International Journal of Business Quantitative Economics and Applied Management Research
Volume-6, Issue-12, 2021 ISSN No: 2349-5677

CHALLENGES OF NO CODE AND LOW CODE APPLICATIONS

Suhas Hanumanthaiah
Independent Research

Abstract

This paper examines the increasing significance of low-code and no-code development platforms (LCDPs) as a means to address the rising demands for rapid application development and digital transformation. These platforms offer user-accessible visual environments that facilitate the creation of software applications with appealing user interfaces and responsive designs, all while demanding minimal programming expertise. This study explores the characteristics, challenges, and optimal practices linked to low-code and no-code applications, gathering insights from online developer communities and empirical assessments. The study identifies key challenges in the adoption and upkeep of low-code and no-code applications, including limitations in customization, complexities in integration, and the accumulation of technical debt. Customization poses a notable obstacle, as developers frequently encounter difficulties in adapting user interfaces and services provided by LCDPs. Furthermore, this paper emphasizes the importance of user experience, noting that usability problems can impede the adoption of low-code platforms. The paper also puts forth best practices to tackle these challenges, emphasizing code quality, thorough documentation, efficient testing, and continuous enhancement. Additionally, it highlights the necessity of strategies to handle technical debt, such as using duplicated code pattern mining algorithms. The importance of selecting an appropriate platform and ensuring a favorable developer experience for sustained maintenance success is emphasized in the paper. The research indicates that, while low-code platforms provide considerable advantages, it is essential to carefully consider their drawbacks and implement well-considered strategies to address maintenance-related issues.

Keywords: Low-Code Development(LCD), No-Code Development, Rapid Application Development, Microsoft Power App, OutSystems, Model-Driven Engineering, Digital Transformation

I. INTRODUCTION TO NO-CODE AND LOW-CODE DEVELOPMENT

In recent years, low-code development (LCD) platforms have gained significant traction as a solution to meet the growing demand for rapid application development in an increasingly digital world. Low-code development has been growing rapidly, with industry analysts like Gartner and Forrester predicting promising growth for its use [1]. Technology giants including Microsoft, Mendix, and OutSystems have launched their own LCD platforms to capitalize on this trend [1]. This growth reflects the need for more efficient application development methodologies in a business landscape that demands digital transformation.

Low-code software development (LCSD) represents an emerging paradigm that combines minimal source code with interactive graphical interfaces to promote rapid application development [2]. Low-code platforms enable the creation and deployment of fully functional applications by mainly using visual abstractions and interfaces and requiring little or no procedural code [3]. This approach shifts the development paradigm from traditional coding to a more visual and component-based methodology.

A key objective of LCSD is to democratize application development, making it accessible to software practitioners with diverse backgrounds [2]. This democratization is critical in an environment where technical talent is scarce and business demands for new applications continue to rise. Low-code software development platforms provide solutions for these challenges by aiming to produce flexible and less costly programs in a shorter time using drag-and-drop components through visual interfaces without requiring deep programming knowledge [4].

According to research findings, LCD provides a graphical user interface that allows users to drag and drop components with little or even no code. Additionally, the equipment of out-of-the-box units (such as APIs and components) in LCD platforms makes them easier to learn and use, while also speeding up the development process [1]. This acceleration of development is particularly valuable in domains that require automated processes and workflows.

II. CHARACTERISTICS OF LOW-CODE AND NO-CODE PLATFORMS

Low-code and no-code platforms share several distinctive characteristics that differentiate them from traditional development environments. Low-code development platforms (LCDPs) provide user-friendly visual environments to create software applications with attractive UI, responsive designs, and require minimal programming skills [5]. These platforms abstract away much of the technical complexity that typically accompanies software development.

The OutSystems Platform exemplifies these characteristics, serving as a development environment composed of several domain-specific languages (DSLs) that allow users to model different perspectives such as interfaces and data models, define custom business logic, and construct process models [6]. This multi-faceted approach enables developers to address various aspects of application development through specialized visual tools.

EasyFL, described as the first low-code federated learning platform, demonstrates how these platforms enable users with various levels of expertise to experiment and prototype applications with little coding while ensuring flexibility and extensibility for customization through unified simple API design, modular design, and granular training flow abstraction [7]. This balance between simplicity and customization capability is a hallmark of effective low-code platforms.

With only a few lines of code (LOC), such platforms empower users with many out-of-the-box functionalities to accelerate experimentation and deployment. These practical functionalities include heterogeneity simulation, comprehensive tracking, distributed training optimization, and seamless deployment [7]. Compared to other platforms, a solution like EasyFL requires just three LOC (at least 10 times less) to build a vanilla application and can expedite distributed training by 1.5 times [7].

The efficiency gains from low-code platforms can be substantial. In addition to reducing development time, such platforms improve the efficiency of deployment,[7] which addresses another critical bottleneck in the application lifecycle. The OutSystems Platform specifically enables fast mobile and desktop application development and deployment, hinging on visual development of the interface and business logic as well as easy integration with data stores and services [8].

III. CHALLENGES OF LOW-CODE AND NO-CODE APPLICATIONS

Despite their advantages, low-code and no-code applications present several significant challenges that organizations must navigate. These challenges span technical limitations, customization issues, integration complexities, and long-term maintainability concerns.

3.1.Technical and Customization Challenges

Given that LCSD is relatively a new paradigm, it is vital to understand the challenges developers face during their adoption of LCSD platforms [2]. Research has identified customization as one of the most significant hurdles. More than 40% of questions about low-code development are about customization, indicating that developers frequently face challenges with customizing user interfaces or services offered by LCSD platforms [2].

The topic of "Dynamic Event Handling" under the "Customization" category is both the most popular (in terms of average view counts per question) and the most difficult. This means that developers frequently search for customization solutions such as how to attach dynamic events to a form in low-code UI, yet most (75.9%) of their questions remain without an accepted answer [2]. This difficulty in resolving customization issues suggests that the platforms may not be providing adequate documentation or support for advanced customization scenarios.

Even in specific domains like recommender systems, these tools are complex and difficult to personalize or fine-tune if developers want to improve them for increasing the relevance of the retrievable recommendations [3]. This limitation can constrain the effectiveness of applications built on these platforms when they need to address unique or evolving business requirements.

3.2. User Experience and Adoption Challenges

User experience issues can significantly hamper the adoption of low-code platforms. For instance, the DSL for process modeling (Business Process Technology or BPT) in OutSystems had a low adoption rate and was perceived as having usability problems hampering its adoption. This was particularly problematic given the language maintenance costs [6]. When platforms fail to provide intuitive and efficient interfaces, they undermine their own value proposition of simplifying development.

Existing platforms are often complex to use and require a deep understanding of the underlying concepts, which imposes high barriers to entry for beginners, limits the productivity of researchers, and compromises deployment efficiency [7]. This complexity contradicts the fundamental goal of making application development more accessible.

To address these issues, platform providers must focus on improving the developer experience. When OutSystems improved their BPT language, evaluations conducted with 25 professional software engineers showed significant improvements: an increase of the semantic transparency

from 31% to 69%, an increase in the correctness of responses from 51% to 89%, an improvement in the System Usability Scale score from 42.25 to 64.78, and a decrease of the NASA Task Load Index score from 36.50 to 20.78. These differences were statistically significant, suggesting the new version of BPT significantly improved the developer experience compared to the previous version [6].

3.3 Integration and Data Management Challenges

Integration with existing systems and data management present another layer of challenges. Research has identified four major categories of challenges that developers face with low-code platforms: Customization, Platform Adoption, Database Management, and Third-Party Integration [2]. These integration challenges can limit the ability of low-code applications to work within complex enterprise environments.

Data integration increasingly means accessing a variety of NoSQL stores. Unfortunately, the diversity of data and processing models, that make them useful in the first place, is difficult to reconcile with the simplification of abstractions exposed to developers in a low-code platform [8]. This tension between simplification and power/flexibility is a fundamental challenge in the low-code development space.

Moreover, NoSQL data stores rely on a variety of general purpose and custom scripting languages as their main interfaces. Some providers have addressed this by building polyglot data access layers that use SQL with optional embedded script snippets to bridge the gap between low-code and full access to NoSQL stores [8]. This approach demonstrates how platforms must evolve to balance simplicity with the ability to leverage advanced data management capabilities.

3.4 Maintenance and Quality Challenges

Maintaining low-code applications over time presents unique challenges. The lower skill floor required by Visual Programming Languages (VPLs) entails that programmers are more likely to not adhere to best practices of software development, producing systems with high technical debt, and thus poor maintainability. Duplicated code is one important example of such technical debt. Observations have shown that the amount of duplication in the OutSystems VPL code bases can reach as high as 39% [9].

Research has found that most questions about low-code development are related to the development phase, and low-code developers also face challenges with automated testing [2]. The lack of robust testing capabilities can compromise the long-term quality and reliability of applications built on these platforms.

Traditional software development methods struggle to produce flexible solutions to dynamic and changing demands on time. Additionally, there are problems such as finding qualified human resources and high costs in writing and updating corporate program codes that can be considered complex on a large scale [4]. While low-code platforms aim to address these challenges, they introduce their own maintenance complexities that must be managed.

IV. BEST PRACTICES FOR LOW-CODE APPLICATION MAINTENANCE

To address the challenges of low-code application maintenance, several best practices have emerged from research and industry experience. These practices focus on code quality, thorough documentation, effective testing, and continuous improvement.

4.1 Addressing Code Duplication and Technical Debt

One of the primary maintenance challenges in low-code applications is managing technical debt, particularly in the form of duplicated code. Literature on duplicated code detection for Visual Programming Languages (VPLs) is very limited. To address this, researchers have proposed novel and scalable duplicated code pattern mining algorithms that leverage the visual structure of VPLs to not only detect duplicated code but also highlight duplicated code patterns that explain the reported duplication [9]. Implementing such tools as part of the maintenance workflow can help identify and reduce technical debt over time.

4.2 Improving User Experience and Platform Selection

Choosing the right platform and ensuring a positive developer experience are crucial for long-term maintenance success. Research findings suggest that developers should consider whether the characteristics of low-code development are appropriate for their projects before adoption [1]. Not all projects are suitable for low-code development, and forcing a misalignment can lead to significant maintenance challenges.

The end users' background with the platform has a relevant impact on the final concrete syntax choices and achieved usability indicators [6]. This suggests that platforms should be selected based not just on features, but on their alignment with the skills and perspectives of the team that will maintain the applications.

4.3 Advanced Strategies for Complex Applications

For more complex applications, advanced strategies may be needed. Choosing the most suitable computational strategy for a given usage is a difficult task which should be automated. Besides, the most efficient solutions may be obtained by the use of several strategies at the same time. This motivates the need for a transparent multi-strategy execution mode for model-management operations [10].

An overview of different computational strategies used in the model-driven engineering ecosystem shows the benefits of mixing strategies for performing a single computation. This approach has led to the design of multi-strategy model-management systems [10]. These advanced approaches can help maintain performance and functionality as applications grow in complexity.

4.4 Security Considerations

Ensure the selection of backend data store is within enterprise managed servers. By creating accessible connectors and templates for business users, consistent solutions can be implemented. Additionally, adding IT oversite to deployment process, by adding IT Architecture team review of application before deployment will help in looking for antipatterns and duplication in design.

4.4.1 Authentication and Authorization

Strong authentication and authorization mechanisms are fundamental to securing No-code and low-code development platforms(NCLCDP) applications. Authentication verifies the identity of users, while authorization determines what resources and functionalities users are allowed to access. NCLCDPs should offer secure authentication methods such as multi-factor authentication (MFA) and integration with identity providers [5]. Furthermore, the platforms should provide granular authorization controls to restrict user access based on roles and permissions. Role-based access control (RBAC) is a common approach that assigns permissions to roles and then assigns users to those roles

4.4.2 Access Control Best Practices

Access control is a critical aspect of security in NCLCDPs, ensuring that only authorized users can access sensitive data and functionalities. Implementing robust access control mechanisms can prevent unauthorized access, data breaches, and other security incidents [11].

Principle of Least Privilege

The principle of least privilege (PoLP) is a fundamental access control principle that states that users should only be granted the minimum level of access necessary to perform their job functions. In NCLCDPs, this means assigning users to roles with specific permissions and restricting access to sensitive data and functionalities based on those roles [11]. By adhering to PoLP, organizations can minimize the potential damage caused by insider threats or compromised accounts.

Multi-Factor Authentication

Multi-factor authentication (MFA) adds an extra layer of security to the authentication process by requiring users to provide multiple forms of identification [12]. In addition to a username and password, MFA may require users to enter a code sent to their mobile device or use a biometric scanner. MFA significantly reduces the risk of unauthorized access, even if an attacker obtains a user's password. NCLCDPs should support MFA and encourage its use for all users, especially those with privileged access.

Regular Access Reviews

Access rights should be reviewed regularly to ensure that users still require the level of access they have been granted. Access reviews should be conducted at least annually or when there are changes in job roles or responsibilities. During access reviews, organizations should identify and remove unnecessary or excessive access rights to maintain a secure environment.

Audit Logging and Monitoring

Audit logging and monitoring are essential for detecting and responding to security incidents. NCLCDPs should provide comprehensive audit logs that track user activities, access attempts, and system events. These logs should be monitored regularly for suspicious activity, such as failed login attempts, unauthorized access to sensitive data, or unusual system behavior [13]. Security information and event management (SIEM) systems can be used to automate log analysis and incident detection.

4.4.3. Data Protection Best Practices

Data protection is paramount in NCLCDPs, as these platforms often handle sensitive information such as customer data, financial records, and intellectual property. Implementing robust data protection measures can prevent data breaches, comply with privacy regulations, and maintain customer trust [14].

Data Encryption

Data encryption is a fundamental data protection technique that renders data unreadable to unauthorized parties. NCLCDPs should provide encryption capabilities for data at rest and data in transit. Data at rest refers to data stored on servers, databases, and storage devices, while data in transit refers to data transmitted over networks [15]. Encryption algorithms such as Advanced Encryption Standard (AES) and Transport Layer Security (TLS) should be used to protect data.

Data Masking and Tokenization

Data masking and tokenization are techniques used to protect sensitive data by replacing it with fictitious or non-sensitive values. Data masking replaces sensitive data with realistic but fictional data, while tokenization replaces sensitive data with unique tokens that have no intrinsic value [14]. These techniques can be used to protect sensitive data in non-production environments, such as development and testing environments.

Data Loss Prevention

Data loss prevention (DLP) measures can be implemented to prevent sensitive data from leaving the organization's control. DLP solutions monitor data in use, data in transit, and data at rest to detect and prevent unauthorized data transfers. NCLCDPs should integrate with DLP solutions or provide built-in DLP capabilities to protect against data leakage.

Regular Data Backups

Regular data backups are essential for disaster recovery and business continuity. NCLCDPs should provide automated backup mechanisms to ensure that data is backed up regularly and stored securely. Backups should be tested regularly to ensure that they can be restored in the event of a data loss incident.

4.4.4. Compliance Best Practices

Compliance with relevant regulations and standards is crucial for NCLCDPs, especially those handling sensitive data in regulated industries such as healthcare and finance. NCLCDPs should provide features and tools to help organizations comply with regulations such as HIPAA, GDPR, and PCI DSS [16].

HIPAA Compliance

The Health Insurance Portability and Accountability Act (HIPAA) sets standards for protecting the privacy and security of protected health information (PHI). NCLCDPs used in healthcare

applications must comply with HIPAA requirements, including implementing security safeguards to protect PHI from unauthorized access, use, and disclosure [16].

GDPR Compliance

The General Data Protection Regulation (GDPR) sets standards for protecting the personal data of individuals in the European Union (EU). NCLCDPs that process the personal data of EU residents must comply with GDPR requirements, including obtaining consent for data processing, providing data access and deletion rights, and implementing data protection measures [16].

PCI DSS Compliance

The Payment Card Industry Data Security Standard (PCI DSS) sets standards for protecting credit card data. NCLCDPs that handle credit card data must comply with PCI DSS requirements, including implementing security controls to protect cardholder data from unauthorized access and use [17].

Regular Security Assessments

Regular security assessments, such as vulnerability scans and penetration testing, should be conducted to identify and address security vulnerabilities in NCLCDP applications. Vulnerability scans automatically scan applications for known vulnerabilities, while penetration testing involves ethical hackers attempting to exploit vulnerabilities to assess the application's security posture [18]. Security assessments should be conducted regularly, especially after major code changes or platform updates.

V. ADOPTION CONSIDERATIONS AND FUTURE DIRECTIONS

As organizations consider adopting low-code and no-code platforms, they should evaluate several factors to ensure success. Research indicates that low-code platforms can facilitate the building of secure and scalable applications with outstanding features and can address various challenges faced by organizations [5]. However, the decision to adopt should be made with a clear understanding of the specific challenges that will be addressed.

The dependence on technology, digital transformation, and the need to work remotely are increasing day by day, and these trends are predicted to increase further. The desire to digitize every object also leads to the need to develop or update many applications software [4]. Low-code platforms are well-positioned to address these needs if implemented thoughtfully.

Platforms like EasyFL aim to increase the productivity of researchers and democratize technology to wider audiences [7]. This democratization trend is likely to continue across various domains, making specialized capabilities more accessible to non-specialists.

VI. CONCLUSION AND RESEARCH GAPS

The review of literature on no-code and low-code applications reveals several significant insights while also highlighting important research gaps. Research findings indicate that LCD may provide a graphical user interface for users to drag and drop with little or even no code;

the equipment of out-of-the-box units in LCD platforms makes them easy to learn and use as well as speeds up the development; LCD is particularly favored in domains that have the need for automated processes and workflows; and practitioners have conflicting views on the advantages and disadvantages of LCD [1].

The research suggests that low-code platforms offer significant benefits in terms of development speed and accessibility. However, they also present challenges in customization, integration, and long-term maintenance that must be addressed through thoughtful platform selection, development practices, and maintenance strategies.

Notable gaps in the existing research include:

1. Limited comparative analysis of specific platforms like Power Apps, Appian, and OutSystems, particularly regarding licensing costs and size limitations
2. Insufficient documentation of best practices specifically for no-code application maintenance
3. Lack of detailed deployment process comparisons across platforms
4. Limited empirical studies on the long-term maintenance costs of applications built on low-code platforms

Future research should address these gaps to provide more comprehensive guidance for organizations adopting low-code and no-code approaches. In particular, more rigorous comparative studies of leading platforms would help organizations make more informed decisions based on their specific requirements and constraints.

Study findings offer implications for low-code practitioners, platform providers, educators, and researchers [2]. By addressing the identified challenges and following emerging best practices, organizations can maximize the benefits of low-code development while mitigating its potential drawbacks.

In conclusion, while low-code and no-code platforms offer promising capabilities for accelerating application development and making it more accessible, successful implementation requires careful consideration of their limitations and thoughtful approaches to addressing maintenance challenges. With appropriate strategies, these platforms can effectively support digital transformation initiatives and help organizations respond more quickly to changing business requirements.

REFERENCES

1. Y. Luo, P. Liang, C. Wang, M. Shahin, J. Zhan, "Characteristics and Challenges of Low-Code Development: The Practitioners' Perspective," International Symposium on Empirical Software Engineering and Measurement, 2021. https://doi.org/10.1145/3475716.3475782
2. M. A. A. Alamin, S. Malakar, G. Uddin, S. Afroz, T. Haider, A. Iqbal, "An Empirical Study of Developer Discussions on Low-Code Software Development Challenges," IEEE Working Conference on Mining Software Repositories, 2021. https://doi.org/10.1109/MSR52588.2021.00018
3. C. D. Sipio, D. D. Ruscio, P. T. Nguyen, "Democratizing the development of

recommender systems by means of low-code platforms," ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, 2020. https://doi.org/10.1145/3417990.3420202
4. E. Şahïnaslan, O. Sahinaslan, M. Sabancıoglu, "Low-code application platform in meeting increasing software demands quickly: SetXRM," AIP Conference Proceedings, 2021. https://doi.org/10.1063/5.0042213
5. K. Talesra, N. G. S., "Low-Code Platform for Application Development," International Journal of Applied Engineering Research, 2021. https://doi.org/10.37622/ijaer/16.5.2021.346-351
6. H. Henriques, H. Lourenço, V. Amaral, M. Goulão, "Improving the Developer Experience with a Low-Code Process Modelling Language," ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, 2018. https://doi.org/10.1145/3239372.3239387
7. W. Zhuang, X. Gan, Y. Wen, S. Zhang, "EasyFL: A Low-Code Federated Learning Platform for Dummies," IEEE Internet of Things Journal, 2021. https://doi.org/10.1109/JIOT.2022.3143842
8. A. Alonso et al., "Building a Polyglot Data Access Layer for a Low-Code Application Development Platform," IFIP International Conference on Distributed Applications and Interoperable Systems, 2020. https://doi.org/10.1007/978-3-030-50323-9_6
9. M. Terra-Neves, J. Nadkarni, M. Ventura, P. Resende, H. Veiga, A. Alegria, "Duplicated code pattern mining in visual programming languages," Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2021. https://doi.org/10.1145/3468264.3473928
10. J. Philippe, H. Coullon, M. Tisi, G. Sunyé, "Towards transparent combination of model management execution strategies for low-code development platforms," ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, 2020. https://doi.org/10.1145/3417990.3420206
11. S. Hosseinzadeh, S. Virtanen, N. Rodrguez, J. Lilius, "A semantic security framework and context-aware role-based access control ontology for smart spaces," None, 2016. https://doi.org/10.1145/2928294.2928300
12. S. Atiewi et al., "Scalable and Secure Big Data IoT System Based on Multifactor Authentication and Lightweight Cryptography," Institute of Electrical and Electronics Engineers, 2019. https://doi.org/10.1109/access.2020.3002815
13. A. Anis, M. Zulkernine, S. Iqbal, C. Liem, C. Chambers, "Securing Web Applications with Secure Coding Practices and Integrity Verification," None, 2018. https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00112
14. M. Yadav, M. Breja, "Secure DNA and Morse code based Profile access control models for Cloud Computing Environment," Elsevier BV, 2019. https://doi.org/10.1016/j.procs.2020.03.317
15. S. Didla, A. Ault, S. Bagchi, "Optimizing AES for Embedded Devices and Wireless Sensor Networks," None, 2007. https://doi.org/10.4108/tridentcom.2008.10409
16. M. Farhadi, H. M. Haddad, H. Shahriar, "Compliance Checking of Open Source EHR Applications for HIPAA and ONC Security and Privacy Requirements," Annual International Computer Software and Applications Conference, 2019.

https://doi.org/10.1109/COMPSAC.2019.00106
17. U. T. Mattsson, "Protecting Web Based Applications: A Best Practices Guide," None, 2008. https://doi.org/10.2139/ssrn.1308922
18. R. Mahmood, Q. Mahmoud, "Evaluation of Static Analysis Tools for Finding Vulnerabilities in Java and C/C++ Source Code," arXiv.org, 2018. https://doi.org/None

ABBREVIATIONS

LCDPs	low-code and no-code development platforms
LCD	Low-Code Development
LCSD	Low-code software development
DSLs	Domain-specific languages
LOC	Lines of Code
VPLs	Visual Programming Languages
NCLCDP	No-code and Low-code Development Platform
MFA	Multi-Factor Authentication
RBAC	Role Based Access Control
PoLP	Principle of Least Privilege
SIEM	Security Information and Event Management
AES	Advanced Encryption Standard
TLS	Transport Layer Security
DLP	Data Loss Prevention
HIPAA	Health Insurance Portability and Accountability Act
PHI	Protected Health Information
GDPR	General Data Protection Regulation
EU	Europe Union
PCI DSS	Payment Card Industry Data Security Standard