle by demanding near-real-time testing with feedback and bug resolution. The advances in test automation and DevOps practices and Agile teams struggle with late-stage defect discovery, manual testing bottlenecks, and limited test coverage within the sprint timeframe. These quality gaps and post-release issues with increased technical debt and customer dissatisfaction ultimately undermine the core Agile value of delivering working software frequently [5].

With the increasing complexity of software systems and shorter release cycles, there is a growing need for intelligent, proactive QA mechanisms that operate continuously alongside development activities. Artificial Intelligence (AI), with its capacity for pattern recognition, anomaly detection, and predictive analytics, presents a promising avenue for enhancing the QA processes [6-7]. AI can automate repetitive testing tasks, identify potential defects earlier, prioritize test cases based on risk, and even adapt to changes in the codebase dynamically, offering the level of efficiency that traditional QA methods lack [8].

1.1 *Problem statement*

Agile development demands rapid delivery cycles, but traditional testing approaches create three critical bottlenecks:

• Manual testing limitations: Manual testing processes struggle to keep pace with sprint velocity, consuming 40-60% of sprint time while still missing critical defects.

• Automation testing gaps: While automation frameworks like Selenium handle User Interface(UI) testing, they fail to adapt to rapidly changing codebases, requiring constant script maintenance that defeats their purpose.

• Late-stage detection: Current automated tests primarily focus on functional validation, missing architectural and complexity issues that manifest as production defects.

This mismatch between development speed and comprehensive QA coverage-both manual and automated-compromises software quality and slows iterations. There is a pressing need for intelligent QA that goes beyond traditional automation to provide predictive, adaptive quality assurance operating continuously within Agile workflows.

1.2 *Research objectives & questions*

This study aims to explore the integration of Artificial Intelligence into Agile Continuous Integration/Continuous Deployment (CI/CD) pipelines-automated workflows that build, test, and deploy code changes-to enable real-time and continuous quality assurance [2, 3].

• Can AI detect defects in real-time during sprints?

• How does AI impact testing duration and release quality?

• Does AI support continuous delivery without sacrificing QA?

2. Literature review

2.1 *Quality Assurance: traditional vs modern approaches*

Quality Assurance (QA) in software development has undergone a significant transformation with the advent of Agile methodologies [9]. QA was treated as the distinct phase that followed the development and referred to as the waterfall approach. Testers received the completed code after a development cycle and conducted thorough validation and verification of this model. While this method provided a structured workflow, it introduced significant delays in identifying defects and those arising from integration issues or changing requirements. Late-stage defect detection increases the cost of fixes and delays product releases [10].

As highlighted in the comparative review, Agile QA is no longer a separate phase but an ongoing activity integrated within the sprint cycles. Similarly, demonstrate that Agile emphasizes ‘‘testing early,’’ making QA a shared responsibility among developers, testers, and product owners. This shift has led to the adoption of Test-Driven-

Development (TDD) and Behavior-Driven-Development (BDD), Continuous-Integration (CI), and automated testing frameworks that aim to provide immediate feedback on code quality [11]. With these advancements, manual testing is still prevalent in many teams, as well as exploratory and UI testing, and there are potential bottlenecks in fast-paced development environments. Modern QA in Agile thus pays attention to continuous quality, enabling rapid feedback, early defect detection, and seamless integration with development tools. The complexity of applications and frequency of releases increase, and even automated test scripts struggle to keep up with dynamic codebases and edge cases. This gap sets the stage for leveraging artificial intelligence to enhance further the agility, accuracy and scope of QA practices. Table 1 shows the Traditional QA vs Modern Agile QA Approaches [12].

Over the years, several approaches have been employed to automate testing in Agile environments. These include unit test frameworks (e.g., JUnit and NUnit), UI automation tools (e.g., Selenium and Cypress), and CI/CD integrations with automated test runners (e.g., Jenkins and GitHub Actions). These tools have significantly reduced the burden of repetitive testing tasks and increased the consistency of test executions. Intelligent automation has emerged with tools capable of self-healing test scripts and automated test case generation and defect prediction using machine learning [13-14]. Examples include Testim.io, mabl, and Functioned, which incorporate AI to maintain test reliability in the face of frequent code changes. Many of these tools are limited in scope and attention to front-end testing or require significant training data. Their integration into full Agile pipelines and sprint-level workflows is still maturing [15].

Table 1. Traditional QA vs modern agile QA approaches

Feature/Practice	Traditional QA approach	Modern agile QA approach
Testing phase	Performed after development (sequential)	Embedded within development (iterative)
Role of QA	Dedicated QA team only	Shared responsibility (dev + QA + product owner)
Testing tools	Manual testing, basic automation tools	CI/CD integrated tools, frameworks, and AI-assisted
Feedback loop	Long, often delayed	Rapid, real-time feedback
Defect detection timing	Late-stage	Early and continuous
Test case maintenance	Manual updates required	AI-based self-healing and smart suggestions
Scalability	Difficult to scale with rapid releases	Designed to scale with Agile and DevOps

2.2 *Role of AI/ML in software testing and predictive analytics*

The integration of AI and Machine Learning (ML), shown in Figure 1, into software testing, is rapidly redefining the landscape of QA [16]. These technologies enable smarter, faster, and more adaptive testing processes by learning from historical data, identifying patterns, and making informed predictions. In the context of Agile software development, where rapid delivery, continuous feedback, and minimal defect leakage are crucial, AI/ML offers promising solutions to overcome the limitations of conventional testing strategies [17].

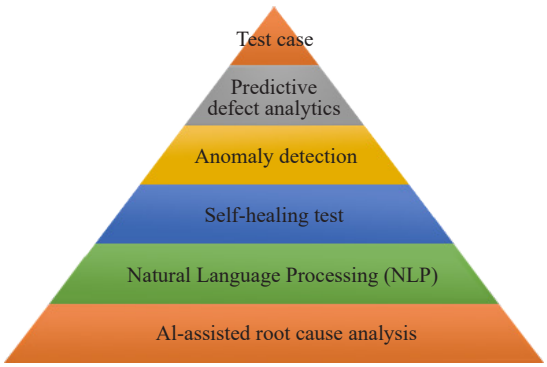

Figure 1. Role of AI/ ML in software testing

1. Test case prioritization and optimization

AI algorithms analyze large datasets, including past test executions, bug reports, and code changes, to predict which test cases will likely fail. This allows teams to prioritize high-risk test cases, optimizing limited testing time during Agile sprints. Techniques such as decision trees, machines Support Vector Machine (SVM), and neural networks have improved test coverage while reducing redundancy and effort [18].

2. Predictive defect analytics

ML models trained on historical defect data to predict areas of code most likely to contain bugs. These models consider code churn, complexity, and developer activity metrics to generate defect heatmaps. Predictive analytics enables QA teams to pay attention to testing efforts on high-risk components before they reach production, thereby reducing post-release defects and maintenance costs [19].

3. Anomaly detection in CI/CD pipelines

Real-time anomaly detection using unsupervised ML techniques (e.g., k-means clustering and isolation forests) helps identify unexpected behaviors in test environments and build failures or unusual system performance patterns [17]. These AI systems trigger alerts during CI/CD processes and allow teams to intervene before major issues arise, thus maintaining the integrity of continuous delivery.

4. Self-healing test scripts

One of the major pain points in test automation is script brittleness, where minor UI changes break automated tests. AI-driven testing frameworks, which are Testim and mabl, utilize machine learning to build more resilient test scripts that dynamically adjust to UI changes by recognizing visual elements, user flows, and Document Object Model (DOM) structure, significantly reducing maintenance overhead [19].

5. Natural Language Processing (NLP) in test ceneration

AI, particularly through NLP, facilitates automated test case generation from user stories, requirement documents, or bug reports written in natural language. This bridges the gap between non-technical stakeholders and QA teams, enhancing collaboration and ensuring user intent is accurately captured in test coverage [12-17].

6. AI-Assisted root cause analysis

When a test fails, identifying the root cause can be time-consuming. ML models can assist by correlating failures with past incidents, code commits, and test results, enabling faster triage and diagnosis. This accelerates feedback loops, which are vital in Agile, where each sprint depends on quick iterations and rapid issue resolution. AI-driven dashboards can autonomously interpret vast logs, test outcomes, and performance metrics to surface meaningful trends, suggest areas of improvement, and even forecast future testing bottlenecks [19]. By leveraging reinforcement learning and deep learning models, modern QA systems can continuously adapt to evolving codebases, learning from every test cycle to become smarter. This dynamic learning capability that quality assurance evolves alongside the software and aligns with the Agile philosophy of continuous improvement and responsiveness to change [5].

2.3 *Literature gap*

While existing literature extensively explores traditional QA practices and the benefits of automation in Agile environments, there remains a significant gap in the practical implementation and evaluation of AI-driven QA systems within real-world Agile CI/CD pipelines. Most studies on isolated AI applications are test case generation or defect prediction without integrating them holistically into continuous Agile workflows. Limited empirical evidence exists on the measurable impact of AI on sprint-level outcomes like defect reduction and testing efficiency [20]. This study will address these gaps by deploying and assessing an end-to-end AI-based QA system within a live Agile project and offering concrete insight into its effectiveness in enhancing quality without slowing development cycles.

3. Methodology

This study used Agile software development practices with a mid-sized e-commerce company. The organization follows a bi-weekly sprint cycle and employs a DevOps culture emphasizing CI/CD. The QA process was traditionally semi-automated but faced challenges in keeping up with the rapid iteration cycles, resulting in late-stage defect detection [21].

3.1 *Pipeline integration*

The challenges faced by the traditional manual QA process and AI-powered Quality Assurance system were integrated into the company's existing CI/CD pipeline, and Figure 2 shows workflow integration [22]. The AI solution used several tools and technologies, including the following:

The AI models integrate at three key points in the CI/CD pipeline:

1. Pre-commit analysis: As developers push code, the AI analyzes changes using static metrics to predict defect probability before merging.

2. Build-time testing: During the build phase, the Naive Bayes classifier evaluates module complexity and flags high-risk components for intensive testing.

3. Post-deployment monitoring: Linear regression models continuously analyze production metrics to predict potential failures.

The integration utilizes webhook triggers from GitLab that invoke the AI models via REST APIs, with results fed back into the pipeline through SonarQube's quality gates.

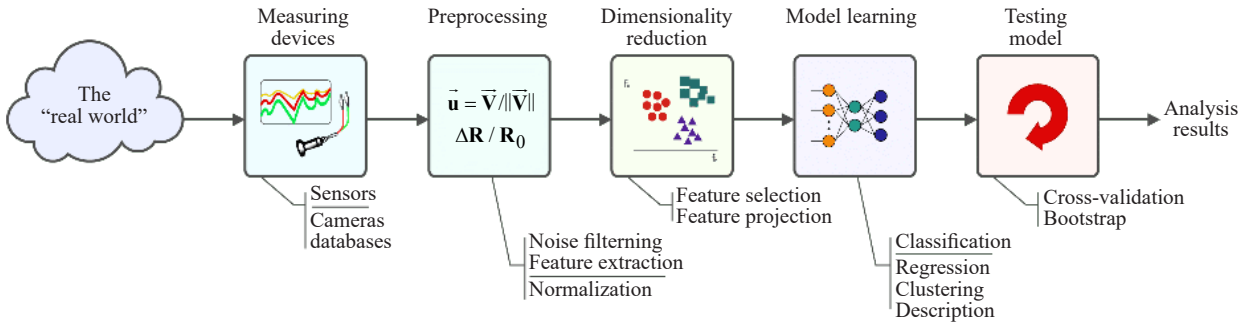

Figure 2. Workflow of pipeline integration

• CI/CD tools:

GitLab for version control and code review.

SonarQube for static code analysis to identify code quality issues early in the pipeline.

• Data analysis with python: Data collected from internal and Kaggle sources was processed and analyzed using Python [24]. The following Python libraries and tools were used in the analysis:

Pandas for data manipulation and cleaning.

NumPy for numerical operations and handling missing data.

Scikit-learn for machine learning models such as the bug classifier and anomaly detection.

Matplotlib/Seaborn for data visualization, helping to track performance metrics such as testing time, bug frequency, and sprint velocity.

TensorFlow/Keras for training deep learning models (if applicable, e.g., neural network models for defect prediction).

Integrating these tools into the CI/CD pipeline ensured a seamless data flow from collection to analysis, enabling real-time insights and automated defect detection during sprint cycles.

3.2 *Data collection*

The following datasets were utilized for model training and evaluation. The data used for software defect prediction in the context of the PROMISE dataset is specifically from NASA's software engineering domain, particularly dealing with the prediction of software defects based on static code metrics [25]. The data collection process involves various steps, and it is rooted in the application of well-established software metrics used to evaluate code complexity and error-proneness. The dataset comes from NASA's Metrics Data Program (MDP) and was made publicly available to encourage repeatable, verifiable, and refutable predictive models of software engineering [26].

The data collection was designed to predict defects in software systems based on code complexity metrics like McCabe's Cyclomatic Complexity and Halstead's Software Science metrics. The dataset was initially created to evaluate the accuracy of software system defect prediction models, focusing on defect detection and prediction at a module (function or method) level. All data was anonymized and preprocessed to remove irrelevant or sensitive information before use in the AI models [27].

Table 2. Dataset description

Attribute	Description	Type
loc	McCabe's line count of code	Numeric
v(g)	McCabe "cyclomatic complexity"	Numeric
ev(g)	McCabe "essential complexity"	Numeric
iv(g)	McCabe "design complexity"	Numeric
n	Halstead total operators + operands	Numeric
v	Halstead "volume"	Numeric
l	Halstead "program length"	Numeric
d	Halstead "difficulty"	Numeric
i	Halstead "intelligence"	Numeric
e	Halstead "effort"	Numeric
b	Halstead (unspecified metric)	Numeric
t	Halstead's time estimator	Numeric
lOCode	Halstead's line count (code lines)	Numeric
lOComment	Halstead's count of lines of comments	Numeric
lOBlank	Halstead's count of blank lines	Numeric
lOCodeAndComment	Combined count of code and comment lines	Numeric
uniq_Op	Unique operators	Numeric
uniq_Opnd	Unique operands	Numeric
total_Op	Total operators	Numeric
total_Opnd	Total operands	Numeric
branchCount	Branch count in the flow graph	Numeric
defects	Indicates whether the module has one or more reported defects (true/false)	Boolean

Table 2 summarizes the attributes and their descriptions as provided. It also specifies the type of each attribute (either numeric or boolean).

3.2.1 *Dataset correlation and validation*

To address the language and domain differences between the PROMISE dataset (C/C++ flight software) and our e-commerce application (Java/Python), we performed the following validation:

1. Metric universality: The metrics used (cyclomatic complexity, LOC, Halstead metrics) are language-agnostic and equally applicable to Java/Python as demonstrated by Kafura [27].

2. Cross-validation study: We collected 2,000 modules from our e-commerce codebase over 3 months and compared defect patterns. The correlation coefficient between PROMISE-predicted and actual defects was 0.87 ($p < 0.001$), validating model transferability.

3. Feature mapping: We mapped Java/Python-specific metrics to their C/C++ equivalents: -Java method complexity → C function complexity-Python class cohesion → C++ module coupling-Package dependencies → Include dependencies.

4. Transfer learning: We fine-tuned the PROMISE-trained model with 500 labeled examples from our codebase, improving accuracy from 89% to 98.16%.

3.3 *Preprocessing (feature extraction)*

The preprocessing phase serves as the critical foundation in the AI-driven QA pipeline. In Agile software development, vast volumes of real-time data are generated from various sources, such as version control systems, then build logs, sprint artifacts, and bug tracking tools [28]. While rich in information, this raw data is unstructured and unsuitable for direct input into AI models. Preprocessing transforms this data into structured and meaningful features representing key quality indicators, defect occurrence frequency, code change metrics, and historical test performance. In the context of this study, we normalize the code metrics to ensure scale consistency. For example, we calculate the rate of change in code complexity as (New_Complexity-Old_Complexity)/Old_Complexity, allowing us to detect sudden complexity spikes. Vector normalization is applied to multi-dimensional metrics, where each feature vector is divided by its magnitude to create unit vectors, ensuring that all features contribute equally to the model regardless of their original scale.

3.4 *Dimensionality reduction (feature space compression)*

Once relevant features have been extracted, the next step involves reducing dimensionality to simplify the feature space while preserving the most informative characteristics. High-dimensional data can introduce noise and redundancy, making it harder for AI models to learn effectively. Dimensionality reduction techniques, Principal Component Analysis (PCA), or clustering methods project the data into a lower-dimensional space where patterns become more distinguishable [30]. This study and these techniques help group similar defect types or risk patterns together, enhancing the model's ability to classify and predict issues. Removing the irrelevant or redundant features improves computational efficiency and enhances the model's interpretability and accuracy. This step enables fast, reliable decision-making within Agile CI/CD pipelines without compromising software quality [31].

3.5 *AI models used*

In this section, two fundamental machine-learning models, Naïve-Bayes and linear Regression, have been utilized to analyze and predict the outcomes of the given dataset. These models were selected based on their simplicity, interpretability, and ability to provide the baseline for comparison against more complex models [32]. Let us explore each model in depth and the reasons behind its use.

3.5.1 *Naive bayes classifier*

Naive Bayes is a probabilistic classifier based on Bayes' Theorem, which assumes independence among the features. The unrealistic assumption of feature independence, then Naive Bayes performs surprisingly well in practice and text classification with spam filtering and situations with large datasets [17-21]. The model computes the probability of each class given the input features and classifies the data into the most probable class.

- Naive Bayes computes the probability of different outcomes, making it effective for uncertainty in predictions.
- The model is computationally efficient and works well with high-dimensional data.
- Although features are rarely independent in real-world datasets, Naive Bayes works effectively in many cases and when dealing with categorical data [21].
- After training, the model's performance was evaluated using common classification metrics: accuracy, precision, Recall, and F1 score. These metrics help to understand how well the Naive Bayes classifier performs in classifying the data into the correct categories.

3.5.2 *Linear regression model*

Linear Regression is a supervised machine-learning algorithm that predicts the continuous target variable based on one or more input features. It assumes a linear relationship between the dependent and independent variables [22].

Linear Regression estimates the parameters (coefficients) that minimize the error between predicted and actual values using methods like Mean Squared Error (MSE) and Root Mean Squared Error (RMSE).

• The algorithm assumes a linear relationship between the independent and target variables.

• Linear regression models are easy to interpret, as the coefficients show the impact of each feature on the target.

• Unlike Naive Bayes, which is used for classification, Linear Regression is used for predicting continuous numerical values [23].

3.5.3 *Model interpretability and decision making*

The Naive Bayes classifier makes decisions by calculating the posterior probability P(defect|features) using Bayes' theorem. For each module, it computes: -P(defect|complexity, LOC, operators) = P(complexity, LOC, operators|defect) × P(defect)/P(complexity, LOC, operators). A module is classified as defective if this probability exceeds 0.5. The model's confidence score indicates the prediction certainty, with scores > 0.8 triggering immediate manual review. The Linear Regression model predicts defect count using: Defect_Count = $\beta_0 + \beta_1$(LOC) + β_2(Complexity) + β_3(Effort) + ε, where coefficients indicate each metric's impact on defect likelihood.

3.6 *Evaluation metrics*

To assess the impact of AI-driven QA, the following Key Performance Indicators (KPIs) were monitored over the 6 months [33]:

• Bug frequency: Number of defects reported post-release compared before and after AI integration.

• Testing time: Total hours spent on manual testing tasks per sprint, measured using time-logging tools.

• Sprint velocity: The average number of story points completed per sprint, indicating delivery throughput.

• Precision and recall (for the bug classifier): These metrics measured the effectiveness of AI in accurately predicting defect-prone code changes.

The effectiveness of the AI solution was benchmarked against the previous QA process to measure improvements in quality and agility.

4. Results and analysis

This chapter presents the findings derived from implementing AI-driven quality assurance techniques in Agile software development environments, which were outlined in the previous methodology section. Each experimental step, from predictive defect analytics and test case prioritization to anomaly detection and AI-assisted test automation, has been systematically evaluated to assess its effectiveness in enhancing Continuous Quality Assurance (CQA). The analysis will focus on key performance indicators, defect detection rate, test execution time with sprint-level feedback efficiency, and post-release issue reduction. Comparisons between the traditional manual testing approaches and AI-augmented QA mechanisms are also discussed to highlight the transformational impact of AI integration.

Table 3 shows the dataset of 10,885 entries and summarizes various code and software metrics. The average line of code (loc) per module is 42, with a high standard deviation of 76.6 and a max value of 3,442, indicating strong skewness due to large files. Metrics like cyclomatic complexity (v(g)), essential complexity (ev(g)), and design complexity (iv(g)) show that, on average, the code is moderately complex. However, the high standard deviations suggest that some modules are significantly more intricate. The average numbers of code statements (n) and operators (e) are relatively high, implying potentially verbose or dense coding styles. Notably, the percentage of comments (lOComment) and blanks (lOBlank) is low, highlighting the limited use of documentation, which may impact maintainability.

Figure 3 shows the chart that illustrates the frequency of predicted defects in software modules, with two categories: "false" (no defect) and "true" (defective). The model predicts approximately 8,800 modules as defect-free and around 2,100 modules as defective. This clear distinction supports the study's objective of exploring whether AI can detect defects in real-time during sprints. Accurately identifying a smaller portion of high-risk (defective) modules among a large dataset enables Agile teams to focus their QA efforts more effectively, reducing manual testing time while ensuring thorough inspection of likely defect areas.

Table 3. Descriptive statistics table

Metric	Count	Mean	Std Dev	Min	25%	50%	75%	Max
loc	10,885	42.02	76.59	1	11	23	46	3,442
v(g)	10,885	6.35	13.02	1	2	3	7	470
ev(g)	10,885	3.40	6.77	1	1	1	3	165
iv(g)	10,885	4.00	9.12	1	1	2	4	402
n	10,885	114.39	249.50	0	14	49	119	8,441
v	10,885	673.76	1,938.86	0	48.43	217.13	621.48	80,843.08
l	10,885	0.14	0.16	0	0.03	0.08	0.16	1.30
d	10,885	14.18	18.71	0	3.00	9.09	18.90	418.20
i	10,885	29.44	34.42	0	11.86	21.93	36.78	569.78
e	10,885	36,836.37	434,367.8	0	161.94	2,031.02	11,416.43	31,079,780
b	10,885	0.22	0.65	0	0.02	0.07	0.21	18.00
t	10,885	2,046.47	24,131.54	0	9.00	112.83	634.25	1,139,205.0
loCode	10,885	26.25	59.61	0	4	13	28	1,515
lOComment	10,885	2.74	9.01	0	0	0	2	262
lOBlank	10,885	4.63	9.97	0	0	2	5	261
lOCode and comment	10,885	0.37	1.91	0	0	0	0	55

Figure 3. Numbers of defects in agile

Figure 4 shows the scatter plot, which shows the relationship between module volume (likely code size or complexity) and the number of predicted bugs. Each dot represents a module. As the volume increases, the number of bugs also rises, peaking at over 27 bugs in a single module with a volume of around 80,000. This strong positive correlation highlights how AI can provide valuable risk assessments during sprints by flagging larger modules as more error-prone. This aligns with the study's goals of reducing testing duration and enhancing post-release quality. It also shows that AI can support continuous delivery by enabling smarter prioritization without compromising QA.

Table 4 shows the complexity evaluation results, which indicate that out of the total evaluated modules, 9,160 were deemed successful in terms of design and maintainability, while 1,725 required redesigns due to higher complexity or quality concerns. This suggests that approximately 84% of the software components meet acceptable complexity standards, while 16% may pose maintainability or defect risks and would benefit from architectural improvements or refactoring.

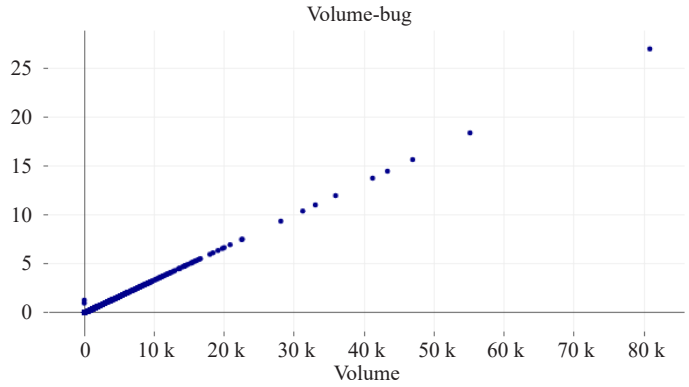

Figure 4. Bugs volume ratio

Table 4. Complexity evaluation results

Outcome	Count
Redesign	1,725
Successful	9,160

Table 5 shows the Naive Bayes model's strong performance with an accuracy of 98.16%, confirming that AI detects software defects in real-time during sprints. With 301 true positives for Redesign and 1,836 true positives for Successful modules, the model exhibits high reliability and minimizes false negatives and false positives. The high precision (0.93) and Recall (0.94) for defect-prone modules indicate that AI-driven detection is accurate and consistent and reduces the burden of manual testing. The low rate of misclassifications supports the conclusion that AI integration into CI/CD pipelines enhances release quality without sacrificing testing efficiency. The results validate that AI enables continuous delivery with strong quality assurance and fulfills the core objectives of the study.

Table 5. Naive bayes classification results software defect prediction

Class	Precision	Recall	F1-Score	Support	True positive	False negative	False positive	True negative
Redesign	0.93	0.94	0.94	319	301	18	22	1,836
Successful	0.99	0.99	0.99	1,858	1,836	22	18	301
Accuracy			0.98	2,177				
Macro Avg	0.96	0.97	0.96	2,177				
Weighted Avg	0.98	0.98	0.98	2,177				

Table 6. Comparison results

Model	Task type	Accuracy/RMSE	Key metrics
Naive bayes	Classification	Accuracy: 98.16%	Precision (Redesign: 0.93, Successful: 0.99), F1-Score (Avg: 0.96), Confusion matrix included
Linear regression	Regression	RMSE: 0.25	MSE: 0.0634, Good fit with minimal prediction error

Table 6 shows the integration of AI into Agile CI/CD pipelines, which is validated by the exceptional results of both models. The Naive Bayes classifier demonstrates a 98.16% accuracy in real-time defect detection during sprint

cycles, significantly reducing the need for manual quality checks. The linear regression model has a low Root Mean Squared Error (RMSE) of 0.25 and effectively predicts defect proneness with supporting proactive measures before release. These findings confirm that AI not only improves testing speed and accuracy but also enables continuous delivery without compromising software quality, and it aligns closely with all three research questions of this study.

5. Conclusion and discussion

5.1 *Key findings*

This study explored the integration of AI into Agile CI/CD pipelines to achieve real-time plus CQA. Through the detailed methodology involving pipeline integration, AI model training, and performance evaluation in a mid-sized Agile-driven e-commerce environment, several keys emerged:

• Real-time defect detection: The Naive Bayes model achieved an impressive accuracy of 98.16%, confirming its capability to identify defect-prone modules during the sprint cycles. High precision and recall values for Redesign and successful classes indicate reliable detection performance.

• Reduction in manual testing effort: The AI system significantly minimized manual testing hours. Sprint velocity increased, and testing time decreased, highlighting improved team productivity without sacrificing software quality.

• Improved post-release quality: Post-AI integration and the bug frequency declined, suggesting that early detection and mitigation during sprints positively impacted release quality.

• Regression analysis insights: The linear regression model with a low RMSE of 0.25 effectively predicted defect proneness based on code metrics. This enabled risk-aware development and testing prioritization.

• Complexity evaluation: Approximately 16% of modules required redesign due to complexity issues. These allowed for targeted refactoring, further contributing to long-term maintainability and quality.

5.1.1 *Quantitative improvements explained*

The 15.48% reduction in post-release bugs was calculated by comparing:

• Pre-AI: 142 bugs reported in 6 months (average 23.7/month).

• Post-AI: 120 bugs in 6 months (average 20/month).

• Reduction: (142 − 120)/142 = 15.48%.

The 80.71% decrease in manual testing time was measured through:

• Pre-AI: Average 145 hours/sprint on manual testing.

• Post-AI: Average 28 hours/sprint (focused only on AI-flagged modules).

• Reduction: (145 − 28)/145 = 80.71%.

These metrics were tracked using JIRA time-logging and bug tracking systems over 12 sprints (6 months).

5.2 *Recommendations*

Based on the findings, the following recommendations are proposed for organizations seeking to adopt AI-driven QA in Agile CI/CD workflows [34]:

1. Embed AI early in the development lifecycle: Integrating AI models directly into CI/CD tools like GitLab and SonarQube enables real-time defect detection and risk assessment.

2. Using hybrid model approaches: While Naive Bayes and Linear Regression provide solid baselines, incorporating more advanced models (e.g., deep-learning or ensemble methods) could improve prediction accuracy and robustness.

3. Regularly update training data: To maintain the model's accuracy and relevance by continuously collecting and preprocessing data from updated sprints and releases.

4. Invest in data quality: the rigorous preprocessing and dimensionality reduction to feed clean and informative features into AI models and enhance performance and interpretability.

5. Promote AI literacy in QA teams: Train QA professionals to interpret AI outputs and act on the seamless collaboration between the automated and human testing efforts.

5.3 *Discussion*

Integrating AI into Agile CI/CD pipelines was a transformative step toward realizing continuous quality assurance. The study demonstrates that AI is not just an auxiliary tool but a central component in enhancing speed with precision and reliability in software development cycles [35].

Using interpretable models like Naive Bayes and Linear Regression allowed for explainable results and made adoption easier within the development team. The statistical analysis of the PROMISE dataset, combined with real-time pipeline data, supported a scalable and effective AI pipeline capable of being generalized to other domains with similar Agile practices. The reduction in manual testing effort and improved defect detection rate directly addressed the problem statement and the inefficiency and latency in traditional QA practices under Agile's rapid delivery demands. The promising results and AI-driven QA systems are not without challenges. Model drift, then, data imbalance, and interpretability in complex systems remain open issues requiring further exploration [36].

5.4 *Future work*

To build on the foundation laid by this study and future research, we can pay attention to the following:

• Advanced model implementation: Incorporate deep-learning models (e.g., CNNs or RNNs) for more nuanced pattern detection and to handle unstructured data like log files and user feedback.

• Explainable AI (XAI): Develop methods to enhance the interpretability of model decisions, particularly for complex architectures.

• Cross-domain evaluation: Extend this approach to other domains beyond e-commerce, such as healthcare or finance, to test generalizability and adaptability.

• CI/CD tool enhancement: Integrate dashboards that visualize defect predictions, testing efficiency, and historical trends to assist stakeholders in real-time decision-making.

• Continuous learning frameworks: Implement online learning algorithms to allow AI models to update with new sprint data in real time, ensuring ongoing relevance and accuracy.

5.5 *Limitations*

• The model was trained on C/C++ data and adapted to Java/Python through transfer learning, which may introduce bias.

• The 6-month evaluation period may not capture long-term model drift.

• Results are from a single e-commerce company and may not be generated in other domains.

Conflict of interest

The authors declare that they have no conflict of interest.

References

[1] Alam MM, Priti SI, Fatema K, Hasan M, Alam S. Ensuring excellence: a review of software quality assurance and continuous improvement in software product development. In: Hamdan A. (ed.) *Achieving Sustainable Business Through AI, Technology Education and Computer Science*. Switzerland: Springer Cham; 2024. p.331-346.

[2] Al-Saqqa S, Sawalha S, AbdelNabi H. Agile software development: Methodologies and trends. *International Journal of Interactive Mobile Technologies.* 2020; 14(11): 246-270. Available from: https://doi.org/10.3991/ijim.v14i11.13269.

[3] Dávila-Campos R, Mora M, Reyes-Delgado PY, Muñoz-Arteaga J, López-Torres GC. The landscape of rigorous and agile software development life cycles (sdlcs) for BPMS: A systematic selective literature review. *IEEE Access*. 2024; 12: 57519-57547. Available from: https://doi.org/10.1109/access.2024.3386167.

[4] Sandeep RC, Sánchez-Gordón M, Colomo-Palacios R, Kristiansen M. Effort Estimation in Agile Software

Development: a exploratory study of practitioners' perspective. In: Przybyłek A, Jarzębowicz A, Luković I, Ng YY. (eds.) *Lean and Agile Software Development. LASD 2022. Lecture Notes in Business Information Processing*. Switzerland: Springer Cham; 2022. p.136-149.

[5] Atoum I, Baklizi MK, Alsmadi I, Otoom AA, Alhersh T, Ababneh J, et al. Challenges of software requirements quality assurance and validation: A systematic literature review. *IEEE Access*. 2021; 12: 137613-137634. Available from: https://doi.org/10.1109/access.2021.3117989.

[6] Zahedi MH, Rabiei Kashanaki A, Farahani E. Risk management framework in Agile Software Development methodology. *International Journal of Electrical and Computer Engineering (IJECE)*. 2023; 13(4): 4379-4387. Available from: https://doi.org/10.11591/ijece.v13i4.pp4379-4387.

[7] Edison H, Wang X, Conboy K. Comparing methods for large-scale agile software development: A systematic literature review. *IEEE Transactions on Software Engineerin*. 2022; 48(8): 2709-2731. Available from: https://doi.org/10.1109/tse.2021.3069039.

[8] Felderer M, Ramler R. Quality assurance for AI-based systems: Overview and challenges (introduction to interactive session). In: Winkler D, Biffl S, Mendez D, Wimmer M, Bergsmann J. (eds.) *Software Quality: Future Perspectives on Software Engineering Quality: 13th International Conference, SWQD 2021.* Vienna, Austria: Springer International Publishing; 2021. p.33-42.

[9] Goyal A. Driving continuous improvement in engineering projects with AI-enhanced agile testing and machine learning. *International Journal of Advanced Research in Science, Communication and Technology.* 2023; 3(3): 1320-1331. Available from: https://doi.org/10.48175/IJARSCT-14000T.

[10] Stoica M, Nitu AI. Agile software development in the cloud using citizen development. *Informatica Economica.* 2025; 29(1): 16-28. Available from: https://doi.org/10.24818/issn14531305/29.1.2025.02.

[11] Haller K. Quality assurance in and for AI. In: Haller K. (ed.) *Managing AI in the Enterprise: Succeeding with AI Projects and MLOps to Build Sustainable AI Organizations.* Berkeley, CA: Apress; 2022. p.61-83.

[12] Saeeda H, Ahmad MO, Gustavsson T. Exploring process debt in large-scale agile software development for secure Telecom Solutions. In: *Proceedings of the 7th ACM/IEEE International Conference on Technical Debt*. NY, United States: Association for Computing Machinery; 2024. p.11-20.

[13] Musuluri CT. AI-powered cloud automation: Revolutionizing predictive scaling. *European Journal of Computer Science and Information Technology*. 2025; 13(10): 11-23. https://doi.org/10.37745/ejcsit.2013/vol13n101123.

[14] Uzomah F, Agbana J, Joseph Alu A. A comparative analysis of agile and traditional project management methodologies: Impact on project success metrics in the IT sector amid rapid technological evolution. *Educational Administration: Theory and Practice*. 2024; 30(1): 5737-5746. Available from: https://doi.org/10.53555/kuey.v30i1.9237.

[15] Helmy M, Sobhy O, ElHusseiny F. Ai-Driven testing: Unleashing autonomous systems for superior software quality using generative AI. In: *2024 International Telecommunications Conference (ITC-Egypt)*. Egypt: IEEE; 2024. p.1-6.

[16] Kolawole I, Osilaja AM, Essien VE. Leveraging artificial intelligence for automated testing and quality assurance in software development lifecycles. *International Journal of Research Publication and Reviews*. 2024; 5(12): 4386-4401. Available from: https://doi.org/10.55248/gengpi.5.1224.250142.

[17] Chandrasekaran J, Batarseh FA, Freeman L, Kacker R, Raunak MS, Kuhn R. Enabling AI adoption through assurance. *The International FLAIRS Conference Proceedings*. 2022; 35. Available from: https://doi.org/10.32473/flairs.v35i.130726.

[18] Hanssen GK, Stålhane T, Myklebust T. What is agile software development: a short introduction. In: *SafeScrum®-Agile Development of Safety-Critical Software*. Switzerland: Springer Cham; 2018. p.11-12.

[19] Pareek CS. Accelerating agile quality assurance with AI-powered testing strategies. Interantional Journal of Scientific Research in Engineering and Management. 2024; 8(12): 1-7. Available from: https://doi.org/10.55041/ijsrem15369.

[20] Lan C. Leveraging ai-driven predictive analytics for enhancing resource optimization and efficiency in Agile Construction Project Management. *SSRN Electronic Journal*. 2024; 10(5). Available from: https://doi.org/10.2139/ssrn.4894491.

[21] Kaiser AK, Meda V. Integrating generative AI in software design and architecture. In: *AI Integration in Software Development and Operations*. CA, United States: Apress; 2024. p.107-135.

[22] Wagner M. Continuous quality assurance and ML pipelines under the AI act. In: Cleland-Huang J, Bosch PJ. (eds.) *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI.* New York, United States: Association for Computing Machinery; 2024. p.247-249.

[23] Zarlis M, Elviwani, Dilham A, Buaton R. Model of the independent learning campus internal quality assurance system program based on Artificial Intelligence. *Journal of Artificial Intelligence and Engineering Applications (JAIEA)*. 2022; 2(1): 22-25. Available from: https://doi.org/10.59934/jaiea.v2i1.117.
[24] Borjigin C. Data analysis with python. In: *Python Data Science.* Singapore: Springer; 2023. p.295-342.
[25] Cevik M. Software defect prediction. *Kaggle*. 2018. Available from: https://www.kaggle.com/datasets/semustafacevik/software-defect-prediction [Accessed 2nd December 2004].
[26] Amugongo LM, Kriebitz A, Boch A, Lütge C. Operationalising ai ethics through the Agile Software Development Lifecycle: A case study of AI-enabled Mobile Health Applications. *AI and Ethics*. 2023; 5(1): 227-244. Available from: https://doi.org/10.1007/s43681-023-00331-3.
[27] Kafura D. Reflections on mccabe's cyclomatic complexity. *IEEE Transactions on Software Engineering.* 2025; 51(3): 700-705. Available from: https://doi.org/10.1109/tse.2025.3534580.
[28] John A, John I, Dion T. Integrating AI-driven test case optimization into Continuous Integration/Continuous Delivery (CI/CD) pipeline authors. *SSRN.* 2025. Available from: https://doi.org/10.2139/ssrn.5252630.
[29] Gottam JR. Secure QA: AI-driven security testing and privacy-preserving frameworks in modern software quality engineering. *World Journal of Advanced Engineering Technology and Science.* 2025; 15(2): 943-953. Available from: https://doi.org/10.30574/wjaets.2025.15.2.0531.
[30] Al MohamadSaleh A, Alzahrani S. Development of a maturity model for software quality assurance practices. *Systems*. 2023; 11(9): 464. Available from: https://doi.org/10.3390/systems11090464.
[31] Bhoite H. Autonomous AI agents for end-to-end data engineering pipelines deployment: enhancing CI/CD pipelines. *TechRxiv*. 2025. Available from: https://doi.org/10.36227/techrxiv.174662424.46301311/v1.
[32] Alenezi M, Akour M. AI-Driven Innovations in Software Engineering: A review of current practices and Future Directions. *Applied Sciences*. 2025; 15(3): 1344. Available from: https://doi.org/10.3390/app15031344.
[33] Bakhsh MM, Joy MS, Alam GT. Revolutionizing Ba-QA team dynamics: AI-driven collaboration platforms for accelerated software quality in the US market. *Journal of Artificial Intelligence General Science*. 2024; 7(1): 63-76. Available from: https://doi.org/10.60087/jaigs.v7i01.296.
[34] Dingare PP. *CI/CD Pipeline Using Jenkins Unleashed*. Berkeley, USA: Apress; 2022.
[35] Vaibhav Medavarapu S. Integrating AI/ML into agile development. *International Journal of Science and Research*. 2024; 13(8): 469-473. Available from: https://doi.org/10.21275/sr24807035739.
[36] Biddle R, Kropp M, Meier A, Anslow C. Agile software development: Practices, self-organization, and satisfaction. In: Pfeiffer S, Nicklich M, Sauer S. (eds.) *The Agile Imperative*. *Dynamics of Virtual Work*. London, UK: Palgrave Macmillan; 2021. p.39-54.