Emerging New Roles for Low-Code Software Development Platforms

Jean-Marie Mottu, Gerson Sunyé

Emerging New Roles for Low-Code Software Development Platforms

Jean-Marie Mottu
jean-marie.mottu@ls2n.fr
Nantes Université, IMT Atlantique, CNRS,
LS2N, UMR 6004
Nantes, France

Gerson Sunyé
gerson.sunye@ls2n.fr
Nantes Université, CNRS,
LS2N, UMR 6004
Nantes, France

ABSTRACT

Low- and no-code development platforms have introduced two new development roles: the platform engineer and the citizen developer. While the former are still software developers who implement the low- and no-code platforms, the latter use them to develop their domain applications. In practice, however, we believe that the citizen developer role is shared by two people. For example, in the teaching domain, the citizen developer role is shared between a teacher and his assistant. The first is the domain practitioner, while the second is the domain engineer. The Domain Engineer uses the development platform to create a tailored platform for the teacher, who uses it to create an application that will be for his students, the end users. To explore the possibility of differentiate these two roles in current low-code development platforms, we used two different low-code platforms—Mendix and OutSystems—to implement two case studies. These case studies reveal the limitations of current platforms for specializing platforms with functionalities close to those of low-code development platforms. To compare these two platforms, we consider a list of features that these platforms must satisfy. The results show that current low-code development platforms cannot fully support these new roles.

CCS CONCEPTS

• **Software and its engineering → Application specific development environments**.

ACM Reference Format:
Jean-Marie Mottu and Gerson Sunyé. 2024. Emerging New Roles for Low-Code Software Development Platforms. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3652620.3688337

1 INTRODUCTION

Low-code and no-code are growing development approaches supported by many platforms [15]. They bridge the gap between development and operations by supporting the active involvement of non-developer domain experts, called citizen developers, in the application development life cycle.

A low-code development platform (LCDP) is designed for domain experts without IT skills, who want to build applications. It migrates application development from manual coding using traditional programming languages to designing applications with graphical user interfaces, pre-built components, and parameters. The user interfaces, business logic, and data services are built using visual diagrams and, in some cases, manual coding. Domain experts who do not want or need to code can use a no-code development platform (NCDP), which is easier to configure but produces less customizable applications.

There are many low- and no-code platforms on the market that are generic for developing applications of any domain. Moreover, some of them are dedicated to a domain, especially Dedicated NCDP. The citizen developer can then develop an application using the concepts, logic, and processes of a specific domain.

For instance, these platforms allow a teacher to independently create and deploy a quiz application for her students. She can choose an LCDP if she needs to set rules using code, an NCDP if the quiz requires less customization, or a dedicated Learning Management System (LMS), such as Moodle[1].

This choice seems to be driven by the amount of code, allowing citizen developers to avoid coding and focus on their domain expertise. A main interest of low-code and no-code is to provide citizen developers with platforms to develop by themselves without outsourcing to software developers. However, it hides how they can apply their domain expertise.

These development platforms generally help domain experts reduce the amount of code they need to write. However, citizen developers who want to concentrate on a specific domain require a platform that is tailored to their needs or the ability to customize their domain. They must then either (i) select a dedicated platform that is designed for their specific domain, (ii) work with a no-code platform despite its customization limitations, or (iii) receive training in building domain-specific applications using a low-code platform.

Dedicated platforms are not a universal solution: there is not a dedicated platform for every domain, and creating such a platform is a typical programming challenge. In addition, while dedicated platforms can meet most of the needs of a particular domain, implementing new features is still the responsibility of a software developer, which is beyond the capabilities of citizen developers.

While a generic NCDP requires the domain expert to accept its limitations, a generic LCDP requires him to learn how to model her domain from generic components and code domain-specific

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3688337

[1] https://moodle.org

rules [13]. Some NCDPs and LCDPs can also be extended by creating dedicated components or specializing execution rules, but this involves coding tasks, whether dedicated or generic.

Increasing the domain specialization of a platform decreases the amount of code, indeed, but reduces the possibility to personalize the platform unless asking the help of developers, again. Hence, a domain expert that faces coding difficulties or limitations of a platform will still be dependent on a software developer. A teacher can be frustrated by a limitation of her LMS but not enough to decide to invest in developing an LMS by her own, even using an LCDP or to ask a developer to develop or extend a platform.

Nevertheless, we still believe that low-code and no-code can benefit domain experts. We have noticed that citizen developer is not just one role, but rather several roles within each domain. We have categorized them into two groups: domain practitioners and domain engineers. These roles, involved in different stages of development, can take advantage of low-code and no-code at different times and frequencies. Their involvement depends greatly on the life cycle of each domain: domain practitioners develop applications for end users, while domain engineers focus on developing applications for domain practitioners. More specifically, they specialize in turning generic low-code development platforms into tailored no-code development platforms.

For example, while a teacher spends most of her time with students, preparing her lessons or evaluating their work (for example, through quizzes), educational engineers are focused on assisting teachers by creating new methods and tools. The teacher is the domain practitioner, she is the citizen with less developing knowledge, while the educational engineer is the domain engineer with more aptitude for development. In the approach we propose, the domain engineers are the low-coders. They can be trained to use low-code platforms to prepare them for the needs of the domain practitioners. They can introduce concepts, logic, and processes into the platform so that the domain practitioner can use it as much as possible as Dedicated No-Code Platforms. For instance, educational engineers can create quiz components, with the concepts of questions, answers, and grades. Then the teachers can fill in the questions, the expected answers, the notation, grading scale, and combine them into multiple quiz applications.

In this paper, we present several experiments conducted to evaluate how two low-code platforms fit into this separation of roles, where the role of citizen developer is split into two new roles. We consider two research questions:

- RQ1: Is it possible to specialize an LCDP to a specific domain?
- RQ2: Can an LCDP manage two roles of citizen developers?

To this end, we develop two applications with the following criteria:

(1) the distinction between the two citizen developer roles is explicit;
(2) while there is already a dedicated development platform, it does not meet our needs. We need to differentiate citizen developer roles to provide them with Low-code/No-code management of their part of the development.

The first case study is an application for teachers who want to design personalized exercise sheets for their students with the help of a pedagogical engineer on the exercise components. The second case study is an application to simulate manufacturing systems. It helps a production systems engineer design machines and interactions between them, while production operators design machine assemblies and simulate their performance before physically reconfiguring a production line.

We use two leading LCDPs: Mendix and OutSystems. We examine the functionality of these platforms to achieve role separation in a given scenario and discuss the limitations we encounter. These preliminary experiments give us a better understanding of how LCDP can integrate different citizen developer roles.

The rest of this paper is organized as follows: Section 2 describes the background of the low-code and no-code domain. In Section 3 we propose an approach of dividing the domain expert into two roles. Section IV discusses a feature list of low-code platforms that we would like the tailored no-code platform to still have once specialized to a domain, and describes a couple of experiments made, and how two LCDPs fit the proposed approach. The paper concludes with a discussion of the limitations and future work in Section V.

2 BACKGROUND: DEMOCRATIZING SOFTWARE DEVELOPMENT WITH LOW-CODE AND NO-CODE

The traditional software development process has historically been the domain of professional developers with specialized programming skills. However, recent advances have led to low-code and no-code approaches that aim to democratize software development by empowering users with limited technical backgrounds. This section presents both approaches, highlighting their key features and potential benefits.

2.1 Low-Code Development: Reduce Coding

Low-code development platforms provide a visual development environment that facilitates the creation of software applications with minimal coding [16]. These platforms target citizen developers, who are business users without formal software development training, but with domain expertise [13]. There are several key features [4] that make LCDPs suitable for these developers:

- Visual Modeling: Logic and workflows are represented visually using drag-and-drop elements, enabling intuitive application design even for non-programmers [14].
- Data Integration: Seamless integration of data from various sources (databases, spreadsheets, cloud applications) allows citizen developers to leverage existing data for their applications [13].
- Pre-Built Components and Templates: Reusability is promoted through pre-built components and templates that serve as building blocks for common functionalities, accelerating development [8].

The integration of citizen developers into the development process through LCDP offers several advantages:

- Increased Innovation: They can rapidly prototype and deploy solutions addressing specific domain challenges due to their deep business understanding [7].

- Improved Agility: Organizations gain the ability to build custom applications internally, enabling faster responses to changing market demands [2].
- Reduced Development Costs: Lower reliance on professional developers can lead to significant cost savings [15].

LCDPs provide easy-to-use visual environments for creating software applications with attractive user interfaces, responsive designs, and minimal programming skills [16]. In particular, visual modeling allows users to drag and drop on the user interface [4]. This makes it easier to use, especially for developers who want to be able to use the platform intuitively.

2.2 No-Code Development: No More Coding

No-code development platforms take the concept of democratization a step further [5]. They enable the creation of software applications entirely without writing code [4]. This caters to an even broader user base, including individuals with minimal technical backgrounds, entrepreneurs, and those seeking to automate personal workflows.

NCDPs accomplish usability through the following features [4]:

- Restrictive Drag-and-Drop Interfaces: Pre-built components can be dragged and dropped onto a visual canvas to build applications.
- Pre-Defined Functionality: The need for custom coding is eliminated through libraries of pre-built features and integrations offered by NCDPs.
- WYSIWYG Editors: What You See Is What You Get (WYSIWYG) editors allow users to build and preview their application in real-time.

2.3 Extending LCDP/NCDP: the Coding Comeback

While LCDP and NCDP main difference is on the use of coding to configure parts of the application, many LCDPs and NCDPs offer extension mechanisms.

Even when considering Dedicated LCDP/NCDP, extension mechanisms requires solid programming skills. For instance, Moodle extension requires PHP and HTML coding[2].

To extend a platform, the citizen developer then requires the help of a software developer and to come back to a classic development life cycle.

2.4 State of the Low-Code/No-Code Market

The low-code development market has exploded in recent years, attracting both service providers and users. It is projected to reach a massive market size by 2025 [17]. This surge in demand, coupled with growing competition, creates a thriving financial landscape for companies and investors. Low-code development offers the potential for increased ROI and reduced development costs. There are several LCDP vendors, with some prominent players listed in a categorization system (Figure 1). The integration of generative AI with low-code development is already underway, promising further market expansion. This could be achieved through AI-powered

Figure 1: Gartner Magic Quadrant [11]

functions that automate tasks or generate code, leading to faster development and potentially lower costs.

Popular LCDPs include Mendix[3], OutSystems[4], Microsoft Power Apps[5], and Salesforce[6]. Popular NCDPs include Bubble[7], Wix[8], Zapier[9], and Glide.[10]

2.5 Empower Domain Expert to Develop with LCDP/NCDP

As explained earlier, citizens with some programming skills are the target users of LCDP. Citizen developers with programming skills can tackle more complex development tasks and take advantage of additional functionality that requires more than drag-and-drop usage. Application domain can differentiate LCDPs from NCDPs. LCDPs are most useful in industries such as healthcare, web and mobile application development, and manufacturing [4]. These industries often require complex, customized solutions that require the integration of specialized functionality and a seamless user experience. NCDPs, on the other hand, are well-suited for building reports, analysis, and monitoring applications [10]. While they excel at simplifying the development of such applications, they have limitations that prevent the customization of graphical user interfaces and the implementation of advanced business rules that are often important for the development of enterprise-level software solutions [3]. As a result, the choice between LCDPs and NCDPs

[2]https://support.moodle.com/support/solutions/articles/80001075418-how-to-create-a-moodle-plugin

[3]https://www.mendix.com/
[4]https://www.outsystems.com/
[5]https://www.microsoft.com/power-platform/
[6]https://www.salesforce.com/platform/low-code-development-platform/
[7]https://bubble.io
[8]https://www.wix.fr
[9]https://zapier.com
[10]https://www.glideapps.com

seems to be guided by the complexity of the application to be developed and the coding ability of the citizen developer instead of the domain itself.

Kirchhof et al. observed that existing LCDPs mostly do not encompass domain knowledge [6]. It is then difficult for a domain expert to focus on their domain, while the effort to model it is a costly preliminary step. One can ask, if currently low-code is not more a progress for developers to gain in productivity while reducing repetitive programming, instead of helping domain experts to develop their domain application by themselves.

As far as we know, domain specialization seems to be reserved for Dedicated NCDPs, platforms with a classic life cycle, designed by the platform engineer to satisfy experts in one domain at a time. Some of them have extension mechanisms or are LCDP platforms that allow advanced users to develop their components. This is useful when a company wants to promote its product by developing components in a dedicated LCDP or NCDP. Let us imagine an industrial company that develops robots and wants its robots to be integrated into platforms dedicated to the simulation of the industrial production chain. Then, the performance of production lines that may include such robots can be evaluated. However, this type of extension is not directly available to citizen developers and is instead outsourced to software developers, who then handle the development life cycle.

3 EMERGING NEW SOFTWARE DEVELOPMENT ROLES

Software development is usually the responsibility of software engineers who receive a specification from a client who wants to propose an application to her users (e. g., a store manager who orders an application for her cashiers). It involves a development life cycle with iterations, and at the very end, a functional application is delivered and hopefully validated. Agile methods and continuous delivery reduce the time it takes to develop and improve an application. However, while users and clients are more involved, the development is still up to the software engineers. This section presents how the citizen developer role emerged, and why it should not be seen as a single role, but as the emergence of a couple of domain application development roles.

3.1 LCDP and NCDPs to Promote Citizen Developers as an Active Role in the Middle of a 3-Role Architecture

LCDP and NCDP promote citizen developers as an active role. They provide tools for citizen developers to develop applications by themselves, managing their own projects. Reducing coding allows citizen developers to consider starting their development projects while keeping control over the development cycle.

3.1.1 The Citizen Developer in the Middle of a 3-Role Architecture. Usually, LCDP and NCDP involve three main roles, as illustrated at the top of Figure 2. First, the *software engineers* create and maintain LCDP and NCDP. Second, the *citizen developer* is a domain expert who uses an LCDP or NCDP to create applications. She uses her technical skills and brings in-depth knowledge of the specific domain in which the software will be used, which guarantees its relevance and usefulness. Third, the *end-user* can use the application developed.

The citizen developer is the key person responsible for creating applications for her users. She may be the sole developer and users, building applications for herself. Alternatively, several citizen developers can collaborate to create large applications for multiple users. They should follow the four main steps of development using LCDP/NCDP [14]. First, they model data using drag-and-drop capabilities to configure the data schema of the application. They create entities, associate them with each other, and set constraints and dependencies. Second, they design the user interface by configuring screens and parts of them. They also manage the authorization processes required to access the application and ensure security. Third, they specify the application behavior by managing and maintaining processes and domain logic rules using visual tools. Fourth, they integrate services through APIs that require careful management of the data structure and databases.

These steps lead to application deployment, which is greatly facilitated by platforms that offer cloud hosting, for most of them. After that, the application can be maintained and updated by the citizen developer according to her own life cycle.

The LC and NC approaches, while functional, can lead to rigid and non-adaptable solutions. Citizen developers may encounter challenges in translating their business knowledge into the various elements required for their tasks. These platforms offer predefined structures with limitations that can be overcome through LCDP coding, but this may hinder the flexibility needed to fully express complex business processes for domain expert citizen developers. There are two main drawbacks. First, these platforms propose non-domain-specific predefined structures, requiring the adaptation of the citizen developer's knowledge. Second, modeling a domain in LCDP is costly and challenging to maintain and update without software development skills. Even platform extensions, including dedicated ones, do not provide an optimal solution for domain expert citizen developers, as they can lead back to dependency on software engineers.

3.2 Sharing the Responsibility in a 4-Role Architecture: Splitting the Citizen Developer Role

We identify a set of roles, not a single role, as citizen developer. It is not a set of roles only based on the amount of coding, which would distinguish citizen developers using LCDP (the ones who may code) and those using NCDP (the ones who do not code, sometimes called *Citizen Makers* [18]). There are a couple of roles based on domain expertise and life cycle. We distinguish between the domain engineer and the domain practitioner, as shown at the bottom of Figure 2. The domain practitioner has a role close to the theoretical citizen developer: she is a domain expert who wants to develop applications focusing on her domain and manage their life cycle. However, we believe that investing in training to master LCDP is too costly for her, while in most situations she already works with other domain experts: the domain engineers.

[11] Images from Freepik Storyset https://fr.freepik.com/auteur/stories

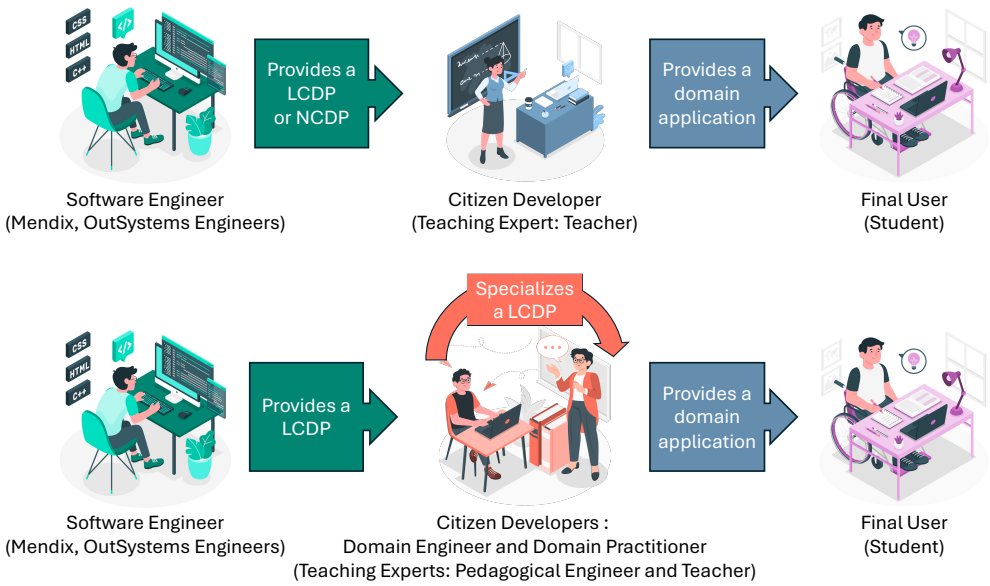

Figure 2: Moving from the Traditional 3-Role Architecture to the Proposed 4-Role Architecture.[11]

The domain engineer helps the domain practitioner improve her methods, procedures, and tools. The domain engineer has almost the same domain expertise, but her life cycle is different because she is not constrained by the practice and rhythm of a domain that depends on its end users.

Therefore, the specialization on a domain increases from the left to the right of Figure 2. A domain engineer can specialize in an LCDP for a domain to be used by several domain practitioners. Each domain practitioner can then develop several applications for different final users. For instance, a pedagogical engineer can specialize a generic LCDP to the teaching domain by introducing components to be incorporated into quiz applications. Then different teachers, let us say in different disciplines, can create quiz applications for math, grammar, and physics evaluations, to be used by their many students of different groups. The separation of the citizen developer roles allows then the domain practitioner to focus on her domain (e. g., a teacher focusing on teaching math), and to create many applications.

The domain practitioner uses the LCDP to periodically create or update applications (e. g., a teacher makes a quiz application at the end of a lesson). The domain engineer manages the specialization of the LCDP with another life cycle, probably a two-step one: a long iteration to specialize the LCDP to her domain, and several shorter iterations to update and introduce new domain components when the domain practitioner requires it. The cycle is supposed to be less constrained than one involving software developers: domain engineer and domain practitioner are supposed to be close to each other, and the first still uses an LCDP, which is easier to use with training than her role can expect her to get.

3.3 Specialization of an LCDP

To the best of our knowledge, the specialization of an LCDP is not a feature of existing platforms. They propose to model the domain and define its rules using prebuilt components, but they do not explicitly propose to distinguish between the two roles of citizen developers. In particular, they do not allow a citizen developer to consider only elements modeled for her domain when building an application.

Another strategy that we will consider in the experiments of the next section is the possibility of developing an application with an LCDP that is domain-specific and, as far as possible, an NCDP, so that it can be used as a Tailored NCDP by the domain practitioner. We distinguish such a Tailored NCDP from a Dedicated NCDP, the latter is developed by a software engineer and the former is a domain specialization of an LCDP that can be done by the domain engineer.

Figure 3 illustrates the expected architecture. Software engineer group develops an LCDP and domain engineer group specializes the LCDP, by modeling the domain, defining the rules, and producing a Tailored NCDP. Domain practitioner group uses the Tailored NCDP to develop domain applications that focus only on their domain. Final users can then use the domain applications. Groups can be one or several actors. In addition, in the life cycle, the domain engineer may require software engineers with LCDP extensions requiring programming skills. In the same way, the domain practitioners may require new or specialized components from the domain engineers.

4 EXPECTED FEATURES OF TAILORED NCDP

In this section, we identify the expected features of a tailored NCDP and what is necessary to specialize in an LCDP to meet the 4-role architecture. We consider two LCDPs and two case studies to evaluate how actual leading LCDPs can manage the 4-role architecture

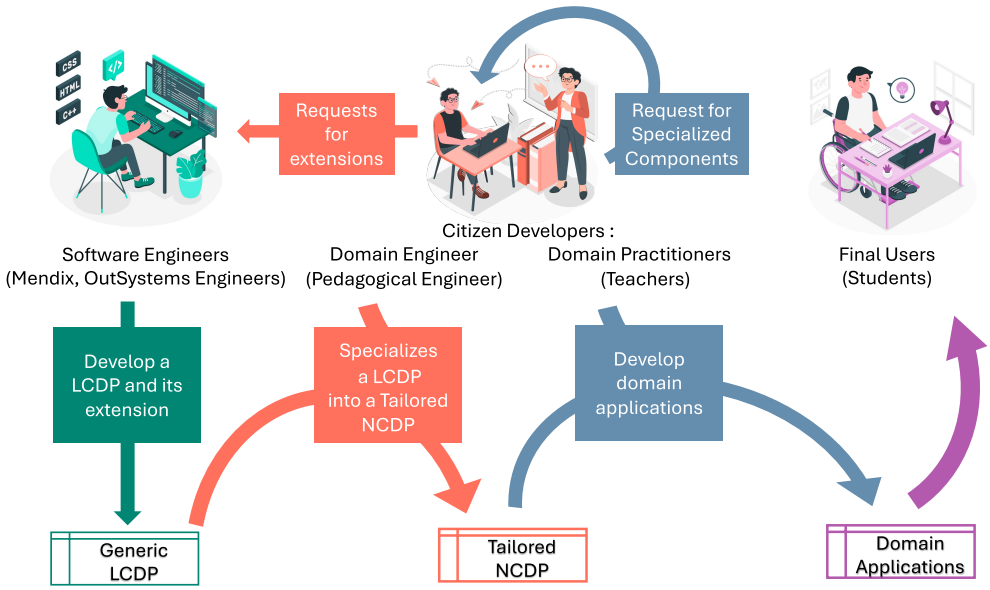

Figure 3: Distribution of roles and life cycle in the proposed 4-Role Architecture

proposed. These preliminary experiments give us a better understanding of how LCDP can integrate different citizen developer roles.

4.1 Case Studies

To this end, we develop two applications with the following criteria:

(1) the distinction between the two citizen developer roles is explicit;
(2) there is already a Dedicated LCDP, but it does not meet our needs. We need to differentiate citizen developer roles to provide them with Low-code/No-code management of their part of the development.

4.1.1 Teaching Application. The first case study is an application for teachers who want to design personalized exercise sheets for their students with the help of a pedagogical engineer on the exercise components. The aim is to create an application in which different types of exercises can be created: multiple choice exercises, exercises with simple questions, and exercises with multipart questions (particularly for mathematical operations).

The first case study involves creating an application for teachers. The application will allow teachers to work with a pedagogical engineer to design personalized exercise sheets for their students. The goal is to develop an application that offers different types of exercises: multiple-choice, simple and multipart questions (for mathematical operations), etc.

As detailed before, splitting the citizen developer role into a couple of roles is well known:

- The Pedagogical Engineer is a domain engineer
- The Teacher is the domain practitioner

There exist dedicated Learning Management Systems (LMS), such as Moodle[12]. However, extending Moodle requires PHP and HTML coding, which is not supposed to be a citizen developer task but a Software Developer one.

4.1.2 Manufacturing Simulation Application. The second case study is an application to simulate manufacturing systems. It assists a production systems engineer design machines and interactions between them, while production operators design machine assemblies, simulate their execution, and assess their performance before physically reconfiguring a production line.

Splitting the citizen developer role is a task that we consider in the RODIC[13] research project, while still being challenging [1]:

- Domain engineer is the production systems engineer
- Domain practitioner is the Production Operator

There are dedicated simulators available, such as FlexSim[14]. However, FlexSim offers many settings when modeling machines and production lines, which is a prohibitive investment for Production Operators. Moreover, extending it, for instance with a new FlexScript class requires C++ programming, which is not supposed to be a citizen developer task but a Software Developer one.

4.2 Mendix and OutSystems LCDPs

We consider two leading LCDPs: Mendix and OutSystems.

Mendix is an LCDP that enables users to create web and mobile applications with minimal coding. It provides a visual development environment where users can drag-and-drop components to create application logic, user interfaces, and data models [12].

[12] https://moodle.org
[13] https://rodic.ls2n.fr/
[14] https://www.flexsim.com/

OutSystems is a platform for the rapid development and deployment of modern applications. Designed to meet the complexity and needs of the enterprise, OutSystems is an enterprise-grade LCDP [9].

4.3 Description of Required Features for LCDP Specialization

In this section, we introduce a feature diagram that presents different terms for describing the functionality required to use an LCDP to create a tailored NCDP. By analyzing the platforms presented in the previous section, we have identified and modeled their variability and commonalities in terms of concrete features. First, we consider five of the features listed in the taxonomy created by Sahay et al. [14]. They outline the features that facilitate the selection and comparison of different LCDPs. From these general characteristics, we have selected those that we believe are important for the transition from an LCDP to a tailored NCDP. To do this, we relied on the implementation of the education case study with the two LCDPs. Then, based on the case studies, we identified three other specific features. All of these features are listed in the Figure 4, which shows a high-level feature diagram where each sub-node represents a major point of variation. Details of the features are given in Table 1.

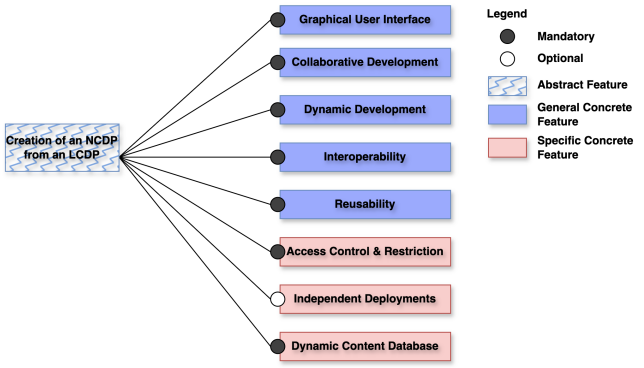

Figure 4: Feature Diagram Representing the Top-Level Areas of Variation to Use an LCDP to Create a Tailored NCDP

Graphical User Interface (GUI). This group of features represents those that allow users to interact with the platform through graphical elements such as buttons, drop-down menus, or icons, using drag-and-drop, point-and-click, and other tools, rather than having to write code.

Both citizen developer roles require an easy-to-use GUI to model their domain and develop domain applications. The domain engineer expects to model the domain and specialize pre-built components with low-code. In addition, the domain practitioner expects to combine specialized components without code. In both situations, graphical elements (drag and drop, point and click, etc.) are required.

In the first case study, the pedagogical engineer uses drag and drop to create exercise components. In the tailored NCDP, the teacher expects to be able to drag and drop these components to combine them into a set of exercises on a screen. If a teacher finds that a component cannot be moved, she expects the pedagogical engineer to provide that option rather than doing it for her, each time it is needed.

Access Control & Restriction. This set of features allows for precise control over user access rights to different application parts. Only authorized users can access and modify specific data or functions. Access control defines specific access rights for users based on their roles and responsibilities as citizen developers.

While access control is natively manageable in LCDP according to a 3-role architecture, split citizen developer roles need to be encapsulated. The domain engineer would like to give the domain practitioners limited rights, for example, to use a specialized component but not to modify it. However, the domain practitioners cannot be considered as classical end users, since the actual end users of the developed domain application should also have access control managed, this time by domain practitioners.

In the second case study, the production systems engineer can define components to model robots with restrictions that he does not want a production operator to lift while using them in a simulation (for safety reasons, for example).

Collaborative Development. This group of features represents functionality that allows several people to work on the same project. Several domain engineers may want to collaborate when specializing an LCDP. Several domain practitioners may want to collaborate when developing a domain application with a tailored NCDP.

In addition, as shown in Figure 2, domain practitioners may ask domain engineers for new or adapted specialized components.

In the first case study, if teachers feel that functionality is missing in the NCDP, they can create a ticket and the pedagogical engineer can see this ticket and make the necessary corrections accordingly.

Dynamic Development. This group of features represents those that allow the application to evolve continuously over time by modifying it without having to redeploy it.

These features are important for the transition between the LCDP and the NCDP because they reduce the dependency between domain engineers and domain practitioners. If the domain engineers need to modify or add components to the tailored NCDP they created, the domain practitioners do not have to wait for a new deployment. Similarly, if the domain practitioners update the application they created, end users do not have to wait for a new deployment.

In the first case study, if the teachers want to go beyond creating exercises by creating an exam, for example, the pedagogical engineer should be able to add a component related to exams in the tailored NCDP without having to perform a deployment that might corrupt existing data (existing exercises).

Independent Deployments. Independent Deployments ensure that the domain application created from a tailored NCDP can be deployed independently of the NCDP. For example, the domain application can run when the tailored NCDP is not in use.

In the second case study, several production operators can configure and run their simulations independently of each other and the production systems engineer.

Dynamic Content Database. This group of features represents a storage system that allows users to store and manage different types of content that can be retrieved, updated, and displayed in applications according to user interactions or predefined rules.

These features facilitate the transition between general data types (in the LCDP) and specific data types (in the tailored NCDP). It allows the domain practitioner to create concrete types that concretize the formal types defined by the domain engineer.

In the first case study, the pedagogical engineer can create an exercise type, while the teacher can specify the exercise type with the subject of her choice, e. g., mathematics or grammar.

Interoperability. This group of features represents functionality that enables collaboration with other platforms and integration of functionality from existing software systems.

This allows domain engineers to include functionality specific to the domain of the tailored NCDP being created. At the same time, it reduces the technical nature of their work. The domain engineers can focus on the functionality required by the domain practitioner.

In the first case study, the pedagogical engineer does not need to redevelop a grade-entry service; she can synchronize her application with an existing grade-entry service. Teachers can also integrate tools tailored to the subject they teach. For example, math teachers can use tools to visualize geometric figures.

Reusability. This group of features represents the reuse of existing data or predefined models. These features reduce development time for the domain practitioner and enable better collaboration with the domain engineers, as well as technical simplicity for the domain practitioner.

They allow the domain practitioner to create a new model from an existing one, created by the domain engineers. In addition, the technical requirements for the domain practitioner are reduced, contrasting with the domain engineers, who must to increase these technical requirements and skills to implement the functionality.

In our case study, the pedagogical engineer is able to create templates on the LCDP (exercise, question) to enable the teacher to create their own useful tools for the students.

4.4 Feature List

In order to effectively assess the capabilities of each platform in the context of our given case study, we constructed a comparison table in Table 1. This table describes the different features and functionalities offered by each platform. By systematically comparing these attributes, prospects can make informed decisions about which platform is best suited for their specific needs.

Table 1 uses the following legend:

- ●: features are natively present on the platform for domain engineer and domain practitioner.
- ◐: features are present for domain engineer but not for the domain practitioner, or LCDP offers an alternative (plugins, etc) for domain practitioner.
- ○: features are present for everyone.

5 CHALLENGES & OPPORTUNITIES

5.1 Observed Limitations

Our two case studies implemented on the top of Mendix and OutSystems, revealed some limitations of these platforms for creating a tailored NCDP, in the education and manufacturing domains. Considering the teaching case study, one of their main disadvantages is that it is only possible to do text-based exercises. This precludes the use of images, advanced mathematical expressions, and different question formats, limiting their use in a variety of subjects. In addition, both platforms have difficulty handling multiple valid solutions, especially for text-based answers with potential variations. As a result, these platforms are less effective for open-ended questions and for promoting deeper learning through more varied and engaging exercise styles. At this stage of our study, we do not know how to address these limitations without requiring an extension of the LCDP. Other problems have arisen with the use of different formats and web browsers. The inability to use data in a variety of formats hinders compatibility with open and widely used formats, which can cause problems for institutions that are required to use these formats. In addition, browser compatibility issues and complex upgrade processes between major versions have been identified.

The choice of platform for its specialization and implementing the 4-role architecture must take these limitations into account.

5.2 Possible Improvements

Recent years have witnessed a revolution in software development, characterized by the rise of low-code and no-code development approaches. The emergence of citizen developers, who have domain expertise but limited programming skills, has changed the role of the developer. Low- and No-coders use platforms with visual interfaces and drag-and-drop tools to simplify application development, democratize software creation, and reduce dependence on traditional coders. While traditional coders focus on critical tasks, low-code, and non-coders accelerate development cycles by taking on simpler aspects such as interface design. Low-code aims to get domain experts more involved in software development by limiting technical elements. LCDP and NCDP provide visual environments for creating applications with a streamlined interface, simplifying the development process.

We pointed out in this paper that citizen developers should not be considered as a single role. It is too restrictive to focus on the amount of coding they can implement, but rather on how they can consider only their domain and manage different development life cycles. The transition to a 4-role architecture instead of a 3-role is motivated by the need to integrate citizen developers more effectively into the software development process. Initially, citizen developers took on the role of both platform developers and end users. However, this approach has its limitations, not only because of the lack of technical skills of the users but also because of the domains that require more people.

For these reasons, a new 4-role architecture is being considered. This involves several citizen developers acting as domain engineers and domain practitioners.

Feature	Description
Graphical User Interface	
Drag-and-drop	Allows users to simply drag and drop graphical elements (such as multiple choice questions, text boxes, images) to build their own customized worksheets.
Logical diagram	Provides a visual representation (in the form of a diagram) of the underlying logic of an application, showing data flows, decisions, and interactions between the various components.
Structural diagram	Provides a visual representation of the entities and their relationships within an application. These diagrams are used to design the structure of the database, defining the entities (in the form of tables) and the relationships between them.
Advanced bug report	Enables advanced error management, including error localization, legible and precise explanation and relevant information (in visual form) to help understand and resolve errors.
Point and click	Allows users to create and modify application components simply by clicking on predefined elements on the graphical interface, such as data fields or buttons.
Access Control & Restriction	
Role creation	Allows administrators or authorized users to create new roles within the platform. These roles define the different levels of authorization and privileges granted to users according to their responsibilities and needs within the system.
Role authorization management	Enables administrators to manage the authorizations associated with each role created in the system. It provides a set of tools for precisely defining the actions and resources that each role is authorized to access.
Widget access control	Enables users to define and manage permissions for accessing and interacting with specific user interface elements or widgets within their applications, ensuring security and data integrity.
Feature access control	Empowers users to regulate and manage permissions for accessing and utilizing specific functionalities or capabilities within their applications, enhancing security and governing user interactions.
Collaborative Development	
Tickets manager	Allows the creation of tickets, individual tasks to be carried out within a project
Version manager	Allows the creation of several versions for the parallel development of several functions. It also allows changes to be reversed.
Advanced reports	Allows users to create detailed reports on the project: how the project works, general problems, updates made, etc.
Dynamic Development	
Platform extensibility	Allows developers to modify or add content to an already deployed application without needing to republish or reinstall the entire application.
Possibility support	Allows users to add additional code (hard code, extensions, etc.) to an existing application to create new features that are not available in the initial features of the LCDP.
Independent Deployments	
Deployment for the domain engineer	Allows the domain engineer to make application updates available to all platform users.
Deployment for the domain practitioner	Allows the domain practitioner to make updates to the application available to end-users of the platform. There is no need to contact the domain engineer for a simple update of the exercises provided.
Dynamic Content Database	
Import / export / filling	Allows users to efficiently work with data from various sources, exchange data with external systems, and populate data fields to build and configure applications.
Add data type	Allows users to select and configure data types when defining data schemes, allowing users to specify the characteristics of their data fields intuitively. This simplifies the process of designing and managing data structures within applications without requiring users to write complex code.
Data schema	Refers to the structure or blueprint that defines the organization and relationships of data within a database or storage system. It specifies the types of data that can be stored, the format in which it is stored, and the relationships between different data entities.
Interoperability	
Plug-ins extensions	Allows the LCDP to be extended by integrating plug-ins or extensions from external sources.
Possibility of using an API	Enables users to integrate third-party services such as time management tools and easily adapt to technological developments and new requirements by integrating new services and updating existing functionality.
Reusability	
Template and model creation	Allows domain engineers to create components that can be reused by the domain practitioners, who in turn, can customize these models without requiring advanced technical skills.
Application extension	Allows domain engineers to extend the features of their platform through the extensions offered by the LCDP, without requiring any coding or changes to the LCDP.

Table 1: Taxonomy to Use an LCDP to Create a Tailored NCDP

Features	Mendix	OutSystems
Graphical User Interface		
Drag-and-drop	◐	◐
Logic diagram	◐	◐
Modeling diagram	◐	◐
Advanced bug report	●	◐
Point-and-click	●	●
Access Control & Restriction		
Role Creation	◐	◐
Role authorization management	◐	◐
Widget access control	◐	◐
Features access control	●	●
Collaborative Development		
Tickets manager	●	○
Version Manager	●	◐
Advanced reports	●	○
Dynamic Development		
Platform extensibility	●	●
Possibility support	●	◐
Independent Deployments		
Deployment for LC-CD	●	●
Deployment for NC-CD	◐	●
Dynamic Content Database		
Import / export / filling	●	◐
Add data type	○	●
Data schema	●	◐
Interoperability		
Plug-ins extensions	●	◐
Possibility of using an API	●	○
Reusability		
Templates and Models creation	●	●
Application extension	○	◐

Table 2: Platform Comparison Table

The goal of the case studies was to show how today's low-code platforms support this type of architecture. The case studies examined two implementations on Mendix and OutSystems and revealed limitations in creating tailored NCDPs for education and manufacturing. These LCDPs have difficulty separating the roles of citizen developers due to technical constraints. The LCDPs tested did not allow us to implement tailored no-code platforms. Plug-ins were conceivable, but not within the reach of every citizen developer.

At this point, we cannot answer the research questions positively, and to accomplish these goals, we would need to experiment with other LCDPs and other case studies.

6 ACKNOWLEDGEMENTS

This work is supported by the French National Research Agency (ANR) [grant number ANR 21 CE10 0017]. We would like to thank all the students of Nantes Université who participated in this study, in alphabetical order, Izzedine Issa AHMAT, Hiba AJABRI, Christella ARISTOR, Nathan DESHAYES, Afi Sabine EKLO, Kylian GERARD, Aurelien PAGEOT, Quentin POISBLAUD, Rayane TABTI.

REFERENCES

[1] Hiba Ajabri, Jean-Marie Mottu, and Erwan Bousse. 2024. Defining KPIs for Executable DSLs: A Manufacturing System Case Study. In *12th International Conference on Model-Based Software and Systems Engineering (MODELSWARD 2024)*. SCITEPRESS - Science and Technology Publications, Rome, Italy, 169–178. https://doi.org/10.5220/0012361000003645

[2] Alexander C Bock and Ulrich Frank. 2021. Low-code platform. *Business & Information Systems Engineering* 63 (2021), 733–740.

[3] Mauro AA Da Cruz, Heitor TL de Paula, Bruno PG Caputo, Samuel B Mafra, Pascal Lorenz, and Joel JPC Rodrigues. 2021. Olp—a restful open low-code platform. *Future Internet* 13, 10 (2021), 249.

[4] Hind El Kamouchi, Mohamed Kissi, and Omar El Beggar. 2023. Low-code/No-code Development: A systematic literature review. In *2023 14th International Conference on Intelligent Systems: Theories and Applications (SITA)*. IEEE, 1–8.

[5] Edona Elshan, Ernestine Dickhaut, and Philipp Alexander Ebel. 2023. An investigation of why low code platforms provide answers and new challenges. *Hawaii International Conference on System Sciences* 56 (2023), 6159–6168.

[6] Jörg Christian Kirchhof, Nico Jansen, Bernhard Rumpe, and Andreas Wortmann. 2023. Navigating the Low-Code Landscape: A Comparison of Development Platforms. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 854–862.

[7] Désirée Krejci, Satu Iho, and Stéphanie Missonier. 2021. Innovating with employees: an exploratory study of idea development on low-code development platforms. In *European Conference On Information Systems, ECIS2021*.

[8] Eder Martinez and Louis Pfister. 2023. Benefits and limitations of using low-code development to support digitalization in the construction industry. *Automation in Construction* 152 (2023), 104909. https://doi.org/10.1016/j.autcon.2023.104909

[9] Ricardo Martins, Filipe Caldeira, Filipe Sa, Maryam Abbasi, and Pedro Martins. 2020. An overview on how to develop a low-code application using OutSystems. In *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*. IEEE, 395–401.

[10] Giorgia Masili. 2023. No-code Development Platforms: Breaking the Boundaries between IT and Business Experts. *International Journal of Economic Behavior (IJEB)* 13, 1 (2023), 33–49.

[11] Oleksandr Matvitskyy, Kimihiko Iijima, Mike West, Kyle Davis, Akash Jain, and Paul Vincent. 2023. *Magic Quadrant for Enterprise Low-Code Application Platforms*. Technical Report. Gartner.

[12] Mendix. 2024. Deliver game-changing software. In *Mendix Low-Code Plateform Features - Low-Code App Development Tools*. (Online) https://www.mendix.com/platform/ -Accessed: (28/04/2024).

[13] Davide Di Ruscio, Dimitris Kolovos, Juan de Lara, Alfonso Pierantonio, Massimo Tisi, and Manuel Wimmer. 2022. Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling* 21, 2 (2022), 437–446.

[14] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. 2020. Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 171–178.

[15] Raquel Sanchis, Óscar García-Perales, Francisco Fraile, and Raul Poler. 2019. Low-code as enabler of digital transformation in manufacturing industry. *Applied Sciences* 10, 1 (2019), 12.

[16] Khushi Talesra and GS Nagaraja. 2021. Low-code platform for application development. *International Journal of Applied Engineering Research* 16, 5 (2021), 346–351.

[17] Jason Wong and Kyle Davis. 2022. *Harness the Disruptive Powers of Low-Code: A Gartner Trend Insight Report*. Technical Report. Gartner.

[18] Taiki Yoshida. 2023. *Microsoft Power Apps and generative AI helps citizen maker upskill and transform career*. https://www.microsoft.com/en-us/power-platform/blog/power-apps/microsoft-power-apps-and-generative-ai-helps-citizen-maker-upskill-and-transform-career/