A managed and sustainable low-code platform for enterprises:

factors to consider for successful implementation and adoption

Adis Jugo
Microsoft Regional Director
Microsoft MVP - Office Apps and Services
Microsoft MVP – Azure

Spencer Harbar
Microsoft MVP – Office Apps and Services
Microsoft Certified Solutions Master
Microsoft Certified Architect

WEBCON®
LOW-CODE, BUT BETTER.

ABOUT THE AUTHORS

Adis Jugo

Microsoft Regional Director,
Microsoft MVP - Office Apps and Services,
Microsoft MVP – Azure

Trusted advisor, leader, and software/solution architect with over 25 years of experience in solution delivery and information technologies. Specialized in Microsoft 365 and Microsoft Azure. Speaker at various technology conferences world-wide for the last 15 years. Awarded as Microsoft Most Valuable Professional in two categories: Office Apps and Services, and Office Development. Founder and spiritus mavens of the European Collaboration Summit, the largest European Microsoft 365/Azure/SharePoint event, with over 1600 attendees. Strongly believes in leading people by example and motivation.

Spencer Harbar

Microsoft MVP – Office Apps and Services,
Microsoft Certified Solutions Master,
Microsoft Certified Architect

Widely recognized as one of the world's foremost Office 365, Identity and SharePoint authorities. With over 25 years industry experience in the architecture, development, deployment and operational service of applications and hosting platforms, his broad base of fundamental skills routinely enables the world's largest organizations to succeed with Office 365 and SharePoint. Spencer is the only person to hold all of the Microsoft Certified Master, Microsoft Certified Solutions Master and Microsoft Certified Architect certifications for SharePoint, and is a 16 time recipient of the Microsoft Most Valuable Professional award. He was the content owner and author for one third of the SharePoint MCSM. He is a strong community advocate, supporting and speaking at the UK SharePoint User Group and helping others via his blog, forums and public events such as the SharePoint Conference, TechEd, SharePoint Evolutions and SharePoint Connections. Spencer is a co-organizer of the European Collaboration Summit and European Cloud Summit.

Table of contents

EXECUTIVE SUMMARY

Low-code development is not new. Many attempts have been made in the past to offer a way to create software applications without writing code, using a variety of terms and names, but what they have in common is to build business solutions with fewer resources, in less time, and with more consistent results.

Low-code is an idea that is independent of, but often loosely interchanged with, citizen development. The former refers to platforms and tools, whereas the latter refers to who is doing the development work (non-professionals, in this case). Professional developers use low-code tools and platforms quite often, since they offer increased productivity in their work. Citizen developers rarely will typically use the low-code platform to implement light-weight business applications on their own.

Key factors when choosing a low-code platform

One must be careful when choosing a low-code environment; it should not be done based on what is fashionable or solely on popularity. Rather, the key decision factors for should be based on platform's features, longevity, and existing expertise.

Composite or Integrated platforms

Composite platforms work by connecting to and aggregating a grouping of disparate resources. A dedicated forms tool might bind to multiple data stores, a workflow might call the APIs of a variety of cloud-hosted web services, and yet another component might be responsible for data presentation and metrics. The application connects its components.
Integrated platforms have you build forms, workflows, data schema, reporting, etc. as a holistic effort. While it might, and it very probably will, include connectivity to external data, for the most part the application contains its components.

Composite platforms promise that you can create new value out of existing, otherwise disconnected assets. This often involves searching for those assets, creating them individually when needed, and making numerous adjustments to each one of them. Deploying them as a set from a development to a test or production environment is usually very challenging.
Integrated platforms are meant to address versioning, packaging, and deployment with relative ease given that the application is built as a coherent whole. They offer all the low-code platform components under one umbrella and within one tool. Developers do not need to change environments or tools to model data, create user interfaces, and author processes.

Data

Low-code tools and platforms vary radically in how well they treat data. Some merely connect to external data and require the solution builder to configure all the connection details every time. Some have you model and manage all data within the application design environment. Some allow for application-managed metadata but external content data. A few platforms allow for all of the above.

Backup, Migration, and Upcycling

Applications need to be restorable after catastrophic incidents. They need to be moved from on-premises data centers to cloud environments (or vice versa). One may need to upgrade the platform on which the application is written.
It would be risky to invest in a low-code environment that did not have solid answers for addressing these issues. Software as a Service (SaaS) has an advantage when it comes to upcycling (the platform vendor is ostensibly doing that work), but backup and migration are a challenge no matter where an application sits.

It is not a matter of if, but rather when, an organization will migrate a solution from one location, or tenancy, or vendor, to another. Platforms that allow for this have distinct advantages.

The more a platform takes a composite approach, the more difficulty one will have in migrating a solution, backing it up, or upgrading its underlying technology. This is because any such undertaking has to be done component by component. The odds of this being doable as one effort, and the odds of such an effort not affecting other applications that might use the same components, are not encouraging.

Process Engine

Almost every platform has an execution engine of some kind; the term of choice may be script, workflow, automation, process, orchestration, recipe, etc. Many approaches are represented among available platforms. Any of them can automate activity. What not all of them can do include:

- Assigning, and reassigning when needed, permissions, responsibilities, and tasks.
- Reacting to a variety of events and activities and taking appropriate action.
- Allowing for processes that can move backward, forward, and sideways depending on circu mstances.
- Being able to effectively communicate the process to business stakeholders.
- Extensibility with custom code and/or connection to external services and automated logic.

Document Handling

While the idea of a paperless office remains an attractive one, business work with a lot of printed documents. Platforms/tools that can generate documents from data, and can generate data from reading/scanning documents, have a distinct advantage over alternatives that lack these capabilities.

Localization

Many organizations operate in more than one language, currency, date format, and/or form of measurement. If that's the case, one can either create several independent location-specific solutions or create a single solution that can adapt.

The latter approach has distinct advantages, but only if the platform itself provides the localization services. If they have to be coded/configured/constructed as part of the business solution, the result will require more work than expected when building and maintaining it.

Responsiveness and Mobile-Friendliness

Solutions must work in a variety of screen sizes, form factors, and operating systems. The ideal is to only have to build an application once on a platform that handles adapting the user experience. The worst-case scenario involves separate efforts, in part or as a whole, to accommodate browsers, phones, tablets, etc.

User Interface

The UI is the part that the users see, and if its usability and optics are subpair, the platform adoption will suffer, regardless how good it otherwise might be. Many approaches exist, and are often a matter of preference. What is universal, however, is whether the user interface is understandable, productive, and at least somewhat pleasant to use.

A toolset might take a WYSIWYG (what you see is what you get) approach, allowing for pixel-perfect placement of form elements, or opt for a layout-based designing, which will approximately define positions of the elements within the screen estate. While the WYSIWYG method was very popular in 2000s, rapidly emerging mobile devices and small screen factors have caused majority of platforms to revert to layout-based designs. Those platforms which still force WYSIWYG will often force developers to create different layouts for computer screens and mobile screens, which leads to the additional efforts in application development and maintenance.

Application Lifecycle Management (ALM)

Low-code solutions should still allow for creating applications in one environment, testing them in another, and deploying them into yet another environment for production use. Not being able to do this means editing applications while they're in use – a recipe for risk and instability.

Integrated solutions have benefits when it comes to ALM; if an application's components were built together as a set, packaging them as a set, and deploying them as a set is far easier.

Integrated solutions might still have external dependencies, though, and these will require attention. Composite solutions, by contrast, are made up entirely of external dependencies.

An environment that knows how to manage connections to external dependencies, and can change them when an application is deployed from one stage to another, has a distinct advantage over one that requires manual post-deployment "clean up".

Managing Change

The ability to package and deploy apps to different target system is a crucial feature that each low-code platform should implement. In addition to that, the low-code platform should support updates and rollbacks, since every application will likely be updated over time. A good low-code platform will not only increase developers' productivity when creating a new application, but will also enable equally productive changes and updates, and will assist with deployment of those changes to end users. Robustness of the low-code platform to handle changes, and assistance with effortless deployment of those changes, might easily be one of the key factors for the low-code platform choice.

Clarity

While all low-code platforms promise to speed up solution development process, the major differentiator is what happens after a solution has been created. The questions such as how easily and how well a solution can be documented and explained to stakeholders, and how much work is needed to train users, will prove to be as or even more important than the development process itself.

- Besides that, the low-code platform needs to support both developers and IT administra tors in gathering metrics and performing monitoring. Information such us which solutions are used the most and by whom, or which solutions are underperforming or causing other issu es within the system, are crucial for overall operations and eventual problems trouble shooting.

Cloud, On-premises, or both?

While majority of low-code platforms today are claiming to be cloud-first, it should not be neglected that there is a huge amount of data and processes which, for forever reason, cannot be moved to the cloud. Ideally, a low-code platform would support hosting both on-premises and in public cloud, as a fully isolated platform hosted in a separate tenant, or is the solution available as Software-as-a-Service shared public cloud platform, or – ideally – both.

Furthermore, it is important to determine if the platform is capable of accessing the resources from both on-premises and cloud environments. In the case of cloud-only platforms, it needs to be assessed how does the solution access on-premises assets, and what does it mean identity, permissions and security-wise?

If the platform is available both in the cloud (either as a cloud-hosted solution, or as a Software-as-a-Service offering), and on-premises, it is also important to determine the feature parity and platform compatibility: is it possible to move or migrate the solutions, data, and processes between on-premises servers and one or more cloud environments?

Conclusion

Recommending a single approach, let alone a single product, would not be responsible or widely successful; different organizations have very different needs, and one size does not fit all. In this whitepaper, we will discuss different aspects, strengths, and weaknesses of different approaches that modern low-code platforms are taking.

What can be recommended is choosing a general path to follow after all the aspects have been considered and analyzed. That path seeks to minimize risk, maximize security, and optimize for constantly changing requirements – and it seeks to have the platform do as much of the work (packaging, deployment, responsiveness, localization, etc.) as possible.

A managed and sustainable low-code platform for enterprises:

factors to consider for successful implementation and adoption

INTRODUCTION

Today there is a plethora of services and products available to assist with the creation of "low-code" business solutions. Significant increases in capability and breadth are being offered at a lower cost all the time, and yet enterprises still face challenges with successfully implementing and adopting low-code platforms.

This whitepaper discusses the fundamental aspects which should be considered when choosing a low-code platform, and how to successfully manage the lifecycle of its applications.

WHAT IS LOW-CODE DEVELOPMENT, AND ARE ENTERPRISES CHOOSING IT?

From the earliest days of computing, the industry has been attempting to make the development of software solutions less complex and accessible to a broader set of users. We have come a long way from entering code in binary format, assemblers, and high-level programming languages to what is today commonly referred to as "low-code development". As the name suggests, low-code development implies developing the majority of business logic through the visual design of forms and processes and using abstract expressions for calculations. The "code" itself, meaning instructions in a traditional programming language, should only be used when necessary, when the low-code platform reaches its boundaries. What and where those boundaries lie are defining factors for success with, and adoption of, any low-code platform.

Since low-code platforms have been typically promoted for use by a group of people that we call "citizen developers" – a broad term that denotes non-professional developers who are developing business applications – "low-code" and "citizen development" are often used interchangeably. That is not entirely accurate; nowadays, low-code platforms are used mainly by professional developers, who are aiming to focus more on developing business logic and less on traditional software development plumbing. That plumbing is generally left to the platform to take care of, especially when leveraging cloud-based services.

This is why it is paramount that the low-code platform is capable of taking care of professional development/deployment tasks, regardless of whether it is used by professional developers, citizen developers, or both. These tasks include translating visual and configuration elements into set of instructions understandable by computers, but also the deployment, governance, maintainability, and sustainability of the applications created with the low-code platform. History teaches us that the rapid creation of business applications is not enough; **it is their maintainability and longevity that matters over the long haul.**

A brief historical perspective

Throughout the history of computing, companies have been looking at increasing the efficiency and quality of business applications developed to support their business processes. In the early days of computing this was barely possible, since development of any kind of software would involve complex programming languages to instruct computers what to do.

This all started to change in the 1980s. The key factor for this change was bringing data, forms, and processes together; with those three ingredients, both professional developers and, for the first time, citizen developers had development platforms with which they could, with little or no coding, create impressive business applications for that time. It should be mentioned here that application frameworks such as dBase and FoxPro, and also spreadsheet programs such as Lotus 1-2-3 and Microsoft Excel, also contributed.

Rising star Microsoft, which was quickly gaining dominance of the personal computer with their new operating system Windows, soon realized the potential those platforms had. By the mid-1990s, they had successfully combined Word, Excel, and Access into one unified package called Microsoft Office. When Excel and Access became widely available, those Microsoft Office components dominated the low-code market within a few years. The number of crucial business processes, which were implemented with Excel worksheets or Access databases, was reaching until-then unthinkable numbers. This, in some ways, changed the software industry; for the first time, a significant portion of the software used in corporations was not implemented with specialized computer languages, but rather with highly abstracted low-code platforms.

What we learned from history: issues with past low-code platforms

However, there were numerous problems with that approach. Hundreds and thousands of Excel sheets and Access databases were floating through companies; multiple versions of those files were typically present, each with slightly different functionality. There was redundancy of data, and above all, they were very difficult to maintain and support. The partial solution for those and other issues was to move those Excel sheets and Access databases to some kind of server infrastructure, where they could be centrally accessed by multiple users and managed by IT professionals.

Enterprise servers and the cloud era

Such an approach was pioneered by Lotus with Notes (which was acquired and evolved by IBM into the Domino platform); soon after, Microsoft would deliver a range of server-based platforms of their own, culminating with SharePoint Server.

SharePoint was (and remains) a web-based collaboration and document sharing platform, intranet, and – with the additions of SharePoint workflows and InfoPath – a platform for composite business application development, which encompassed documents, data (SharePoint lists), user interfaces (InfoPath, SharePoint Designer), and processes (SharePoint workflows). When this was not enough to meet business requirements, SharePoint offered extensibility through traditional development using Microsoft's .NET Framework. The success of SharePoint was enormous, and to this day it is considered to be one of the most successful Microsoft products. SharePoint very soon became the low-code platform of choice for the vast majority of developers of information worker solutions within the Microsoft enterprise ecosystem.

With the birth of cloud computing came a completely new approach to software deployment and making it available to end users. One of the first companies to realize this was Salesforce, which soon became a leader in the Software as a Service (SaaS) field with their CRM platform. One element of Salesforce that appealed to end users was the ability to easily customize it; users could create data entities and relationships between them, design a UI, and define processes. The first low-code platform in the cloud was born.

Microsoft very quickly became a major cloud player with its own cloud platforms that today include Azure, Microsoft 365, Dynamics 365, and more. However, Microsoft has taken a different approach from Salesforce with respect to low-code platforms in the cloud; there are multiple approaches to data storage (the Common Data Service, SharePoint Online Lists, and SQL Server databases being just some of the options) as well as multiple workflow options (Power Automate and Azure Logic Apps are new players in the field, and the lifespan of Workflow Manager has also been extended). Perhaps the greatest confusion is in respect to UI tools – after multiple unsuccessful attempts to create a new UI designer platform with Access Services and FOSL (Forms On SharePoint Lists), Microsoft extended the support-window for InfoPath and started working on Power Apps, a next generation tool for creating user interfaces for Microsoft's cloud platform. Due to the heterogenous origins of Power Apps, it is necessary to differentiate between model-driven Power Apps which are more geared towards Dynamics 365, and canvas-driven Power Apps, which are targeted to everyone else and are being widely accepted by Microsoft 365 developers.

As we can see, Microsoft have offered a wide range of interesting technology elements, but as opposed to other cloud-based, low-code development platforms, Microsoft did not offer "one recommended way." In order to make right choices, companies would need to engage architects to identify their needs and determine the most viable path forward.

Third party low-code platforms in the Microsoft ecosystem

The history of third-party low-code platforms within the Microsoft ecosystem started in the late 1980s with products such as dBase and Fox Pro. In the 2000s, various Microsoft partners had identified weak or missing spots and started to offer their own solutions for workflows and forms on top of the SharePoint platform. Those solutions quickly gained popularity because they typically offered an integrated platform for forms and workflow development, which further streamlined and expedited the development of business applications.

Those partner platforms have evolved significantly with the move to cloud-based services. Some of them are implemented as server-based solutions which can be hosted either on-premises or in the cloud, and some of them are made from scratch as cloud-native Software as a Service solutions. We have also seen that many of them are gradually decreasing their dependency on SharePoint, offering standalone low-code platforms which can work with SharePoint data as well as other data sources. The vast majority of them are still integrated platforms which offer data modeling and encapsulation, creation of user interfaces, and development of workflows and business processes within a single platform.

FUNCTIONAL REQUIREMENTS FOR THE CHOICE OF A LOW-CODE PLATFORM: DATA, WORKFLOWS, AND FORMS

In the past three decades of low-code platforms, we have learned many lessons about what works, and especially what does not. Hosting both the platform and customer data in the cloud helps the vendors with quick deployment of new features and quick responses to issues. At the same time, however, any mistakes made in that operational model are immediately propagated and visible to all customers.

Most of the low-code platforms present on the market today manage to avoid the most obvious mistakes of the past. However, not all platforms put the same level of importance on the different aspects of what defines a good low-code platform. Some are better with processes and workflows, some are better with forms, and some of them have managed to create good management, governance, and operations models.

Understanding requirements, ranking them, and choosing a low-code platform that fits can save companies a lot of time and lot of money in the process.

For **developers**, making the right choices means that they will not reach dead ends or functional blockers in their projects, and avoiding obstacles which are either impossible to overcome or for which a solution is only possible through heavy additional custom code.

If these aspects are not considered carefully, it leads to frustration, and possibly abandoning the project or even the platform. It can also mean the implementation of an unsustainable collection of low-code elements and bolt-on dependencies.

> Change and maintenance, important part of each project, also play a major role in developers' acceptance. If developers are faced with obstacles in maintaining and evolving their own – or someone else's – applications, that will also not positively contribute to the platform acceptance.

For **companies and stakeholders**, making the right choice means that the low-code development platform of choice is, to a degree, future-proof, that financial and time investments made into the platform will not be obsolete in the short or medium period of time. Furthermore, for companies it is important to have a clear and understandable licensing model and cost structure with which it is possible to plan, and without hidden or unexpected costs.

> Furthermore, for companies, interplay between users and developers (regardless of whether they are professional or citizen developers) should play a major role: the platform which offers better understanding, provides easier feedback, and perhaps even collaboration on certain segments of application development (such as UI and Data model) will prove more valuable, more efficient, and more accepted.

For **IT departments**, making the right choices means that they will be able to support the platform, support the developers and users, provide maintenance, and ensure confidentiality, availability, integrity, and business continuity.

Integrated versus composite platforms

During the mid-2000s, Microsoft heavily promoted the idea of a "composite business applications" to describe low-code development based upon SharePoint Server, a "composite platform." That meant that developers could use various SharePoint components and services (e.g., Business Connectivity Services), together with additional tools such as SharePoint Designer and InfoPath, to get the job done. This also meant that not all of the choices and all of the components were straightforward; developers were often faced with choices among different possibilities which were doing the same thing with nuanced pros and cons.

Opposite to the idea of a composite platform, **integrated platforms** offer all the low-code platform components under one umbrella and within one tool. Developers do not need to change environments or tools to model data, create user interfaces, and author processes. Those integrated platforms create an abstraction layer between developers and the underlying technologies, about which the developers do not need to know all the mechanisms and specifics.

During the mid-2000s, Microsoft heavily promoted the idea of a "composite business applications" to describe low-code development based upon SharePoint Server, a "**composite platform**." That meant that developers could use various SharePoint components and services (e.g., Business Connectivity Services), together with additional tools such as SharePoint Designer and InfoPath, to get the job done. This also meant that not all of the choices and all of the components were straightforward; developers were often faced with choices among different possibilities which were doing the same thing with nuanced pros and cons.

Opposite to the idea of a composite platform, **integrated platforms** offer all the low-code platform components under one umbrella and within one tool. Developers do not need to change environments or tools to model data, create user interfaces, and author processes. Those integrated platforms create an abstraction layer between developers and the underlying technologies, about which the developers do not need to know all the mechanisms and specifics.

Data

Data is one of the three main components for every citizen development platform. Generally, when a new project is started, one of the first decisions that is made is if existing data is going to be reused (for example, in modernization projects) or if a new data model and storage is going to be created (for example, in greenfield projects).

In enterprise scenarios, where a decision on low-code development platform has been made, it is the role of IT to define and approve data sources. It is strongly recommended that corporate IT analyses the needs, reviews existing data sources that could be reused, and makes recommendations to developers which will be easy enough to follow yet fulfill all security and manageability criteria. In the case that a fully integrated platform has been (or will be) chosen as a low-code development platform of choice, where the underlying data store has been hidden from the developers, corporate IT still should know what that underlying source is and what its security and manageability characteristics are.

It is important to note that file-based databases are not recommended for low-code development in enterprise scenarios. File-based databases store all the data in one file, which is then typically dedicated to one instance of the application. Typical file-based databases are Microsoft Access Database and Excel Workbooks. File-based databases have difficulties with multi-user and data concurrency scenarios, and are prone to data redundancy, which was one of the main causes of headaches with low-code development platforms in the past.

When choosing a data source, the following parameters and criteria should be considered.

Data modelling

Each low-code platform has some sort of data modeler, which is used either to reference data, to import existing data, or to create new data entities and relationships between them. Different low-code platforms take very different approaches with a data modeler:

- Some low-code platforms expect developers to know the specifics and behavior patterns of the underlying data source.
- Some low-code platforms create a strong abstraction layer and handle all the specifics of the underlying data source on its own, hidden from the developers.
- Some low-code platforms take a hybrid approach, and let developers create independent data objects, which can interact with each other, but they need to be manually associated with the underlying data sources.

The way low-code platforms handle underlying data, and how the data modeler is implemented, are important decision criteria for the choice of a low-code platform. If the way the platform handles underlying data is not fully in accord with the data sources already present in the company, it may pose a significant obstacle to developers delivering the expected quality and efficiency of their applications.

Core Data

Core or reference data, sometimes also called "Master data", is the consistent and uniform set of identifiers and extended attributes that describe the core entities of the enterprise; examples include customers, prospects, citizens, suppliers, sites, hierarchies, and the chart of accounts.

The crucial factor here is whether such data can be consumed by the designated low-code development platform.

Another important input parameter is whether the low-code development platform of choice has its own core data management system. This usually happens for three reasons:

1. Central core data is missing entities the applications need. In this case, platform-level core data will suffice.
2. Central core data is too difficult to use; it may be in a data platform that requires more experti se than most users of a low-code platform.
3. Central core data isn't performant if used from a single location; the platform might be acting in this case as a distributed copy of central core data.

A typical example of such systems would be Managed Metadata in Microsoft 365's SharePoint Online. If that is the case, then the limits of the system should be well-known and considered. In the Microsoft 365 example, SharePoint Managed Metadata can store up to 200,000 terms (data). The key pieces of information on core data, which need to be taken into account when choosing a low-code development platform, are:

- Has the company already implemented a core data management system?
- Can the designated low-code development platform connect to the existing system?
- Does the designated low-code development platform have its own core data system?
- Can the existing core data system and the platform's core data system be kept in sync, and what effort would it take?

Transactions and concurrency

When determining the approach to data storage, which will be used for transactional data, there are some questions which must be answered:

- Is transactional data used by the applications which will be developed typically going to be used by more than one user concurrently?
- Will there be a need to perform data locking and data isolation?
- Will there be a need to perform transactional operations, such as rollbacks?

Of course, it is difficult to answer all these questions unanimously for all the applications that are going to be created with the low-code platform. Still, it is important to be aware if these requirements are going to be asked for, and to what degree. If a certain amount of the applications developed with a low-code platform will need to take into account things such as concurrency and transactional operations, it will be necessary to choose a low-code platform which supports a data source which can support these requirements.

Relational data

Relational data is in one way or another present in all modern data sources. However, there are different types of relationship behaviors. If a "parent" element has been deleted, or changed in a way that the "child elements" should be changed too, what happens with the child elements?

When choosing a low-code platform, it should be determined if the underlying database

- Supports different flavors of one-to-many and many-to-many relationships
- Supports Entity and Referential Integrity preservation after transactions

Some database engines typically used in modern low-code platforms, support these requirements only to a certain degree, or only on a very basic level (such as SharePoint, for example). If a certain amount of applications which will be created with the low-code platform will require a stronger enforcement of relational data constraints, this should also be taken in account by the choice of platform.

Logging and auditing

When choosing a low-code platform, it is important to determine if the applications created with that platform will need to support logging and auditing for the data and processes used in the application. Very typical basic scenarios include workflow history lists or tables used for auditing workflows, as well as versioning for data and files.

The questions which should be answered here are:

- Is full transaction logging needed to enable full scale auditing over the transactions perfor med on the data? This is needed, for example, in auditing scenarios to be able to answer qu estions about who has changed data, and when.
- Is data history (versioning) needed, and to what degree? Can previous versions be restored, and how does that affect relational data? For example, if we restore one version of data in a parent table, how does that affect the data in the related child tables?
- Is there a process and workflow logging system easier to implement and use than stan dard transaction logging? A good example here would be SharePoint Workflow History lists, despite their historical constraints and limits.

Avoiding Data Redundancy

Data Redundancy was one of the main problems with low-code platforms using file-based data in the past, examples being Microsoft Access and Microsoft Excel. Those applications would typically be shared among users in a way that a copy of the file would be passed between the employees. That would usually lead to creation of redundant copies of data, where it became difficult to determine which is the "master copy" of either the application, the data, or both.

By hosting data and the application centrally, either on a server or in the cloud, that problem has been reduced for the most part. However, there are still scenarios where data redundancy can easily occur and create the same problems as Access databases were creating in the past.

Microsoft Power Apps can use file-based data sources, such as Excel tables, as data sources. One of the first official Power Apps demos by Microsoft demonstrated Power Apps using an Excel table stored in a consumer Dropbox. Even though that demo was removed relatively soon after its initial appearance, it was a good example of what can go wrong when a low-code platform is used in the wrong way.

Besides such obvious cases, certain scenarios where Power Apps are used in combination with SharePoint Lists can indirectly lead towards data redundancy. Since Power Apps do not support a staging-testing-production release cycle, in a way that the data references cannot be changed and rebuilt easily, one of the workarounds is to package and copy the complete data sources, and to deploy them together with the app into a new environment. If not used carefully, this approach can also lead to a data redundancy problem.

Backup

Backup has been a fundamental aspect of data management since the beginning. The goal is to be able to restore the data in a case of data corruption (or data loss) that would prevent normal operations.

When the data is created and used by a low-code platform application, data restoration has to be considered in the light of restoring the functionality of the whole application.

It is typically easier to restore application functionality if an integrated low-code platform is used, since those usually take on the responsibility for managing internal data as part of the application itself; the application includes the data storage and access. By comparison, with composite platforms this is usually a more complex task, since there are typically connectors, or similar abstraction layers, present between the application and the data sources – and all data is external to the application. Restoring and configuring those connectors, their permissions, and their context, can often be a very challenging task.

The following should be considered when choosing a low-code platform:

- Does the low-code platform itself have backup/restore capabilities, which would back up application, data, and eventually application instances?
- Are the data sources used by a low-code platform already part of a backup plan thro ugh that data source? This is typically the case if a database server, such as Microsoft SQL Server, is used as a data source.
- If the data is backed up through the database server backup plan, what is the procedure for restoring the application's functionality after the data is restored? Are there data connectors, or any similar intermediary abstraction layer settings, which need to be manually restored/ configured or refreshed?
- What is the downtime during and after data restoration in all the above-mentioned scenarios?

Upcycling and migration

In many scenarios, a need will emerge to "modernize" existing applications and data sources; this has certainly been true of Microsoft Excel, Microsoft Access, and previous attempts at composite SharePoint applications.

It becomes even more challenging if the data itself goes through an upgrade or migration process. For external data, this is a very real possibility; there could be updates to database engines, movements of SharePoint lists, changes to APIs, etc.

This does not include only the data migration from one data source to another, although that would be a first necessary step if modernization projects are planned.

Although a fully automated migration from a legacy platform to a new low-code platform will never be possible for numerous reasons, different low-code platforms offer various helpers in order to make such projects less painful.

If it is clear beforehand that modernization of old applications is one of the goals, it should be proved if, and how much, the designated low-code platform can help with that process and what additional tooling or processes will be needed.

Process Engine (aka Workflow)

Low-code development platforms usually come with a workflow engine, or some other type of process engine. This is very often the core component of the whole platform. Ideally, it should be possible to create processes, regardless if they are user triggered or background processes, in a structured way, with as little coding as possible.

Still, there are significant differences in the features that workflow or other type of process engine platforms can offer

Easy process assignment and permissions management

Process assignments, process delegations, and permissions management have been core requirements for workflow engines from the early days. Even the simplest approval processes will sometimes require a change of ownership and delegation, and potentially change of permissions on the underlying data.

Still, various workflow engines have different approaches to this requirement. When choosing a low-code platform it is important to check how the platform deals with the following tasks:

- Process reassignment (when a process is completely reassigned to another person or group)
- Instance delegation (when a process is delegated to another person or group, but the initial assignee can still perform the action)
- Group assignments (when any member of the group can perform the action)
- Permission changes (when process reassignment or delegation causes permissions changes on the underlying data)

Triggers and background processes

In software development in general, we are differentiating between user-initiated actions, and non-user-initiated actions, also known as daemons or background processes. Daemons are usually triggered by another action, or at a scheduled time, and can include various calculations, reporting, notification or cleanup actions.

When choosing a low-code platform, it is important to check if that platform supports those background processes, and to what degree.

- Are non-user-initiated actions (daemons), such as background processes, supported and to what degree?
- Are time scheduled triggers included?
- Which kind of event and action-based triggers are supported?
- Are triggers from relevant external systems supported, and if not, can that be compensated for by redirecting signals (polling, service buses) from those systems to one of the supported systems?

State Machine Workflow support

There are multiple types of workflows, but the two typically most used are Sequential workflows and State Machine Workflows. While sequential workflows are typically executed from start to finish, and include branching, looping and other conditional controls, state-machine workflows are event driven. That means that they rely on external events (for example, user input, received email, changes to documents and forms) to drive the workflow to completion.

Not all workflow engines in modern low-code platforms support state machine workflows. And while most of them can emulate state machine behaviors, using loops and exits, that kind of workflow is prone to errors, less performant, and much more difficult to maintain.

Before deciding on a low-code platform, an analysis of whether state machine workflows are required for the projects which will be implemented on top of that platform.

> In processes of any real complexity, where the flow of work might move backwards or sideways, support for the state machine workflows will be one of crucial requirements, and a low-code platform with a native support for state machines should be preferred.

Extending workflows

Even with the extensive set of actions that most modern workflow engines have, developers will sometimes find themselves in a situation where it is necessary to create their own custom actions, often as a discrete interface to external systems. Although, with a good workflow engine, the need for this should be very small, it can never be completely excluded.

Custom Actions

Custom actions perform operations on data which are not possible with the built-in workflow actions. This usually happens when the requirements are too specific and cannot be implemented with the built-in actions, or when the implementation with built-in actions would be so complicated that the workflow would become unmaintainable.

Various workflow engines have implemented development of custom actions in different ways, but, usually, well-known programming languages and development techniques are used for their implementation.

There should be awareness of the implementation options for custom actions before making a decision about the low-code platform, because this factor can have a significant impact on the platform acceptance and platform use.

External web service calls

Many software applications support APIs and web services, which allow them to communicate and share data with other systems. Therefore, a modern workflow platform should include calls to external web services and should support various authentication methods for those services. Preferably they will do so using industry standard protocols.

If one of the requirements for the designated low-code platform is communicating with external systems, support for two-way web service calls should be regarded as one of the key decision factors. Ideally it should also be possible that external web services can initiate or move workflows within the low-code platform.

Connecting to other data sources

If a low-code platform's workflow engine needs to interact not only with its native data source but also with external systems, it would be advantageous if it could easily connect to those systems through connectors rather than custom code. In this case it would be preferable if connectors for the most common external systems to already be available.

However, the experience shows that it will often be necessary to create custom connectors for the proprietary systems or data sources which are not covered by the standard connectors. This is different than calling external APIs (explained in the previous chapter), because custom built connectors offer standardization in accessing those proprietary systems, in a way that once created connector can be reused many times, whereas calling APIs directly has to be done each time from scratch. Having a standardized and well documented methodology for creating custom connectors is a very strong argument pro low-code platform systems that need to connect to proprietary systems.

Document Generation

One of the important outcomes of many business processes are different kind of documents, such as Microsoft Office or PDF documents. This is most common for reports, summaries, and generation of official documents such as purchase orders and contracts. Therefore, if applications built on top of the low-code platform can generate documents as a part of a business process, this will be an important criterion for the choice of the platform.

It should be checked if the low-code platform supports following document generation features and scenarios:

- Templates, which will be filled in with application data (Templates are typically provided as Microsoft Office documents) and possibly refreshed on demand
- Repeating tables and sections. Most of the documents will be based on relational data, so it will be important to present that data in a parent-child format within the documentation
- Calculations. Scenarios where a summary or other aggregation operation should be perfor med on data are fairly common and should be included in the document generation features.
- PDF conversion. Document templates and documents are commonly generated from, and as Microsoft Office documents. Often however PDF conversion is an important part of the process, as PDF is widely seen as an appropriate "read-only" and accessible format. Therefore, the various protection and locking options for PDF documents should be possible.

Multilingual support and localization

If a company operates in different countries and languages, multilingual support and localization are important features of the low-code platform. While it is of more importance within the user interface (which will be discussed later in this document), multilingual support and localization are also an important part of the authoring and administration interfaces.

- All workflow notifications and status reports should be presented in a language preferred by the user
- All the standard workflow task labels should be in the language preferred by the user
- Process overview and workflow stages should be presented in the language preferred by the user

Mobile Friendly

With the increasing number of tasks being performed on mobile devices, it is becoming important that the process visualization and tasks can be displayed in a mobile-friendly way, which can be used on smartphones and tablets.

At minimum, the designated low-code platform should include:

- Mobile friendly visual representations of the process and its current stage
- Mobile representation of tasks
- Mobile friendly representations of the most common workflow actions (approvals, delega tions etc.)

While the definition of mobile-friendly is certainly a matter of interpretation, it should not be achieved by trying to fit all the process elements and process map on a smaller screen. Process-focused mobile views should focus on providing user-friendly interaction with processes, rather than focusing on fitting all the information present on desktop into mobile views.

At the same time, though, the mobile experience should not require twice the amount of development and maintenance. With web applications, we would opt for what is known as responsive design, which adapts to various screen sizes. Any low-code platform worth trusting will take a similar approach.

User Interface designer (aka Forms)

Form designer

A form designer is typically one of the major differentiating factors with low-code platforms. These differences are usually caused by other elements of the platform. A major difference revolves around how the data model is created and represented: are forms created from the data model (model-driven), or does data have to be specifically included on the form by the developer (canvas-driven)? Furthermore, if forms should be displayed only on computer screens, many form designers will offer a WYSIWIG, pixel-perfect-design experience, but if the forms should be used on both desktop and mobile devices, most designers will fall back to canvases and zones, where developers can position the form controls.

If creating forms and user interfaces is of importance for the applications which will be created on the low-code platform (and it very seldomly is not) it should be determined what are the decisive criteria for the forms designer.

Some of that requirements that might influence the choice of platform are:

- Availability of the data model within the form-designer. The data from the model should al ways be available, so that it can be easily included in the UI. Ideally, the form designer sho uld be able to create an initial form solely based on the model behind that form.
- Ideally a WYSIWYG forms designer which helps developers to create forms more efficiently
- If WYSIWIG is not viable due to other factors, such as responsive form design, or heavy depen dency on the data model, a zone-based designer with canvases and zones should be available.
- Forms designer should offer standard grouping controls, such as tabs and control groups.
- It should be possible to create form styles and save them into a library for future re-use, possibly across applications. Ideally one style based upon corporate identity should be cre ated. That style would include design for all form elements, controls and tables, and it would be reused in all forms and all applications across the platform.

Differentiation between display, new and edit forms

In modern low-code application platforms, developers will be creating a number of data-focused and process-focused forms. Whereas data-focused forms will enable user to browse, view and edit data, process-focused forms are there to enable users to complete tasks related to the process.

As mentioned above, data-focused forms are intended to provide user-friendly data manipulation.

Each data-focused form must usually support multiple different states

- Browse/filter/search (navigating through existing data)
- Display (showing existing data)
- New (entering new data)
- Edit (editing existing data).

Depending on the low-code platform sometimes these are completely different forms (which will then eventually have to be kept in sync), sometimes it is one form with multiple different states (which then makes form development and control states more difficult to maintain), or a hybrid approach between the two (one display form, and a combined new/edit form).
In process-focused applications, forms may be tightly coupled to a task to be performed rather than a data state. In such a scenario, there could be a different form (or a different rendering of a single form) for each step in a process. It could also be a form that automatically adapts itself to whatever the current step in the process is.

When choosing a low-code platform, it is important to be in the know about how the platform implements data-focused forms and process-focused forms, and their form states. The implementation of this feature might pose a challenge to developers delivering, and later evolving, user friendly and easily adopted applications.

Parent-child relations through repeating tables and repeating sections

For each data model where relational data is used, representing the parent-child relationships is of paramount importance both for the data representation, and for users understanding the data. This is usually achieved with parent data being displayed in the form header, and the child data being displayed in the form of repeating tables, or repeating sections.

The low-code platform of choice should fulfill the following requirements:

- Easily placing the repeating tables and repeating sections on the form, and choosing one of the existing relations from the data model to serve as the relationship for the repeating table or repeating section
- It should be possible to place multiple repeating tables or repeating sections on the form
- Ideally, the low-code platform should be able to generate parent-child forms from the data model, which can then be edited and modified by the developer.

Intuitive expression language

This section applies to the expressions used within the user interface, within forms, and thus should be considered as front-end expressions and calculous. They are used to support user data entry and manipulation, form flow, and in-form calculations. They are not used for background processes and calculations as that task is performed by the workflows, which have been discussed in the previous section.

The expression language for the UI (forms) is one of the crucial features of each low-code platform, and helps the platform to be accepted by the developers or rejected as too limited for serious use. One example of a well-designed expression language, which has propelled the platform, is Microsoft Excel: millions of professional and citizen developers are using it daily to achieve their desired results.

Some of the requirements for a good expression language are:

- Easy to understand syntax. From the experience, platforms which offer syntax which is intuiti ve, easy to learn, and easy to understand, are much more likely to be accepted.
- Expression authoring should be possible using formula builders and/or syntax checkers. It's important to offer both; experienced developers can type quickly, but novices need cueing and prompting.
- Syntax error detection and error marking and correction within an expression editor, so that simple syntax errors are detected before saving and testing.
- Enough expressions, operations and functions to reduce the need for traditional coding to the minimum, and the ability for high- or low-level custom code to be used alongside the code-less expressions (as their part rather than replacing them),
- Grouping expressions into sequences or functions.
- Easy access to the data and environment variables. It should be easy for developers to inc lude the application data, such as field names and variables, into the expressions. This also applies for the environmental data such as user data, user permissions, and context data (ap plication specific environmental data).
- Expression testing on real data within the expression editor. The process of writing or editing expressions, saving and deploying, and then post-deployment testing is too cumbersome, and it decreases developers' efficiency. An expression editor within the form designer should enable in-editor, pre-deployment testing on real data; doing so significantly improves deve lopers' efficiency.
- Example expressions and good documentation.
- Automatic adoption of expressions so they are correctly interpreted regardless of the brow- -ser or device form factor in use.
- Application staging readiness – very often, expressions include environmental information, such as paths, user information and other environment variables. It is important to know if the expression language supports environment variables, which will make those expres sion automatically update by deployment to other environment, or is post-deployment ma nual work required to ensure application functionality.
- Can expressions blocks (functions, procedures, commonly used code blocks) be stored in a kind of managed expression repository, and reused across the applications? The ability to track where the expression is being used is crucial to eliminate costly errors in future.

Calculated fields

It is often the case that ad-hoc calculations must be displayed within the form, in special calculated fields. This is one of the key requirements for each forms engine, regardless of the low-code platform.

Forms should be able to display calculated fields regardless of if they will be persisted in the data source or not. The expression language, or custom code in the rare cases when this is not enough, should be used for these calculations.

Form flow and form control states

The forms are rarely static and immutable within one application. They usually dynamically change based on the data displayed in the form, environment variables, or current user and their permissions.

The forms engine within the low-code UI platform should be able to support following scenarios:

- Dynamically displaying or hiding form controls based on form data, environment, or user information.
- Dynamically making form controls editable or read-only based on form data, environment, or user information.
- Dynamically making form controls required, based on form data, environment, or user infor--mation.
- Dynamically displaying or hiding control groups, such as tabs, based on form data, environ ment, or user information.
- Creation of multi-page "walk-through-wizards".
- Ideally, it should be possible to create a set of operations which would be executed imme diately after the form has been loaded ("on form load"), or before and after the form has been submitted ("pre-save" and "post-save"). This is useful for different types of initialization, validation and cleanup scenarios.

Validation

Validation is an important part of every data entry or manipulation within the form. Developers will typically want to prevent submission of incomplete or corrupt data, and they will try to intercept these situations within the form, through a process called validation.

Forms engines within the low-code platform should support the following validation requirements:

- Field input field validation, which will ensure that all the form field inputs are in the correct format and range (for example, that selected month must have a value between 1 and 12).
- Form validation, which will ensure that the correlation between inputs in different fields is meaningful and not corrupt (for example if the selected month is "March", the day must be within the range of 1 and 31)

- Preventing form submission. If validation has failed on a field, or on a complete form, it sho uld not be possible to submit the form before the input data has been corrected.
- Using expression language for the form validation. Validation can sometimes be way more complex than a simple range check, and developers should have the possibility to use com plex expressions to perform validation.
- Using code behind for validation. In rare situations when even the expression language is not enough for more complex validation scenarios, it should be possible to use code behind for validation purposes.

Seamless integration with the process engine

One of the key factors for each low-code platform regardless of whether it is integrated or composite, is that the UI part (forms) and process part (workflows) can interact with each other.

Each low-code platform should be able to support the following scenarios:

- Triggering workflows from within the form.
- Workflows should be able to pass back data to the forms from which they have been triggered.
- Ideally it should be possible for each workflow to send notifications to and manipulate data of all, or specific forms, regardless of the form which triggered the operation.

Code-behind for the advanced scenarios

Sometimes calculations and validation in forms cannot be fully accomplished by using the expression language alone. This is where so called "code behind" is necessary: one of the standard programming languages is then used to accomplish calculation or validation tasks within the form. Good examples for this are Visual Basic for Applications in Excel, or C# within InfoPath.

However, it needs to be clearly stated that the code-behind scenarios are not desired and should be used only as the last resort within the low-code platform. The source code is typically stored within the low-code platform's data source, which creates a set of challenges for deployment, running and maintenance issues. Applications which use code behind are more difficult to maintain, and especially to deploy to the target environments. The code behind should only be used when needed, and when there is really no other alternative to achieve the goal.

One of the remaining outstanding migration challenges with composite applications to the cloud is the migration of validation logic implemented in custom code, which is difficult to discover if the system has not been previously documented.

Since most of the form components of modern low-code platforms are web based, the natural choice for the code-behind language is JavaScript. Low-code platforms should ensure that it is possible for code-behind to directly reference the application data, as well as environment and user data.

Multilingual support and localization

Multilingual support and localization have been discussed previously in the section about workflows. However, for companies that need to ensure multilingual support, this is even more important with forms.

There are three components of the user interface which are directly impacted by multilingual and localization efforts:

- Platform chrome – all the menus, icons, and other elements common for all applications developed on the platform should be displayed in the preferred language of user. This is a task for the platform developer, to ensure localization of the platform on the major languages. It is important to know which languages are available for the platform chrome, if the missing languages can be added, and existing languages eventually updated.
- Application interface – all the labels, messages, application-specific menus etc. should be displayed in the preferred language of user. This is a task for the application developer, to enable localization of the application that is being developed. However, the platform needs to enable developers to support localizations. This in practice means, that all the labels, tab names, messages and other text elements of the user interface, need to be stored separately from the form, so that they can be translated, and displayed to the user in her preferred language.
- Content translation. In some specific cases, it will be required that even the content of the application, which is usually created by the users or by background processes, is multilingual and available in multiple languages. There are multiple ways to deal with that, but if this is a requirement, the low-code platform needs to offer a multi-lingual storage and manual or automated translation for text blocks created by the users.

Mobile-friendly and responsive forms

The requirement for all the forms to be responsive and mobile friendly is practically a standard today, due to the concepts of modern workplace, and enabling people to work from all device form factors. It can be even be argued that unresponsive, pixel-perfect, full-size forms are becoming more and more of an exception, and they are relevant only for specific applications where a large quantity of data must be entered and viewed within one form.

There are different ways to achieve responsive and mobile friendly forms. Most of them include giving up on WYSIWYG form editors, and falling back to editors with form zones, where form controls are then repositioned and reflowed based on the screen size or other factors (such as high contrast displays).

When choosing a low-code platform, it is very important to understand if the platform supports mobile-friendly, responsive forms, and how that is achieved. The implementation of this feature might pose a significant obstacle to the developers attempting to deliver user friendly and easily adopted applications.

OPERATIONAL AND GOVERNANCE REQUIREMENTS

In the previous section, we discussed the functional requirements that should play an important role when a low-code platform in the enterprise is chosen.

Equally important are the operational and governance requirements. Who is going to use the applications developed on top of the platform? How are they going to be maintained? How can changes be managed? Who will be providing support, and how?

The low-code platforms of the past have been for the most part a complete failure in this regard. The terms "Excel hell" and "Evil Access databases" are still prevalent, describing the situation where Access databases or Excel workbooks are still being used in production, sometimes for business critical processes, without any documentation, with support based on peers, and where the original developer has long since left the company. We witness at present a huge increase in the projects of migrating InfoPath forms/applications to modern technologies, and all the problems and issues that come with that migration.

One of the main goals which needs to be set when choosing a new low-code platform, is sustainability of that platform. Companies need to make sure that the platform has a secured longevity and that it is future proof, that the applications developed on the platform can be documented, supported and managed, even with the original developers gone. Furthermore, it is of crucial importance that frequent changes are possible and maintainable.

What does lifecycle management mean for low-code apps?

One of the reasons why low-code development often has a bad reputation, is the way in which those applications are sometimes developed. It is too often the case that they are developed in production, with live data. After some point of time, no change is possible without impacting the production data, the application becomes unmaintainable, but it quite possibly cannot be decommissioned because it has become an important tool for many people.

The way application lifecycle should be approached with low-code platforms and applications is exactly the same as how it should be approached with traditional development, and it should include all phases such as planning, development, deployment and change. The necessary accompanying activities, such as application documentation and organizing support are also as important as with traditional development.

Still we are witnessing that significant parts of low-code platforms which are available today are still neglecting basic Application Lifecycle Management (ALM) aspects, and that something that is not acceptable with traditional development, still seems acceptable with low-code development. Or worse, such neglect is promoted as "you don't need to bother with that when doing citizen development".

Therefore it is of extreme importance that when choosing a low-code platform, to be aware of what that platform offers in terms of lifecycle management, and more specifically how is development, deployment and change management supported. Neglecting to do so will most probably lead to a repeat of "Excel Hell" and another painful future migration challenge.

Developer teams

Just like traditional development, a lot of low-code applications will often be built by multiple developers.

When choosing a low-code development platform, the following should be checked:

- Can multiple developers work on one application (application ownership)?
- If multiple developers can work on one application simultaneously, what is the level of isola tion?
- How are possible conflicts handled?
- If it is being logged who made the changes and it these changes can be tracked?

Release cycle / Release management

A fundamental lesson from decades of software development experience, is that a clearly defined process of development, release and deployment has to be followed, in order to ensure quality and maintainability. There is no difference with low-code development.

Ideally the following environments should be available in the process of software development:

- development environment (where one or more developers are developing the application).
- integration environment (where the work of multiple developers is being brought together)
- testing environment (where the application functionality and performance can be tested)
- staging environment (where application should be tested under the same conditions as in production)
- production environment (where end-users use the application with the production data).

With smaller organizations, or with a smaller number of developers, some of these environments, and release steps can be safely omitted, or in some cases combined. With low-code development, an integration environment is typically not needed, since there is no source code to merge. With smaller organizations and on smaller projects, testing and staging environments are also often merged into one.

At a minimum, a low-code platform should include:

- Development + Integration environment, used for development of apps
- Testing + Staging environment, used for testing application by QA team and selected users
- Production environment, where the users will work with production data

When choosing the low-code platform, companies should be aware if the platform allows the application to be packaged and deployed to a different (target) environment. Those application platforms which do not allow packaging and deployment, should not be taken into serious consideration for the enterprise scenarios.

Managing application packages and deployments

As discussed in the previous section, one of the major requirements for every low-code platform is the ability to deploy the finished applications to a different environment. If a low-code platform fails to do so, that would mean developing in the production environment, and most probably with the live (production) data. That scenario should be avoided at all costs.

Furthermore, in order to avoid multiple physical copies of one application, and thus possible data redundancy which would in the long run make the application unmaintainable and changes and upgrades very difficult, application developers and the IT departments should have an overview of where an application has been deployed, and which version is in use. It will often be a requirement that one application is used by multiple departments, project teams, or other groups of people, but possibly in different versions. Managing that situation is of a great importance for overall sustainability, and in the end, for the success of the low-code platform.

When deciding on a low-code platform, the following should be taken in account:

- Is it possible to package a version of application and to deploy it to a different target environment?
- Does this package contain all modules of the application (data, workflows, forms, reports, dashboards), or do some parts of the application need to be manually deployed or configured after the initial deployment?
- Can the application packages contain initial data necessary for a first run of the application – e.g. dictionaries for dropdowns?
- Can application developers and corporate IT have an overview of where the application has been deployed, and in which version?
- If the master data is kept and maintained within the low-code application platform, does it have to be deployed as well, and how are those deployments handled?
- Can an application be gracefully retracted from the target system, leaving no trace that it has been deployed?

Managing change

As mentioned in the previous section, the ability to package and deploy apps to different target system is a crucial feature that each low-code platform should implement.

In addition to that, the low-code platform should support updates and rollbacks: every application should and will be updated over time. Sometimes those updates will be just minor bug fixes, and sometimes the updates will mean adding new functionally, including creating new or changing existing data structures.

If those updates are not managed in a structured and organized way, so both application developers and corporate IT have an overview of who is using which version of the application and why, the risk of revisiting "Excel hell" is omni-present.

Ideally the low-code platform should support the following scenarios:

- Provide an overview of all deployments of one app, including the number of currently de ployed version(s).
- Update to a newer version of an app. Ideally, delta-updates should be possible, where only the change is deployed.
- Roll back to the previous version of an app should be possible (in the case of buggy updates)
- Data updates. This is usually implemented in a way, that when new data structures are re quired and available in the new app version, that they will be added during the update pro cess to the target environment, without redeploying complete data sources.
- If the data structures have been deleted in the new version, the update process should offer to those performing deployment to choose if those data structures should be also deleted from the target environment, or merely hidden from the user interface.
- Not require extensive post-deployment cleanup and/or configuration. Deploying an applica tion update only to have to reconfigure every data connection to use a production database instead of the development one is a disruptive act.

Roles, permissions and security

In an enterprise environment, where different people will be involved in the in the different phases of creating and managing an app which is built on top of a low-code platform, it is crucial that the low-code platform supports different roles and permissions.

Different low-code platforms approach this issue in very different ways. Some of them tie their own permissions concept into the permission concept of the underlying data source (this is typical for the low-code platforms which are built on top of SharePoint), and some of them offer their own permissions and security concepts.

Typically, the permissions a low-code platform should support are:

- App developers – people who can create new apps or modify existing apps.
- App administrators – creating app packages and deploying, updating and retracting apps.
- Global administrators – global settings, permissions management, and security settings.

Usually those roles are also set per app. Each app should have clearly defined roles - who can develop and modify the app, who can package, deploy, and update an app, and people who can change the security settings for that app ("app owners").

Operationally, permissions also define which features users can see, let alone use. They can even govern the flow of execution and the ability to create/modify/delete data.

Different low-code platforms handle app users in different ways, and that is foremost dependent on the licensing model.

Before choosing a low-code platform for the enterprise, companies should understand which security and permission concepts are present, and how that affects licensing.

Creating and managing documentation

Developers as well as IT departments know how important documentation is: it is easier to understand what has been done and why, what is the purpose of different application elements. Without documentation, applications remain "locked" to their original developers, and are very difficult to maintain and support.

One of the advantages of low-code platforms over traditional development is that development, modifications, and changes are easier to track, understand, and reproduce compared to traditional source code. The drawback is that the configuration can be performed in various different places in the app, so getting a comprehensive overview of all customizations might pose a challenge (and, in fact, the mark of a good tool is that it can provide that kind of comprehensive overview).

Some low-code development platforms offer the auto-generation of application documentation. This typically includes documentation of the data model, external connectors, all forms and workflows. Ideally such a generated documentation would include the app versions when configurations have been made.

Before choosing a low-code platform for the enterprise, it should be checked if the platform offers auto-documentation capabilities, and to what degree.

Metrics

For each enterprise it is of huge importance to be aware of the IT resources used within the company. That information can help to optimize those resources, to invest more in the applications which are used by many, or to analyze what are the reasons for low usage of other applications.

Low-code platforms should provide this usage information on per-application level. The developers, and the IT department, should know how many users are using an application, and how frequently they are doing so.

This information should be typically presented in a form of a dashboard.

Before choosing a low-code platform for the enterprise, it should be checked if the platform offers such app metrics capabilities, and to what degree.

Support for hybrid environments

Even in a predominantly cloud world, the vast majority of companies have data which are hosted on premises, and which are – for all different reasons – possibly never going to me migrated to the public cloud. There are various ways, how different low-code application platforms are handling those hybrid environments, or if they can handle this at all.

- Native support for hybrid environments – a low-code platform can access both cloud and on premises data without separate proxies and gateways. When this is the case, it is import ant to know whether the low-code platform must be hosted on premises or whether the organization has the option to host it either on premises or in the cloud. If it can be hosted in the cloud, it is important to see what needs to be taken care of in order that it can reach on premises data and services.
- Support through a gateway or proxy. This is usually the case with a cloud-only platform, which then needs a separate piece of software to connect to the on-premises data. In this case, the questions that need answers are:
- How complicated and sensitive is setting up the gateway, which is then becoming a poten tial single point of failure in this scenario
- Will on-premises data be sent to cloud for processing? Which compliance and security proce dures are set in place to prevent data breaches?
- If a company is operating a hybrid IT infrastructure, and if the low-code platform should work both with hybrid and cloud data and services, those questions should be considered among the criteria for choice of the low-code platform.

Licensing

Last, but not least, the low-code platform should have a clear and understandable licensing model, which will enable enterprises to perform reliable cost planning, and to know the financial impact of the wide-spread use of the platform within the company. The licensing model should ideally be protective of enterprises, without sudden and surprising pricing and licensing changes, which can happen during or after implementation.

The licensing models vary widely between different low-code platforms, and it is often difficult to compare them side by side. The only viable approach to comparison is to make a projection of the number of applications and users which are going to use those applications, and then to calculate the expected price.

Typical licensing questions and options include:

- Is the platform licensed by number of users, by application, by something else, or is it a com bination of these factors?
- Are both subscription and one-time-purchase models available? What is the cost prognosis with both models for mid-term and long-term periods?
- What is the cost of the yearly software assurance with one-time-purchase model?
- Is there a customer protection clause against unexpected changes in licensing and price models, which would change the yearly costs to a large degree?
- Is it possible to move between licensing models without impacting the hosting of applications?

CITIZEN DEVELOPERS AND THEIR ROLE IN USING THE LOW-CODE PLATFORM:

Low-code platforms have often been associated with so called "citizen development", where as "citizen developers" are considered those who are not professional developers, but still create some kinds of business applications with the tools available to them.

That is not an incorrect assumption. From the early days of low-code platforms, such as Microsoft Access database and especially Microsoft Excel spreadsheets, technology-prone business users were creating worksheets, databases, and SharePoint lists. It is fair to say that low-code platforms have historically been used as much by citizen developers as they were by professional developers. Some, such as Microsoft Excel, even more so.

The issue with citizen development, from the very beginning, has been the fact that it has almost always happened under the radar, in the shadows. IT did not endorse or support citizen development, and those business users have been left to develop, maintain, update, and support the applications they have created on their own.

At the same time we have learned from history that citizen development has been needed, and it is still needed. Business users have a large amount of knowledge in their respective fields. The business analysis and requirements scoping phases with citizen development are reduced to the minimum. The issues that came out of citizen development are almost exclusively due to the lack of knowledge about specific technology, and lack of the knowledge about all the things one application needs during and after the development.

And those are exactly the reasons why citizen development should be endorsed, but also guided and supported by IT, and more widely by the organizational culture towards IT.

Citizen developers on their own

A typical process in citizen development has historically looked like this:

1. Citizen developer has an idea for an app, based on needs and subject matter knowledge.
2. Using a low-code platform, citizen developer uses simple app construction methods to build the app.
3. Citizen developer identifies existing data sources or creates new ones.
4. Citizen developer uses the app and makes it available to the other users.

This approach to a large degree, led to "Excel hell" and "evil Access databases". Even if the citizen developers could get acquainted with the tool, they seldom would know about or consider the other implications addressed by software development disciplines. And while the applications they create would help them and their colleagues in the short term, they would cause a lot of headaches in the mid and long term.

Alliance between Citizen Developers and IT departments

What has been proven as a method to successfully empower citizen developers, is to form an alliance between citizen developers, and the IT department. The IT department would endorse, and support citizen developers, making a supported low-code development platform available to them. Citizen developers would accept the guidance and help of the IT department, to ensure the quality, maintainability and longevity of the created applications.

In such a situation, the process of citizen development will look slightly different:
1. Citizen developer has an idea for an app, based on her needs, and on their subject matter knowledge
2. Citizen developer identifies existing data sources or create new ones. The IT department helps with this step, ensuring that the adequate and approved data sources are used.
3. Using the low-code development platform, citizen developer uses simple app construction methods to build the app.
4. IT works with the citizen developer to apply standards and guidelines, to ensure support and governance
5. IT department organizes the support process for the application
6. Application is deployed/published to all users who need to use the application
7. Application needs a change: the process starts from the beginning

This way the IT department leaves citizen developers with the possibility to develop the business apps that they need and which otherwise could not be developed because of the overstretched IT resources or other reasons. At the same time, the IT department is there to support and guide citizen developers, to ensure quality, maintenance, and support.

Ideally, the low-code application platform would be used by professional and citizen developers alike.

Fundamentally, the lifecycle of applications and their confidentiality, availability, and integrity must be considered intrinsic elements of the successful implementation and adoption of low-code platforms to avoid the common problems of previous approaches to citizen development.

CHOOSING A LOW-CODE DEVELOPMENT PLATFORM: CHECKLIST

The following check list is a rough guide to help enterprises when choosing the low-code development platform. Each enterprise will have its own preferences and areas of special interest in that process. This check list does not claim to be definite, or the only basis for that decision. It is intended to help enterprises to retain an overview of relevant aspects of low-code development platforms.

FEATURE	LCP-1	LCP-2	LCP-3
DATA			
Visual data modeler	☐	☐	☐
Support for relational data models	☐	☐	☐
History data, previous versions, workflow logging, auditing	☐	☐	☐
Automated data backup	☐	☐	☐
Migration from legacy data sources	☐	☐	☐
PROCESS ENGINE (WORKFLOWS)			
Process assignment and delegation	☐	☐	☐
Setting permissions from workflows	☐	☐	☐
Triggers and background processes	☐	☐	☐
Support for state machine workflows	☐	☐	☐
Extending workflows – custom actions	☐	☐	☐
Extending workflows – custom connectors	☐	☐	☐
Extending workflows – web service and external API calls	☐	☐	☐
Office and PDF documents generation in the workflow	☐	☐	☐
Multilingualism and localization in workflow tasks and stages	☐	☐	☐
Mobile friendly presentation of workflow tasks and stages	☐	☐	☐
USER INTERFACE (FORMS)			
WYSIWYG Forms Designer	☐	☐	☐
Zones-based Forms Designer	☐	☐	☐
Parent-child relations: repeating tables	☐	☐	☐
Parent-child relations: repeating sections	☐	☐	☐
Expression language	☐	☐	☐
Calculated fields on form	☐	☐	☐
Form flow, form control states, wizard forms	☐	☐	☐
Field validation and form validation	☐	☐	☐
2-ways Integration with workflows	☐	☐	☐
Code behind (extending forms through programming)	☐	☐	☐
Multilingual and localized forms	☐	☐	☐
Mobile friendly and responsive forms	☐	☐	☐
OPERATIONAL AND GOVERNANCE REQUIREMENTS			
Support for developer teams	☐	☐	☐
Support for deployment to different environments	☐	☐	☐
App versions, app updates and rollback process	☐	☐	☐
Deployment cockpit – where is an app deployed	☐	☐	☐
Different user roles (administrators, developers, operations)	☐	☐	☐
Automatic generation of app documentation	☐	☐	☐
App usage metrics	☐	☐	☐
Support for hybrid environments	☐	☐	☐

WEBCON®

WEBCON develops the low-code Business Process Automation platform, WEBCON BPS (Business Process Suite).
This enterprise-grade system allows organizations to embrace digital transformation by digitalizing their workflows
and building comprehensive, scalable, process-centric and future-proof applications applying the agile DevOps model.

At WEBCON, digital transformation isn't a buzzword – it's a way of life; it's about reducing steps, eliminating mistakes, ensuring compliance, connecting assets, and encouraging continuous improvement. Applications made with WEBCON BPS are scalable, process-centric, low-to-no-code, equally at home online or on-premises, and happily used on both desktops and mobile devices. WEBCON's unique InstantChange™ technology lets customers adapt/evolve processes
to address changing needs immediately and painlessly. WEBCON processes are clearly understood and easily governed, and they can be connected to line of business systems, documents, forms, messages, and collaboration workspaces.

Over 350 happy clients worldwide underline the platform's success in effective business process management.
WEBCON partners with top-class international IT system integrators from 12 countries who deliver business solutions for industry-leading enterprises.

Want to learn more & find out what we can do for your company?

LEARN MORE >

webcon.com
office@webcon.com

600 Stewart St., Suite 400
Seattle, WA. 98101, USA