# Supporting the understanding and comparison of low-code development platforms

Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, Alfonso Pierantonio
*Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica*
*Università degli Studi dell'Aquila*
L'Aquila, Italy
*name.surname*@univaq.it

***Abstract*—Low-code development platforms (LCDPs) are easy to use visual environments that are being increasingly introduced and promoted by major IT players to permit *citizen developers* to build their software systems even if they lack a programming background. Understanding and evaluating the LCDP to be employed for the particular problem at hand are difficult tasks mainly because decision-makers have to choose among hundreds of heterogeneous platforms, which are difficult to evaluate without dedicated support. Thus, a detailed classification is needed to elaborate on the existing low-code platforms and to help users find out the most appropriate platforms based on their requirements.**

**In this paper, a technical survey of different LCDPs is presented by relying on a proposed conceptual comparative framework. In particular, by analyzing eight representative LCDPs, a corresponding set of features have been identified to distil the functionalities and the services that each considered platform can support. The final aim is facilitating the understanding and the comparison of the low-code platforms that can best accommodate given user requirements.**

***Index Terms*—Low-Code Development Platforms, Model Driven Engineering, Software Development**

## I. Introduction

Low-code development platforms (LCDPs) are provided on the Cloud through a Platform-as-a-Service (PaaS) model, and enable the development and deployment of fully functional software applications utilizing advanced graphical user interfaces and visual abstractions requiring minimal or no procedural code [21]. Thus, with the primary goal of dealing with the shortage of highly-skilled professional software developers, LCDPs allow end-users with no particular programming background (called *citizen developers* in the LCDP jargon) to contribute to software development processes, without sacrificing the productivity of professional developers.

By using low-code platforms, citizen developers can build their software application without the help of several developers that were earlier involved in along the full-stack development of fully operational applications. Thus, developers can focus on the business logic of the application being specified rather than dealing with unnecessary details related to setting up of the needed infrastructures, managing data integrity across different environments, and enhancing the robustness of the system. Bug fixing and application scalability and extensibility are also made easy, fast and maintainable in these platforms by the use of high-level abstractions and models [4]. Procedural code can be also specified in these platforms to achieve further customization of the application on one's own preferences.

Forrester and Gartner document the growth of LCDPs in their reports [16], [18], [20] that forecast a significant market increase for LCDP companies over the next few years. Major PaaS players like Google and Microsoft are all integrating LCDPs (Google App Maker and Microsoft Power Platform, respectively) in their general-purpose solutions. According to a Forrester report [18], the low-code market is expected to represent $21B in spending by 2022. In a recent Gartner report [20], 18 LCDPs were analyzed out of 200. Consequently, understanding and evaluating candidate LCDPs that best fit the particular problem at hand can be a strenuous and challenging task.

In this paper, a technical survey is provided to distil the relevant functionalities provided by different LCDPs and accurately organize them. In particular, eight major LCDPs have been analyzed to provide potential decision-makers and adopters with objective elements that can be considered when educated selections and considerations have to be performed. The contributions of the paper are summarized as follows:

- Identification and organization of relevant features characterizing different low-code development platforms;
- Comparison of relevant low-code development platforms based on the identified features;
- Presentation of a short experience report related to the adoption of LCDPs for developing a simple benchmark application.

To the best of our knowledge, this is the first paper aiming at analyzing different low-code platforms and discuss them according to a set of elicited and organized features.

The paper is organized as follows: Section II presents the background of the work by showing the main architectural aspects of low-code development platforms. Section III introduces the eight LCDPs that have been considered in this work. Section IV presents the taxonomy, which has been conceived for comparing LCDPs as discussed in Section V. Section VI presents a short experience report related to the adoption of LCDPs for developing a simple benchmark application. Section VII concludes the paper and discusses some perspective work.

## II. Background

Low-code development platforms[1] are software platforms that sit on the cloud and enable developers of different domain knowledge and technical expertise to develop fully-fledged applications ready for production [16]. Such applications are developed through model-driven engineering principles and take advantage of cloud infrastructures, automatic code generation, declarative and high level and graphical abstractions to develop entirely functioning applications [20]. These platforms capitalise on recent developments in cloud computing technologies and models such as Platform-as-a-service (PaaS), and proven software design patterns and architectures to ensure effective and efficient development, deployment and maintenance of the wanted application.

At the heart of low-code platforms, there are model-driven engineering (MDE) principles [11] that have been adopted in several engineering disciplines by relying on the automation, analysis, and abstraction possibilities enabled by the adoption of modelling and metamodeling [12].

[1]Hereafter, the terms *low-code platforms* and *low-code development platforms* are used interchangeably.

978-1-7281-9532-2/20/$31.00 ©2020 IEEE
DOI 10.1109/SEAA51224.2020.00036

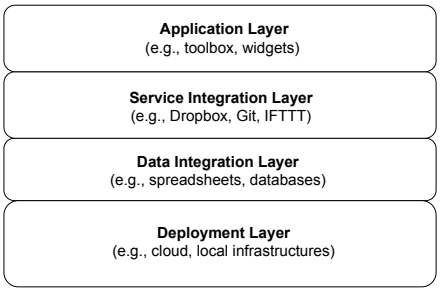


Fig. 1. Layered architecture of low-code development platforms

### A. A bird-eye view of low-code platforms

From an architectural point of view, LCDPs consist of four main layers, as shown in Fig. 1. The top layer (see `Application Layer`) consists of the graphical environment that users directly interact with to specify their applications. The toolboxes and widgets used to build the user interface of the specified application are part of this layer. It also defines authentication and authorisation mechanisms to be applied to the specified artefacts. Through the modelling constructs made available at this layer, users can specify the behaviour of the application being developed. For instance, users can specify how to retrieve data from external data sources (e.g., spreadsheets, calendars, sensors, and files stored in cloud services), how to manipulate them by using platform facilities or utilising external services, how to aggregate such data according to defined rules, and how to analyse them. To this end, the `Service Integration Layer` is exploited to connect with different services by using corresponding APIs and authentication mechanisms.

A dedicated data integration layer permits to operate and homogeneously manipulate data even if heterogeneous sources are involved. To this end, the `Data Integration Layer` is concerned with data integration with different data sources. Depending on the used LCDP, the developed application can be deployed on dedicated cloud infrastructures or on-premise environments (`Deployment Layer`). Note that the containerization and orchestration of applications are handled at this layer together with other continuous integration and deployment facilities that collaborate with the `Service Integration Layer`.

### B. Main components of low-code development platforms

By expanding the layered architecture shown in Fig. 1, the peculiar components building any low-code development platform are depicted in Fig. 2 and they can be grouped into three tiers. The first tier is made of the application modeler, the second tier is concerned with the server side and its various functionalities, and the third tier is concerned with external services that are integrated with the platform. The arrows in Fig. 2 represent possible interactions that might occur among entities belonging to different tiers. The lines shown in the middle tier represents the main components building up the platform infrastructure.

As previously mentioned, modelers are provided with an *application modeler* enabling the specification of applications through provided modeling constructs and abstractions. Once the application model has been finalized, it can be sent to the platform back-end for further analysis and manipulations including the generation of the full-fledged application, which is tested and ready to be deployed on the cloud.

Figure 3 shows the application modeler of Mendix [6] at work. The right-hand side of the environment contains the widgets that

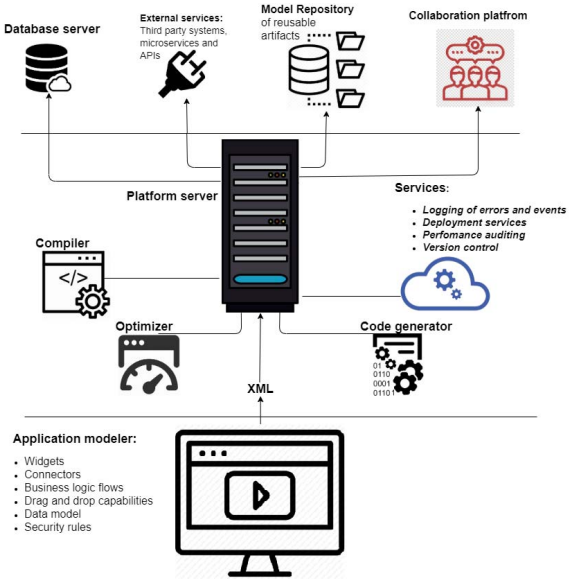


Fig. 2. Main components of low-code development platforms

modelers can use to define applications, as shown in the central part of the environment. The left-hand side of the figure shows an overview of the modeled system in terms of, e.g., the elements in the domain model, and the navigation model linking all the different specified pages. The application modeler also permits to run the system locally before deploying it. To this end, as shown in Fig. 2, the middle tier takes the application model received from the application modeler and performs model management operations including code generations and optimizations by also considering the involved services including database systems, micro-services, APIs connectors, model repositories of reusable artifacts, and collaboration means [8].

Concerning *database servers*, they can be both SQL and NoSQL. In any case, the application users and developers are not concerned about the type of employed database or mechanisms ensuring data integrity or query optimizations. More in general, the developer is not concerned about low-level architecture details of the developed application. All the needed *micro-services* are created, orchestrated and managed in the back-end without user intervention. Although the developer is provided with the environment where she can interact with external *APIs*, there are specific connectors in charge of consuming these APIs in the back-end. Thus, developers are relieved from the responsibility of manually managing technical aspects like authentication, load balance, business logic consistency, data integrity and security.

Low-code development platforms can also provide developers with repositories that can store reusable modeling artifacts by taking care of version control tasks. To support *collaborative development* activities, LCDPs include facilities supporting development methodologies like agile, kanban, and scrum. Thus, modelers can easily visualize the application development process, define tasks, sprints and deal with changes as soon as customers require them and collaborate with other stakeholders.

### C. Development process in LCDPs

The typical phases that are performed when developing applications by means of LCDPs can be summarized as follows.

1) *Data modeling* - usually, this is the first step taken; users make us of a visual interfaces to configure the data

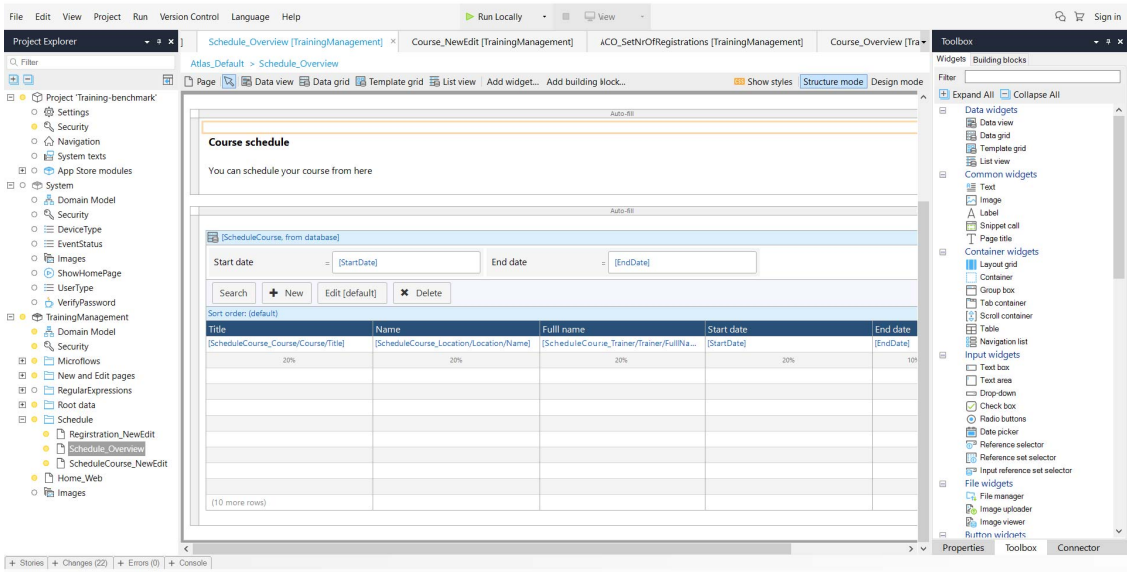


Fig. 3. The application modeler of Mendix at work

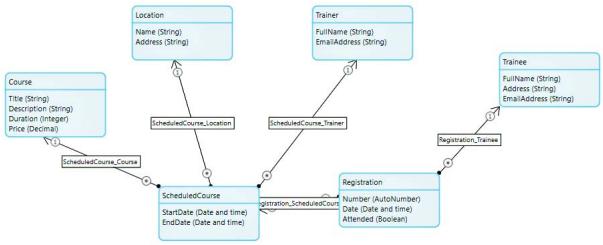


Fig. 4. A simple data model defined in Mendix

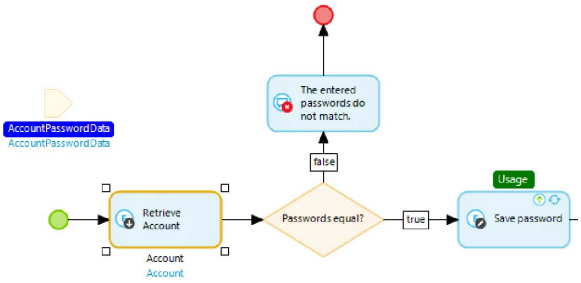


Fig. 5. A simple logic defined in Mendix

schema of the application being developed by creating entities, establishing relationships, defining constraints and dependencies generally through drag-and-drop facilities. A simple data model defined in Mendix is shown in Fig. 4.

2) *User interface definition* - secondly, the user configures forms and pages (e.g., see Fig. 3) used to define the application views, and later define and manage user roles and security mechanisms across at least entities, components, forms, and pages. It is here that drag-and-drop capabilities play a significant role to speed up development and render the different views quickly.
3) *Specification of business logic rules and workflows* - Third, the user might need to manage workflows amongst various forms or pages requiring different operations on the interface components. Such operations can be implemented in terms of visual-based workflows and to this end, BPMN-like notations can be employed as, e.g., shown in Fig. 5.
4) *Integration of external services via third-party APIs* - Fourth, LCDPs can provide means to consume external services via integration of different APIs. Investigating the documentation is necessary to understand the form and structure of the data that can be consumed by the adopted platform.
5) *Application Deployment* - In most platforms, it is possible to quickly preview the developed application and deploy it with few clicks.

## III. An Overview of Representative Low-Code Development Platforms

This section presents an overview of eight low-code development platforms that have been considered as leaders in the related markets from recent Gartner [20] and Forrester [18] reports. These eight low-code platforms are assumed to be representative platforms for the benefit of our analysis that encompasses diverse feature capabilities mentioned in Table I.

**OutSystem [8]** is a low-code development platform that allows developing desktop and mobile applications, which can run in the cloud or in local infrastructures. It provides inbuilt features which enable to publish an application via a URL with a single button click. OutSystems has two significant components. First, it has an intermediate Studio for database connection through .NET or Java and secondly, it has a service studio to specify the behaviour of the application being developed. Some of the supported applications in this platform are billing systems, CRMs, ERPs, extensions of existing ERP solutions, operational dashboards and business intelligence.

**Mendix [6]** is a low-code development platform that does not require any code writing and all features can be accessed through drag-and-drop capabilities while collaborating in real-time with peers. There is a visual development tool that helps to reuse various components to fasten the development process from the

data model setup to the definition of user interfaces. Users can create some context-aware apps with pre-built connectors, including those for the IoT, machine learning, and cognitive services. Mendix is compatible with Docker[2] and Kubernetes[3], and it has several application templates that one can use as starting points. Mendix's Solution Gallery[4] is an additional resource that permits users to start from already developed solutions, and that might be already enough to satisfy the requirements of interest.

**Zoho Creator [10]** offers drag-and-drop facilities to make the development of forms, pages and dashboards easy. The provided user interface supports web design where the layout of the page reflects the resolution of the screen of the user (e.g., in the case of mobile or desktop applications). It also offers integration with other Zoho apps and other Salesforce[5] connectors. Customized workflows are essential features of Zoho Creator.

**Microsoft PowerApps [7]** supports drag-and-drop facilities and provides users with a collection of templates which allows reuse of already developed artifacts. A user can follow model-driven or canvas approaches while building applications. PowerApps integrates with many services in the Microsoft ecosystem such as Excel, Azure database[6] or similar connectors to legacy systems.

**Google App Maker [3]** allows organizations to create and publish custom enterprise applications on the platform powered by G Suite[7]. It utilizes a cloud-based development environment with advanced features such as in-built templates, drag-and-drop user interfaces, database editors, and file management facilities used while building an application. To build an extensive user experience, it uses standard languages such as HTML, Javascript, and CSS.

**Kissflow [5]** is a workflow automation software platform based on the cloud to help users to create and modify automated enterprise applications. Its main targets are small business applications with complete functional features which are essential for internal use, and human-centred workflows such as sales enquiry, purchase request, purchase catalogue, software directory, and sales pipeline. It supports integrations with third-party APIs, including Zapier[8], Dropbox[9], IFTTT[10], and Office 365[11].

**Salesforce App Cloud [9]** helps developers to build and publish cloud-based applications which are safe and scalable without considering the underlying technological stacks. It exhibits out-of-the-box tools and operations for automation by integrating them with external services. Some of the peculiar features are the extensive AppExchange marketplace[12] consisting of pre-built applications and components, reusable objects and elements, drag-and-drop process builder, and inbuilt kanban boards.

**Appian [1]** is one of the oldest low-code platform, which permits to create mobile and Web applications through a personalization tool, built-in team collaboration means, task management, and social intranet. Appian comes with a decision engine which is useful for modeling complex logic.

[2]https://www.docker.com/
[3]https://kubernetes.io/
[4]https://www.mendix.com/solutions/
[5]https://www.salesforce.com/it/
[6]https://azure.microsoft.com
[7]https://gsuite.google.com/
[8]https://zapier.com
[9]https://www.dropbox.com/
[10]https://ifttt.com/
[11]https://products.office.com/it-it/home
[12]https://appexchange.salesforce.com/

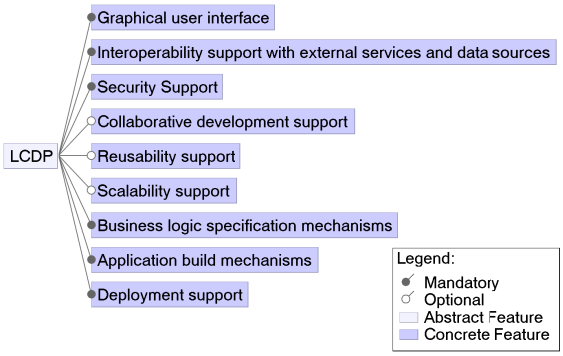


Fig. 6. Feature diagram representing the top-level areas of variation for LCDPs

## IV. Taxonomy

In this section we introduce preparatory terms, which can facilitate the selection and comparison of different LCDPs. The features are derived by examining the requirements in building an application along with the capabilities that a low-code platform could offer in achieving the making of an application. In particular, by analysing the low-code development platforms described in the previous section we identified and modeled their variabilities and commonalities. Our results are documented using feature diagrams [13], which are a common notation in domain analysis [14]. Fig. 6 shows the top-level feature diagram, where each subnode represents a major point of variation. Table I gives details about the taxonomy described in the following.

- *Graphical user interface:* This group of features represents the provided functionalities available in the front-end of the considered platform to support customer interactions. Examples of features included in such a group are drag-and-drop tools, forms, and advanced reporting means.
- *Interoperability support with external services and data sources*: This group of features is related to the possibility of interacting with external services such as Dropbox, Zapier, Sharepoint, and Office 365. Also, connection possibilities with different data sources to build forms and reports are included in such a group.
- *Security support*: The features in this group are related to the security aspects of the applications that are developed by means of the employed platform. The features included in such a group include authentication mechanisms, adopted security protocols, and user access control infrastructures.
- *Collaborative development support*: Such a group is related to the collaboration models (e.g., online and off-line) that are put in place to support the collaborative specification of applications among developers that are located in different locations.
- *Reusability support*: It is related to the mechanisms employed by each platform to enable the reuse of already developed artifacts. Examples of reusability mechanisms are pre-defined templates, pre-built dashboards, and built-in forms or reports.
- *Scalability support*: Such a group of feature permits developers to scale up applications according to different dimensions like the number of manageable active users, data traffic, and storage capability that a given application can handle.
- *Business logic specification mechanisms*: It refers to the provided means to specify the business logic of the application being modeled. The possibilities included in such a group are business rules engine, graphical workflow editor, and API support that allows one application to communicate with

other application(s). Business logic can be implemented by using one or more API call(s).
- *Application build mechanisms*: It refers to the ways the specified application is built, i.e., by employing code generation techniques or through models at run-time approaches. In the former, the source code of the modeled application is generated from the specified models and subsequently deployed. In the latter, the specified models are interpreted and used to manage the run-time execution of the application.
- *Deployment support*: The features included in such a group are related to the available mechanisms for deploying the modeled application. For instance, once the system has been specified and built, it can be published in different app stores and deployed in local or cloud infrastructures.

In addition to the top-level features shown in Fig. 6, LCDPs can be classified also with respect to the *Kinds of supported applications*. In particular, each LCDP can specifically support the development of one or more kinds of applications including Web portals, business process automation systems, and quality management applications.

## V. Comparison of relevant LCDPs

In this section, we make use of the taxonomy previously presented to compare the eight low-code development platforms overviewed in Sec. III. Table II shows the outcome of the performed comparison by showing the corresponding supported features for each platform. The data shown in Table II are mainly obtained by considering the official resources of each platform as referenced by [8], [6], [10], [7], [3], [5], [9], [1], and by considering the experience we gained during the development of a benchmark application as discussed in the next section.

### A. Features and capabilities

The essential and distinguishing features and capabilities of the analyzed low-code platforms can be summarized as follows: *OutSystems* provides developers with a quick mechanism to publish developed applications, the capability to connect different services, to develop responsive mobile and web-apps, security mechanisms and real-time dashboards. *Mendix* supports collaborative project management and end-to-end development, pre-built templates with app stores and interactive application analytics. *Zoho Creator* has an easy to use form builder, user-friendly and mobile-friendly user interfaces, and the capability of app-integration among different Zoho CRM apps, Salesforce, etc. It also supports pre-built templates and customized workflows. *Microsoft PowerApp* supports integration with Office 365, pre-built templates, easy mobile and tablet application conversion, and the capability to connect with third-party applications for basic application development. *Google App Maker* has a drag-and-drop function similar to most of the analyzed low-code platforms, app preview, reusable templates, deployment settings, means to specify access roles, built-in tutorials and google analytic integration. *Kissflow* supports progress tracking, custom and pre-built reports, collaborative features, and the possibility to use third-party services such as Google Doc, and Dropbox documents. It also supports Zapier to integrate different systems. *Salesforce App Cloud* has an extensive app market place for pre-built apps and components, reusable objects and elements, in-built kanban boards and a drag-and-drop process builder. *Appian* supports native mobile apps, drag-and-drop tools, collaborative task management, and a decision engine with AI-enabled complex logic.

### B. Additional aspects for comparing LCDPs

The taxonomy discussed in the previous section plays an important role when users have to compare candidate LCDPs and select one among possible alternatives. Further than the features previously presented, we identified additional aspects that are orthogonal to the presented taxonomy, and that can be taken into account when decision-makers have to decide if a low-code development platform has to be adopted and which one.

*Type of solutions to be developed:* there are two main types of applications that can be developed employing LCDPs, namely B2B (Business to Business) and B2C (Business to Customer solution). B2B solutions provide users with business process management (BPM) functionalities such as creation, optimization, and automation of business process activities. Examples of B2B solutions include hotel management, inventory management, and human resource management. Multiple applications can be combined in a B2B solution. B2C solutions provide more straightforward answers for end customers. B2C solutions are for developing single applications such as websites and customer relations management applications. The interactivity aspects of B2C is much more crucial than B2B ones.

*Size of the user's company/organization:* another dimension to be considered when selecting LCDPs, is the size of the company/organization that is going to adopt the selected LCDP. Organizations fall under three possible categories: *small* (with less than 50 employees), *medium* (if the number of employees is in between 50 to 1000), *large* (if the number of employees is higher than 1000). Thus, the decision-maker must keep in mind the organization size to identify the optimal solution according to her needs. Any organization who wishes to scale their enterprise at an optimum cost need to select an LCDP based on the strength of the company. LCDPs such as Salesforce app cloud, Mendix, and OutSystems support large enterprises, and they are used to develop large and scalable applications. Google App Cloud, Appian, Zoho Creator are instead mainly for supporting small to medium scale enterprises and they are relatively cheaper.

*Cost and time spent to learn the platform:* the time spent on the development, testing and deployment of an application may vary from one low-code platform to another. To be proficient in such processes, users must spend time to learn all the related aspects of that platform. Also, decision-makers have to consider potential training costs that have to be faced for learning the concepts and processes of that particular low-code platform.

*The price of the low-code platform:* it is one of the most critical criteria, especially for small or medium-scale companies. The price of the platform can be estimated as the price of using the platform for one developer per month. Moreover, the dimensions that contribute to the definition of the price include *i)* the number of applications that need to be deployed, and *ii)* where data are going to be stored, i.e., in on-premise databases, in cloud environments, or in hybrid configurations.

*Increase in productivity:* The adoption possibilities of low-code development platforms have to be assessed by considering the potential number of developed applications with respect to the time spent to learn the platform, the price incurred in training and to buy the licenses to use the considered platform.

## VI. Using LCDPs: a short experience report

The making of such platforms capable of giving citizen developers the ability to build fully-fledged applications faster and efficiently comes on a cost. Critical architectural decisions are made to ensure minimal coding, speed, flexibility, less upfront

TABLE I
TAXONOMY FOR LOW-CODE DEVELOPMENT PLATFORMS

| Feature | Description |
|---|---|
| *Graphical user interface* | |
| Drag-and-drop designer | This feature enhances the user experience by permitting to drag all the items involved in making an app including actions, responses, connections, etc. |
| Point and click approach | This is similar to the drag-and-drop feature except it involves pointing on the item and clicking on the interface rather than dragging and dropping the item. |
| Pre-built forms/reports | This is off-the-shelf and most common reusable editable forms or reports that a user can use when developing an application. |
| Pre-built dashboards | This is off-the-shelf and most common dashboards that a user can use when developing an application. |
| Forms | This feature helps in creating a better user interface and user experience when developing applications. A form includes dashboards, custom forms, surveys, checklists, etc. which could be useful to enhance the usability of the application being developed. |
| Progress tracking | This features helps collaborators to combine their work and track the development progress of the application. |
| Advanced Reporting | This features enables the user to obtain a graphical reporting of the application usage. The graphical reporting includes graphs, tables, charts, etc. |
| Built-in workflows | This feature helps to concentrate the most common reusable workflows when creating applications. |
| Configurable workflows | Besides built-in workflows, the user should be able to customize workflows according to their needs. |
| *Interoperability support* | |
| Interoperability with external services | This feature is one of the most important features to incorporate different services and platforms including that of Microsoft, Google, etc. It also includes the interoperability possibilities among different low-code platforms. |
| Connection with data sources | This features connects the application with data sources such as Microsoft Excel, Access and other relational databases such as Microsoft SQL, Azure and other non-relational databases such as MongoDB. |
| *Security Support* | |
| Application security | This feature enables the security mechanism of an application which involves confidentiality, integrity and availability of an application, if and when required. |
| Platform security | The security and roles management is a key part in developing an application so that the confidentiality, integrity and authentication (CIA) can be ensured at the platform level. |
| *Collaborative development support* | |
| Off-line collaboration | Different developers can collaborate on the specification of the same application. They work off-line locally and then they commit to a remote server their changes, which need to be properly merged. |
| On-line collaboration | Different developers collaborate concurrently on the specification of the same application. Conflicts are managed at run-time. |
| *Reusability support* | |
| Built-in workflows | This feature helps to concentrate the most common reusable workflows in creating an application. |
| Pre-built forms/reports | This is off-the-shelf and most common reusable editable forms or reports that a user might want to employ when developing an application. |
| Pre-built dashboards | This is off-the-shelf and most common dashboards that a user might want to employ when developing an application. |
| *Scalability* | |
| Scalability on number of users | This features enables the application to scale-up with respect to the number of active users that are using that application at the same time. |
| Scalability on data traffic | This features enables the application to scale-up with respect to the volume of data traffic that are allowed by that application in a particular time. |
| Scalability on data storage | This features enables the application to scale-up with respect to the data storage capacity of that application. |
| *Business logic specification mechanisms* | |
| Business rules engine | This feature helps in executing one or more business rules that help in managing data according to user's requirements. |
| Graphical workflow editor | This feature helps to specify one or more business rules in a graphical manner. |
| AI enabled business logic | This is an important feature which uses Artificial Intelligence in learning the behaviour of an attributes and replicate those behaviours according to learning mechanisms. |
| *Application build mechanisms* | |
| Code generation | According to this feature, the source code of the modeled application is generated and subsequently deployed before its execution. |
| Models at run-time | The model of the specified application is interpreted and used at run-time during the execution of the modeled application without performing any code generation phase. |
| *Deployment support* | |
| Deployment on cloud | This features enables an application to be deployed online in a cloud infrastructure when the application is ready to deployed and used. |
| Deployment on local infrastructures | This features enables an application to be deployed locally on the user organization's infrastructure when the application is ready to be deployed and used. |
| *Kinds of supported applications* | |
| Event monitoring | This kind of applications involves the process of collecting data, analyzing the event that can be caused by the data, and signalling any events occurring on the data to the user. |
| Process automation | This kind of applications focuses on automating complex processes, such as workflows, which can takes place with minimal human intervention. |
| Approval process control | This kind of applications consists of processes of creating and managing work approvals depending on the authorization of the user. For example, payment tasks should be managed by the approval of authorized personnel only. |
| Escalation management | This kind of applications are in the domain of customer service and focuses on the management of user viewpoints that filter out aspects that are not under the user competences. |
| Inventory management | This kind of applications is for monitoring the inflow and outflow of goods and manages the right amount of goods to be stored. |
| Quality management | This kind of applications is for managing the quality of software projects, e.g., by focusing on planning, assurance, control and improvements of quality factors. |
| Workflow management | This kind of applications is defined as sequences of tasks to be performed and monitored during their execution, e.g., to check the performance and correctness of the overall workflow. |

investment and out-of-box functionalities that deliver the full application faster. However, decisions that are usually taken during the usage of LCDPs can give place to some issues that might emerge later on. In particular, to get insights into LCDPs, we developed the same benchmark application by employing different platforms, and in particular Google App Maker, Mendix, Microsoft PowerApps and OutSystems. The benchmark application is a course management system intended to facilitate trainers and trainees to manage their courses, schedules, registrations and attendance. Despite the simplicity of the application, it exhibits general user requirements that are common during the development of typical functionalities such as management of data, their retrieval and visualization. Moreover, we had the possibility of integrating external services via third-party APIs. We managed to investigate how reusable code and artefacts developed in one platform can be integrated into other low code platforms hence smoothing the path toward discovery and reuse of already proven artefacts across different platforms.

The first performed activity to develop the benchmark application was the elicitation of the related requirements. We came up with the corresponding use cases, and thus with the functional requirements of the system. According to the performed experience, software applications can be built in LCDPs by following two main approaches:

- *UI to Data* - the developer starts building the application by creating a user interface and then linking it with the needed data sources. Forms and pages are defined first followed by the specification of business logic rules and workflows,

TABLE II
COMPARISON OF ANALYSED LOW-CODE DEVELOPMENT PLATFORMS

| Feature | OutSystems | Mendix | Zoho Creator | MS PowerApp | Google App Maker | Kissflow | Salesforce App Cloud | Appian |
|---|---|---|---|---|---|---|---|---|
| *Graphical user interface* | | | | | | | | |
| Drag-and-drop designer | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ |
| Point and click approach | | | | ✓ | | | | |
| Pre-built forms/reports | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Pre-built dashboards | ✓ | | ✓ | ✓ | | ✓ | ✓ | |
| Forms | | | ✓ | ✓ | | | | |
| Progress tracking | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Advanced reporting | | | | | | ✓ | | |
| Built-in workflows | | | ✓ | | | ✓ | ✓ | |
| Configurable workflows | | | ✓ | | | ✓ | ✓ | |
| *Interoperability support* | | | | | | | | |
| Interoperability with external service | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ |
| Connection with data sources | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| *Security Support* | | | | | | | | |
| Application security | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Platform security | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| *Collaborative development support* | | | | | | | | |
| Off-line collaboration | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| On-line collaboration | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ |
| *Reusability support* | | | | | | | | |
| Built-in workflows | | | ✓ | | | ✓ | ✓ | |
| Pre-built forms/reports | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Pre-built dashboards | ✓ | | ✓ | ✓ | | ✓ | ✓ | |
| *Scalability* | | | | | | | | |
| Scalability on number of users | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Scalability on data traffic | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| Scalability on data storage | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| *Business logic specification mechanisms* | | | | | | | | |
| Business rules engine | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Graphical workflow editor | ✓ | ✓ | | | | ✓ | ✓ | |
| AI enabled business logic | ✓ | | | | | ✓ | ✓ | ✓ |
| *Application build mechanisms* | | | | | | | | |
| Code generation | ✓ | | | | | | | |
| Models at run-time | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| *Deployment support* | | | | | | | | |
| Deployment on cloud | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Deployment on local infrastructures | ✓ | ✓ | | | | | ✓ | ✓ |
| *Kinds of supported applications* | | | | | | | | |
| Event monitoring | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Process automation | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ |
| Approval process control | | | | | ✓ | | | |
| Escalation management | | | | | | ✓ | | |
| Inventory management | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Quality management | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Workflow management | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

which then lead to the integration of external services before the application deployment. LCDPs such as Mendix, Zoho Creator, Microsoft PowerApps, and Kissflow can follow this approach.

- *Data to UI* - it is a data-driven approach that starts from data modeling and then builds the user interface of the application followed by the specification of business logic rules and workflows. Afterwards, it leads to the integration of external services, if needed before the deployment of the application. LCDPs such as OutSystems, Mendix, Zoho Creator, Microsoft PowerApps, Salesforce App Cloud and Appian can follow this approach.

Specification of business logic rules, workflows, and the integration of external services can be swapped according to the developer's style in both the approaches mentioned above.

By developing the considered benchmark applications with the considered LCDPs, we managed to identify some challenges that users and developers are likely to face along the course of development in LCDPs such as interoperability issues among different low-code platforms, extensibility limitations, steep learning curves, and scalability issues [15] [19] [20]. Below we discuss such challenges that transcend most of the low-code development platforms we surveyed. We will not discuss potential challenges that might also occur concerning code optimization or security and compliance risks, because we were not able to deeply assess these features due to the lack or limited visibility of the considered low-code platforms. However, we acknowledge that such aspects should be investigated in the future to give a more broad perspective about potential challenges that might affect LCDPs.

**Low-code platforms' interoperability**: this characteristic ensures interaction and exchange of information and artefacts among different low-code platforms, e.g., to share architectural design, implementation or developed services. Such a feature is also essential to mitigate issues related to vendor lock-ins. Unfortunately, most low-code platforms are proprietary and closed sources. There is a lack of standards in this domain by hampering the development and collaboration among different engineers and developers. Thus, they are unable to learn from one another, and the reuse of already defined architectural designs, artefacts and implementations are still hampered.

**Extensibility**: the ability to add new functionalities not offered by the considered platform is hard in such proprietary platforms or even impossible. Due to lack of standards, some of them require extensive coding to add new capabilities, which have to adhere to architectural and design constraints of the platform being extended.

**Learning curve**: most of the platforms have less intuitive graphical interfaces. For some of them, drag-and-drop capabilities are limited, and they do not provide enough teaching material, including sample applications and online tutorials to learn the platform. Consequently, the platform adoption can be affected. The adoption of some platforms still requires knowledge in software development, thus limiting their adoption from citizen

developers who are supposed to be the main target of these platforms and products.

**Scalability**: Low code platforms should be preferably based on the cloud and should be able to handle intensive computations and to manage big data, which get produced at high velocity, variety, and volume [17]. However, due to lack of open standards for such platforms, it is very challenging to assess, research and contribute to the scalability of these platforms.

Overall, LCDPs are suitable for organizations that have limited IT resources and budget because they can deliver fully-featured products in a short time, as in the case of CRM applications. However, the development possibilities depend on the functionalities provided by the available modules, and users might need accommodating their initial requirements depending on the options offered by the employed platform. Third-party integration and the management and maintenance of the developed applications can be hampered depending on the extensibility capabilities of the employed LCDPs.

## VII. Conclusion and future work

Over the last years, the interest around LCDPs has significantly increased from both academia and industry. Understanding and comparing hundreds of low-code platforms [2] can be a strenuous and challenging task without the availability of an appropriate conceptual framework underpinning their evaluation. In this paper, we analyzed eight low-code platforms that are considered as leaders in the related market, to identify their commonalities and variability points. An organized set of distinguishing features has been defined and used to compare the considered platforms. A short experience report has been also presented to discuss some essentials features of each platform, limitations and challenges we identified during the course of development of our benchmark application.

In the future, we plan to refine the proposed taxonomy by considering additional LCDPs to achieve a comprehensive and thoroughly validated set of features. Such a refinement process might also involve the different LCDP providers to validate the crated taxonomy further and to share with them the challenges and lessons we learned during the development of the discussed benchmark application. Moreover, we plan to focus more on the reusability and interoperability facilities that are needed by low-code platforms. The main goals are *i)* the design and development of a repository supporting the reuse of already developed low-code artifacts, *ii)* and development of generic mechanisms enabling the interoperability of different low-code development platforms.

## Acknowledgment

This work is funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie – ITN grant agreement No 813884.

## References

[1] Appian platform overview. https://www.appian.com/. Accessed: 2020-03-23.
[2] Best low-code development platforms software. https://www.g2.com/categories/low-code-development-platforms. Accessed: 2020-05-26.
[3] Google app maker platform guide. https://developers.google.com/appmaker/overview. Accessed: 2020-03-23.
[4] An introduction to low-code platform. https://www.mendix.com/low-code-guide/. Accessed: 2020-03-23.
[5] Kissflow platform overview. https://kissflow.com/process-management/. Accessed: 2020-03-23.
[6] Mendix platform features. https://www.mendix.com/platform/. Accessed: 2020-03-23.
[7] Microsoft powerapps platform overview. https://docs.microsoft.com/en-us/powerapps/maker/. Accessed: 2020-03-23.
[8] Outsystem platform features. https://www.outsystems.com/platform/. Accessed: 2020-03-23.
[9] Salesforce app cloud platform overview. https://developer.salesforce.com/platform. Accessed: 2020-03-23.
[10] Zoho creator platform features. https://www.zoho.com/creator/features.html. Accessed: 2020-03-23.
[11] Francesco Basciani, Juri Rocco, Davide Di Ruscio, Amleto Di Salle, Ludovico Iovino, and Alfonso Pierantonio. Mdeforge: An extensible web-based modeling platform. volume 1242, 09 2014.
[12] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*, volume 1. 09 2012.
[13] Krzysztof Czarnecki. *Generative programming - principles and techniques of software engineering based on automated configuration and fragment-based component models*. PhD thesis, Technische Universität Illmenau, Germany, 1999.
[14] Krzysztof Czarnecki. *Domain Engineering*, pages 433–444. American Cancer Society, 2002.
[15] Justice Opara-Martins, R. Sahandi, and Feng Tian. Implications of integration and interoperability for enterprise cloud-based applications. pages 213–223, 10 2015.
[16] C. Richardson and J. R. Rymer. The forrester wave: Low-code development platforms, q2 2016. tech. rep. *Forrester Research*, 2016.
[17] Juri Rocco, Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. Collaborative repositories in model-driven engineering [software technology]. *IEEE Software*, 32:28–34, 05 2015.
[18] John R. Rymer. The forrester wave: Low-code platforms for business developers, q2 2019. *Forrester Research*, 2019.
[19] Amandeep Singh, Pardeep Mittal, and Neetu Jha. Foss: A challenge to proprietary software. *IJCST*, 4, 2013.
[20] Paul Vincent, Kimihiko Iijima, Mark Driver, Jason Wong, and Yefim Natis. Magic quadrant for enterprise low-code application platforms. *Gartner report*, 2019.
[21] Robert Waszkowski. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, 52(10):376–381, 2019.