Complex Systems Informatics and Modeling Quarterly (CSIMQ)
eISSN: 2255-9922
Published online by RTU Press, https://csimq-journals.rtu.lv
Article 200, Issue 36, September/October 2023, Pages 68–86
https://doi.org/10.7250/csimq.2023-36.04

Exploring Low-Code Development: A Comprehensive Literature Review

Karlis Rokis[1] and Marite Kirikova[2*]

[1] Alumni of Riga Technical University, 6A Kipsalas Street, Riga, LV-1048, Latvia
[2] Institute of Applied Computer Systems, Riga Technical University, 6A Kipsalas Street, Riga, LV-1048, Latvia

rokis.karlis@gmail.com, marite.kirikova@rtu.lv

Abstract. Low-code development provides the ability to create software using visual application development tools, predefined components, and automation, minimizing the reliance on manual coding. Furthermore, it enables individuals with limited programming expertise to actively contribute to software development in collaboration with information technology (IT) professionals, improving the alignment between customer or business requirements and the developed solution. This approach speeds up the development process, fosters closer collaboration between software development and business teams, and enhances the organization's ability to respond to business and market demands or changes. In this article, existing knowledge of low-code development from multiple perspectives is summarized, including the definitions, used tools, applied development lifecycles, application areas, potential benefits, challenges, and, consequently, relevant development and delivery principles.
Keywords: Low-Code Development, Low-Code Platforms, Low-Code Benefits, Low-Code Principles, Software Development.

1 Introduction

Low-code development is a modern approach to software development that utilizes visual tools, component reusability, and automation to minimize the need for manual coding and speed up the delivery process. Furthermore, low-code development enables a wide range of users, including non-professional developers from business backgrounds, to participate in software development. This fosters better collaboration between IT and business departments, improves responsiveness to business and market requirements, and helps address the shortage of skilled developers. It is important to note that the scope of low-code development extends beyond specific application domains, encompassing various domains such as employee, customer, or partner-facing web or

[*]Corresponding author

Reference: K. Rokis and M. Kirikova, "Exploring Low-Code Development: A Comprehensive Literature Review," Complex Systems Informatics and Modeling Quarterly, CSIMQ, no. 36, pp. 68–86, 2023. Available: https://doi.org/10.7250/csimq.2023-36.04

Additional information. Author's ORCID iD: M. Kirikova – https://orcid.org/0000-0002-1678-9523. PII S225599222300200X. Received: 18 May 2023. Accepted: 30 October 2023. Available online: 31 October 2023.

mobile applications, The Internet of Things (IoT), social media, manufacturing, marketing, and more.

According to [1], low-code development can be viewed as a tool-based approach. However, there are some software development and delivery peculiarities that distinguishes it from other approaches [2]. To effectively implement low-code development in an organization and capitalize on its potential advantages, it is relevant to identify principles, capabilities, and other factors that impact the realization of low-code development's potential. While the popularity of low-code development is growing, the research on it is scattered, focusing on a limited number of perspectives. The objective of the article is to summarize existing knowledge on low-code development from multiple perspectives – used tools, applied development lifecycles, application areas, potential benefits, and challenges and derive relevant development and delivery principles that would be useful for utilizing existing knowledge in the application of low-code development in companies and identifying areas of research for further enhancements in low-code development methods and tools. The following research questions are addressed by the article:

- RQ1: What is low-code software development?
- RQ2: What are low-code development platforms?
- RQ3: What development lifecycles are described in the literature related to low-code development?
- RQ4: What are the application areas of low-code development?
- RQ5: What are the benefits of low-code development?
- RQ6: What are the challenges of low-code development?
- RQ7: What are the low-code development principles?

To fulfill the objective and to answer the research question, a comprehensive literature review on low-code development was conducted. The remainder of this article is structured as follows: Section 2 describes the research method. Section 3 amalgamates the foundations and characteristics of low-code development including its definition (Subsection 3.1), low-code development platforms (Subsection 3.2), applied development lifecycles (Subsection 3.3), applications areas (Subsection 3.4), low-code benefits (subsection 3.5), and challenges (subsection 3.6). Section 4 points to the main principles to be followed in the use of low-code approach. Conclusions are provided in Section 5.

2 Methodology

A literature review serves as a valuable instrument for comprehending the existing knowledge within a particular domain, enabling the identification of gaps and directions in which further research is needed and it has been selected as the method for answering the research questions of this article. [3]. The review was carried out following the methodologies outlined by Levy and Ellis [3]. At the beginning, the understanding of the current status of the Body of Knowledge of the topic was gained, and the research goals and research questions were established. Then according to [3], the literature review process was executed consisting of three stages: (i) input, (ii) processing, (iii) output. The "input" stage covers aspects of the search and identification of high-quality literature, and the procedure for the collection of literature. During the "Processing" stage the literature is comprehended, applied, analyzed, synthesized, and evaluated. At the "Output" stage research findings are synthesized and these findings are articulated in the written literature review [3].

First, a keyword search in scientific databases was conducted, using relevant keywords as suggested by Levy and Ellis [3]. Afterward, a backward reference search was conducted [3]. In this step, the references of the articles identified during the keyword search were reviewed.

The initial keyword search was carried out in scientific databases – IEEE Xplore Digital Library, ACM Digital Library, ScienceDirect, and SpringerLink. The initial keyword-based search was conducted in the last quarter of 2022. Additionally, other electronic resources, such as Scopus, were utilized to access papers identified through the backward search method. Based on the research questions, specific keywords were identified, and a search string was formulated. An automatic search in databases was executed within the papers' title, abstract, and keyword fields. The obtained results were filtered to include only papers related to the information technology domain while removing irrelevant content types (e.g., posters). The papers retrieved in the search were validated against the selection criteria and any duplicate papers were eliminated. The inclusion and exclusion criteria used for paper selection are described in Table 1.

Table 1. Inclusion and exclusion criteria of the articles

Selection	No.	Criteria
Inclusion	I1	The article is related to low-code software development
Exclusion	E1	The article is not published in English
	E2	The full text of the article is not available
	E3	The paper does not contain answers to any of the research questions

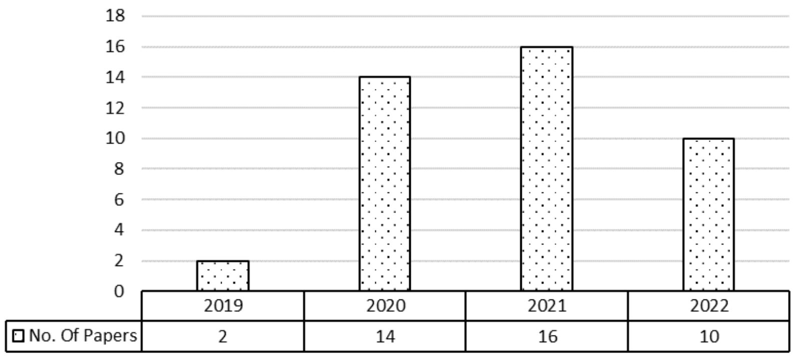
In the initial keyword search, database search engines retrieved a total of 281 papers that were relevant to the topic of low-code development. The number of articles found in each library is displayed in Table 2 in column "Number of results".

Table 2. Number of results per scientific library

Database	Number of results	Selected studies
IEEE	133	20
ACM	61	12
ScienceDirect	63	2
SpringerLink	24	5

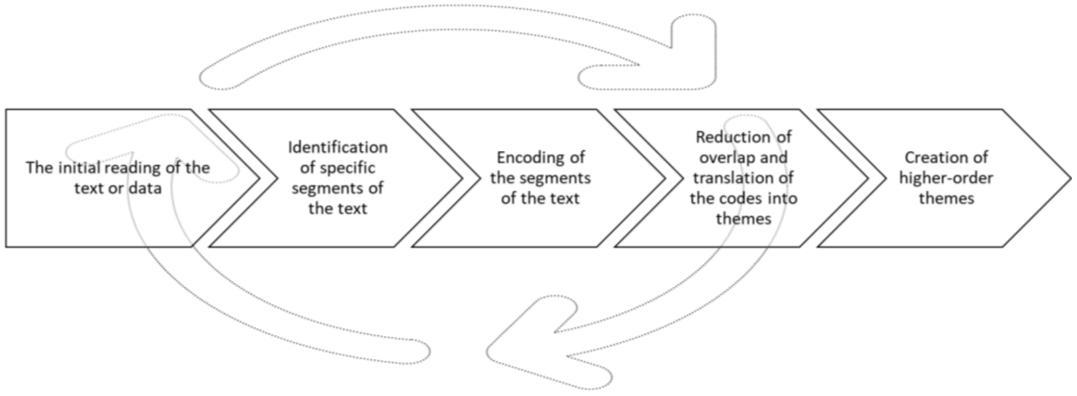
The obtained results were reviewed by applying the criteria defined in Table 1. Additionally, the full text was examined for multiple papers to ensure their relevance and applicability to the paper. Finally, a total of 39 articles were identified as relevant for the review from the keyword search step. The number of selected studies in each library is displayed in Table 2 in column "Selected studies". Then the backward search process was conducted. The backward search process began by searching for potential articles within the available resources. Afterward, inclusion and exclusion criteria were applied. The backward search step added three papers to the literature list. In total, 42 unique articles were selected from scientific databases for the literature review. The distribution of the selected papers by year is visualized in Figure 1. The data shows that papers concerning low-code development emerged in 2019, and there has been an increase in the number of articles over the past two years. This trend indicates that the topic is relatively new.

In the article, to summarize, consolidate, and compare the gathered information related to the defined research questions, the authors employed the research synthesis method known as thematic synthesis. The thematic synthesis method is applied to identify explanations or patterns essential to considered review questions through descriptive synthesis. This method involves identifying recurring themes across multiple studies, interpreting, and explaining them, and at the end drawing conclusions [4]. As proposed in [4] the process of thematic synthesis in software engineering consists of five steps, among which the authors move iteratively. These steps are visually represented in Figure 2. After selecting the relevant studies, *the sources are read*, and *data are extracted*. Then *descriptive labels – codes – are assigned* to chunks of the text, taking into account the context of the findings. Codes enable the organization and grouping of common data into categories and initiate themes. In this review, the codes were developed using an integrated approach that combines both inductive and deductive ("start list" of codes) development of codes. Then these *codes are translated into themes*, which involves grouping them into smaller sets to generate more meaningful units. These identified themes can then be explored and used to *interpret higher-order themes*.



	2019	2020	2021	2022
▫ No. Of Papers	2	14	16	10

Figure 1. Number of selected papers by year (at the end of 2022)



The initial reading of the text or data
Identification of specific segments of the text
Encoding of the segments of the text
Reduction of overlap and translation of the codes into themes
Creation of higher-order themes

Figure 2. Process of thematic synthesis (adapted from the description in [4])

The limitation of the structured literature review is related to subjectivity in literature selection, interpretation, and synthesis leading to biases [3]. The limitations of the thematic synthesis are related to subjective biases during data extraction and data coding, which can lead researchers to perceive patterns based on their expectations rather than the actual content of the text. Another concern is the risk of coding data at too general a level or out of context. Furthermore, the trustworthiness of the synthesis depends on the quality of the studies used [4]. To some extent, these biases were possible to overcome by involving researchers with different experience in low-code development.

3 Foundation and Characteristics of Low-Code Development

This section provides an answer to the research questions about the definition of low-code development, its lifecycle, as well as related characteristics and specifics.

3.1 Defining Low-Code Development (RQ1)

According to [5]–[8] the term “low-code” originates back to 2014 when it was defined in a Forrester Research paper called “New Development Platforms Emerge For Customer-Facing Applications” by Richardson C. et al. Since then, multiple definitions have been delivered in the current research literature.

In [9] by Waszkowski R., the term “low-code programming technique” is used. The author describes it as a derivate of fourth-generation programming (4GL) in combination with Rapid

Application Development (RAD) principles. The low-code programming technique allows programmers and practitioners to develop applications by focusing on designing aesthetics and functionality while reducing the effort required for coding [9].

In another paper, by Alamin et al. [10], low-code development is defined as a paradigm that enables minimal hand-coding, the use of visual programming with a graphical interface, and model-driven design for software development. Furthermore, the authors emphasize that low-code software development embodies the principles of End User Software Programming by enabling practitioners with diverse backgrounds and varying levels of software development experience to engage in software development activities [10]. Although the provided definition focuses more on the technical realization of low-code development, the paper establishes a connection between Agile methodology and low-code software development. From this follows that it would be important to respect development methodology principles when formulating a definition for low-code development.

Somewhat similar aspects in defining low-code development are described by Luo et al. [11]. According to this paper, practitioners define low-code development as a process that involves minimal coding effort, drag-and-drop capabilities, visual programming, the availability of pre-designed templates, and development that is friendly to non-professional developers [11].

Similar definitions appear in other peer-reviewed papers as well – [1], [6], [7], [12]–[22]. Consequently, to capture the multifaceted nature of low-code development and to ensure comprehensiveness, the following definition is proposed and used in this article: *Low-code software development is a development approach that enhances rapid, flexible, and iterative software development by enabling quick business requirements translation through visual programming with a graphical interface, visual abstraction, and minimal hand-coding; and involving practitioners with various backgrounds and software development experience*.

3.2 Low-Code Development Platforms (RQ2)

As indicated by the definitions provided above, low-code development is closely associated with low-code development platforms (LCDPs). Therefore, it can be claimed that low-code development is a tool-based approach [1]. LCDPs are platforms, as they include application development, deployment, lifecycle, and platform management features [8]. These platforms are cloud- (delivered through Platform-as-a-Service (PaaS) model) or on-premises-based platforms, on which low-code development is accomplished using visual tools, predefined components, and their customization and configuration [19], [23].

In general terms, a low-code platform is made of multiple layers and, from the architectural perspective, according to [19], there can be distinguished the following layers:

- Application layer. This layer is made of a graphical environment that includes toolboxes and widgets for user interface definition. It also includes mechanisms for authentication and authorization. Platform users engage with this layer, utilizing its modeling constructs and abstractions to develop applications and specify their behavior [19].
- Service integration layer. The role of this layer is to provide integration with and utilization of various services, such as application programming interfaces (APIs) and authentication mechanisms [19].
- Data integration layer. A layer dedicated to enabling integration, operation, and manipulation of data that can come from various sources [19].
- Deployment layer. A layer that concerns the deployment of the developed application on a dedicated cloud or on-premise environment. In collaboration with the service integration layer, the containerization and orchestration of the developed application are performed [19].

There is a vast number of low-code development platforms currently available. The authors in [14], based on Forrester Research paper called "Vendor Landscape: The Fractured, Fertile Terrain of Low-Code Application Platforms" by Richardson C. and Rymer J.[†], distinguished five major segments: (i) General-purpose platforms, (ii) Process app platforms, (iii) Database app platforms, (iv) Request handling platforms, and (v) Mobile app platforms. General-purpose platforms offer various features for software development, including ones for user experience, database, integration development, access control, deployment, and others. Process app platforms are used to develop coordination and collaboration applications with the visual process and case-modeling tools. Database app platforms specialize more in data management. Request-handling app platforms are dedicated to the creation of request processing, handling, and tracking apps. Mobile applications specialize in mobile app development with dedicated tools [14].

Low-code development platforms combine multiple traditional components and therefore include multiple features – supported functionalities and services. However, differences of supported functionalities among platforms exists. Various scientific literature sources [5], [13], [19], [24], [25] have introduced feature diagrams and lists to compare the commonalities and variabilities of platforms. However, by reviewing other peer-reviewed papers and applying the thematic synthesis approach, additional features supported by platforms were identified. In this article, these features are summarized and integrated with the feature lists introduced in the previously mentioned sources. It is important to note that not all low-code development platforms support all functionalities or address specific features to the same extent or in the same way. Thus, the significant variability of supported LCDP features poses a challenge in defining a consistent set of core features [5], [19], [24]. The obtained list of LCDP features is organized into two feature levels and is represented in Table 3. Top-level features are shown in column 1, corresponding base-level features are listed in column 2, and feature descriptions are given in column 3.

Table 3 Features supported by LCDP

Top-level features	**Features**	**Description**
Requirement modeling support	Requirement management tools	Built-in functionality for requirements specification. Platforms typically provide tools for requirements management, including data collection, checklist creation, user stories, and the ability to import them into sprint plans. This feature is important for ensuring the correct implementation of requirements, as well as for requirements traceability, and verification [10], [25]
Visual development tools	Visual designer	Visual modeling tool utilized to develop the software. Through it, all aspects of the software are developed by selecting, arranging, configuring, and connecting components [26], [27]
	Drag and drop approach	A drag-and-drop or "point and click" feature allows developers to create an app by simply grabbing and dragging the required elements to their desired positions on the interface [19]
	Integrated development environment	A dedicated integrated development environment for developers for a complete development lifecycle [27]
	Forms	The feature includes dashboards, custom forms, surveys, checklists, and more to enhance the user interface and user experience of the developed application [19]
	Advanced coding components	A feature that allows achieving further customization to build an extensive user interface and user experience, integrations, or custom algorithms supporting various technologies [11], [19], [24], [28]

[†] https://informationsecurity.report/Resources/Whitepapers/0eb07c59-b01c-4399-9022-dfc297487060_Forrester%20Vendor%20Landscape%20The%20Fractured,%20Fertile%20Terrain.pdf

Table 3 Continued

Top-level features	Features	Description
Reusability support	Predefined components	Pre-built components (various pallets of widgets, structures, models, services, connectors, and other components), and templates that are available for the development of applications [19], [27], [29]–[31]
	Predefined templates	
	Predefined configuration	A feature allowing to utilize the predefined configuration of settings and solutions [1], [12]
	Built-in workflows	A feature that provides a platform with predefined common reusable workflows that can be used for application development [19]
	Pre-built forms/reports/dashboards	Reusable set of common forms, reports, or dashboards available for editing and use in development [19], [32]
	Artifacts export mechanisms	Mechanisms to export created models or generated artifacts for future reuse [25]
	Artifacts import mechanisms	Mechanisms to import and reuse previously created or other external artifacts [25]
	Artifacts storage and future reuse	Feature for management of inbuilt or previously developed reusable artefacts locally or in cloud infrastructure [25]
Data source specification mechanisms	Data structure specification	A component that enables data structure specification using a conceptual modeling tool [5]
	Internal databases	A feature that allows data to be stored in the internal database [24]
	External data sources	A capacity to access external data sources using various mechanisms [24]
	Data binding	Feature for data binding mechanisms [32]
Interoperability support	Connection with data sources	A feature that supports the connection of application to various data sources – relational databases, non-relational databases, or files [19], [24]
	Interoperability with external services	A feature that provides functionality to incorporate various external services, commonly through APIs [19], [24]
Business logic specification mechanisms	Business rules engine	A feature that allows for establishing the logic and executing business rules helping to manage data according to requirements [8], [19]
	Graphical workflow editor	A feature that enables visual-based specification of business rules, where workflows are defined using graphical diagrams similar to business process model and notation [6], [19]
	AI-enabled business logic	A feature that is learning attributes behavior and replicates it conforming to artificial intelligence-based learning mechanisms [19]
Development automation features	Automatic development	A feature that enables the automatic generation of applications (for instance, automatically generating an application from a spreadsheet) [9], [12]
	Automation of low-level details	A feature that automates low-level concerns in application development, such as object-relational mapping, load balancing, data integrity, query optimization, services, messaging, authentication, and more [8], [19]
Collaborative development support	Online collaboration	Mechanisms for simultaneous work and collaboration by multiple developers and conflict management at runtime [19]
	Offline collaboration	Mechanisms that support multiple developers work locally in offline mode and commit changes to the server in which changes are merged [19]

Table 3 Continued

Top-level features	Features	Description
Artificial intelligence (AI)	Internal artificial intelligence components	A feature that enables internal components for the use of AI [24], [32]
	Integrability of external artificial intelligence services	A feature that enables the use of specialized APIs for AI services [24]
Testing and verification support	Automated testing	A feature for automated testing realization [12]
	Integration with testing platforms	Mechanisms to integrate external testing platforms for UI, API, and End-to-End automated testing [12]
	Requirements used in testing	Functionality to map system requirements to the test cases [12]
	Testing environment	A feature that covers inbuilt testing workbench, model checking, and validation tools [25]
Deployment support	Automatic deployment support	A tool to automate the deployment of the application [1]
	Deployment on a cloud	A feature that enables application deployment on a cloud infrastructure [19]
	Deployment on local infrastructures	A feature that enables local application deployment on an organization's infrastructure [19]
Security support	Application security	A feature that enables such applications' security mechanisms as confidentiality, integrity, and availability through authentication mechanisms, security protocols, and user access control [19]
	Platform security	A feature that enables security and role management to ensure confidentiality, integrity, and availability on a platform level [19]
Lifecycle-management features	Lifecycle-management components	Features that enhance collaborative work employ task, document, and requirement management as well as deployment, staging, maintaining, and other functionalities. Integration with external tools is also possible [10], [19], [22], [33]
	Repositories	A feature that supports repository and version control tasks of developed artifacts [19]
Analysis environment	Analysis Features	Dashboards, monitors, alarms, and other performance and measurement tools to gather statistics, identify code anti-patterns, analyze the performance, and monitor CPU, memory, and other components [1], [25], [34]
	Advanced Reporting	A feature that enables application usage reporting in a tabular and graphical way [19]
Extensibility	Easy to add modeling features	Mechanisms to add new features [25]
	Easy to modify the existing features	Mechanisms to refine or extend developed functionality [11], [25]
Scalability	Scalability on the number of users	Features to scale applications regarding the number of manageable active users [19]
	Scalability on data traffic	Features to scale applications regarding the data traffic [19]
	Scalability of data storage	Features to scale applications regarding data storage [19]
Other features	Developer Assistance	Mechanisms (for example, recommender systems) that assist developers in the development process [7], [33]

3.3 Low-Code Development Lifecycle (RQ3)

To deliver software through a low-code approach, multiple software development steps need to be performed. Any software development methodology can be applied for low-code development, but organizations frequently use fast delivery practices [35].

Typical low-code software development stages are described in papers [10] and [19]. The authors in [10] and [19] describe a low-code development approach consisting of seven phases to develop applications using low-code development platforms:

1. *Requirements and feasibility analysis*.
2. *Data modeling* – the application's data schema is defined and configured. This involves establishing the structure of entities, defining their relationships, implementing constraints, and identifying dependencies within the data model.
3. *Definition of the user interface* – in this phase, the application's forms, pages, and views are created and user roles and security mechanisms are configured.
4. *Implementation of business logic and workflows* – application workflows and business logic are defined and configured.
5. *Integration of external services* – required integrations with third-party services are established.
6. *Testing and deployment*.
7. *Customer feedback and additional features* – after application deployment, feedback from stakeholders is received, and, if required, changes and additional features are defined, followed by the subsequent cycle.

Multiple papers describe the use of Agile concepts within low-code development. According to researchers, Agile methodology and low-code software development conform well [10], as it promotes an iterative, frequent delivery approach with continuous stakeholder involvement. In the literature, the use of various frameworks is described. For instance, the concepts of Extreme Programming are applied [36]; the papers [19], [32] and [37] mention that low-code development fits well with the use of Scrum and Kanban methods; the use of Lean software development methodology is also proposed [38]. The relation between low-code development stages and Agile methodology is offered by [10], introducing the following approach:

1. *Requirements analysis* – requirements regarding the developed software are identified, and platform requirements management tools can be used.
2. *Planning* – analysis and planning of feasibility, schedule, interdependences, risks, and complexities are performed.
3. *Application design* – based on the defined requirements, the design is specified and reviewed by stakeholders considering architecture, modularity, and extensibility.
4. *Application development* – the application is being developed using low-code platform features specifying user interface, business logic, integration, and other elements.
5. *Testing* – tests are performed to verify the implementation of requirements within the software [12].
6. *Deployment* – the application is deployed either on-cloud or locally.
7. *Maintenance* – maintenance is provided once the application is released, and if functionality should be extended, additional features can be added, or a new iteration can be initiated.

Multiple peer-reviewed sources have mentioned the utilization of RAD within the context of low-code development (e.g., [9], [14], [20]). However, these papers did not elaborate on the specifics of the development lifecycle steps associated with RAD in the context of low-code development.

We can observe that, although the provided lifecycles differ in the naming of focus of several phases, all approaches have a dedicated phase for requirements, testing, and after-deployment activities. In our paper [39], the literature on life cycles has been amalgamated and the low-code development lifecycle model is proposed. The model consists of seven low-code development phases which are executed iteratively and are similar to the Agile development lifecycle phases: ideation and requirement analysis, planning, application design, development, testing, deployment, and maintenance [39].

3.4 Low-Code Development Application Areas (RQ4)

According to Bucaioni et al. [7], 21 application areas have been identified in peer-reviewed and grey studies. The most common are web, mobile, enterprise services, business processes, and IoT. Other areas include healthcare, education, databases, request handling, recommender systems, manufacturing, industrial training, domain-specific language (DSL) engineering, social media, process, marketing, desktop, blockchain, automotive, AI and aeronautics [7]. Additionally, the e-commerce and Extract-Transform-Load applications are mentioned in the literature [11].

Based on the available sources, currently, there are no restrictions reported concerning any application areas.

3.5 Benefits of Low-Code Development (RQ5)

There are various benefits identified in the peer-reviewed literature that businesses can gain from low-code development. Identified benefits are amalgamated in Table 4 and described in the following paragraphs.

Table 4. Identified low-code development benefits

ID	Benefit	Sources
BE1	Acceleration of the development cycle	[1], [5], [6], [10], [11], [13], [14], [19], [26], [27], [29], [32], [35], [40]–[42]
BE2	Involvement of citizen developers	[1], [5], [6], [10]–[12], [14], [19], [26], [35], [38], [41]–[43]
BE3	Decreased costs	[1], [5], [6], [11], [14], [19], [27], [32], [35], [40], [44]
BE4	Increased responsiveness to business and market demands	[1], [5], [6], [10], [35], [36]
BE5	Lowered maintenance effort	[5], [6], [10], [19], [36], [45]
BE6	Improved collaboration among the development team and business	[11], [41]
BE7	Promoted digital innovation	[6], [14]
BE8	Mitigation of shadow IT	[6]

Low-code development allows full application delivery faster to face time constraints [14], [19]. This highlights the first low-code development benefit - *acceleration of the development cycle* (BE1). One factor that contributes to this is the platforms' functionalities, which cover the entire development process and speed it up by minimizing the need for manual coding, visual application development, drag-and-drop capabilities, reusability, and automation [13], [27].

The benefits gained from the *involvement of citizen developers* (BE2) can be examined from various perspectives. First, it helps to mitigate the problem related to the lack of developers as citizen developers gain the opportunity to implement applications themselves or with the lower involvement of IT professionals. Additionally, from this perspective, it also contributes to the first benefit of providing an opportunity to deliver applications faster (BE1) [9]. Secondly, this contributes positively to reducing misinterpretation of requirements as citizen developers come from business backgrounds so that they can define and fulfill them precisely [10], [12].

Another advantage offered by low-code development is *cost reduction* (BE3). Cost savings are achieved through several means – (i) reduction in coding time, as low-code development enables faster application creation, (ii) promoting IT and business collaboration using the available resources more effectively, (iii) reduction of maintenance costs, (iv) limiting the need to hire new developers or outsource [5], [11], [32].

Business and market situation is dynamic and constantly changing, often with a tight timeline and narrow market window. Therefore, low-code development can be beneficial as it *increases responsiveness to business and market demands* (BE4). It can be realized due to its fast development characteristics and user feedback enabling the realization of new opportunities and meeting requirements [1], [36].

Low-code development *lowers maintenance effort* (BE5) by minimizing the occurrence of bugs and integration-related issues due to the use of predefined components. Simultaneously, low-code development supports the continuous evolution of requirements to ensure alignment between the developed application and business needs. Therefore, more attention can be paid to innovation [6], [36].

Another benefit of low-code development is *improved collaboration among the development team and businesses* (BE6). Low-code development platforms and approaches promote frequent collaboration between IT and business stakeholders throughout the entire development lifecycle. Furthermore, it is easier to collaborate as understandable visual models can be used, and feedback gathered. In such a way, the business side can directly contribute to the project and development team to identify the right work to do [11], [22], [31].

Low-code development promotes *digital innovation* (BE7) by empowering such cultural aspects as learning and experimentation, creativity, innovation, and fast time-to-market delivery. Furthermore, digital innovation is enhanced through increased collaboration between IT and business teams. This allows us to keep up with changing requirements by improving existing solutions, validating ideas, or addressing new challenges to remain competitive [6], [14], [18].

Low-code software development provides tools administrated by the organization to non-IT professionals. This approach *reduces the risk of the rise of shadow IT* (BE8), which is defined as unsanctioned IT solutions and systems that are used without the approval of the organization's IT department and therefore possess security, data governance, and other risks [6].

3.6 Challenges in Low-Code Development (RQ6)

Besides the potential benefit that can be obtained from low-code development, there are multiple challenges to consider. These challenges have been addressed in our previous publication [23], where 23 low-code development challenges were described, and their possible mitigation approaches were overviewed according to the Agile development phases. The challenges are depicted in Table 5.

Table 5. Challenges in low-code development adopted from [23]

Relation to Agile SDLC	Challenge
Requirement Analysis	Requirements specification; Changing requirements
Planning	Selection of the platform; Vendor lock-in
Application Design	Extensibility limitations; Interoperability; Consideration of scalability; UI design; Data storage design
Development	Implementation of business logic; Integration; No access to source code; Customization of UI; Debugging
Testing	Limited testing and analysis support; Dependence on third-party testing tools; Testing of non-functional requirements
Deployment	Performance; Configuration issues; Accessibility issues; Version control
Maintenance	Debugging; Use of maintenance features

4 Low-Code Development Principles (RQ7)

Besides using development platforms, low-code development usually includes changes in software development and delivery principles. As the comprehensive list of low-code development principles was not available, it was established based on knowledge obtained in the literature review aiming at the list of principles that can help to realize identified low-code development benefits (see Section 3.5). The principles were identified using the thematic synthesis qualitative data analysis method [4]. They are described further in this section, and, for each principle, the related works are listed to relate the principle to the sources that describe the ways the principle can be implemented. The following principles were identified:

1. Select the right low-code platform.
2. Comprehend and master the platform.
3. Embrace visual application development and utilization of predefined components, elements, and templates.
4. Enhance reusability.
5. Embrace platform automation capabilities.
6. Extend the functionality with additional customization when required.
7. Empower citizen developers and establish a fusion teams approach.
8. Promote IT-business collaboration.
9. Address the knowledge gap of citizen developers.
10. Establish governance.
11. Establish and follow an iterative development lifecycle.
12. Embrace the test-and-learn culture for innovation.
13. Support changing requirements.

Select the right low-code platform. Low-code development platform takes an important aspect of the low-code development approach. As stated by [19] low-code development platforms can help enterprises of any size, especially those with limited IT resources and budget, in scaling their organizations efficiently and rapidly delivering feature-rich products at an optimal cost. Hence, the selection of a platform should be approached thoroughly to avoid waste of time and resources [1], [10]. Furthermore, while most low-code platforms can be considered general-purpose platforms that support various use cases, there are also specific platforms for process apps, database-related segments, request handling, mobile development, and even IoT [14], [25]. For selecting the platform, the following suggestions are available in the literature:

- Use feature lists to facilitate comparison and selection. Such feature lists are presented in Section 3.2 of this article and in [13], [19], [24], [25];
- Apply the intended scenarios and recognize the organization's needs for platform selection [11].

It is also important to consider the aspect of vendor lock-in when evaluating low-code platforms. The paper [5] discusses that representations created among low-code platforms might not be compatible raising the risk of investment protection and deadlock if the platform's vendor stops the support of the platform.

Comprehend and master the platform. A low-code development platform integrates various traditional development components in a single environment and takes a significant role in low-code development. Initial development on low-code development platforms is typically straightforward and may not require formal training, but to release the full potential of the platform, users should possess knowledge and understanding of utilizing the provided features. Having a programming background can be advantageous in this context [5], [32], [35]. Hence, comprehension of the platform will determine the development output.

Learning to start using the platforms can be easy, and it usually does not require training for developers, and the effective way to become proficient is a learning-by-doing approach [16], [27],

[43]. Users should explore platform facilities and development environments that make the use of the platform easy. It is essential to understand how to create integrations with external services and different data sources using built-in connectors. Users should discover how to enhance integration possibilities by configuring custom connectors. Finally, the work principles of workflow definitions should be clear [19], [21], [27], [32]. Users should familiarize themselves with the use of lifecycle management and collaboration facilities, requirement management tools, component libraries, and repositories. Exploring the inbuilt testing facilities of the platform, understanding their principles, and investigating the potential integration of external testing tools are also important. Capabilities of cloud and on-premise deployment, as well as maintenance features, should be revealed [6], [10], [12], [19], [27], [28].

Embrace visual application development and utilization of predefined components, elements, and templates. Low-code platforms facilitate the delivery of applications through visual development, minimizing the need for manual coding. Therefore, in order to achieve fast application delivery without compromising quality, it is essential to effectively utilize visual application development, predefined components, elements, and templates provided by the platform [19], [27]. As presented by [36] and [27], such mechanisms allowed for delivering projects on time, which would not be possible using traditional methods.

The platforms offer visual tools for developing applications, encompassing the interface, data model, business logic, control and data flow, APIs, integration, and security model [27], [31]. Developers are provided with a visual environment that allows them to specify applications using provided components and elements through visual modeling, drag-and-drop functionality, and abstraction techniques. While there may be a need for some minor manual coding, for instance, for configuration, the significant reduction in extensive manual hand-coding leads to an increased pace of application delivery [12].

The use of predefined components, elements, and templates eliminates the need to create them from scratch or customize them for every project, resulting in time savings. In addition, settings configurations, for instance, component behavior, are also predefined [12], [28].

Additionally, maintenance, deployment, and other functionality, including scalability and extensibility mechanisms, are provided through an easy-to-use visual environment, thus reducing routine tasks and the effort for maintenance [19].

Provided mechanisms are non-professional developers-friendly and include assistance facilities (for instance, recommender systems) to support them in development activities [11], [33].

Enhance reusability. Low-code development emphasizes the principle of reusability, and platforms incorporate features to support it. Reusability allows for the efficient utilization of previously created artifacts, leading to increased productivity, faster delivery speed, and reduced maintenance effort [19], [24], [28]. Reuse principles are facilitated across various components, ranging from setting up the data model to defining UI elements. Additionally, reusability is supported during testing through reusable test data and cases. Furthermore, platforms offer repositories and handle version control [12], [19].

Distributed architectures are common systems architecture, and the principles of reusability promote that in low-code development. In such a way, the development, deployment, and maintenance efforts are reduced [12], [19].

Embrace platform automation capabilities. Essential automation mechanisms are incorporated into low-code development platforms, accelerating the application delivery process [1], [6]. Automation of low-code platforms is present throughout the whole lifecycle and can take care of the following:

- Generate applications automatically (for instance, from spreadsheets) [12];
- Take care of low-level architectural details [19], [27];
- Deal with such technical aspects as authentication, load balancing, consistency of business logic, integrity and security of the data, microservice creation, orchestration, and management [19];

- Convert applications interface from design tools (for instance, Figma[‡], InVision[§]) to a low-code platform [28];
- Test case generations and configurations (using such mechanisms as "record and play", UI testing frameworks, and others). It can be done using inbuilt features or external tools [12], [44];
- Transform an artifact into executable representation and generate code [19], [24], [38],
- Deploy application [5], [45];
- Scale and tune performance depending on the number of users [19];

Consequently, developers should take advantage of these automation opportunities whenever possible, enhancing development speed.

Extend the functionality with additional customization when required. Various platforms allow the extension of their user interface, flows, data services, connectors, and other functionality using additional customization features. This further customization is achieved using technologies such as HTML, CSS, JavaScript, Java, C#, Python, and others. Additional customization features enable a higher degree of personalization and facilitate the implementation of advanced requirements. However, developers should be careful with the implementation of advanced customization, and it should be only applied when required and when it delivers additional value, as every additional customization takes additional development time and increases the complexity of maintenance [11], [19], [22], [24], [35].

Empower citizen developers and establish a fusion teams approach. Empowering people (typically "business end-users") with minimal or without software engineering experience and merging with experienced IT professionals provides fast development and better alignment with business or customer needs [5], [6], [25]. According to Microsoft[**], such a formation can be considered a fusion team – a group that merges business, analytics, and technology professionals. Citizen developers are empowered to build apps quickly and are involved in other software development lifecycle phases. Moreover, their knowledge of business promotes that needs are addressed precisely [5], [15], [22]. Meanwhile, IT professionals deliver invaluable expertise in various software development aspects (e.g., security and risks) and are pivotal in delivering more advanced cases. They also ensure addressing non-functional and quality challenges [38], [46]. Additionally, architects and IT leaders are in charge of low-code development platform selection, adoption, configuration, and customization [7]. The formation of such teams also ensures that citizen developers can proactively turn for advice to more experienced colleagues [33].

Promote IT-business collaboration. The promotion of collaboration improves the understanding of requirements and enhances the accuracy of their implementation by reducing the path of requirements transfer [9], [10].

Both sides should be responsive, working within low-code development. Together fast translation of business requirements and, also, their refinement into a functional application can be achieved [10], [16]. Getting feedback faster and regularly is essential. It is important to engage in continuous feedback gathering and refinement through quick iterations [31]. For this, the low-code code nature of visual and quick development fits well, as such visual representation can improve communication [17], [22].

Ultimately, the collaboration benefits both developers and the business. Developers can allocate their resources to delivering the required functionalities, while the business receives precisely what they require.

Address the knowledge gap of citizen developers. Citizen developers, as non-professional developers from the business side with various backgrounds, might lack a background in software development. The knowledge of software development best practices and platform features should

‡ https://www.figma.com/
§ https://www.invisionapp.com/
** Microsoft. What are fusion teams and development? Retrieved from https://powervirtualagents.microsoft.com/en-us/fusion-teams-development/

be gained to adopt low-code development successfully and ensure the professional growth of citizen developers [19], [33]. To address the knowledge gap, appropriate training and learning-by-doing might be required to become proficient in the platform [9], [19], [43].

Establish governance. While citizen developers prioritize fast application delivery, there is a potential risk of neglecting security and other IT governance aspects. It is essential to take proactive actions to prevent these concerns and ensure that proper security measures and IT governance practices are in place [1], [46].

Establish and follow an iterative development lifecycle. Iterative and fast development methodologies with regular releases, rapid prototyping or minimum viable product creation, and continuous stakeholder involvement in gathering feedback are suggestable as this way, acceleration of release time and adjustments can be gained.

As shown in Section 3.3, current sources describe low-code development in use with concepts of Agile (Scrum, Kanban, Extreme Programming, Lean, DevOps) or Rapid Application Development methodologies [1], [14], [19], [20], [36]–[38]. However, other methodology can also be used for low-code development [35]. Nevertheless, the methodology's key characteristics are iterative manner, regular releases, prototype or minimum viable product creation, incrementally growing product, and validation with a customer. Furthermore, typically development process consists of multiple phases, and steps cover requirements analysis, planning, design, development, testing, deployment, and maintenance activities [10], [14], [31], [32].

Embrace the test-and-learn culture for innovation. A culture that values experimentation through test-and-learn approaches to enhance innovation is essential for business on the quick market, business opportunity realization, and digital transformation acceleration. An organization can leverage low-code development to facilitate the delivery of innovative ideas. However, these ideas need to originate from the collaborative efforts between the business and IT departments. The test-and-learn approach using low-code methods is well-suited for fast innovation [16], [18], [29]. The test-and-learn approach prescribes that people should be open to generating ideas, and their requirements (even if there is only perception about relevant features), building a minimum viable product, testing in real-life scenarios, obtaining feedback, and deciding on further continuation [6], [7], [18]. If business units are ready to develop their applications, timely adoption of low-code development ensures tools for non-professionals and thus reduces the risk of "shadow IT" [6], [18].

Support changing requirements. To retain customers and meet evolving business needs, organizations must have the ability to respond to changing markets and business situations. Low-code development, through agility, plays a crucial role in delivering this responsiveness. Developers need to be open-minded and adapt to supporting changing requirements, allowing organizations to quickly and effectively address market dynamics and meet business demands [5], [29], [35]. Furthermore, it has been demonstrated in practice by [36] that low-code development can deal with even daily requirements changes to provide a functional solution to accommodate client's needs and deliver a solution that meets their expectations.

The relations of identified low-code development principles and benefits and their correspondence to relevant business objectives are demonstrated in the earlier publication [39]; where, also, capabilities to follow identified principles are stated and their relation to low-code development lifecycle, low-code development platform, and other low-code development issues are demonstrated.

Conclusion

The objective of the article was to summarize existing knowledge on low-code development from multiple perspectives – used tools, applied development lifecycles, application areas, potential benefits, challenges, and relevant development and delivery principles.

The article amalgamates in an organized way knowledge from research papers on low-code development that had been published by the last quarter of 2022. With this, it helps other researchers to get an insight into:

- Current understanding of the concept of low-code development,
- Currently employed low-code development lifecycles,
- Currently available features of low-code development platforms,
- Benefits and challenges of low-code development.

By leveraging the capabilities of low-code development platforms, organizations can achieve acceleration of development, promote the involvement of citizen developers and collaboration between IT and business, enhance innovations and responsiveness to changing market dynamics, and lower costs, maintenance effort, and risks for shadow IT. However, it is essential to consider various challenges and possible changes in software development and delivery principles. Therefore, the article proposes how different aspects of low-code development can be organized in low-code development principles. For instance, those are principles related to platform selection and comprehension, embracing its full capabilities and features, promoting a culture of innovation, and experimentation as well as empowering citizen developers, and enhancing business-IT collaboration and iterative development lifecycle. These principles, as well as provided lists of platform features and low-code development benefits and challenges, can be used as a reference frame to suggest and evaluate new models, methods, and tools in low-code development areas.

Low-code development concept continues to grow in popularity and evolves, therefore described characteristics might require revisioning in the future. Nevertheless, this article can serve as a basis for further characterization of low-code development, expanding identified characteristics with more specific details or practices.

References

[1] S. Rafi, M. A. Akbar, M. Sánchez-Gordón, and R. Colomo-Palacios, "DevOps Practitioners' Perceptions of the Low-code Trend," Association for Computing Machinery (ACM), Sep. 2022, pp. 301–306. Available: https://doi.org/10.1145/3544902.3546635

[2] C. Richardson and J. R. Rymer, "New Development Platforms Emerge for Customer-Facing Applications," 2014. Available: https://www.forrester.com

[3] Y. Levy and T. J. Ellis, "A systems approach to conduct an effective literature review in support of information systems research," *Inf Sci*, vol. 9, pp. 181–211, 2006. Available: https://doi.org/10.28945/479

[4] D. S. Cruzes and T. Dybå, "Recommended steps for thematic synthesis in software engineering," in *International Symposium on Empirical Software Engineering and Measurement*, IEEE Computer Society, 2011, pp. 275–284. Available: https://doi.org/10.1109/esem.2011.36

[5] A. C. Bock and U. Frank, "Low-Code Platform," *Business and Information Systems Engineering*, vol. 63, no. 6, pp. 733–740, Dec. 2021. Available: https://doi.org/10.1007/s12599-021-00726-8

[6] R. Sanchis, Ó. García-Perales, F. Fraile, and R. Poler, "Low-code as enabler of digital transformation in manufacturing industry," *Applied Sciences (Switzerland)*, vol. 10, no. 1, Jan. 2020. Available: https://doi.org/10.3390/app10010012

[7] A. Bucaioni, A. Cicchetti, and F. Ciccozzi, "Modelling in low-code development: a multi-vocal systematic review," *Softw Syst Model*, 2022. Available: https://doi.org/10.1007/s10270-021-00964-0

[8] D. Di Ruscio, D. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, and M. Wimmer, "Low-code development and model-driven engineering: Two sides of the same coin?" *Softw Syst Model*, vol. 21, no. 2, pp. 437–446, Apr. 2022. Available: https://doi.org/10.1007/s10270-021-00970-2

[9] R. Waszkowski, "Low-code platform for automating business processes in manufacturing," in *IFAC-PapersOnLine*, Elsevier B.V., 2019, pp. 376–381. Available: https://doi.org/10.1016/j.ifacol.2019.10.060

[10] M. A. Al Alamin, S. Malakar, G. Uddin, S. Afroz, T. Bin Haider, and A. Iqbal, "An empirical study of developer discussions on low-code software development challenges," in *Proceedings - 2021 IEEE/ACM 18th*

International Conference on Mining Software Repositories, MSR 2021, Institute of Electrical and Electronics Engineers Inc., May 2021, pp. 46–57. Available: https://doi.org/10.1109/MSR52588.2021.00018

[11] Y. Luo, P. Liang, C. Wang, M. Shahin, and J. Zhan, "Characteristics and challenges of low-code development: The practitioners perspective," in *International Symposium on Empirical Software Engineering and Measurement*, IEEE Computer Society, Oct. 2021. Available: https://doi.org/10.1145/3475716.3475782

[12] F. Khorram, J. M. Mottu, and G. Sunyé, "Challenges & opportunities in low-code testing," in *Proceedings - 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS-C 2020 - Companion Proceedings*, Association for Computing Machinery, Inc, Oct. 2020, pp. 490–499. Available: https://doi.org/10.1145/3417990.3420204

[13] I. N. Oteyo, A. L. S. Pupo, J. Zaman, S. Kimani, W. De Meuter, and E. G. Boix, "Building smart agriculture applications using low-code tools: The case for discopar," in *IEEE AFRICON Conference*, Institute of Electrical and Electronics Engineers Inc., Sep. 2021. Available: https://doi.org/10.1109/AFRICON51333.2021.9570936

[14] C. Di Sipio, D. Di Ruscio, and P. T. Nguyen, "Democratizing the development of recommender systems by means of low-code platforms," in *Proceedings - 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS-C 2020 - Companion Proceedings*, Association for Computing Machinery, Inc, Oct. 2020, pp. 471–479. Available: https://doi.org/10.1145/3417990.3420202

[15] A. Calçada and J. Bernardino, "Experimental Evaluation of Low Code development, Java Swing and JavaScript programming," Association for Computing Machinery (ACM), Aug. 2022, pp. 103–112. Available: https://doi.org/10.1145/3548785.3548792

[16] P. M. Gomes and M. A. Brito, "Low-Code Development Platforms: A Descriptive Study," in *Iberian Conference on Information Systems and Technologies, CISTI*, IEEE Computer Society, 2022. Available: https://doi.org/10.23919/CISTI54924.2022.9820354

[17] X. He, L. Tang, and Y. Liu, "MerGen: A Smart Code Merging Approach for Automatically Generated Code; MerGen: A Smart Code Merging Approach for Automatically Generated Code," 2022. Available: https://doi.org/10.1109/COMPSACS54236.2022.00141

[18] V. Phalake, S. Joshi, K. Rade, and V. Phalke, "Modernized Application Development Using Optimized Low Code Platform," in *2022 2nd Asian Conference on Innovation in Technology (ASIANCON)*, IEEE, Aug. 2022, pp. 1–4. Available: https://doi.org/10.1109/ASIANCON55314.2022.9908726

[19] A. Sahay, A. Indamutsa, D. Di Ruscio, and A. Pierantonio, "Supporting the understanding and comparison of low-code development platforms," in *Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020*, Institute of Electrical and Electronics Engineers Inc., Aug. 2020, pp. 171–178. Available: https://doi.org/10.1109/SEAA51224.2020.00036

[20] A. Jacinto, M. Lourenço, and C. Ferreira, "Test mocks for low-code applications built with OutSystems," in *Proceedings - 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS-C 2020 - Companion Proceedings*, Association for Computing Machinery, Inc, Oct. 2020, pp. 530–534. Available: https://doi.org/10.1145/3417990.3420209

[21] A. Sahay, D. Di Ruscio, and A. Pierantonio, "Understanding the role of model transformation compositions in low-code development platforms," in *Proceedings - 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS-C 2020 - Companion Proceedings*, Association for Computing Machinery, Inc, Oct. 2020, pp. 431–435. Available: https://doi.org/10.1145/3417990.3420197

[22] F. Gurcan and G. Taentzer, "Using Microsoft PowerApps, Mendix and OutSystems in Two Development Scenarios: An Experience Report," in *Companion Proceedings - 24th International Conference on Model-Driven Engineering Languages and Systems, MODELS-C 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 67–72. Available: https://doi.org/10.1109/MODELS-C53483.2021.00017

[23] K. Rokis and M. Kirikova, "Challenges of Low-Code/No-Code Software Development: A Literature Review," in *Perspectives in Business Informatics Research. BIR 2022. Lecture Notes in Business Information Processing*, E. Nazaruka, K. Sandkuhl, and U. Seigerroth, Eds., Springer, Cham, 2022. Available: https://doi.org/10.1007/978-3-031-16947-2_1

[24] A. C. Bock and U. Frank, "In Search of the Essence of Low-Code: An Exploratory Study of Seven Development Platforms," in *Companion Proceedings - 24th International Conference on Model-Driven Engineering Languages and Systems, MODELS-C 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 57–66. Available: https://doi.org/10.1109/MODELS-C53483.2021.00016

[25] F. Ihirwe, D. Di Ruscio, S. Mazzini, P. Pierini, and A. Pierantonio, "Low-code engineering for internet of things: A state of research," in *Proceedings - 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS-C 2020 - Companion Proceedings*, Association for Computing Machinery, Inc, Oct. 2020, pp. 522–529. Available: https://doi.org/10.1145/3417990.3420208

[26] T. C. Lethbridge, "Low-Code Is Often High-Code, So We Must Design Low-Code Platforms to Enable Proper Software Engineering," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Science and Business Media Deutschland GmbH, 2021, pp. 202–212. Available: https://doi.org/10.1007/978-3-030-89159-6_14

[27] R. Martins, F. Caldeira, F. Sa, M. Abbasi, and P. Martins, "An overview on how to develop a low-codeapplication using OutSystems," in *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*, 2020.

[28] M. Bexiga, S. Garbatov, and J. C. Seco, "Closing the gap between designers and developers in a low code ecosystem," in *Proceedings - 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS-C 2020 - Companion Proceedings*, Association for Computing Machinery, Inc, Oct. 2020, pp. 413–422. Available: https://doi.org/10.1145/3417990.3420195

[29] N. Krishnaraj, R. Vidhya, R. Shankar, and N. Shruthi, "Comparative Study on Various Low Code Business Process Management Platforms," in *5th International Conference on Inventive Computation Technologies, ICICT 2022 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 591–596. Available: https://doi.org/10.1109/ICICT54344.2022.9850581

[30] J. Pacheco, S. Garbatov, and M. Goulao, "Improving Collaboration Efficiency Between UX/UI Designers and Developers in a Low-Code Platform," in *Companion Proceedings - 24th International Conference on Model-Driven Engineering Languages and Systems, MODELS-C 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 138–147. Available: https://doi.org/10.1109/MODELS-C53483.2021.00025

[31] S. Saay and T. Margaria, "Model-Driven-Design of NREn Bridging Application: Case Study AfgREN," in *Proceedings - 2020 IEEE 44th Annual Computers, Software, and Applications Conference, COMPSAC 2020*, Institute of Electrical and Electronics Engineers Inc., Jul. 2020, pp. 1522–1527. Available: https://doi.org/10.1109/COMPSAC48688.2020.00-39

[32] R. Arora, N. Ghosh, and T. Mondal, "Sagitec Software Studio (S3) - A Low Code Application Development Platform," in *2020 International Conference on Industry 4.0 Technology, I4Tech 2020*, Institute of Electrical and Electronics Engineers Inc., Feb. 2020, pp. 13–17. Available: https://doi.org/10.1109/I4Tech48345.2020.9102703

[33] L. Almonte, I. Cantador, E. Guerra, and J. De Lara, "Towards automating the construction of recommender systems for low-code development platforms," in *Proceedings - 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS-C 2020 - Companion Proceedings*, Association for Computing Machinery, Inc, Oct. 2020, pp. 451–460. Available: https://doi.org/10.1145/3417990.3420200

[34] I. P. Fernandes, M. Terra-Neves, and J. C. Seco, "Automated Refactoring of Unbounded Queries in Software Automation Platforms," in *Companion Proceedings - 24th International Conference on Model-Driven Engineering Languages and Systems, MODELS-C 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 417–426. Available: https://doi.org/10.1109/MODELS-C53483.2021.00065

[35] M. A. A. da Cruz, H. T. L. de Paula, B. P. G. Caputo, S. B. Mafra, P. Lorenz, and J. J. P. C. Rodrigues, "Olp—a restful open low-code platform," *Future Internet*, vol. 13, no. 10, Oct. 2021. Available: https://doi.org/10.3390/fi13100249

[36] J. Varajão, "Software Development in Disruptive Times," *Commun ACM*, vol. 64, no. 10, pp. 32–35, 2021.

[37] N. Jesse, "Agility eats legacy-the long good-bye," in *IFAC-PapersOnLine*, Elsevier B.V., Nov. 2019, pp. 154–158. Available: https://doi.org/10.1016/j.ifacol.2019.12.464

[38] Y. Wang, Y. Feng, M. Zhang, and P. Sun, "The Necessity of Low-code Engineering for Industrial Software Development: A Case Study and Reflections," in *Proceedings - 2021 IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 415–420. Available: https://doi.org/10.1109/ISSREW53611.2021.00112

[39] K. Rokis and M. Kirikova, "An ArchiMate-Based Thematic Knowledge Graph for Low-Code Software Development Domain," 2023, pp. 465–476. Available: https://doi.org/10.1007/978-3-031-42941-5_40

[40] W. Nurharjadmo, M. A. Khadija, and T. Wahyuning, "Modern No Code Software Development Android Inventory System for Micro, Small and Medium Enterprises," in *Proceedings - 2022 IEEE International*

Conference on Cybernetics and Computational Intelligence, CyberneticsCom 2022, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 191–195. Available: https://doi.org/10.1109/CyberneticsCom55287.2022.9865265

[41] C. V. K. Iyer *et al.*, “Trinity: A No-Code AI platform for complex spatial datasets,” in *Proceedings of the 4th ACM SIGSPATIAL International Workshop on AI for Geographic Knowledge Discovery, GeoAI 2021*, Association for Computing Machinery, Inc, Nov. 2021, pp. 33–42. Available: https://doi.org/10.1145/3486635.3491072

[42] S. Sinha *et al.*, “Auto-generation of domain-specific systems: Cloud-hosted devops for business,” in *IEEE International Conference on Cloud Computing, CLOUD*, IEEE Computer Society, Oct. 2020, pp. 219–228. Available: https://doi.org/10.1109/CLOUD49709.2020.00041

[43] A. Bucchiarone *et al.*, “What is the future of modeling?” *IEEE Softw*, vol. 38, no. 2, pp. 119–127, Mar. 2021. Available: https://doi.org/10.1109/MS.2020.3041522

[44] K. Schneid, L. Stapper, S. Thone, and H. Kuchen, “Automated Regression Tests: A No-Code Approach for BPMN-based Process-Driven Applications,” in *Proceedings - 2021 IEEE 25th International Enterprise Distributed Object Computing Conference, EDOC 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 31–40. Available: https://doi.org/10.1109/EDOC52215.2021.00014

[45] G. Daniel, J. Cabot, L. Deruelle, and M. Derras, “Xatkit: A Multimodal Low-Code Chatbot Development Framework,” *IEEE Access*, vol. 8, pp. 15332–15346, 2020. Available: https://doi.org/10.1109/aCCESS.2020.2966919

[46] G. Hurlburt, “Low-Code, No-Code, What’s under the Hood?” *IT Professional*, vol. 23, no. 6. IEEE Computer Society, pp. 4–7, 2021. Available: https://doi.org/10.1109/MITP.2021.3123415