BULGARIAN ACADEMY OF SCIENCES
PROBLEMS OF ENGINEERING CYBERNETICS AND ROBOTICS • 2024 • Vol. **82**, pp. 35-47
p-ISSN: 2738-7356; e-ISSN: 2738-7364
https://doi.org/10.7546/PECR.82.24.03

Overview of Low-Code Technologies and Foundations for Architectural Best Practices

Borislav K. Shumarov[1], Ivan G. Garvanov[1]

[1] University of Library Studies and Information Technologies, Sofia, Bulgaria
Emails: borislav.shumarov@gmail.com, i.garvanov@unibit.bg

Abstract: Low-code development platforms are transforming the software development landscape by enabling rapid application delivery through visual interfaces and declarative programming. However, the adoption of these platforms introduces new challenges in maintaining code quality and establishing development standards. This paper presents a comprehensive study of low-code technologies, focusing on foundational guidelines for best practices and architectural design. Through market analysis and a review of an empirical study of OutSystems practitioners, we review critical gaps in current development standards and foundation for solutions. Our previous survey revealed, that 78% of low-code developers support the implementation of standardized code style guidelines, similar to traditional programming environments. We establish a basis for theoretical framework based on a 3-Layer Canvas architecture, together with concrete guidelines for maintaining scalable and maintainable architecture for low-code applications. The study includes specific recommendations for preventing architectural antipatterns, managing dependencies, and implementing effective code organization strategies. Our results contribute to both the theoretical understanding of low-code development and provide practical guidance for organizations adopting these technologies. The research highlights that while low-code platforms simplify development processes, they require adaptation of established software engineering principles to ensure long-term application sustainability.

Keywords: *Low-code development platforms, Software architecture, Best practices, OutSystems, Code style guidelines, Enterprise applications, Software quality*

1. Introduction

The software development landscape is experiencing a paradigm shift with the emergence of low-code development platforms (LCDPs). As organizations face increasing pressure to digitize operations and deliver applications rapidly, traditional software development approaches often struggle to meet these demands due to resource constraints, technical complexity, and lengthy development cycles. Low-code technologies have emerged as a potential solution, promising to democratize application development through visual interfaces and declarative programming approaches while maintaining the robustness required for enterprise applications.

Despite being present for some time already, low-code software development is still a relatively unknown field in software development. Furthermore, despite the growing adoption of low-code platforms, there remains a significant gap in our understanding of what software engineering principles and best practices should be adapted and applied in low-code environments [1]. While these platforms aim to simplify development, they introduce new challenges in maintaining code quality, ensuring architectural integrity, and establishing standardized development practices [2]. In contrast to the well-known software applications [3, 4], low-code development platforms have no clear and comprehensive guidelines specifically tailored to low-code development potentially undermining the long-term maintainability and scalability of applications built on these platforms.

This paper addresses these challenges through three main objectives. First, we provide a comprehensive overview of what constitutes a modern low-code development platform and the current low-code technology landscape, examining market trends and identifying key players shaping the industry. Second, we examine empirical research investigating the needs and challenges faced by low-code practitioners, particularly within the OutSystems ecosystem, regarding development standards and best practices. Third, we establish theoretical foundations for architectural design in low-code environments, on the basis of a review of theoretical guidelines and validation principles.

The study particularly focuses on the OutSystems platform as a representative case study, while drawing broader implications for low-code development in general. Through this multi-faceted approach, we aim to bridge the gap between traditional software engineering practices and the emerging requirements of low-code development.

The findings presented in this paper contribute to both the theoretical understanding of low-code development and provide practical guidance for practitioners. By outlining foundational guidelines for best practices in low-code development, we aim to support organizations in leveraging these technologies

effectively while maintaining software quality and architectural integrity, while also paving the way for future deeper and more concrete research in the area.

2. Low-code technologies. Software development guidelines and low-code systems

In recent years, the landscape of software development has undergone a significant transformation with the emergence of low-code technologies. These platforms promise to revolutionize how applications are built, offering rapid development capabilities, and reducing the traditional dependency on extensive coding expertise. This section explores the foundational aspects of low-code technologies, examines their growing prominence in the software industry, and dives into the topic of low-code software development standards and guidelines and best practices.

The first section about the broader notion of Low-Code Software Development describes the paradigm shift from conventional coding methodologies. At its core, low-code aims it enables developers to create applications through visual interfaces and declarative programming rather than extensive hand-coding. This approach aims to democratize app development by empowering business users and citizen developers to participate actively in the creation process. Key features include intuitive drag-and-drop tools for GUI design, pre-built templates for common functionalities, and seamless integration capabilities with external systems.

There are numerous challenges, that low-code software development aims to solve. As clients' demands for quicker service and higher quality software are pushing the market forward, the need of new and better approaches to answering these demands is evident. This can be seen in Figure 1, showing the results of a survey by Forrester Research, conducted among 41 companies [5].

Most LCDPs today offer standard offline help, responsive UI, mobile capabilities, authentication, UI, repository control and role-based-access to data [6]. Based on the work of [7, 8], five of the most significant characteristics of an excellent LCDP can be outlined as follows:

- Simple GUI configurability;
- Convenient integration options to other tools and platforms;
- Mobile integration;
- Scalability across the organization;
- Support of the complete software lifecycle.

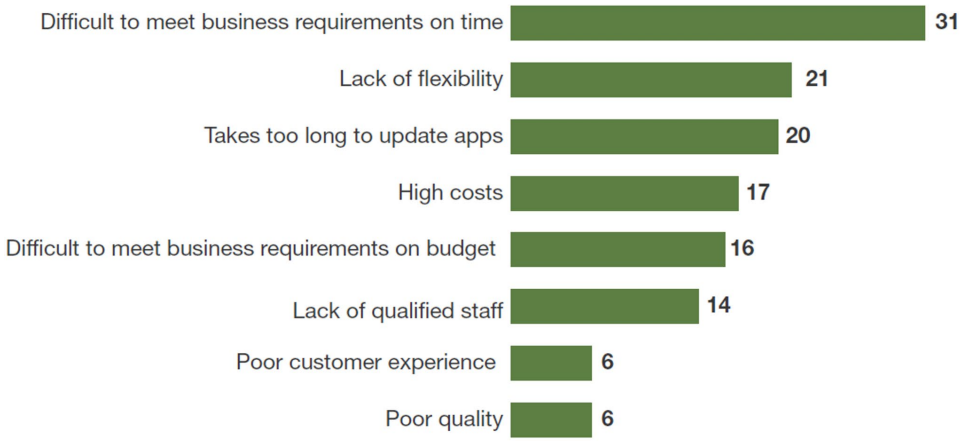

Fig. 1. Challenges for building custom applications using traditional coding with programming languages, frameworks, and middleware

The enterprise low-code development and one of the leading players in the area: the low-code provider OutSystems, as well as the adoption of low-code platforms across industries. It is primarily driven by their ability to accelerate time-to-market and foster innovation. Market leaders such as Mendix, OutSystems, Salesforce, and Microsoft have pioneered advancements in low-code technologies, each offering unique strengths and strategic approaches. Industry reports from renowned research firms highlight the exponential growth and transformative impact of low-code platforms, positioning them as catalysts for digital transformation initiatives globally.

Forrester and Gartner are two market research organizations, which have been tracking the low-code market in recent years. Fig. 2 depicts the top vendors of LCDPs, in the realm of professional application development and delivery (AD&D), according to Forrester [9].

They are positioned on the graph, according to two criteria: how strong they perceive the current offering is and how they assess the future strategy of the respective company. Forrester and Gartner [10, 11] seem to reach an almost perfect consensus about who the leaders in the industry are:

- Mendix,
- OutSystems,
- Salesforce, and
- Microsoft.

Fig. 2. Low-code development platform providers for AD&D professionals Q1 2019

After investigating the current market, functionalities and trends in the modern low-code software development, the next section further delves into the realm of software standards within low-code systems. It aims to outline a comprehensive study that evaluates the effectiveness of existing practices and methodologies in the low-code development environment. The study [12] is needed to better understand the needs of the low-code and OutSystems professionals, regarding best practices, guidelines and coding design patterns and their particular specificity.

For this reason, the study has been undertaken primarily among software development cadres and more specifically: the particular relatively small niche of OutSystems developers and professionals. The methodology involves the qualitative and quantitative research techniques aimed at capturing insights from

the practitioners and industry experts. Surveys conducted among a diverse group of stakeholders provide empirical data on the challenges and opportunities associated with low-code development. By examining participant feedback and industry benchmarks, the study aims to identify gaps in current practices and propose frameworks that enhance productivity and code quality.

Exploratory data analysis has been performed to the data from the questionaire. It provides descriptive statistics that summarize the central tendency, dispersion, and shape of a dataset's distribution. Overall, respondents express a moderately strong inclination toward the benefits of adhering to a code style guide, as evidenced by the mean score of 4.44 out of 5. This suggests a general consensus among the participants regarding the positive influence of code style guidelines on the overall quality of a codebase.

Furthermore, the respondents demonstrate a similar level of agreement when it comes to the belief that following a code style guide aids in understanding code written by others, particularly during refactoring or code reading situations. The mean score of 4.48 indicates a high degree of confidence in the ability of style guidelines to enhance code comprehensibility and maintainability.

Regarding the manner of maintaining a style guide, the responses reveal an interesting disparity. While a majority of respondents (75%) believe that maintaining the style guide in written form would be beneficial, a smaller proportion (25%) express concerns about its potential negative impact on the speed of software development. This discrepancy suggests that maintaining the style guide in written form could have varying implications across different teams or contexts.

Examining the experiences of the respondents, we find that a significant number have encountered challenges related to the code style guide. Approximately 58% of participants reported spending a substantial amount of time refactoring or enforcing rules to conform to the style guide. This indicates that some members of the team have had to invest additional effort to align with the guidelines, potentially impacting productivity and development timelines.
Moreover, around 41% of respondents admitted to experiencing difficulties in understanding the style guide. This highlights the importance of providing clear and concise documentation, as well as offering comprehensive training or support to ensure effective adoption and comprehension of the guidelines.

Additionally, a considerable portion of the respondents (42%) expressed the need to seek clarification or guidance on the code style guide, suggesting that clear communication and support systems are crucial for facilitating the implementation of the guidelines within the team.

Interestingly, a substantial number of participants (53%) also provided suggestions for improvements to the code style guide. This feedback highlights the iterative nature of style guides and the value of incorporating input from team members to enhance its effectiveness and relevance. It also shows, that the small

Code Style Guide, provided by the low-code platform provider (OutSystems) is limited and could be built upon.

Particulary interesting are the answers to question “I think that having a code style guide, similar to these for traditional tech stacks (Java, Python, JavaScript, C#, C++, etc.) is appropriate for Outsystems and other low-code technologies.” (Fig. 3). 78% of respondents agree with the statements, highlighting the need of a standardized code style guideline for OutSystems and similar technologies, despite being low-code.

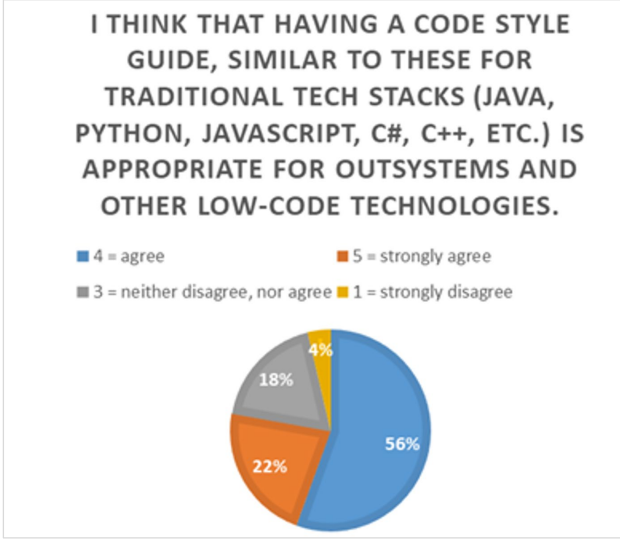

Fig. 3. I think that having a code style guide, similar to these for traditional tech stacks (Java, Python, JavaScript, C#, C++, etc.) is appropriate for Outsystems and other low-code technologies

As to which areas are the most sought after to be built upon, results gathered for question “I have suggestions for additional rules to the Outsystems basic best practices in the area of (multiple choice):” highlight the following areas (Fig. 4):

- Naming
- Back-End Design Patterns
- Front-Ent Design Patterns
- Architecture
- Formatting

These results are especially valuable for the low-code and OutSystems community and suggest a gap in the current body of knowledge and practices in the specified areas.

Fig. 4. I have suggestions for additional rules to the Outsystems basic best practices in the area of (multiple choice).

3. Theoretical foundations for conventions and good practices

Building on the insights gathered from the survey on code style guidelines, this section explores the essential theoretical principles and best practices necessary for establishing a standardized framework in low-code development. The overarching aim is to streamline design and development processes through adherence to established conventions and guidelines from the broader software engineering industry while integrating specific practices tailored to low-code development, particularly within OutSystems.

This chapter commences with a detailed exploration of typical architecture best practices for a system built with the LCDP OutSystems. First, the architectural pattern of the so-called 3-Layer Canvas is being reviewed. Next, the significance of validating the architecture's structure is highlighted to ensure robustness and adherence to design principles throughout various development stages.

The 3-Layer Canvas [13] is a structured approach in OutSystems architecture, organizing applications into three distinct layers. It is depicted on Fig. 5.

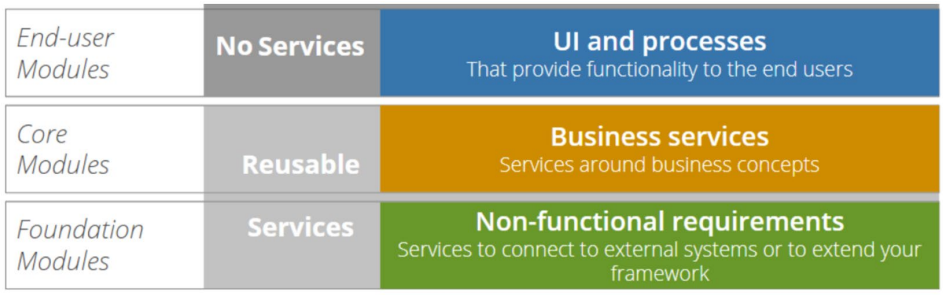

Fig. 5. 3-layer Canvas

It consists of End-user layer; Core layer and Foundation layer.

- End-user layer: This top layer focuses on user interactions through user interfaces and processes, utilizing core and foundational services. It includes role-based UIs to support specific use cases, business logic tailored to these use cases, auxiliary entities and structures for UI status, workflow definitions, and permissions for access control. Modules in this layer should remain independent and not provide services to other modules, ensuring total lifecycle independence.
- Core layer: This middle layer manages core business services, including services around business concepts, rules, entities, transactions, and widgets. These services should be system-agnostic and built on foundation services to abstract integration details. It isolates all reusable services or components, such as business processes available for use by users or other processes, core entities, business web blocks, change operations, audit trails, and integration logic. Permissions are defined to control service-specific access.
- Foundation layer: The foundational layer includes all reusable non-functional requirements like services for external system connections, UI pattern libraries, and themes. It covers normalized APIs, exception handling, single sign-on and session logic, structures for inputs and outputs, non-core entities, UI widgets, themes, layouts, exception flows, and common roles for domain-specific users.

An extended view of the 3-layer Canvas is depicted on Fig. 6.

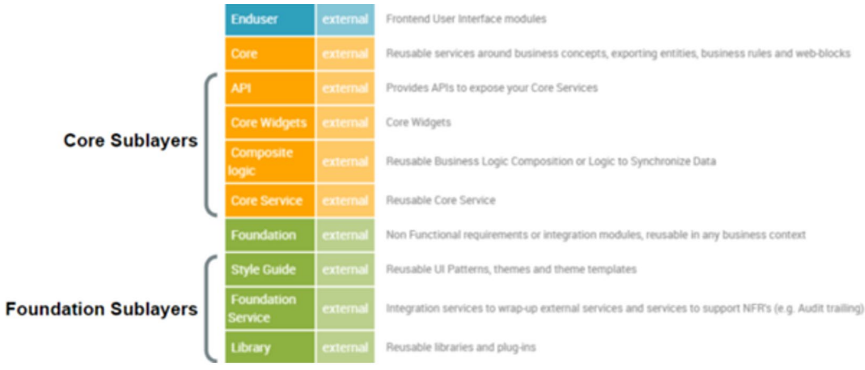

Fig. 6. 3-layer Canvas: extended overview

To ensure the integrity and efficiency of an architectural framework and to prevent the emergence of monolithic or convoluted structures, it is imperative to adhere to a set of prescribed guidelines and recommendations [14, 15].

- Prohibition of upward references across the three layers (see Fig. 7). Given the hierarchical nature of the architecture, foundational services must remain independent of core business concepts and reusable services should not depend on end-user interfaces. Upward references create clusters of interdependent modules, leading to circular dependencies. This not only enlarges the runtime footprint unnecessarily but also exposes modules to changes in other modules, compromising their lifecycle independence.

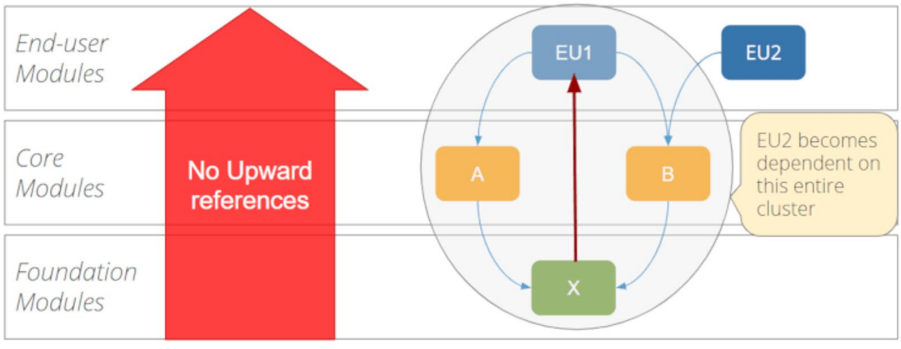

Fig. 7. Prohibition of upward references across the three layers

- Avoidance of side references among end user modules/layers (Fig. 8). End-user modules/layers must be strictly isolated and should not provide reusable services. This isolation ensures that end-user modules maintain their independence in lifecycle management, preventing any unintended dependencies and potential disruptions.

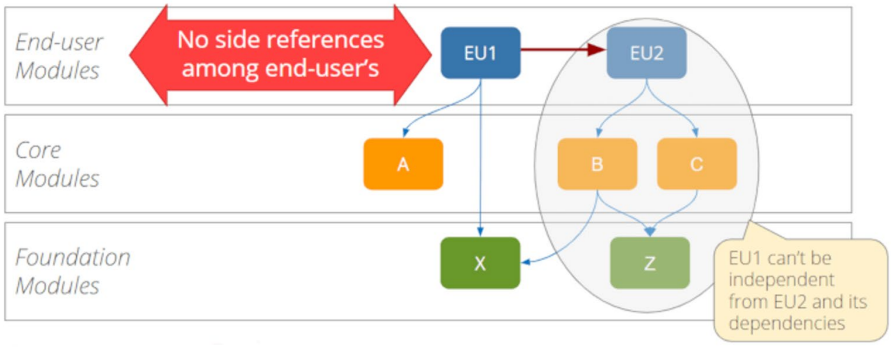

Fig. 8. Avoidance of side references among end user modules

- Prevention of circular references between core and foundational modules (see Fig. 9). The circular dependencies among modules are undesirable as they complicate code management and indicate improper abstraction of concepts. To avoid such cycles, modules that are strongly interconnected should either be consolidated or have their dependencies restructured to align with proper conceptual relationships. The following rules refer to strong references across modules or applications. This means, that circular references are still conceptually undesirable, but are not an issue by service actions or reference/redirect to screens.

Fig. 9. Prevention of circular references between core and foundational modules

4. Discussion and Conclusion

This research provides a comprehensive examination of low-code development platforms and establishes foundational guidelines for best practices in this rapidly evolving field. Through analysis of market trends, industry surveys, and empirical

research, several key findings emerge that contribute to the body of knowledge in software engineering and low-code development.

The study reveals a significant market shift towards low-code technologies, driven primarily by the need to address traditional software development challenges such as lengthy development cycles and resource constraints. Market leaders like Mendix, OutSystems, Salesforce, and Microsoft have emerged as pioneers in this space, as confirmed by both Forrester and Gartner analyses.

Our survey of OutSystems professionals yielded crucial insights into the practical needs and challenges of low-code development. The strong support for standardized code style guidelines (78% approval) indicates that despite the "low-code" designation, practitioners still value and require structured development approaches. The survey identified key areas requiring additional standardization, including naming conventions, back-end and front-end design patterns, architecture, and formatting.

Continuing with the architectural foundations for low-code development: the 3-Layer Canvas architecture provides a robust foundation for organizing low-code applications. The clear delineation of end-user, core, and foundation layers, coupled with strict guidelines for data flow and dependency constraints, together offer a practical approach to maintaining scalable and maintainable low-code applications.

These findings suggest that while low-code platforms simplify development processes, they do not eliminate the need for sound software engineering principles. Rather, they require adaptation and refinement of established best practices to suit the unique characteristics of low-code development environments. Future research should focus on developing more detailed guidelines for the identified areas of concern and investigating how traditional software engineering principles can be effectively translated into the low-code context.

The conclusions drawn from this study have significant implications for both practitioners and platform providers in the low-code development space. They highlight the importance of establishing and maintaining comprehensive development standards, even in environments designed to minimize traditional coding, and provide a foundation for future work in this rapidly growing field.

Acknowledgement

This work is supported by the Bulgarian National Science Fund, Project title “Innovative Methods and Algorithms for Detection and Recognition of Moving Objects by Integration of Heterogeneous Data”, KP-06-N 72/4/05.12.2023.

References

1. Alamin, M.A.A., Uddin, G., Malakar, S., Afroz, S., Haider, T., Iqbal, A.: Developer discussion topics on the adoption and barriers of low code software development platforms. Empirical Software Engineering 28, 4 (2023), https://doi.org/10.1007/s10664-022-10244-0.
2. Pinho, D., Aguiar, A., Amaral, V.: What about the usability in low-code platforms? A systematic literature review. Journal of Computer Languages, 74, 101185, (2023), https://doi.org/10.1016/j.cola.2022.101185.
3. Borissova, D., Mustakerov, I.: A framework for designing of optimization software tools by commercial API implementation. Int. Journal of Advanced Engineering, Management and Science, 2(10), 1790–1795 (2016).
4. Borissova D., Mustakerov, I., Bantutov, E.: Web-based architecture of a system for design assessment of night vision devices. International Journal of Information Science and Engineering, 7(7), 62–67 (2013).
5. Rymer, J.: The Forrester WaveTM: Low-Code Development Platforms for AD&D Pros, Q4 2017. (2017).
6. Richardson, C., Rymer, J.R.: New Development Platforms Emerge for Customer-Facing Applications. (2014).
7. Nepal, M.: Launchpads to Build Powerful Apps Easily. (2018).
8. Rymer, J.: Vendor Landscape: A fork in the road for low-code development platforms. (2017).
9. Rymer, J., Koplowitz, R.: The Forrester WaveTM: Low-code development platforms for AD&D professionals, Q1 2019. (2019).
10. Rymer, J., Seguin, B.: The state of low-code platform adoption, 2018. (2019).
11. Vincent, P., Iijima, K., Driver, M., Wong, J., Natis, Y.: Magic quadrant for enterprise low-code application platforms. (2019).
12. Shumarov, B.: Code style guidelines by low-code technologies and Outsystems. In: Knowledge Society and 21st Century Humanism The 20 th International Scientific Conference Sofia, 1st November 2023, 534–542 (2023).
13. OutSystems: The architecture canvas – OutSystems best practices, https://success.outsystems.com/documentation/best_practices/architecture/designing_the_architecture_of_your_outsystems_applications/the_architecture_canvas/, (2024).
14. Menezes, F.: Application architecture: Best practices for future-proofing your apps built with low-code, https://www.outsystems.com/blog/posts/application-architecture/, last accessed 2024/06/21.
15. OutSystems: Validating your application architecture – OutSystems best practices, https://success.outsystems.com/documentation/best_practices/architecture/designing_the_architecture_of_your_outsystems_applications/validating_your_application_architecture/, last accessed 2024/06/22.