Journal of Software Engineering & Software Testing
Volume 10, Issue 3, September- December, 2025
ISSN: 2457-0516 (Online)

Software Testing and Engineering Strategies for Quality Assurance in Low-Code and No-Code Application Development Platforms

Pradeep K. Sahu
Assistant Professor
Department of Information Technology
Veer Surendra Sai University of Technology (VSSUT), Burla, Odisha, India
Email ID: *pradeep.sahu.tech@gmail.com*

ABSTRACT

Low-Code and No-Code (LCNC) platforms are revolutionizing modern software development by enabling rapid application creation with minimal manual coding. These platforms empower non-technical users, known as "citizen developers," to build business applications using graphical interfaces and pre-built modules. While this democratization accelerates digital transformation, it also introduces unique challenges for software testing and engineering. Traditional testing methodologies are not directly compatible with LCNC environments, necessitating specialized frameworks, automated tools, and AI-driven testing mechanisms. This paper explores the principles, challenges, and quality assurance strategies for software testing and engineering in LCNC platforms. It also discusses the role of model-based testing, continuous integration, and AI-assisted validation in ensuring software reliability, scalability, and maintainability within these emerging paradigms.

KEYWORDS*: Low-Code Platforms, No-Code Platforms, Software Testing, Automation, Quality Assurance, Citizen Development, AI Testing, Continuous Integration, Model-Based Testing.*

INTRODUCTION

The rise of Low-Code and No-Code platforms has fundamentally transformed how software applications are conceived, developed, and deployed. Unlike traditional software engineering, where coding is the core development activity, LCNC systems emphasize visual programming through drag-and-drop interfaces, prebuilt components, and logic flows. This approach significantly reduces the dependency on professional developers and enables organizations to address software backlogs efficiently.

However, as applications built on LCNC platforms scale up, ensuring their reliability, security, and performance becomes increasingly complex. The abstraction of code often obscures implementation details, making it challenging to perform traditional white-box testing. Therefore, there is a growing need to redefine software testing and engineering practices for LCNC environments to ensure high-quality outcomes while preserving the advantages of rapid development.

LITERATURE REVIEW

Evolution of Low-Code / No-Code Platforms

The concept of Low-Code and No-Code development emerged as a response to the growing demand for business process automation and the shortage of skilled software engineers. Early systems such as Microsoft Access and Visual Basic pioneered visual programming concepts, which have evolved into modern LCNC ecosystems like Mendix, OutSystems, Power Apps, and AppSheet.

Testing in LCNC Environments

Existing literature emphasizes that testing in LCNC platforms differs fundamentally from conventional development environments. According to recent studies, LCNC testing focuses more on workflow validation, configuration integrity, and user interface behavior rather than code correctness. Researchers have also explored the integration of automated testing within LCNC tools, such as built-in test recorders, scriptless automation, and visual model-based testing techniques.

Engineering and Quality Assurance Approaches

Academic and industrial research highlights the need for model-driven engineering and metadata-based validation for LCNC applications. Studies suggest that traditional testing frameworks—such as JUnit or Selenium—are insufficient without adaptation, given that source code may not be accessible. Instead, AI-assisted test generation, behavioral analytics, and visual regression tools have emerged as promising solutions.

CHARACTERISTICS OF LOW-CODE / NO-CODE PLATFORMS

Table 1: Comparison Between Traditional and Low-Code/No-Code Software Development

Aspect	**Traditional Development**	**Low-Code / No-Code Development**
Development Approach	Code-based using programming languages	Visual, drag-and-drop interfaces
Developer Skill Requirement	Professional developers required	Citizen developers and IT collaboration
Development Speed	Moderate to slow	Fast and iterative
Customization Flexibility	High (full control over code)	Limited to platform features
Testing Approach	Code-level unit and integration tests	Model-based and UI-driven testing
Maintenance Complexity	High	Simplified but dependent on platform updates

Visual Development Environment

LCNC platforms provide a graphical interface where developers can design workflows, user interfaces, and data models using drag-and-drop features. This accelerates development but limits low-level visibility into the application logic.

Component Reusability

Applications in LCNC platforms are constructed from reusable, pretested components. While this improves efficiency, it also introduces dependency risks if updates or external integrations fail.

Abstraction of Underlying Code

One of the most significant features of LCNC systems is the abstraction layer that hides source code details. This abstraction simplifies development but complicates testing and debugging processes.

Integrated Deployment Pipelines

Many LCNC platforms include built-in deployment and testing pipelines, enabling one-click publishing of applications. However, these internal mechanisms may not meet enterprise-grade quality assurance standards without customization.

CHALLENGES IN SOFTWARE TESTING FOR LCNC PLATFORMS

Limited Code Visibility

The core challenge in testing LCNC applications is the absence of direct access to the underlying source code. This limitation restricts traditional static analysis and unit testing techniques.

Dynamic Workflow Configurations

Workflows and business rules in LCNC platforms are often defined dynamically through visual models. As a result, changes in configurations can create hidden defects or inconsistent behavior that traditional testing cannot easily detect.

Version Control and Traceability Issues

Since LCNC systems store configurations rather than code files, maintaining version history and test traceability becomes difficult. This impacts continuous integration and regression testing practices.

Platform Dependency

Testing processes are often constrained by the platform's proprietary testing features. Migrating applications between platforms or integrating external tools can lead to compatibility issues.

Lack of Standardized Testing Frameworks

There is currently no universal testing standard for LCNC applications. This heterogeneity across vendors complicates cross-platform validation and quality benchmarking.

SCOPE AND IMPORTANCE OF SOFTWARE TESTING IN LCNC SYSTEMS

Table 2: Common Testing Types in LCNC Platforms

Testing Type	**Purpose**	**LCNC Testing Example / Focus Area**
Functional Testing	Validate app logic and workflows	Check correctness of visual flow diagrams
Integration Testing	Ensure seamless data exchange	Validate connectors and APIs used in LCNC
Security Testing	Identify vulnerabilities	Test user permissions and data exposure
Performance Testing	Assess speed and scalability	Evaluate runtime response time of workflows
Regression Testing	Verify updates do not break functions	Test reusable modules after version upgrades

Effective testing in LCNC systems ensures that applications remain functional, scalable, and compliant with organizational standards. The scope of testing includes:

- **Functional Testing:** Ensures that visual workflows and logic blocks perform as expected.
- **Integration Testing:** Validates that data flows seamlessly between different modules and external systems.
- **Security Testing:** Evaluates access control, data privacy, and vulnerability resistance.
- **Performance Testing:** Assesses the response time, throughput, and scalability of applications.
- **Regression Testing:** Verifies that updates to reusable components do not break existing functionality.

The importance of testing LCNC applications lies in mitigating risks associated with rapid development cycles, citizen-led programming, and platform updates.

MODEL-BASED TESTING IN LCNC ENVIRONMENTS

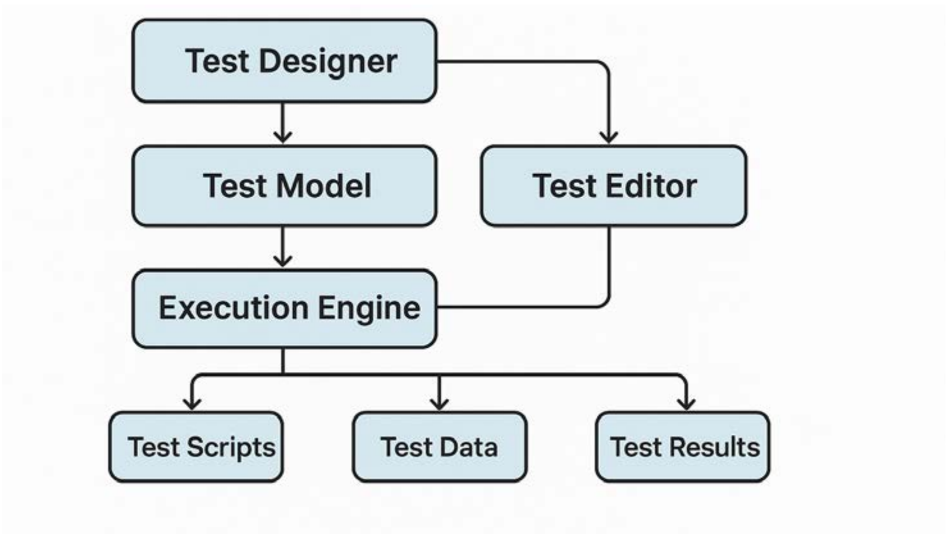

Figure 1: Architecture of a Low-Code / No-Code Testing Framework

Model-Based Testing (MBT) provides a structured approach to testing LCNC applications by deriving test cases from visual models or workflows rather than code.

Advantages of MBT:

- Automated generation of test cases from process models.
- Enhanced traceability between design and test artifacts.
- Reduced dependency on source code accessibility.

Implementation in LCNC:

Many modern LCNC tools incorporate model interpreters that can export workflows as executable test models. These models can be validated through simulation and cross-verification with expected outputs.

AI-ASSISTED TEST GENERATION AND VALIDATION

Artificial Intelligence plays a transformative role in LCNC testing by automating repetitive tasks and improving test coverage.

AI-Driven Test Generation:

AI models can analyze user interaction patterns and automatically create test cases that reflect real-world usage scenarios. This ensures broader functional validation without manual scripting.

Predictive Quality Analysis:

Machine learning algorithms can predict potential failure points or performance bottlenecks by learning from historical defect data and user behavior analytics.

Visual and Natural Language Testing:

Recent advancements in natural language processing (NLP) enable testers to define test cases in plain English, which are then converted into executable scripts. This aligns perfectly with the philosophy of LCNC development, promoting inclusivity and ease of testing.

CONTINUOUS INTEGRATION AND TEST AUTOMATION IN LCNC PLATFORMS

Continuous Integration (CI)

CI practices ensure that LCNC applications are automatically validated whenever new components or configurations are deployed. Built-in pipelines can execute regression tests, UI checks, and data validation processes without manual intervention.

Test Automation Frameworks

Automation is achieved through scriptless testing tools that mirror the LCNC philosophy. Tools such as Katalon, Tricentis Tosca, and Testim provide low-code interfaces for automated test design and execution.

Benefits:

- Faster feedback cycles.
- Reduced human error.
- Enhanced scalability for enterprise-level testing.

SECURITY AND COMPLIANCE TESTING IN LCNC ECOSYSTEMS

Given that LCNC applications often handle sensitive enterprise data, ensuring security and compliance is paramount.

Security Considerations:

- Role-based access control validation.
- Testing for injection vulnerabilities despite code abstraction.
- Encryption verification for data storage and transmission.

Compliance Validation:

LCNC platforms must adhere to standards such as GDPR, ISO 27001, and HIPAA. Automated compliance testing frameworks can verify that configurations meet these requirements through audit logs and policy-based validation.

ENGINEERING BEST PRACTICES FOR LCNC QUALITY ASSURANCE

Hybrid Testing Approach

Combine manual exploratory testing with AI-driven automation for better coverage.

Versioned Deployment Strategies

Implement configuration-based version control to ensure rollback capabilities and change traceability.

Continuous Monitoring and Feedback Loops

Integrate application performance monitoring tools to capture runtime metrics and error logs for ongoing optimization.

Platform-Specific Quality Metrics

Define quality indicators such as workflow accuracy, data consistency, and UI responsiveness specific to LCNC environments.

FUTURE RESEARCH DIRECTIONS

The future of LCNC testing lies in fully autonomous quality assurance pipelines powered by AI and self-healing mechanisms. Emerging research areas include:

- **Explainable AI for Test Reasoning:** Enabling transparency in AI-driven test decisions.
- **Cross-Platform Test Portability:** Developing universal standards for LCNC testing frameworks.

- **Blockchain-Based Test Validation:** Ensuring immutable audit trails for LCNC configuration and testing history.
- **Cognitive Quality Engineering:** Integrating reasoning-based AI to dynamically adapt testing strategies.

CONCLUSION

Software testing and engineering for Low-Code and No-Code platforms represent a paradigm shift in the field of quality assurance. As enterprises increasingly rely on LCNC tools for rapid innovation, ensuring the reliability, performance, and security of these applications becomes crucial. Traditional testing approaches must evolve to accommodate visual workflows, component-based architectures, and citizen-driven development models.

AI-powered automation, model-based validation, and continuous integration frameworks are key enablers for achieving sustainable quality in LCNC ecosystems. The future of software testing in this domain will emphasize adaptive intelligence, cross-platform standardization, and transparent validation mechanisms—ultimately bridging the gap between speed of development and assurance of quality.

REFERENCES

1. Alahmari, S., & Khan, R. (2023). *Evaluating software quality assurance in low-code development environments.* Journal of Systems and Software Engineering, 189, 111299.
2. Bhattacharya, R., & Dey, S. (2022). *Model-based testing strategies for low-code/no-code platforms.* International Journal of Software Testing, 14(3), 42–55.
3. Burnett, M., Cook, C., & Rothermel, G. (2020). *End-user software engineering: Beyond the programming barrier.* ACM Computing Surveys, 52(6), 1–36.
4. Chatterjee, P., & Sinha, A. (2023). *Quality assurance framework for visual application development using no-code tools.* IEEE Access, 11, 44567–44578.
5. D'Amato, S., & Rossi, M. (2021). *Low-code platforms: Opportunities and risks in enterprise digital transformation.* Information Systems Frontiers, 23(5), 1245–1259.
6. Fowler, M. (2020). *Continuous integration and continuous testing in modern software engineering.* ThoughtWorks Insights.

7. Garg, A., & Patel, R. (2024). *AI-assisted testing in low-code development pipelines.* Software Quality Journal, 32(1), 77–92.
8. Haffar, A., & Nuseibeh, B. (2019). *Challenges of testing visual programming environments: A systematic review.* Empirical Software Engineering, 24(5), 2891–2921.
9. Jain, V., & Kumar, P. (2023). *Automation in low-code/no-code application testing using AI and NLP.* Journal of Emerging Software Technologies, 17(2), 98–113.
10. Katalon, Inc. (2024). *Scriptless test automation for low-code platforms: A technical whitepaper.* Katalon Research Labs.