

# Low-Code vs. Model-Driven Architecture

Markus Reiter

University of Innsbruck, Austria

**Abstract.** In this paper we present a survey conducted by evaluating various low-code tools in order to get a better understanding of the advantages and drawbacks. The evaluation entails creating a test application with each low-code tool and assessing factors such as ease of use and scalability. Using the results from this evaluation, we try to determine if and when low-code platforms are a viable alternative or even preferable to traditional software development using a model-driven architecture.

**Keywords:** Low-Code · Model-Driven · Development

## 1 Introduction

In this paper, we compare Low-Code Platforms to Model-Driven Architecture and try to determine when and where to use them over Model-Driven Architecture or vice-versa.

In order to make clear to the reader the difference between the two approaches we firstly provide definitions and characteristics of each paradigm in the following sections.

### 1.1 What is Model-Driven Architecture?

With Model-Driven Architecture or Model-Driven Engineering, the goal is to standardise on models in a given domain. This is done in order to reduce code duplication and speed up development. Especially when bootstrapping a new project, it is helpful keep boilerplate code to a minimum. In addition to reducing development time, Model-Driven Engineering also tries to increase productivity by maximising compatibility between systems by using the aforementioned standardised models. [1] Generally, Model-Driven Architectures are geared towards developers, i.e. people who also have a good understanding of the underlying programming language(s) of the respective architecture. One common way Model-Driven Engineering is used in practice is code generation. Using code generation can speed up many processes: For example, given an API specification (e.g. using the industry standard OpenAPI [2]), one can generate all code that is necessary for a client for this API. Swagger [3] is one such tool which can generate API clients for many different programming languages. There are also tools which can generate a complete class hierarchy from an UML diagram; many IDEs offer such functionality either natively or via plug-ins. These are only a few examples of what falls into the category of Model-Driven Architecture, the main point is that all of them help developers reach their goal faster than doing the same tasks manually.

### 1.2 What is a Low-Code Platform?

A low-code platform, as the name implies, requires a minimal amount of code to be written. Similarly to conventional programming (i.e. using a programming language), an IDE is used for creating programs using low-code platforms. Often, the IDE is part of the low-code platform itself instead of a standalone program as is the case for conventional programming. In most cases, the IDE of the platform and the low-code platform cannot be used separately so the term low-code platform is also commonly used to refer to the IDE interface itself. Low-code platforms are targeted towards users with little to no programming experience, which means the actual programming of a low-code system is done mainly via

visual building blocks. These building blocks contain pre-built templates and functionality and can be combined to form a coherent user interface. Oftentimes, custom functionality can be implemented using a click-and-drag pattern between building blocks and by assigning an action accordingly. The available building blocks vary by platform which makes them less portable than a system written using a programming language. This is one common point of critique on low-code platforms.

Another sub-category of low-code platforms are so called no-code platforms [4]. Where to draw the line between low-code and no-code is not clearly defined, especially since low-code platforms can sometimes be used in a way that cannot be differentiated from no-code platforms. For the purposes of this paper, low-code platforms are general purpose visual programming languages [5] aimed at both developer and non-developer users while no-code platforms are closer to software as a service (SaaS) platforms targeted mostly (or only) towards non-developers. Another way to look at it is that low-code platforms still give developers the freedom to add custom building blocks for their specific use case and no-code solutions are usually purpose-made for a specific line of business or a specific type of department. Some examples include monday.com [6], a no-code platform for project management tasks; FileMaker [7], which provides a database engine and a corresponding no-code platform for creating graphical user interfaces for interacting with databases; and Airtable [8], a no-code platform for creating databases connected to interactive spreadsheets.

## 2 Evaluation of Low-Code Tools

### 2.1 Selection of Low-Code Tools

Before evaluating low-code tools, we had to first determine which tools to evaluate. Given the ever evolving landscape of low-code platforms, there are countless companies offering low-code solutions. The following is an incomplete list of low-code platforms we found:

- Oracle APEX (Application Express) (originally released as Oracle Flows in 2000) by Oracle (founded in 1977)
- OutSystems (founded in 2001)
- Mendix (founded in 2005), a subsidiary of Siemens
- Simplifier (founded in 2012 as iTiZZiMO, renamed 2019 to Simplifier, the same name as its low-code platform)
- OSBP (Open Standard Business Platform), an open-source low-code platform (developed since 2016) under the umbrella of the Eclipse Foundation
- Corteza Low Code, part of the open-source Corteza Project (released in 2019), originally developed by Crust Technology

We wanted our evaluation to contain at least one open-source platform (e.g. OSBP), one platform backed by a well-known corporation (e.g. Oracle), one platform by a relatively new/unknown corporation (e.g. Simplifier), one relatively old/well-established platform (e.g. Oracle APEX or OutSystems) and one relatively new platform (e.g. Simplifier or Corteza). With the given list of low-code platforms, we could satisfy this requirement.

To ensure we could properly test all of these tools, we conducted a brief first inspection of each. Therefore, the following section will give a brief explanation of each platform, any problems encountered during the initial setup and the steps necessary before being able to use the low-code platform.

**OSBP (Open Standard Business Platform)** The Open Standard Business Platform is an open-source development environment which combines no-code and low-code with traditional application development. [9] It is the community version of OS.bee [10], a commercial product developed by COMPEX only available to business users. The interface for the OSBP platform is implemented via a plug-in for the Eclipse IDE. The user first needs to install the Eclipse IDE, after which they can add the link to the plug-in repository and install it directly from within the IDE. Unfortunately, the latest version of OSBP was released more than one year ago and is incompatible with the latest version of the Eclipse IDE. At the time of writing (Jan 2, 2021), the last commit in the OSBP Git repository was on May 9, 2019. The latest version of the Eclipse IDE was 2020-12. The OSBP website recommended downloading Eclipse Neon, a version of the Eclipse IDE released in 2016. A non-functioning IDE plug-in for an outdated version of said IDE which has not been updated in over a year is a red flag. We therefore deemed OSBP unsuitable and decided to exclude it from further evaluation.

**Corteza** Corteza Low Code is part of the Corteza Project [11] initiated by Crust Technology. The Corteza Project encompasses Corteza CRM Suite, an open-source customer relationship management tool built entirely on Corteza Low Code; Corteza Messaging, a messaging solution similar to the likes of Slack; Corteza One, a unified workspace from which to access and run web applications and finally Corteza Low Code, a low-code platform designed for building record-based applications. It is possible to try out Corteza directly in the browser by creating an account or by using an existing GitHub or Google account to sign up. We chose to set up a local demo deployment using Docker. The Corteza Project provides very good documentation on how to get started, including a ready-made `docker-compose.yml` file containing all necessary services for a full Corteza installation. After a simple call to `docker-compose up -d`, the Corteza server was running locally. The last step before being ready to use was to set up an administrator account in the local Corteza instance.

**Oracle APEX (Application Express)** Oracle Application Express [12] is a web-based application development platform developed by Oracle. It is backed by an Oracle database and allows building web applications using a low-code interface. Oracle offers three ways to set up APEX. The first is to use it as part of their Oracle Cloud offering, secondly, by requesting a free workspace or lastly by self-hosting using the Oracle APEX On Premise variant. For our purposes, we chose to request a free workspace for testing. We chose a name for our namespace and registered using an email address and shortly thereafter we were sent an approval email with the login details. After that, Oracle APEX was ready to use.

**Simplifier** Simplifier is, like Oracle APEX, a web-based application development environment on low-code basis. Simplifier offers a demo platform called “Simplifier Playground”, an instance of the Simplifier platform specifically set up for trialling the platform. The trial instance is reset to defaults every night, so all data stored on it will not be persisted. They also offer a “Freemium” instance, available by registering via an email address, which has no such limitation, we therefore applied for such an instance. Shortly afterwards, we received a confirmation email with the URL to our instance, which unfortunately returned a “400 Bad Request” error. We contacted the Simplifier support team about this and decided to get familiar with the platform using the Simplifier Playground in the meantime.

**Mendix** Similar to Oracle APEX and Simplifier, Mendix is also a web-based environment for low-code application development. Mendix offers a free tier which allows hosting unlimited applications with 1GB of memory and 0.5GB of storage per application. For more advanced needs, Mendix offers an enterprise option starting at 1875€ per month, allowing on-premises hosting and hosting on private clouds, e.g. AWS, IBM etc. For our evaluation purposes, the free tier is more than enough given there are no restrictions on the provided functionality. Creating an account was straight-forward and we were able to log in within minutes of registering. There are two options for developing applications using Mendix: The first is the web-based environment called Mendix Studio, the second a native desktop application called Mendix Studio Pro, offering more advanced options for professional users. Unfortunately Mendix Studio Pro is only available for Microsoft Windows, so we were not able to test it, the basic functionality for creating low-code application is the same in both versions of Mendix Studio, however.

## 2.2 Developing a TODO List Application

In order to evaluate each platform, we decided to build a simple TODO list application. The user should be able to add items to the list containing a title, body text and the status of the TODO item (either “done” or “outstanding”).

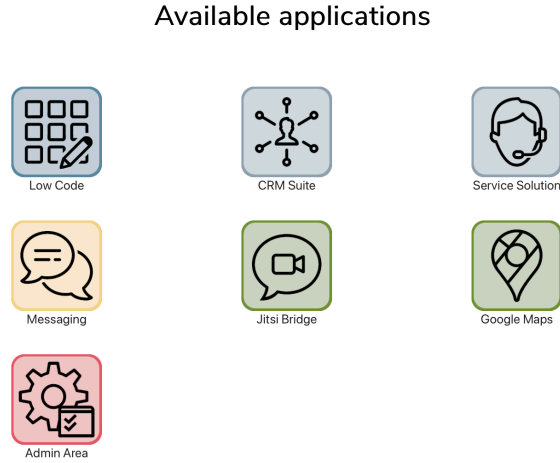


Fig. 1: Corteza Home Screen

**Corteza** In figure 1, we can see the initial screen of the Corteza platform. By selecting “Low Code”, we reach a new view containing all existing “Low Code Namespaces”. A namespace can be thought of as an individual application within the low-code environment. In order to start with our application, we create a `todo-list` namespace. Next we need to create a module. A module in Corteza is similar to creating a database schema for a single table. For our application, we create a module containing text fields for `title` and `body` and a drop-down field for the `status`. We also mark `title` as a required field, as seen in figure 2. Next, we need to create a page which lets us add TODOs to our list. To do this, we create a blank page and add the pre-built “Record List” block. In the “Record List” block, we select our “TODO Module” as the data source and select all fields so they are displayed in the list. The list also provides an option to allow inline editing of records. Without this option, the user has to click on the list item to open the detail page in order to edit it. Unfortunately, for some reason when enabling the checkbox to allow inline editing, we were unable to save the page with this configuration. Nevertheless, the basic functionality of our application was completely implemented in less than ten minutes.

Edit module

Module name

Todo Module

Handle

todo\_module

Manage record fields

Name	Title	Type	Multi value	Required	
status	Status	Select / dropdown	<input type="checkbox"/>	<input type="checkbox"/>	
title	Title	Text input (string)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
body	Body	Text input (string)	<input type="checkbox"/>	<input type="checkbox"/>	

+ Add new field

Fig. 2: Creation of a Corteza Module

There are also some other features in Corteza: The “Chart Builder” allows creating charts for data in a module and it is possible add custom buttons with automations. The custom automations however, have to be fully implemented in JavaScript, so we didn’t evaluate them any further given our focus on low-code.

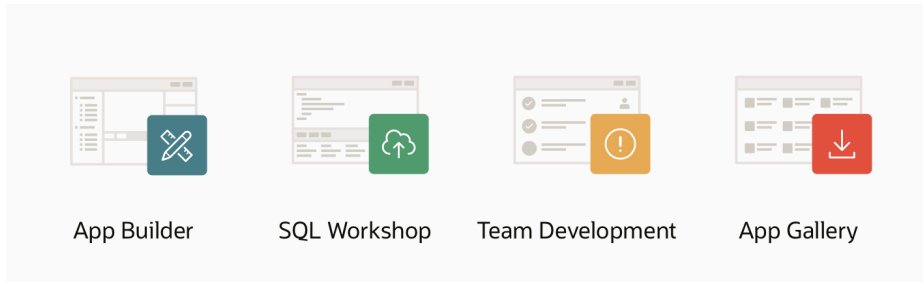


Fig. 3: Oracle APEX Home Screen

**Oracle APEX (Application Express)** When first logging into Oracle APEX, the user is greeted with a an overview of all applications in the workspace and a quick menu for accessing the major parts, App Builder, SQL Workshop, Team Development and App Gallery, as seen in figure 3. App Builder is the low-code development interface, allowing the user to create page-based web applications, SQL Workshop is an interface to create database tables for use in low-code applications. Team Development is an integrated project management tool that

helps coordinate tasks across team members when multiple people are working together in the same namespace. And finally, App Gallery is a gallery containing sample applications to use as either templates or guidance when developing a new low-code application.

Before starting with our TODO list application, we decided to set up a sample application called “Sample Database Application”, which contains a basic inventory management application, including a product list which we used as a template for our TODO list.

For our TODO list, we created a new application called “TODO List”. The setup wizard allows customising the colour theme of the application, the menu style (e.g. side menu or top menu), language as well as the application icon. Once created, we had a blank application containing a login page and a home page. The next thing we needed to do was set up a database table for our TODO items. Basic SQL knowledge is required to do this since properly setting up a primary key and a unique constraint for it is necessary to be able to insert data later on. We now had a `TODOS` table containing `ID`, `STATUS`, `TITLE` and `BODY` and were now able to add a new page to our application.

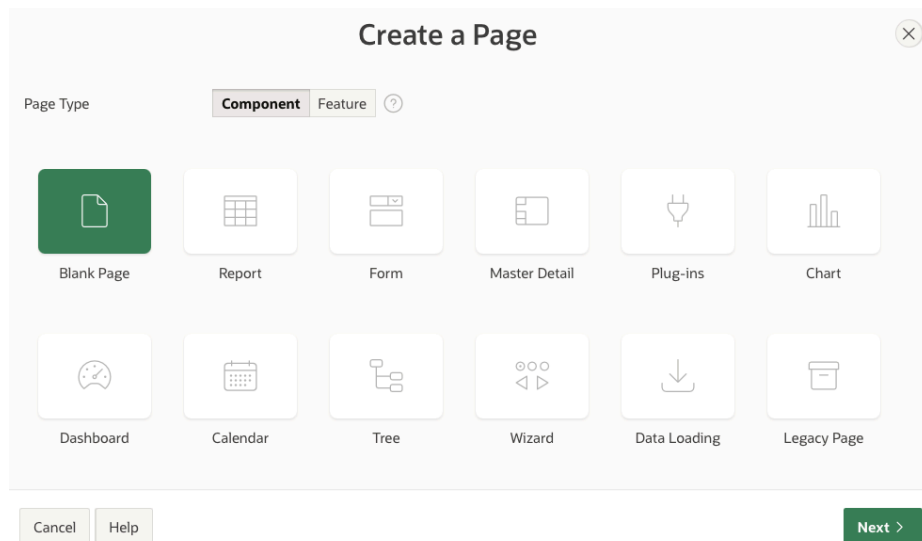


Fig. 4: Oracle APEX Create Page Dialog

There are many different page styles to choose from, as seen in figure 4. For our use case, we chose the “Form” page template. The user simply assigns a page name and subsequently selects the corresponding database table. The result is a form page with all fields of the database table which lets the user add new tasks



and edit existing ones. Some minor visual adjustments were necessary for the form page, i.e. hiding the automatically generated ID and setting the **STATUS** to “outstanding” by default when adding a new task. The next step was to create a list containing our TODO items with a button that opens the form page to add new ones. Similar to the page templates, there are many different building blocks to choose from. We used the same as the one used in the sample application, the “Interactive Report”, a table view with filtering options. Lastly, we added a **CREATE** button to the top of this component, linking it to the form page created earlier.

Oracle APEX provides many advanced options for all building blocks which allows building custom logic by connecting them via events (e.g. mouse click or value change). It is also possible to create custom events: For example, a form page (in dialog form) provides a **Cancel Dialog** event which is fired when the **CANCEL** button is pressed.

**Simplifier** Upon logging into a Simplifier instance, the user sees the Simplifier dashboard, seen in figure 5. The dashboard provides an overview of all elements represented by tiles. The “Applications” tile opens a list of applications, allowing to edit them and create new ones. “Widgets” allows viewing existing building blocks and also creating new reusable components. “Connectors” allow specifying connections to external services, e.g. for sending emails or accessing a ReST API. The “Business Objects” tile is used to create workflows by assigning connectors, plugins and functions to business objects. There is also a “Database Designer” which is used for creating database schemas.













<b>Applications</b> 10  Create, manage and configure applications, widgets and libraries. Process mapping defined within user stories.	<b>Widgets</b> 440  Create, manage and configure widgets, widget groups and widget categories.	<b>Connectors</b> 14  Create, manage and configure the interfaces, respective logins and certificates to connect to different systems and devices.	<b>Business Objects</b> 17  Merge the connectors, plugins and business objects for easy and fast reuse of complex business requirements.
<b>Database Designer</b> 0  Create and deploy persistent data structures. These can store user-defined data of your apps and be accessed by connector via OData.	<b>Data Types</b> 86  Create, manage and configure domain types, structures and collections as well as define validation rules.	<b>Users</b> 1  Create, administrate and configure all of your Simplifier users, groups and roles with their corresponding user permissions.	<b>Transports</b> 0  Migration of applications and individual components to other Simplifier instances, inc. simulation and validation of transports.
<b>Plugins</b> 2  Offers the possibility to extend or change the core functions of the Simplifier with the help of any external plugin.	<b>Logs &amp; Monitoring</b>  Central monitoring and filtering of all user and system activities. Provides detailed information which are very helpful for debugging.	<b>Jobs</b> 0  Create and administrate jobs for the execution of business objects. These are based on flexibly configurable time intervals.	<b>Templates</b> 6  Creation and definition of reusable HTML text components. These can be personalized by using of different, predefined placeholders.

Fig. 5: Simplifier Dashboard

For our TODO List application, we first needed to create a connector to a database. Simplifier allows creating a connector to MySQL, PostgreSQL and other common databases. We opted for a **SQL** connector using the SQLite back end to avoid having to set up an external database server. Next, we opened the “Database Designer” and created a schema for a TODO list item, choosing the **SQL** connector we created as data source. The schema can then be edited using an entity-relationship (ER) diagram editor. We added a entity called **task** with fields for **status**, **title** and **body**. With a click of a button, we could then deploy the changes in the ER diagram to the database.

Next, we created a new application called **TODO\_List**. We added a custom header bar in order to add a **add\_task** button for adding new tasks and a list view for displaying the tasks. We also added a second screen to our application containing a dialog box for adding a new task. We then opened the “Process” tab, which allows creating workflows. Workflows are represented as stories, which can be edited using a graphical interface. First, we created a workflow that opens the “Add Task dialog” which can be seen in figure 6 and figure 7.

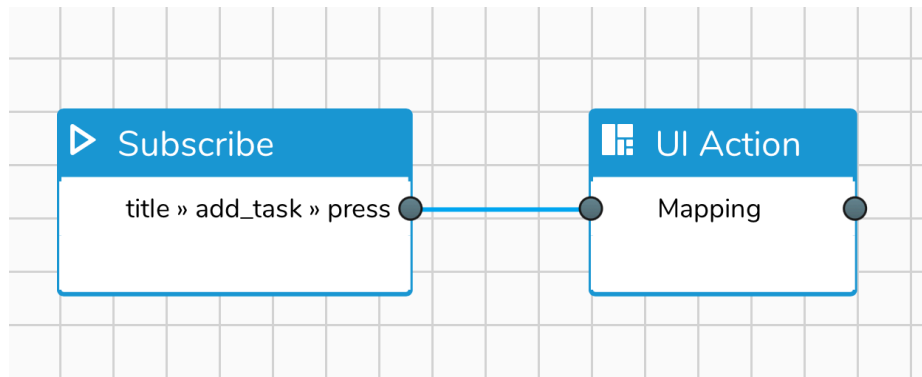


Fig. 6: Simplifier application process story: If the **add\_task** button is pressed, execute a **UIAction**. The inner logic of the **UIAction** is shown in figure 7.

Unfortunately, when creating a second workflow that is supposed to create a new task in the database from the inputs in the dialog, we received a “Maximum call stack exceeded.” error. The Simplifier Playground offers two similar applications to ours, a TODO List using a ReST back end and a shopping list using a SQLite back end. We took a closer look at the shopping list application and saw that it defines custom functions for accessing the database. Our TODO List application on the other hand uses the “Data Service” connector automatically generated for the database schema we created. There were no other demo applications using the “Data Service” connector, so we could not compare our process

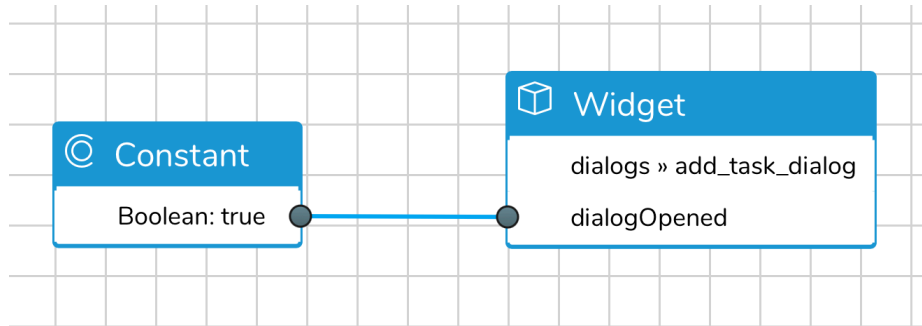


Fig. 7: Inner logic of the `UIAction` component shown in figure 6: Open the `add_task_dialog` dialog by setting the `dialogOpened` property to `true`.

to an existing working solution. Additionally, the TODO List demo application did also experience an error when trying to create new tasks.

**Mendix** Mendix offers three ways for creating a new application: Creating a blank application from scratch, generating an application from spreadsheet data or using one of many templates as a starting point. For our TODO list application, we chose to create an application from scratch. After creating an application, you can either edit it using the web interface or open it in the Mendix Studio Pro desktop application. We developed our TODO list application entirely using the web interface. The Mendix interface provides four main menu items: Pages, Domain Models, Microflows and Navigation. Pages is used to manage the different pages needed for the application. Domain Models contains the entities for our application, in our case we needed an entity for items in our TODO list. Microflows are used to add custom logic (micro workflows) to the application, e.g. when clicking a button. An interesting feature is Mendix Assist, an AI-based assistant making suggestions to the user when creating Microflows. Lastly we have Navigation, which is simply the representation of the application menu hierarchy.

For our application, we started by adding a new page for creating a task on our TODO list. There are numerous page templates to choose from, we used the “Form Horizontal” template as seen in figure 8. The resulting page (`add_task`) contains a so called data view. Every data view needs a data source, so we were prompted to create a new entity. We added a new entity called `task` and while editing the form, we were prompted to add attributes to that entity. We added two `String` attributes for `title` and `body`. We also needed to add a `status` attribute; for this, we edited the entity directly, adding an attribute of type `Enumeration` with values for “done” and “outstanding” and set the default value to “outstanding” accordingly.

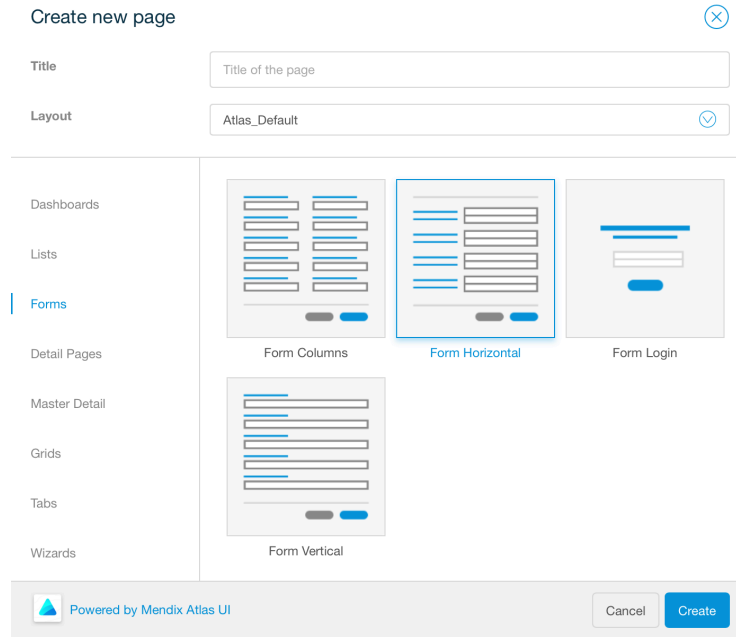


Fig. 8: Mendix Create Page Dialog

Next, we added a list view to the **home** page, choosing the **task** entity as data source in the process. We could now drag text fields into the list view and choose the corresponding attributes to be displayed. In order to actually add new tasks, we also added a “New task” button. For the click action, the user can choose to open a page or run a Microflow or a predefined action such as saving changes on the current page. We chose our page for creating a new task as the target and also needed to enable the “Create object” option. With this option, we pass an empty **task** object to the page, since the data view on the page expects a **task** as data source.

Lastly, we created another form page (**edit\_task**), also containing the **status**, in order to be able to edit a **task**. To access it, we added a click event to the list view on the **home** page. Since the list view data source is a **task**, it is automatically passed to the data view on the **edit\_task** page.

### 3 Assessment of Low-Code Tools

In this section, we assess whether the evaluated low-code platforms are suitable for serious usage by rating the following aspects:

- ease of use

- customisability
- portability
- scalability
- suitability for mission-critical applications

**OSBP (Open Standard Business Platform)** Given that we were not able to evaluate OSBP, we cannot assess whether applications created using it are scalable. The same is true for determining whether it is easy to use and customisable. It not working with the latest version of the Eclipse IDE (at the time of writing) also makes it very much unsuitable for mission-critical applications.

**Corteza** The Corteza-Low-Code platform is meant for building record-based management applications, making is very easy to use. Creating a simple TODO List application was possible in less than 10 minutes and should be possible in less than 5 minutes once you are familiar with the platform. Apart from choosing the order of predefined building blocks, customising the layout of an application is hindered by the fact that it is specialised for building management applications. The selection of pre-built blocks is very limited, as is the selection of colour schemes. Corteza uses a MySQL database in the back end, making at least the stored data very portable. The created applications are virtually unportable since they are tightly integrated into the Corteza platform, custom scripts are at least partially portable since they are written in JavaScript. In terms of scalability, creating complex applications will be difficult using the small selection of building blocks. Process management applications which don't rely on many different UI elements, like the included Corteza CRM, can definitely be implemented in a scalable manner. Given the overall simplicity of the Corteza platform, it is certainly suitable for creating mission-critical applications.

**Oracle APEX (Application Express)** Creating an application Oracle APEX requires basic SQL knowledge and the page designer is quite complex. Nevertheless, we were able to create our TODO List application in less than two hours by reviewing a sample application. It is also possible to completely generate the basic structure of an application using a wizard by importing data from a file, making the initial setup even easier. For further development, there is a small learning curve for getting familiar with the interface. The default theme offers three different menu types and four colour schemes and there is an option to use a custom theme. The page editor offers all basic building blocks in addition to some specialised ones, the ability to execute custom actions on page/button events and advanced options for each component. Given the flexibility and advanced options of the page designer, it is definitely possible to create scalable applications which can also be used for mission-critical usage. Oracle APEX uses

their proprietary Oracle Database for the back end and custom logic has to be written in PL/SQL, a proprietary programming language also by Oracle. This makes an Oracle APEX application completely unportable and vendor lock-in is a definite issue.

**Simplifier** The Simplifier application editor offers an extensive list of UI components for composing an application with advanced options for each component, which is of course associated with a small learning curve to get familiar with the interface. A different theme can be uploaded manually. All things considered, this make an application very customisable and the platform itself moderately easy to use. Adding custom functionality can be done easily by adding so called “stories” using a graphical editor as well, making it very scalable since these “stories” can then be composed by creating custom events. Leaving aside the technical difficulties we had during the creation of our TODO List application, we still think it is suitable for creating mission-critical applications, given the commercial support would most likely be able to sort out these issues. A Simplifier application relies heavily on “stories”, which are stored in a proprietary format and there for not portable. The data, however, is highly portable since it is stored on external databases via connectors and many different databases are supported.

**Mendix** Mendix Studio offers many building-blocks and widgets to choose from and also offers a marketplace where developers can publish custom ones, making the platform very flexible. The colour scheme of an application can also be completely customised and the marketplace also offers the ability to use custom themes. Due to the fact that the data and building blocks are tightly coupled, it is very easy to create basic application logic. Application data can be downloaded from backups in SQLite format and external data sources can be installed from the marketplace, making the data very portable. The application itself it tightly coupled to the Mendix platform and not at all portable. Given the vast amount of building-blocks available (counting those available from the marketplace), Mendix is certainly suitable for building scalable and mission-critical applications.

## 4 Findings & Possible Future Work

During the research process for searching low-code tools we could evaluate, we found that there is a very small number of open-source low-code platforms. Our assumption is that people in open-source community are most often developers and therefore the need for low-code platforms is simply not there. It would be interesting to investigate further into this direction, also including other types of

platforms and comparing their presence in the open-source vs. the commercial space. In general, the commercial low-code platforms in our evaluation also seem to offer a more advanced feature set compared to open-source tools.

Another thing that stood out is the user-interface design of the low-code platforms. There is a clear trend towards a more streamlined user interface in the more mature products like Oracle APEX and Mendix. The user interface of, say Corteza, while simple, is not as thought-through and the Simplifier interface, while functional, looks like a generic UI prototype. Whether this makes a platform more usable than another lies of course in the eye of the beholder.

A common point of critique of low-code platforms is the lack of portability. According to our evaluation, this point is definitely valid. Of the platforms we evaluated, none of the applications are portable; at most, the stored data is easily portable by using a common database back end like MySQL. There is no standard definition of what a low-code platform looks like let alone any standard defining low-code building blocks which could be used to make porting an application from one platform to another possible. Given that many low-code platforms are commercial products, it is also doubtful if there is any incentive to make it easy for customers to switch platforms.

## 5 Conclusion

Taking the assessments in section 3 into account, low-code platforms are a valid alternative to model-driven development. Some caveats apply however: By their very nature, low-code platforms will not be as flexible, since with model-driven architecture, developers can take advantage of the full feature set of the software stack and customise all aspects of it, while low-code platforms are limited in the ways they can be extended. This is of course highly dependent on the platform, which means one cannot argue for or against low-code in all cases. The choice between low-code and model-driven architecture cannot be made without evaluating multiple different low-code platforms and assessing whether they can satisfy the requirements for a particular project, similar to choosing a programming language or framework that is most suitable for a particular task.

## References

- [1] Wikipedia contributors. *Model-driven engineering*. 2020. URL: [https://en.wikipedia.org/w/index.php?title=Model-driven\\_engineering&oldid=995747770](https://en.wikipedia.org/w/index.php?title=Model-driven_engineering&oldid=995747770) (visited on 12/27/2020).
- [2] *OpenAPI Initiative*. 2020. URL: <https://www.openapis.org/> (visited on 12/27/2020).
- [3] *Swagger*. 2020. URL: <https://swagger.io/> (visited on 12/27/2020).
- [4] Wikipedia contributors. *No-code development platform*. 2021. URL: [https://en.wikipedia.org/w/index.php?title=No-code\\_development\\_platform&oldid=996724223](https://en.wikipedia.org/w/index.php?title=No-code_development_platform&oldid=996724223) (visited on 01/01/2020).
- [5] Wikipedia contributors. *Visual programming language*. 2021. URL: [https://en.wikipedia.org/w/index.php?title=Visual\\_programming\\_language&oldid=991997087](https://en.wikipedia.org/w/index.php?title=Visual_programming_language&oldid=991997087) (visited on 01/01/2021).
- [6] Wikipedia contributors. *monday.com*. 2021. URL: <https://en.wikipedia.org/w/index.php?title=Monday.com&oldid=993059669> (visited on 01/01/2021).
- [7] Wikipedia contributors. *FileMaker*. 2021. URL: <https://en.wikipedia.org/w/index.php?title=FileMaker&oldid=991237143> (visited on 01/01/2021).
- [8] Wikipedia contributors. *Airtable*. 2021. URL: <https://en.wikipedia.org/w/index.php?title=Airtable&oldid=992831018> (visited on 01/01/2021).
- [9] *OSBP (Open Standard Business Platform)*. 2021. URL: <https://www.eclipse.org/osbp/index.html> (visited on 01/02/2020).
- [10] *OS.bee*. 2021. URL: <https://www.osbee.org/home/> (visited on 01/02/2020).
- [11] *Cortez Project*. 2021. URL: <https://cortezaproject.org/> (visited on 01/02/2020).
- [12] *Oracle APEX*. 2021. URL: <https://apex.oracle.com/> (visited on 01/02/2020).