International Journal of Scientific Research and Engineering Development-– Volume 8 Issue 5, Sep-Oct 2025
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Comparative Study of Application Development - Low Code/No Code Method V/S Traditional Method

Nihal M Jagirdar*, Afreen Mujawar**
*(Department of Computer Science and Engineering, SIET, Vijayapura
Email: anihajagirdar.2000@gmail.com)
** (Department of Computer Science and Engineering, SIET, Vijayapura
Email: mujawarafreen19@gmail.com)

----------**********----------

Abstract:

This paper presents a systematic comparative study between traditional application development and Low-Code/No- Code (LCNC) development approaches. Using a Fitness Tracker application as a case study, two implementations were created: a traditional web application using Django (Python, HTML, CSS, SQLite) and a low-code Canvas app using Microsoft PowerApps connected to Dataverse. We evaluate both approaches across development time, code volume, technical complexity, perfor- mance, scalability, security, cost, and maintainability. Experi- mental measurements and qualitative analysis show that LCNC platforms dramatically reduce development effort and accelerate deployment at the cost of flexibility and vendor-dependence; conversely, traditional methods provide greater control and long- term extensibility. We conclude with practical recommendations and propose a hybrid strategy that leverages both approaches.

Keywords— **Low-Code, No-Code, PowerApps, Django, Data- verse, Software Development, Comparative Study**

----------**********----------

I. INTRODUCTION

The demand for faster software delivery and democratized application development has driven the emergence of Low- Code/No-Code (LCNC) platforms in industry. Traditional de-velopment frameworks (e.g., Django, Spring Boot) provide comprehensive control over architecture, performance, and customization but often require significant developer expertise and longer delivery cycles. LCNC platforms, such as Microsoft PowerApps, Mendix, and OutSystems, offer drag-and-drop UI composition, visual logic, and managed backend services that allow citizen developers to create production-capable applications with minimal hand-written code.

This research performs an empirical comparative study by building a Fitness Tracker application using both methodologies. The Fitness Tracker was selected because it covers a representative set of functionalities (authentication, CRUD for activities, dietary logs, and weight tracking) that allow a fair evaluation of typical application development concerns.

II. BACKGROUND AND MOTIVATION

Organizations, especially small and medium enterprises, face pressure to deliver digital solutions rapidly while keeping costs under control. LCNC platforms promise to reduce time- to-market by enabling rapid prototyping and lower development effort. However, concerns remain around customization, vendor lock-in, scalability limitations, and long-term maintenance. Prior studies highlight the trade-offs between LCNC and traditional development: LCNC is often preferred for internal tooling and prototypes, while traditional frameworks remain the default for complex, high-performance systems [1]– [3].

III. PROBLEM STATEMENT AND OBJECTIVES

A. Problem Statement

The choice between LCNC and traditional development methods is non-trivial. Organizations need empirical evidence and practical guidelines to decide which approach fits a particular project. There exists limited direct comparative work where the *same* application is implemented using both paradigms and evaluated across multiple dimensions.

B. Objectives

1) Implement a Fitness Tracker application using Django (traditional) and PowerApps + Dataverse (LCNC).
2) Create a comparison framework to evaluate development time, code volume, performance, scalability, security, cost, and maintainability.
3) Produce recommendations and explore a hybrid approach.

IV. RELATED WORK

Research into LCNC adoption shows consistent benefits in development speed and accessibility, but frequent concerns about integration with legacy systems and vendor lock-in [1], [7]. Usability studies indicate higher productivity for non- expert users on LCNC platforms, while experienced developers prefer traditional coding for fine-grained control [5]. Market analyses (Forrester, Gartner) recognize the maturity of platforms like Microsoft Power Platform for enterprise scenarios [8]. Studies comparing specific projects implemented both ways (e.g., NexusBRaNT) found reduced complexity with LCNC but decreased flexibility for specialized needs [6].

V. METHODOLOGIES

A. Design Choices

To ensure an equitable comparison, both implementations were designed to provide equivalent features:

- User authentication and profile management.
- Fitness activity logging (type, duration, calories).
- Dietary entry logging (food items, calories, macro nutrients).
- Weight entry history with date tracking.

B. Technologies Selection

TABLE I: Technology choices for both approaches

Aspect	Choice
Traditional Framework	Django (Python)
Database (Traditional)	SQLite (local development)
Frontend (Traditional)	Django templates, HTML/CSS
LCNC Platform	Microsoft PowerApps (Canvas app)
LCNC Database	Microsoft Dataverse
Authentication (Traditional)	Django auth (session-based)
Authentication (LCNC)	Microsoft Entra ID (M365 SSO)

C. Implementation Steps

1) Django (Traditional): A Django project was created with a 'fitness' app. Core models included `UserProfile`, `FitnessActivity`, `DietaryLog`, and `WeightEntry`. Views, forms, and templates were written to provide CRUD operations (limited to Create and Read for scope control). Static CSS created a consistent UI.

2) PowerApps + Dataverse (LCNC): Dataverse tables mir- roring the Django models were created. A Canvas app was built with screens for registration, login (via M365), activity addition, diet logging, and weight tracking. Power Fx expressions handled filtering, basic validation, and data submission. The app was published to be available on web and mobile.

D. Testing and Validation

Both applications underwent a functional test suite verifying:

- Account creation and login
- Adding fitness activities, dietary logs, and weight entries
- Viewing and filtering records
- UI responsiveness on desktop and mobile view ports

VI. SYSTEM ARCHITECTURE

Figure 1 presents a conceptual architecture comparison. Replace the placeholder image with your actual diagrams.

VII. DATA SCHEMA

The data schema for both implementations shares the same logical entities:

- **UserProfile:** user id, name, DOB, height, baseline weight, fitness level.
- **FitnessActivity:** user id (FK), date, activity type, duration, calories burned.
- **DietaryLog:** user id (FK), date, food item, calories, carbs, proteins, fats.
- **WeightEntry:** user id (FK), date, weight.

Using an identical logical schema ensures fairness of comparison.

VIII. RESULTS

We summarize both quantitative and qualitative differences observed during development and testing.

A. Development Effort and Code Volume

TABLE II: Development metrics

Metri	Django	PowerApps
Development Time	2.5 weeks	4–5 days
Lines of Code / Expressions	~1500 LOC	~50Power Fx expr.
Deployment Complexity	Manual server setup	Automated cloud hosting

B. Performance Benchmarking

Benchmarking was performed in controlled conditions (local dev host for Django; cloud-hosted Dataverse for PowerApps). Results are indicative and depend on network and hosting environment.

TABLE III: Performance benchmarking (approximate)

Operation	Django (local)	PowerApps + Dataverse
Average page load (home)	0.9 s	1.4 s
DB insert (single record)	0.05 s	0.12 s
Retrieve 50 records	0.08 s	0.15 s
Deployment time	1–2 hours	<15 minutes

C. Feature Integration Comparison

TABLE IV: Feature comparison summary

Feature	Django (Traditional)	
Authentication	Django auth, custom roles	M365 / Entra ID / SSO
UI Customization	Full control (HTML/CSS)	Drag and drop components
Data Storage	SQLite	Dataverse
Integration	Any API (flexible)	Easy MS integration
Scalability	High	Managed but platform Limits
Maintenance	Requires dev expertise	Platform -managed Updates
Cost Model	Open Source	Subscription

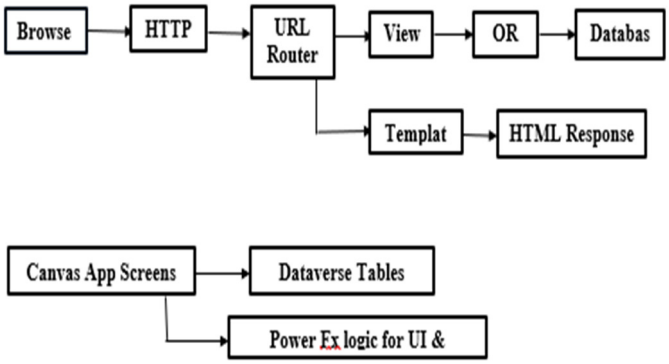

Fig. 1: Comparative architecture: Django (above) versus Low Code No Code (below)

IX. DISCUSSION

A. Advantages of LCNC

LCNC significantly lowers the barrier to entry and accelerates the development lifecycle. Key advantages observed include:

- Rapid prototyping and deployment.
- Reduced hand-coded logic and fewer syntactic errors.
- Built-in connectivity to enterprise services (Office365, Azure).
- Managed hosting and security compliance features.

B. Limitations of LCNC

Despite benefits, LCNC comes with trade-offs:

- Vendor lock-in: apps and data tied to vendor ecosystem.
- Limited ability to implement specialized algorithms or custom logic.
- Platform-specific limits (API call limits, delegation is- sues).
- Long-term cost can grow with scale (licensing, storage).

C. Advantages of Traditional Development

Traditional frameworks like Django provide:

- Full control over architecture, optimization, and deployments.
- Portability across hosting providers and no vendor lock- in.
- Easier integration of custom algorithm and complex business logic.

D. Limitations of Traditional Development

These include longer development cycles, greater reliance on skilled developers, and more effort for deployment and maintenance.

X. SECURITY CONSIDERATIONS

Security is central in applications that handle personal health and fitness data. Observations:

- Django: developers must explicitly implement secure practices (CSRF protection, proper session handling, encryption at rest, TLS). This yields flexibility but requires expertise.
- PowerApps/Dataverse: platform provides built-in encryption, role-based access control, and compliance certifications (GDPR, ISO), but visibility into low-level security controls is limited.

For regulated scenarios (medical data), organizations must evaluate data sovereignty and compliance implications before choosing LCNC.

XI. COST ANALYSIS

A high-level cost comparison:

- Traditional: minimal software licensing costs (open- source), but hosting, DevOps, and developer salaries are primary expenses.
- LCNC: lower initial setup and faster delivery, but recur- ring licensing and per-user / per-capacity costs grow with scale.

A total cost of ownership (TCO) model should be constructed for each organization based on expected scale, required SLAs, and compliance needs.

XII. RECOMMENDATIONS AND HYBRID STRATEGY

For many organizations, a hybrid approach offers best value:

- Use LCNC for internal tools, rapid UI prototyping, and citizen-developer driven features.
- Implement core systems, heavy computation, and sensitive processing with traditional frameworks.
- Ensure exportable data models and well-defined API contracts to avoid lock-in.

XIII. LIMITATIONS

This study deliberately limited scope to CRUD-driven features and did not integrate wearable device streams, large-scale load testing, or advanced analytics modules. Benchmarks are measured in a controlled environment and serve as relative indicators rather than exhaustive performance claims.

XIV. CONCLUSION AND FUTURE WORK

This paper compared two development paradigms through a Fitness Tracker application. LCNC (PowerApps + Dataverse) drastically lowered development time and code volume while offering rapid deployment and integration in the Microsoft ecosystem. Traditional development (Django) afforded greater performance, scalability, and flexibility at the expense of longer development cycles and greater developer effort. We recommend a hybrid development strategy

combining LCNC's agility with traditional backends where appropriate.

Future work should:

- Extend experiments to additional LCNC platforms (Mendix, OutSystems) and traditional frameworks (Flask, Spring Boot).
- Perform large-scale load and stress testing.
- Integrate device telemetry (wearables) and ML-driven analytics.
- Conduct economic TCO studies across multi-year horizons.

REFERENCES

[1] J. Berkemeier, A. Mu¨ller, and T. Schneider, "Adoption of Low-Code Platforms in SMEs," *Journal of Information Systems and Technology Management*, vol. 17, no. 3, pp. 45–60, 2020.

[2] A. Zimmermann, "Low-Code Platforms in Digital Transformation: Opportunities and Risks," in Proc. 15th Int. Conf. Digital Enterprise Transformation, 2020, pp. 112–120.

[3] OutSystems, "The State of Application Development Report: Low-Code 2021," OutSystems Industry Report, 2021.

[4] P. Mohagheghi, S. Sæther, and T. Sta˚lhane, "Software Quality in Low- Code Development: Opportunities and Challenges," *Empirical Software Engineering*, vol. 26, no. 5, 2021.

[5] J. Nachtigall, F. Langer, and H. Vogelsang, "Usability Evaluation of Low-Code vs. Traditional Platforms: An Empirical Study," in Proc. 45th Int. Conf. Software Engineering (ICSE), 2023, pp. 188–198.

[6] D. Aveiro, L. Guerreiro, and J. Tribolet, "Traditional vs. Low-Code Development in the NexusBRaNT Experiment," *Enterprise Modelling and Information Systems Architectures*, vol. 18, 2023.

[7] Deployd Research, "Cost Analysis of Low-Code vs. Traditional Development," Deployd Industry Report, 2025.

[8] Gartner, "Magic Quadrant for Enterprise Low-Code Application Plat- forms," Gartner Research Report, 2025.

