

# How (not) to fail a Microsoft Power Platform Project



Hint: failing is easy, you don't even need to try hard

*By Alex Shlega*

*Power Platform Solution Architect, Developer, Consultant, and, surprisingly,  
still a business applications MVP*

<b>1. Introduction</b>	<b>3</b>
<b>2. The promise of Power Platform</b>	<b>5</b>
<b>3. Why go Agile (and what is Agile?)</b>	<b>7</b>
<b>4. Power Platform projects and Agile</b>	<b>9</b>
<b>5. What's the common pattern?</b>	<b>10</b>
<b>6. Fit-gap analysis is not a guarantee</b>	<b>11</b>
<b>7. Face these Dataverse-related questions early, or face the consequences later</b>	<b>14</b>
<b>7.1. Security</b>	<b>16</b>
<b>7.2. Search</b>	<b>20</b>
<b>7.3. Reporting</b>	<b>24</b>
<b>7.4. Translations</b>	<b>26</b>
<b>7.5. Deep customization and advanced business logic</b>	<b>29</b>
<b>7.6. Data migration and data integration</b>	<b>32</b>
<b>7.7. Resourcing</b>	<b>35</b>
<b>7.8. User Interface</b>	<b>38</b>
<b>7.9. Application Lifecycle Management</b>	<b>42</b>
<b>7.10. Infrastructure, Tools, Resources</b>	<b>45</b>
<b>7.11. Licensing</b>	<b>47</b>
<b>8. Feeling overwhelmed? Here is what may help</b>	<b>49</b>
<b>9. Conclusion</b>	<b>50</b>
Appendix A - What's not included and will be added later	<b>51</b>

# 1. Introduction

This book has simply happened. Actually, it still keeps happening, since I am not sure the first “edition” is really going to include everything I have in mind, but it definitely was not in the plans even a few months ago.

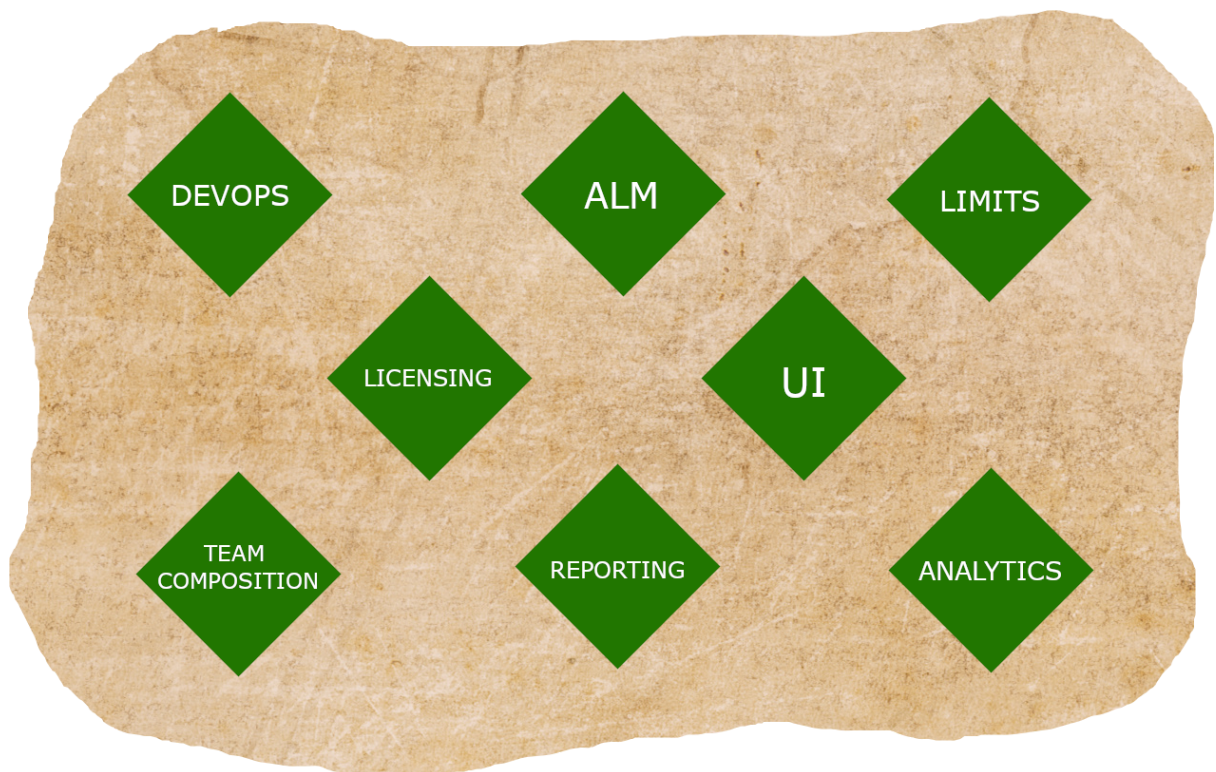
However, it has always been the case that I would leave one project, join the other, and that would kick off a bit of a thought process which would have resulted in more blog posts, or, as it happened this time around, it has resulted in a book. Who would have thought...

It's not that I would not have anything to say. I've worked on quite a few Dynamics / Power Platform projects in all these years, but I'm not even going to pretend I know it all.

Still, there are some patterns, and one can't help but notice them after a while.

Why is it called “How to Fail a Power Platform Project”, though? There are lots of great resources out there which would tell you how to do certain things, and, at some point, I realised that there is not a lot I can add to what's already been said by the other folks. If you have a technical question about Power Platform, most likely you can “just google it”, and you'll get at least some answer.

However, what if you were able to see potential issues before you run into them? It's not that you are certainly going to experience those issues first hand on each and every project, but, once you know they are lurking there, you may, at least, start taking some precautions beforehand.



Knowledge is power, after all, Power Platform has lots of goodies, but it also has some traps. Don't walk into those traps, stick to the goodies, and you and your clients will love your Power Platform experience.

So the purpose of this book is to show you those traps. Usually, there are different ways to navigate around them, but that's not what this book is about.

## 2. The promise of Power Platform

Power Platform is one of the leading low-code development platforms out there, and I am sure you have heard about it. For the purpose of this book, I will be assuming that you also have some idea of what Power Platform is, and how it works. The goal of this book is not to introduce you to the Power Platform, but, instead, to discuss one very peculiar aspect of Power Platform projects, which is that they tend to get out of control easily, and, sometimes, it seems all they need to do that is just a bit of time..

I will get into the details further down, but, right now, let's just look at what Microsoft has to say about low-code development, which is one of the pillars of the Power Platform:

*Low-code development can facilitate your company's digital transformation. Instead of relying heavily on programming, low-code platforms simplify application development with techniques like drag-and-drop functionality and visual guidance. This means that anyone in your company, regardless of their technical expertise or abilities, can build apps so that the business can offload some tasks from IT.*

[\(https://powerapps.microsoft.com/en-ca/low-code-platform/\)](https://powerapps.microsoft.com/en-ca/low-code-platform/)

In other words, low-code development, by definition, unlocks the door for non-developers to build their own applications without having to hire professional developers.

This is a great concept, and Microsoft has been working hard on evolving Power Platform into a tool that can live up to those expectations; however, there are certain things that may get in the way, and a lot of them have to do with how low-code development is not the same as no-issues development.

I've been working with Power Platform, and, before it, with Dynamics, for quite some time now, so let me share a secret with you. There are definitely ways to make Power Platform projects fail!

The purpose of this book is to show how that happens, and, then, you can choose what to do with that knowledge.

If, however, you decide to use this book as a manual when trying to fail your next project, just make sure not to tell Microsoft where you are getting those ideas from, since they may decide to do something nasty about it. As in, those folks from the product team will always come up with all sorts of workarounds / suggestions, and they'll even release product hotfixes just to prevent you from failing. So beware.

And, with that, let's start, should we? We'll need to lay out a bit of a foundation, though, so bear with me - I will do it in the first few chapters, and, then, we'll dig into some of the details.

And the very first question is going to be: why does Power Platform lend itself so readily to the Agile process, and why is it that almost every Power Platform project will be agile these days?

### 3. Why go Agile (and what is Agile?)

According to the Agile Alliance:

*“Agile is the ability to create and respond to change. It is a way of dealing with, and ultimately succeeding in, an uncertain and turbulent environment”*

(<https://www.agilealliance.org/agile101/>)

In some existential way, Power Platform is a manifestation of the Agile approach Microsoft has clearly adopted on their side to begin with. There was a need for a new, low-code application development paradigm, and that's what Microsoft has come up with. Along the way, Microsoft has developed the whole ecosystem of tools such as Power Apps, Power Automate, Power BI, Power Virtual Agents, etc. A lot of these developments have been guided by the feedback from the clients / community, and I am not sure anyone at Microsoft had known where this was all going a few years ago, yet the current plans are more or less certain for the next 6-12 months, and more remote ones seem to always depend on the current prioritisation.

One might argue things may have happened the other way around - as in, perhaps, some of the tools which are currently considered part of the Power Platform had already existed when it all started, and, so, they were brought together under the Power Platform umbrella rather than developed from scratch. Which is true, to some extent, but, even so, those tools had to be adjusted / somewhat reinvented to live happily in the Power Platform ecosystem. In either case, those details are not, really, relevant. No matter how we choose to look at it, it would be difficult to deny the agile nature of Power Platform.

That does not stop there, though. You will rarely, if ever, see a Power Platform project that's following a waterfall approach - Agile and Power Platform seem to go side by side, it's almost a natural fit, and this extends to the solutions built on top of the Power Platform.

Again, look at it this way:

- Power Platform itself is always changing. There are two major releases per year, not to mention all the smaller ones in between. With those changes, there is always something to look forward to, and, depending on which features get released, our own Power Platform projects have to adapt to those changes, so that means we have to assume some flexibility from the beginning
- Resources availability, team maturity, even our users' familiarity with the tools offered by Power Platform is never a given. Today, you may have someone on the team who is more familiar with the model-driven applications, and they might choose model-driven as an option. Tomorrow, you might get citizen developers involved, and they would rather do their development with Canvas Apps. Your project may get a sudden boost when a new feature gets released by Microsoft, or you may have to figure out a workaround for a feature that's been delayed or not working.

In other words, more often than not, a Power Platform project would be an Agile project to a large extent since it's almost impossible to apply rigid long-term waterfall processes to these projects. At most, you may be able to come up with a high-level plan, but, then, you'll likely be better off to go about the implementation with Agile in mind.

Now what is required for any agile project to succeed? It's the flexibility, of course. Everyone and everything has to be flexible - from the resourcing, requirements, functionality, timeline, and pretty much from any other perspective.

Agile has been around for quite some time, though, and almost everyone would have at least some idea of what it is; however, generally speaking Agile applies more to the development teams than to the business stakeholders, and, usually, it's more about adapting to the always changing priorities than it is about adapting to the technology.

Power Platform, on the other hand, introduces an additional layer of uncertainty where everyone has to adapt to the technology itself as well.

How often will your familiar Javascript framework be updated? If anything, it'll experience gradual improvements year after year, but it's not as if it would be re-built from the ground up. As for the Power Platform, there are always new additions / changes:

- Power FX is replacing javascripts
- Power Automate is replacing classic workflows
- Canvas Apps are merging with Model-Driven
- Power BI is replacing SSRS reports
- Power Automate word templates are replacing classic word templates
- Licensing fees tend to change periodically
- First-Party applications are getting updated all the time
- Power Portals are getting renamed to Power Pages, and that comes with some licensing changes, too

See what I mean?



## 4. Power Platform projects and Agile

Agile is all about being flexible - this is what's expected of the development teams. They need to be able to adapt to the always changing environment, and, usually, development teams are used to doing it well. However, Power Platform introduces yet another, subtle level of uncertainty, that takes flexibility expectation to a whole new level.

Is this specific to the Power Platform projects? Not quite, it's just that there is a combination of factors that come into play in this case:

- PowerPlatform is always evolving - there is always something brewing there, there are features getting ready to shipped, there are other features getting deprecated, completely new functionality may show up that expands your ability to implement required functionality, etc. This, of course, is due to the excellent work of the Power Platform product teams which keep maintaining their pace after all these years. Usually, this actually helps to deliver the projects faster. But it may also introduce some unexpected delays when the team needs to deal with the features being deprecated or when there is a dependency on the preview features which have not been pushed to the GA yet
- There are certain aspects of PowerPlatform which tend to be overlooked. We always hear about how the platform is capable of just about anything, including low-code development, security, reporting, etc. The devil is in the details, though. Occasionally, you might be lucky enough to work on a project where you don't need to deal with those details. However, as time goes on, you will almost inevitably start hitting minor or major limitations and inconveniences which you will have to deal with. The ability of the team (and, perhaps more importantly, the willingness of the stakeholders) to adapt will be crucial to the ongoing success of the project

Those two, together, may give you a clue of where this is going. You have a platform that's quickly evolving, it's always growing and expanding, certain features are changing, others are getting deprecated, and the level of control you have there is not even close to where it would be if your team were developing an application completely from scratch using Java or .NET.

Your development team might be somewhat accustomed to this, though it is never a given and a lot depends on the experience of your team members. Yet your stakeholders might find this sort of uncertainty even more troublesome, since they might be much more used to how custom development is, usually, capable of accommodating their needs (given time and resources, of course), and Power Platform will sometime be "no can do", so everyone on the team, including the stakeholders, will have to work together on the alternative ways to design the system and to achieve desired outcomes.

## 5. What's the common pattern?

In reality, this whole situation is ripe for failure.

Think of the opportunities there. You have stakeholders that can easily get frustrated, you have a development team that can easily get somewhat out of sync with the platform capabilities, you may even find yourself making wrong architectural decisions today since your experience from the past does not necessarily reflect the current situation.

Besides, there are forces outside of the Power Platform that might leap into action when you least expect that. For instance, in many cases, there are regulatory and other requirements that cannot be easily handled, and, if you let them go unnoticed for long enough, they might grow into a huge obstacle on your way to production.

A lot of this is about now knowing when to raise the alarm and/or about raising false alarms where it's not needed. Starting with the stakeholders, and down to the level of each individual developers, it's not uncommon for the team members to misunderstand certain aspects of the Power Platform, and, once that misunderstanding gets built into the final solution, at some point it becomes a minor inconvenience, a major defect, or a full-blown show stopper.

For example, quite often it would be required to provide full logging capabilities. They are available out of the box, but there are important questions to answer, such as permissions, retention, reporting capabilities, etc. Different people may have different idea of what logging means, and, given size/reporting/security and/or retention limitations of the Power Platform logs, it would be reckless to promise full logging capabilities to the stakeholders without actually clarifying what those capabilities are.

In other words, there are certain risks, a lot of them are well-known, but, with agile, not all of them will be dealt with soon enough. There are lots of things that can be done on a project before having to dig into the really risky (and, often, time-consuming) areas, and all one needs to do to ensure eventual failure is to keep delaying that moment until such time when it's, already, too late to change anything.

Normally, this would all be uncovered while working on the fit-gap analysis, so let's talk about it in the next chapter.

## 6. Fit-gap analysis is not a guarantee

The usual purpose of fit-gap analysis is to identify the gaps, since, where there is a gap, there is a risk. By uncovering the gaps, we are also uncovering the risks, and, from there, we can start mitigating them. Or, perhaps, we can even say that there is no fit.

However, imagine you were shopping around for a used car. You could be talking to a salesperson in the dealership, or you could be talking to your trusted fellow mechanic, and you might be getting somewhat different responses to the same questions. That analogy is a little stretched to be fair. I've met a lot of great salespeople who are very knowledgeable and helpful; however, ultimately, the point is: when the devil is in the details, you need to be talking to someone who understands your situation and who does have an in-depth understanding of the subject to begin with.

In case with the fit-gap analysis, this is where things can easily fall apart, since:

- We might not be aware of some requirements while doing the fit-gap
- We might have limited budget, and, so, may have to stick to the high-level analysis only
- We might not know about some features and/or limitations of the platform either, and that might affect the quality of our fit-gap analysis

Even when the fit-gap is done with the best intentions in mind, it may turn out to be useless in retrospect due to the unfortunate combination of those three factors above.

There are, however, additional techniques that are meant to increase the credibility of fit-gap analysis - typically, you would be looking at the following options:

- Implementing a proof of concept solution
- Implementing a minimum viable product (MVP) solution

The two are not mutually exclusive, since the idea is that you may still need a proof of concept to validate some assumptions about those areas which are not going to be implemented in the MVP, for instance. Besides, a project would, often, start with the proof of concept in either case, then evolve into an MVP.

Ultimately, though, the purpose of fit-gap, POC, and, to some extent, even of the MVP is to ensure that the development team can figure out possible functionality gaps, demonstrate the ability to work around those gaps, and also, figure out how flexible the requirements/expectations of the broader project team are. Once that's been done, presumably it becomes easier to set high-level milestones for the project implementation, to get some idea of the overall effort, to set the expectations properly, and to start moving towards the end goal, which is to have the solution implemented.

And, of course, there are so many ways to plant seeds of failure at this stage! The platform is evolving, it's huge, every organisation is somewhat unique, all those regulatory

requirements are adding subtle complexities to the implementations all the time, so, in reality, it's not even possible to review all the gaps.

Fit-gap analysis will almost always be relatively high-level, and it will be for the POC / MVP implementations to look at the details more closely. And, usually, that's the intent of the POC/MVP, but it's important to remember that, when dealing with the time/budget/resource constraints, certain aspects of that work are always going to be sacrificed.

Even with the best intentions, we often can't guarantee anything, and that actually opens up some interesting scenarios where the same project can be considered a success at some point, and a failure at another, and that can easily change within the span of a few months.

After all, your MVP could be successful, but there might be missing features which are hugely important for the next release, and which might not be achievable without redesigning a significant piece of the MVP.

For example, your stakeholders may agree that, because of the limited scope of the MVP, it's not that important to look at the security model right in the MVP. However, if you are supposed to cover security in the post-MVP release, that's when you may come to the realisation that you now need to start moving things around, introducing the business units, starting to use access teams, and, even so, you may not be able to achieve all intended goals without redesigning the data model.

You cannot guarantee that your fit-gap analysis will be absolutely accurate, and you cannot, really, investigate all the risks in the proof of concept / minimum viable product solutions, but you can, and probably should, identify as many of the "generic" risks as possible to the client where you see them potentially affecting intended implementation. That would not be to turn the client away from the Power Platform, but it would be to make them think about those risks sooner than later.

There are some risks which keep affecting various Power Platform projects over and over, though, and, to a large extent, this seems to be happening because of the overselling at different levels. As in, we all know that Dataverse, as one of the main data storage options on the Power Platform projects, has a robust security model, there are great search capabilities in the model-driven application, there is Power BI for reporting... But, of course, what has just been left out of that statement above is that there is no record-level security in Dataverse, some security rules might be challenging to set up, search functionality is not, always, that user-friendly, and Power BI expects the users to build up yet another set of skills.

In my mind, the easiest way to fail a Power Platform project in the long term would be to completely ignore those common issues until such time when they become a show-stopper. The list of such issues is always evolving, some are becoming less important, but others may suddenly show up. This is where a good Power Platform consultant may be able to raise those issues to the project team early enough to ensure solution development takes the right turn at the right time.

Let's have a look at some of those and see why they matter.

## 7. Face these Dataverse-related questions early, or face the consequences later

The choice of when to deal with the questions below is yours, of course, but, as I alluded to before, one easy way to fail a Power Platform project would be to keep delaying the inevitable till later, since, quite frankly, it might turn out to be just too late. In either case, no matter what you decide to do, let's go over some of those questions in no particular order.

For this chapter, I am going to make one assumption, though, which is that your data is stored in Microsoft Dataverse. Reason being, it's all too easy to think of the Dataverse as of yet another database and forget that it's, actually, not. Which is where most of the questions / issues, if left unattended, may easily start derailing the project.

To begin with, what is Microsoft Dataverse?

It would be difficult to give a single-sentence definition of what it really is, but here is how it can be defined according to Microsoft

(<https://learn.microsoft.com/en-us/power-apps/maker/data-platform/data-platform-intro>):

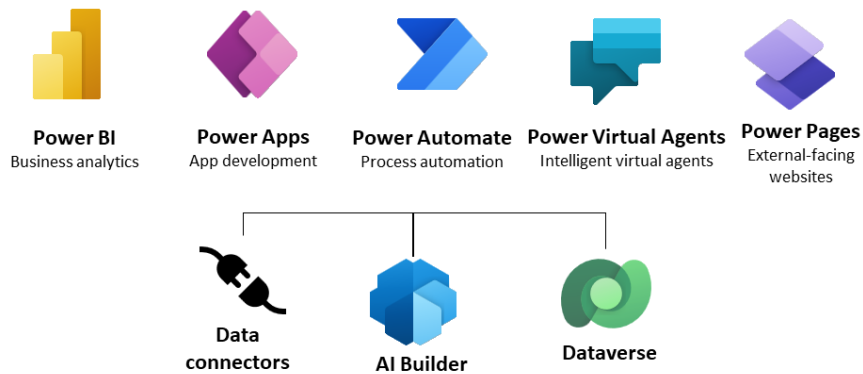
- Dataverse lets you securely store and manage data that's used by business application
- It's easy to manage – both the metadata and data are stored in the cloud. You don't need to worry about the details of how they're stored.
- It's easy to secure – data is securely stored so that users can see it only if you grant them access. Role-based security allows you to control access to tables for different users within your organisation.
- Dynamics 365 applications data is also stored within Dataverse, allowing you to quickly build apps that use your Dynamics 365 data and extend your apps with Power Apps
- It provides rich metadata – data types and relationships are used directly within Power Apps.
- There is logic and validation – define calculated columns, business rules, workflows, and business process flows to ensure data quality and drive business processes.
- There are productivity tools – tables are available within the add-ins for Microsoft Excel to increase productivity and ensure data accessibility.

And the diagram below (from the same link above) explains where Dataverse belongs in the grand scheme of things:



## Microsoft Power Platform

The low code platform that spans Microsoft 365, Azure, Dynamics 365, and standalone apps.



According to the bullet points above, with Microsoft Dataverse you get secure storage where it's easy to access and manage data, where you can add advanced logic and validation, and which you can use to easily build apps on top of it.

Most of it is correct most of the time, but, then, sometimes it's a bit more complicated than that.

## 7.1. Security

Dataverse security relies on a few security permissions you can assign to your users, but, also, those permissions can be given on different levels. The permissions are:

- Create, Read, Write, Delete
- Append, Append To
- Assign
- Share

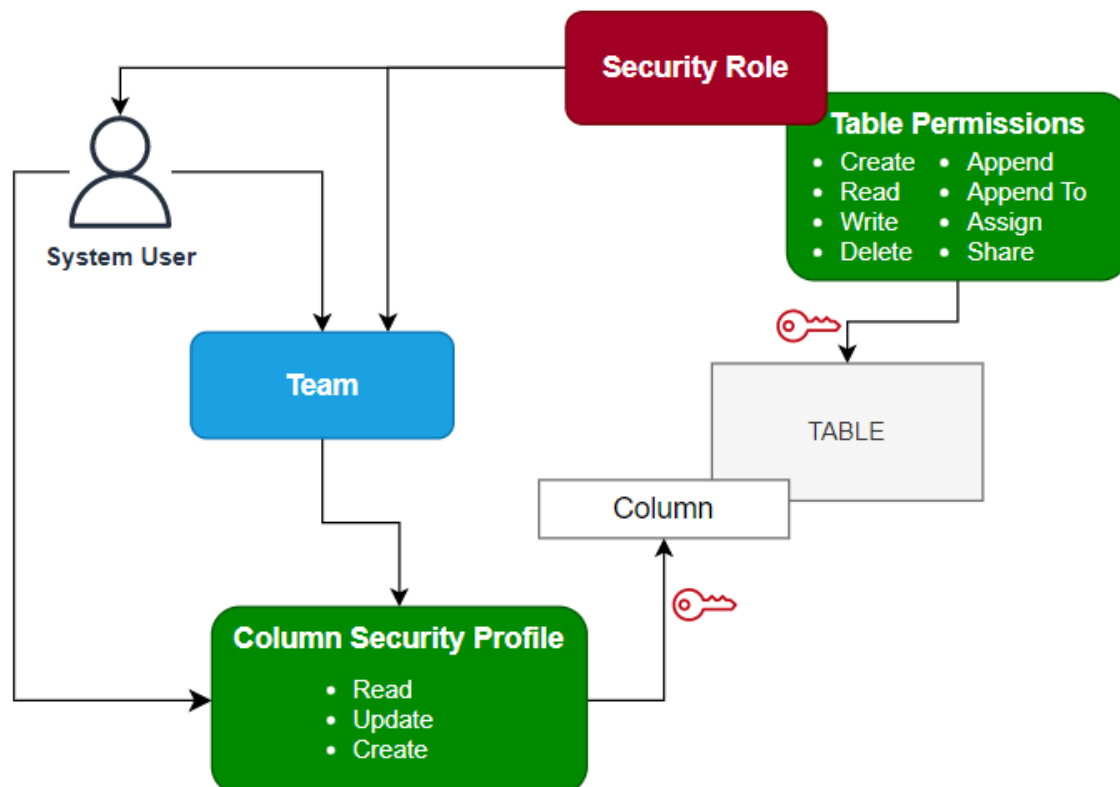
And the levels are: User, BU, Business Unit and Child Business Units (Parent Child), Organisation.

(If you wanted a more detailed walkthrough, have a look here:

<https://learn.microsoft.com/en-us/power-platform/admin/wp-security-cds>)

Those permissions can be given per table, and, of course, they can be organised into security roles, they can be given to the teams, then users can either be given those security roles directly or through the teams.

You can also protect individual columns in the tables with the field security profiles, and, just like with the security roles, you can assign those profiles to the users directly or using teams.





*Please note that the diagram above is overly simplified, since it does not reflect the concept of ownership / business units (which is how the “level” of table permissions is defined).*

This model is quite flexible, but there are some notable exceptions and caveats.

First of all, there is no record-level security. Well, there kind of is, since you can use “user” level security settings to only give users access to their own records in the tables. However, normally this is not what you want - instead, you may want to organise data access around team membership, for instance, and, even though there is a concept of the access teams, which is, really, an extension of the “share record” capability, it might be difficult to properly share records in an automated way. Those scenarios can quickly become overly complicated, since you may need to automate sharing not only when a new record gets created, but also, when users are moved to different teams, when certain fields get updated on the records, etc.

However, for a given record, you cannot deny access to that record for a specific user if that user has already been given access through the other security roles.

What if there were a user who had a conflict of interests on a specific sales opportunity, and you would not want that user to have access to that particular sales opportunity? Typically, your security roles would define table-level access, but this situation is different, and you may have to work around it by placing certain records in a dedicated business unit, for instance, and starting to share such records with those users who need access. That complicates the security model quite a bit, requires some automation, and, ultimately, increases the complexity of the project.

There are some edge cases, too. If you ever used Notes in your applications, you may have noticed how notes get re-assigned when the records they have been added to gets reassigned to another owner. This is happening because all note relationships are configured to cascade “assign” operation:

Type of behavior \* ⓘ

System ▾ \*

Delete \* Cascade All ▾

Assign \* Cascade All ▾

Share \* Cascade All ▾

Unshare \* Cascade All ▾

Reparent \* Cascade All ▾

And it's not, actually, configurable. Given that notes are often treated as “user notes”, and the creator of a note is expected to have full access to their note, this ends up being a problem, since, once a note has been re-assigned, the original owner may not be able to update it anymore.

Conceptually, Dataverse security works best when there is a way of putting your users into buckets so that you can, then, organise those “buckets” around the business units, and, within each business unit, you can create security roles with different levels of access. Quite often, though, either the users are not getting assigned to the business units forever, or the data has to be shared in a more generic way so it does not fit those buckets either.

The real issue with all those security scenarios is that they are, often, underestimated from the complexity perspective. In the proof of concept / minimum viable product, the project team may choose to disregard them, since, in general, the security model looks very flexible and capable of handling a lot of different scenarios. However, by the time those limitations are discovered, they often become a show stopper, and, since those are platform limitations, they may be difficult to work around.

From that standpoint, trying to uncover related security requirements early usually makes a lot of sense, since that allows you to bring potential issues up to the stakeholders earlier than later, and, perhaps, to steer them away from some of those requirements which might be difficult, or even impossible, to implement.

Aside from the few scenarios above, you might also want to consider if any of the below scenarios apply, since this is where you may need to do some digging:

- Are you planning to enable Sharepoint integration? Do you need to secure Sharepoint documents in the same way their associated records in Dataverse have been secured?

- Do you want users from different business units to have access to the activities in the context of the main record such as case, for instance, while having limited permissions on the parent record itself? That's going to take some playing around with the roles, record sharing, may require automation etc
- Are you planning to configure different permissions for different types of activities (phones, emails, letters, etc)? Can't do, may need to re-consider the requirements
- Do you intend to share the data in general (such as contact and/or accounts), but still lock down some of those records when required by a certain business group? That might not be covered by the security model per se - you may have to implement certain customizations

As soon as you get more than one user and/or more than one business group working with your application, you will almost inevitably have to deal with those security scenarios, and the sooner you do it the better, since, otherwise, you may end up having to re-model some pieces of your application, which means you may end up with having to spend more time not only on the implementation, but, also, on re-training the users, migrating the data, etc.

## 7.2. Search

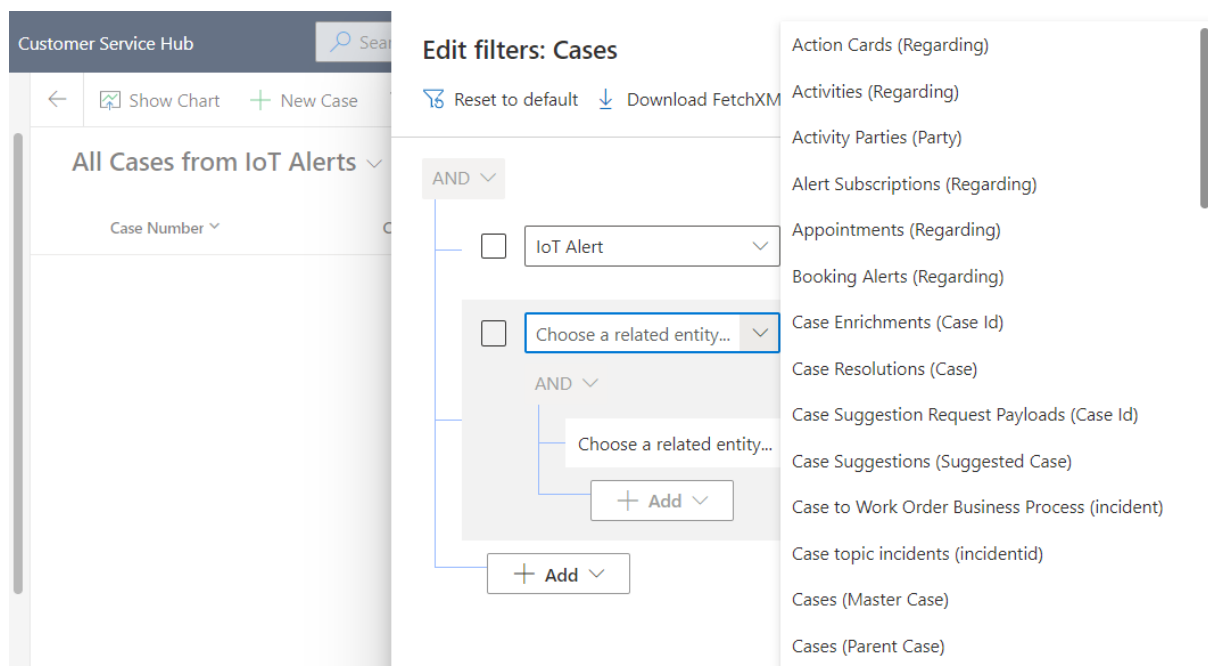
In the model-driven applications, search functionality comes right out of the box, and it has at least three different flavours:

- You can use Dataverse search
- You can use quick find
- You can use “advanced filters” (former advanced find)

In the canvas applications, you have none of that, so you need to build out a user interface to let users perform queries against the Dataverse (or against whatever other datasource you may be utilising in the application).

It may look as if model-driven applications have certain advantages in that sense; however, advanced filters, while offering a lot of useful functionality, can be difficult to grasp for the regular users who have no time and/or no training to be able to fully understand all those relationships in the data model.

Have a look at the screenshot below:

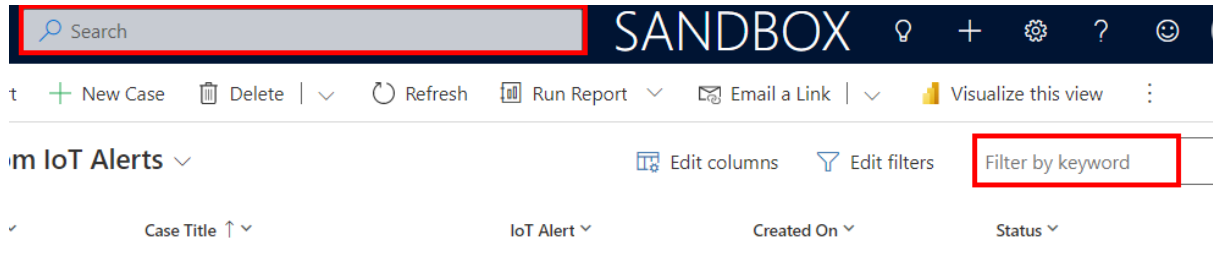


There are a few potential issues with this:

- You cannot expect your business users, who would normally be focused on the operational efficiency (the notorious “one click does it all” approach) to really like this kind of advanced filtering, since it definitely takes more than one click to use it properly
- Even though the query builder is very powerful, there are some notable caveats. For example, we cannot group criteria from different related tables. We cannot add

columns from the child tables to the output. Besides, those tables names / display labels you see on the screenshots are not easily configurable (and they are definitely not configurable per user role)

Usually, advanced filtering turns out to be more suitable for the advanced users, and, normally, there is only a limited number of those. Most of the other model-driven application users will likely have to rely on the quick find and/or on the dataverse search:



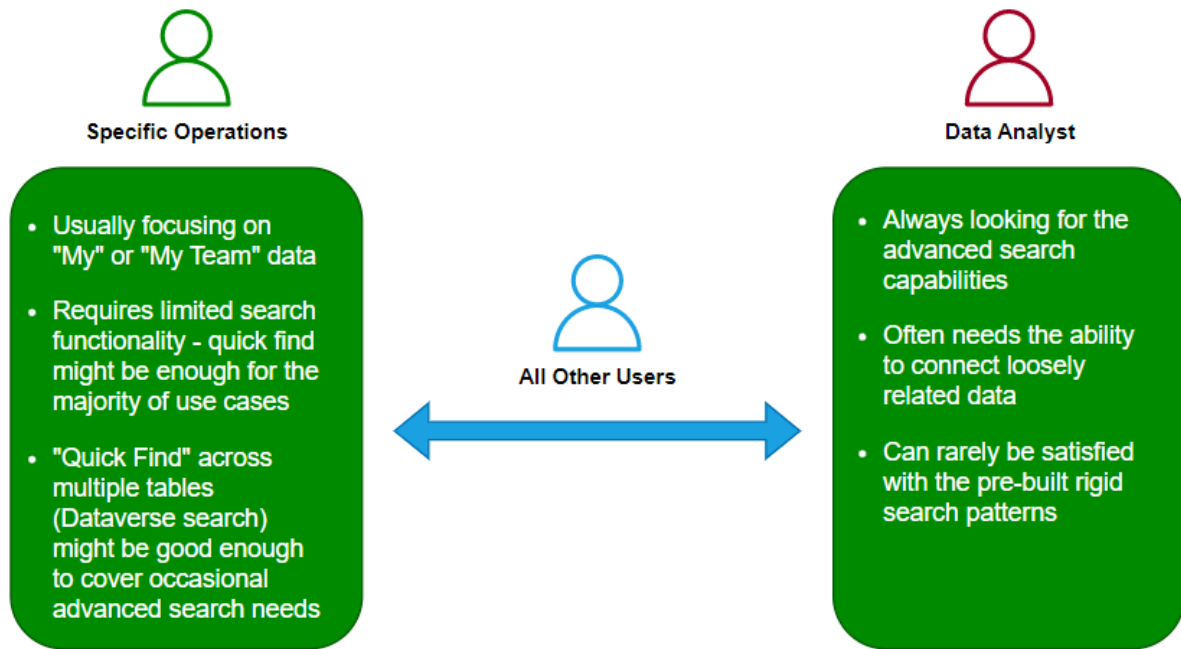
Can't be simpler than that, but this functionality is limited to the keyword search (well, in reality Dataverse search supports advanced syntax, but learning that syntax would likely require additional training to be delivered to the application users).

The issue here is that there is no "middle ground". It's either quick find or advanced filtering, the former is too simple and the latter can be perceived as overly complicated.

Sometimes, you may be able to come up with a targeted Canvas application / custom page to support specific search functionality. It may not be doable, though, since, if and when you have to build a query builder as part of that functionality, you may be looking at having to rebuild advanced filtering, just differently, and it may not be worth the effort.

You may even start thinking about implementing some sort of custom search functionality using javascript / React / etc - of course sky's the limit in that case, but this takes project complexity to a whole new level since you may need to figure out how to engage a professional developer, how to perform maintenance moving forward, and so on.

All those questions start showing up as you start encountering users learning to the right on the diagram below:



For those users who require very specific search patterns because of the well-defined process they need to follow in the day-to-day operations, coming up with the required views, filters, and even reports is, usually, not that complicated.

However, as you start talking to the users who don't, necessarily, know exactly what they are looking for, since they need to be able to find just about anything just about everywhere, you have to start thinking about how (and if) you can achieve that flexibility. Since, if you just keep moving assuming it'll be easy to do with Advanced Find / Filtering, you might find it a bit less (or, perhaps, a little too) advanced than it needs to be.

All that said, I've been mostly looking at it from the "query builder" perspective so far in this chapter. There is, also, another side of the search functionality, which is how search results are going to be presented.

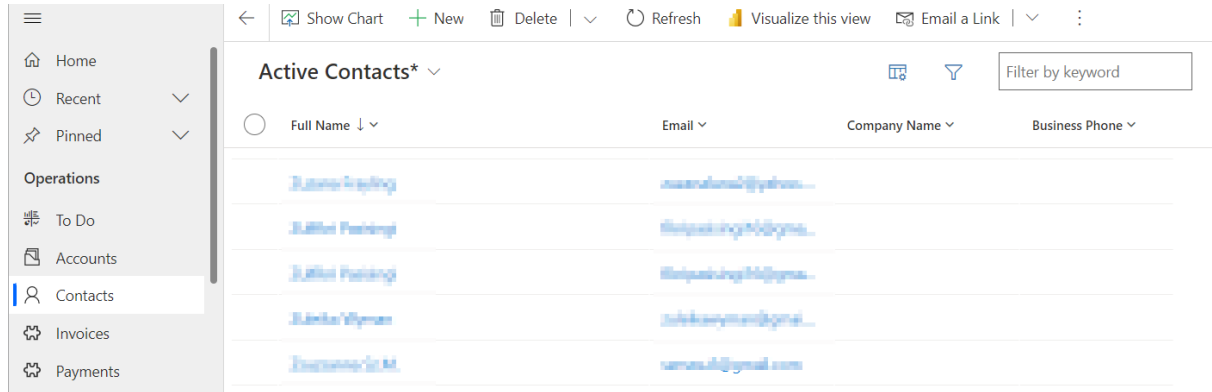
Of course, if you are working on a canvas application, you can present those search results the way you want. Assuming you have development resources for that.

In the model-driven app, at least out of the box, you are stuck with the views. They are great, but there are some limitations there:

- For the 1:N relationships, you can configure filters, but you can't add columns from the child records to the output. For example, imagine an account record, and assume there are multiple cases linked to it. You can search for an account, you can search for an account that has certain keyword in one of the child records, but you are only going to be able to display case columns in the output - you won't be able to bring in child records into the same view
- You have no control over column headers - they will display whatever has been configured as a display label for the related attribute. Every now and then, your users would ask you to rename a column, and you'd have to explain that, by renaming it in one place, you'd be affecting how that name shows up in all other places, too

Of course you might decide to create a PCF component to change view behaviour, but that would require additional time and resources, and, ultimately, that just screams complications and scope creep. Unless you look into it early, bring it up to the stakeholder, and mark it as a concern that requires attention sooner than later.

That said, search functionality often serves another purpose, which is reporting. Since, after all, if you need a report which shows you all active contacts in the system, you can easily do it by creating a view instead:



Still, let's talk about reporting-related questions separately below.

## 7.3. Reporting

Reporting is, often, an umbrella term. Different people will put different meaning into it, but, in general, it usually implies one or more of the following:

- Generating pre-built reports
- Utilising out of the box “dynamics reporting” features
- Document / pdf generation
- Performing data analysis on the data

From that standpoint, reporting discussions in the context of Power Platform are no different from that kind of discussions on any other platform or solution. However, there are some caveats:

- Since we are talking about Dataverse, which is not a database, but it's a whole service around it, so our ability to bring data into the reports can be significantly affected by the API limits, licensing, security, available query language, and, also, by the tools we have at our disposal.
- As far as the tools go, Canvas apps or Power Pages have no integrated reporting out of the box (well, they sort of have since we can add Power BI dashboards there, but it's a little different from how Model-Driven applications are integrated with SSRS). Model-Driven apps offer some neat features around SSRS integration, but that flavour of SSRS comes with a very special FetchXML datasource. Which, of course, has some limitations, too. Also, any view in the model-driven application can be thought of as a form of report as well, yet there is a advanced find / advanced filters, so MDA-s have a lot to offer
- Model-driven applications are also offering the ability to generate word documents and/or export data to Excel, which can be very useful in many scenarios. Except that this cannot be automated.
- Then we have Power BI. This is a great tool, the dashboards and paginated reports can be used independently from Power Apps or they can be embedded into the applications, but, of course, there are licensing questions
- Power BI is, also, the only tool we can really use to perform data analytics work on the data we have in Dataverse. We can do some of that in the model-driven applications, too, using advanced filters and built-in view filters, but it's not the same
- Finally, if we were to talk about automated document generation, we would probably have to look at the Power Automate and word templates there

There are other, third-party options, as well. The main problem with those is that our data may have to be loaded to the external servers, and, depending on the data residency requirements on your project, this may or may not work.

Most of the time, any solution which is not a personal productivity application would require some combination of pre-built reports, dynamic reporting, data analytics, and document generation. Given how diverse the technologies are and how they are, sometimes, not integrated that well from the user interface perspective, this one may easily catch us by surprise if we just assume reporting is easy to do.



For example:

- Model-driven applications views will only bring in distinct rows into the output
- There is no way to bring in rows from the “N” side of the 1:N relationship into the view
- We cannot group conditions from different related tables
- Navigation through those advanced filters assumes quite a bit of familiarity with the data model, and, also, some level of comfort when using the tool - that only comes with experience and it's almost impossible to push it to the new users right away
- Data analytics can be really challenging if it's done over the operational database because of the nature of those queries - they can be heavy, time consuming, et all. And, hence, they will almost inevitably start running into the Dataverse API and service protection limits. Most likely, you would need to set up a different database for this kind of reporting, and, subsequently, you would need a process for exporting data into that database
- Then there could be data model and security issues around reporting. You need to have required ta in the database to begin with before you can report on it. And, also, since operational data security might be slightly different from the reporting data security, that may be an additional consideration here, too.

Does it mean this is not doable? Not at all, but it's one of those areas which is often overlooked at the start of the project simply because the usual thinking would be “as soon as we have the data, we can report on it”. And, in case with Dataverse, it's kind of true, but there are all those caveats above.

Well, we don't necessarily have to get stuck on the reporting requirements, but, to save ourselves some troubles in the later stages of the project, it's usually better to start looking at them sooner to avoid potential firefighting later.

## 7.4. Translations

I live in Canada. It's a great country with lots of great people, but there is one peculiar aspect of doing software development here which can, sometimes, drive you nuts:

### Question:

What is the purpose of the *Official Languages Act*?

### Answer:

The purpose of the [Official Languages Act](#) is to ensure that federal government institutions can communicate and provide services in both English and French so that Canadian citizens can comfortably speak in the official language of their choice. The

(As per <https://www.clo-ocol.gc.ca/en/resources/frequently-asked-questions>)

This simple statement above essentially requires all publicly facing government software to be bi-lingual. And there is a push to apply the same expectations to the internal-only software, too. And, of course, if you are in the private sector, you might want to support two languages, too.

When it comes to the Power Platform, this means two things:


- You need the ability to support translations from the technical perspective (think emails, portal pages, labels, notifications, error messages, etc)
- And you may need the actual translator to do the translations for you if you are not fluent in the other language (as in, I have only a rudimentary knowledge of French)

In general, when developing Power Platform applications, whether those are Canvas, Model-Driven, or Portal applications, you have a certain level of translation support provided by the platform, at least from the technical perspective.

This way or the other (depending on the type of application you are developing), you can provide label translations for most of the metadata (as in, you can translate choice labels, you can translate display labels, you can translate section headers on the model-driven forms, etc).

However, when it comes to implementing bi-lingual data, things are, suddenly, getting more complicated. For example, look at the screenshot below:

## My Active Accounts ∨

	 Account Name <span>↑ ∨</span>	Main Phone <span>∨</span>
	English	

What if you wanted to display French version of the account name in that first column to those users who have configured “French” to be their application language? Here is what it would look like:

## Mes comptes actifs ∨

 Mod

	 Nom du compte <span>↑ ∨</span>	Téléphone pr... <span>∨</span>
	English	

In the model-driven applications, there is just no way of conditionally using certain columns in the query builder and/or in the view output depending on the user’s language selection.

Canvas apps, from that standpoint, have an advantage, since you have full control over the formulas used for querying the data and for presenting it. However, model-driven applications come with so many other features right out of the box - Canvas apps don’t stand a chance in comparison once the application becomes complex enough. More often than not, you’d be using Canvas to build very targeted applications for specific groups of users... or when you have no Dataverse behind your app, which is a completely different scenario anyways.

Then, of course, there is the question of public-facing applications, and this is where we are talking about Power Pages (former portals). They do have essentially the same problem as model-driven applications, though, since data translation is hardly possible there.

All that said, there are only a couple of ways to do data translations in the model-driven applications / power pages:

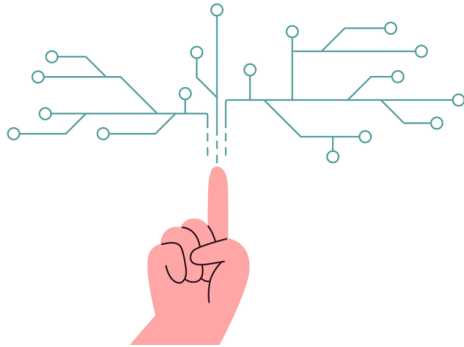
- We can ignore the problem altogether and just start concatenating labels in different languages (as in “English / Français”). This may work for two languages, but this depends, a lot, on what your stakeholders think of it. Also, simply because of how much space those labels are going to take, this will start affecting your application design and usability at some point, not to mention that this is not a scalable option (what if you need to support more than two languages?)

- We can, potentially, use Javascript to do “on-the-fly” substitutions in Power Pages. Which is going to complicate development quite a bit, and that does not work for model-driven
- We can, potentially, use plugins in the model-driven applications, which is going to complicate development quite a bit, and that does not work for Power Pages (because of the caching mechanism there)

Do you need to support different languages?

Think about it early, start figuring out if your stakeholders will be ready to compromise here and there, see if there are regulatory requirements which you absolutely have to cover. Don't wait until later, since some of those design decisions may affect both the data model and the development scope.

## 7.5. Deep customization and advanced business logic



You will rarely see an application with no business logic embedded into it. So you might be thinking right now “what’s the point of talking about business logic in the context of Power Platform specifically”?

Well, the reality is, Power Platform has multiple layers, and all of them can be customized to some extent. Depending on what business logic you want to build into your application, you may have to work with one of more of those layers.

To begin with, we can think of the client and server sides first. Well, when working with Power Platform there is no, really, “server side” - it may be more appropriate to say “cloud side”, but still.

On the client side, you can start simple, and, from there, you can go as complex as you wish:

- In the model-driven applications, you can use business rules, javascripts (including PCF components), even Power FX
- In the Canvas applications, you have quite a bit more control over the user interface, so you can embed business logic there, but, in reality, there seem to be a limit to how far we want to take Canvas apps in that sense before they become difficult to maintain
- In Power Pages, you can use Javascript or Liquid to perform client-side customizations, so that can be a very powerful combination, too

However, this kind of customizations, even though they are very useful from the user interface perspective, usually cannot enforce business rules across the whole set of applications working with your data. Since, after all, if you decided to add Javascript to a model-driven application form to do some calculations, you may have to implement similar logic in each and every other application working with the same data. Over time, this is not going to be sustainable.

Therefore, client-side customizations are, mostly, user-interface customizations, and they can rarely cover the business logic - at most, they can make it easier for the users to follow certain business processes, which may be equally important.

This is not to say you can’t take client-side customizations to the extreme and implement a completely custom user-interface within a model-driven application, for example. You definitely can, I’ve seen it done, and, even though I’m always getting impressed when I see

those examples, I can't help but wonder if, perhaps, it would be easier to go with a different platform / architecture instead.

The main goal of Power Platform is to facilitate development by using low-code tools, and to only use classic development tools and frameworks sporadically when it's absolutely a must (which is where Powerapps Component Framework, or PCF, comes in by the way).

Then there are server-side customizations. We are still talking about Dataverse-based applications, so there are a few options here:

- Power Automate flows. This is a "go-to" solution for when you need to perform server-side calculations, integrations, etc. However, there is no such thing as "synchronous" Power Automate flows. In other words, if you need to validate some value whenever a record is being updated, or if you need to perform some calculations right at that moment, Power Automate is not going to fit such requirements. Also, and this might be a bit of a personal preference, but I feel that the more complex your flows become, the more you wish you had implemented them using pro-code (that would be plugins/ Azure Functions, etc)
- Classic workflows. That's another low-code option, but, for better or worse, it's not the recommended one. Except that it's the only low-code choice you have when dealing with the "real-time" server side execution
- Plugins. This is, likely, the most flexible option, though there are some limitations, too. Plugins can only run in response to some event (such as a record update event), they have a 2 minutes execution limit, and, of course, they do require a pro-developer to create and maintain them
- And there is a myriad of options in between which can also be extended with Azure Functions, Logic Apps, Service Bus, Web Services, existing API-s, custom connectors, etc

Point being, though, every project will inevitably have some business rules to implement. For the simple rules which, perhaps, do not require strong enforcement and can be "a-synchronous", you may be able to stick to the low-code customizations and tools.

As the rules become more complex, though, you will need to engage pro-developers on the project, and this will start affecting the budget, duration, testing, maintenance, even the perception of the project from the stakeholders perspective.

However, and this is where it starts getting complicated, Microsoft is doing what it can to promote low-code development, it's not that unusual for the project owners to, sometimes, insist on not using pro-code at all, and, when this happens, we can find ourselves staring at those business requirements and thinking that there is no way to implement them under such circumstances.

You don't want to go halfway through the project to find yourself in that situation - instead, look at the requirements, see where they do require this kind of deep customization, and mark those as risks to the project stakeholders. It's up to them to decide whether they should allow such customizations, or whether they would rather prefer to change the requirements, but the sooner those expectations are set, the better it will normally work out.

The table below may help to compare those options a little better:

	Supported Complexity	UI	Server-side	Real-time (Sync)	A-Sync	Canvas	Model-driven	Power Pages
<b>Business rules</b>	Low	x	x	x		x	x	x
<b>Plugins</b>	High		x	x	x	x	x	x
<b>Javascript</b>	High	x		x	x	x	x	x
<b>Power FX</b>	Medium	x		x		x		
<b>Power Automate</b>	High		x		x	x	x	x
<b>Classic workflows</b>	Medium		x	x	x	x	x	x

It should be more or less self-explanatory except, perhaps, for the “supportedAb complexity” column. Well, if you compare Classic Workflows to Power Automate, for instance, you’ll find that Power Automate is quite a bit more sophisticated tool compared to the classic workflows, and, hence, more complex customizations are doable there. If not for the “real-time” support, or the lack of it, there would be no reason to even mention classic workflows.

## 7.6. Data migration and data integration

Why did I have to mention Data Migration here? Everyone knows data migration can be complicated, it can easily go wrong, it may lead to data inconsistencies, yet the effort required to do it is almost always underestimated when the project starts.

There is no need to discuss, let's skip this chapter and go to the next one.

Although, wait, there is something I wanted to mention.

Here is what differentiates Dataverse from your regular database:

- Dataverse is not a database, it's a service
- As a service, it comes with all the bells and whistles, including some that help and some that make it more complicated

For example, we can use SQL to access Dataverse over the TDS endpoint, which is great. However, that is not a "real" SQL connection - it's an emulated one. Specifically, it is still subject to the service protection API limits, there are memory limits, and the version of SQL you work with is somewhat different from the TRANSACT-SQL

(<https://learn.microsoft.com/en-us/power-apps/developer/data-platform/how-dataverse-sql-differs-from-transact-sql?tabs=supported>). Also, this SQL connection only allows "read" operation.

Those limits, even though you may not be thinking of them that way when starting the project, can become relatively restrictive once you start looking at the large volume of data, which is exactly what tends to happen with the data migration.

Below is an excerpt from the service protection limits

(<https://learn.microsoft.com/en-us/power-apps/developer/data-platform/api-limits?tabs=sdk>):

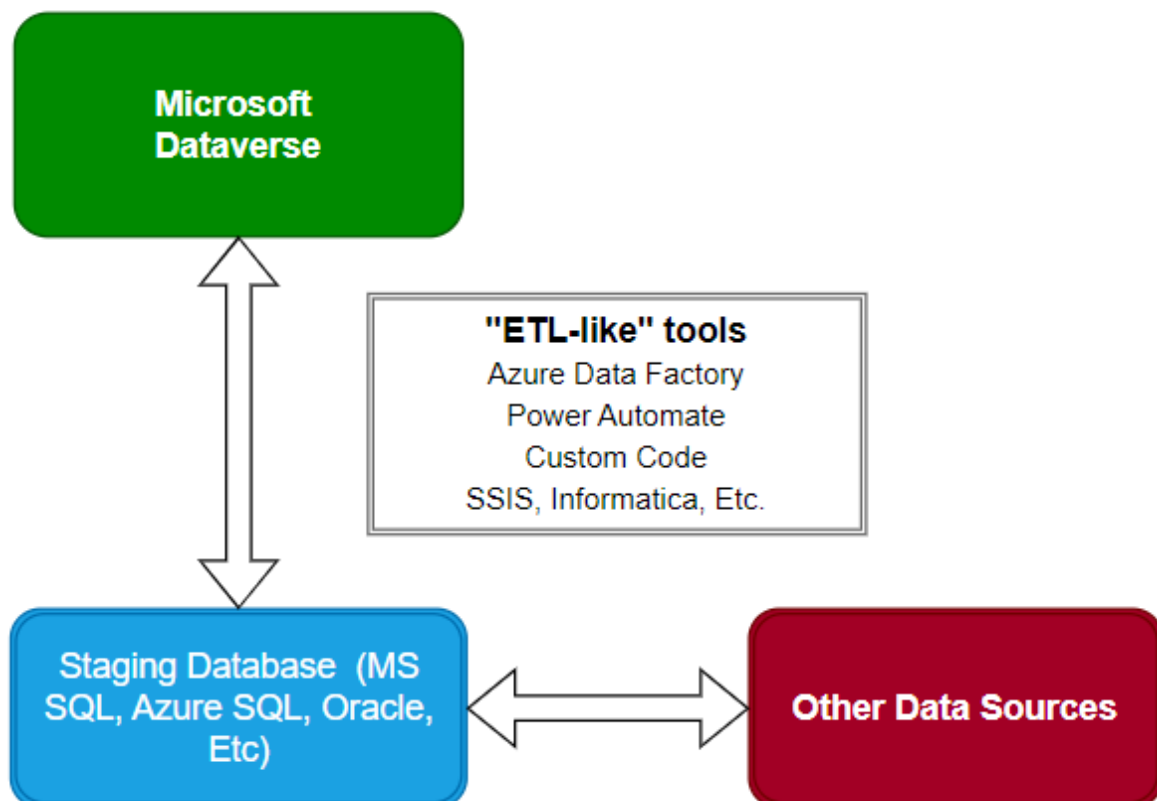
Measure	Description	Limit per web server
Number of requests	The cumulative number of requests made by the user.	6000 within the 5 minute sliding window
Execution time	The combined execution time of all requests made by the user.	20 minutes (1200 seconds) within the 5 minute sliding window
Number of concurrent requests	The number of concurrent requests made by the user	52 or higher

6000 requests per 5 minutes may look like a lot, but, ultimately, if you go about your data migration the usual way (which is treating Dataverse in the same way you'd be treating a SQL database by running all the read / update / delete operations right there), you may hit that limit sooner than expected. The problem is, of course, once and if that happens, you will be limited in terms of the workarounds you may be able to apply, since, ultimately, a limit is a limit.



And it's not the only limit. You will also need to account for the API limits imposed by your licensing - of course, you may be able to use a service account to tap into the tenant API allotment, but, even so, you may need that somewhere else, too.

This is why, unless you only need to migrate a small amount of data, you may need to look into setting up a framework/infrastructure just for the purpose of the data migration. It often works best when you are able to offload most of the data-related processing to a different database:



For example, if you used the approach depicted on the diagram above, you'd have a dedicated staging database where you'd be able to run SQL queries directly into the database without having to worry that much about the API limits and/or SQL dialect performance/compatibility. It may look like a small thing, but, in the long term, it pays off more often than not.

That said, do you have required cloud resources / subscriptions / connections to the on-prem databases if that's what you are going to use, etc? Every now and then, you'd find yourself working in the environment where all those infrastructure-related questions have already been answered; however, again more often than not, you'd actually have to deal with them as you go, and, since a lot of that would be about procuring additional resources and/or dealing with the security concerns, this would take time to do. You don't want to push this off until the very last phase of the project.



## 7.7. Resourcing

I may be biased, but I think the concept of citizen development is, often, oversold. Remember that quote I provided above? Let me repeat it here:

*Low-code development can facilitate your company's digital transformation. Instead of relying heavily on programming, low-code platforms simplify application development with techniques like drag-and-drop functionality and visual guidance. This means that anyone in your company, regardless of their technical expertise or abilities, can build apps so that the business can offload some tasks from IT.*

To some extent, Power Platform is able to achieve that. However, I can't help but think of a lot of potential disclaimers to that statement above, and, in the end, it all comes down to the project planning from the resourcing perspective.

Yes, business users can easily create new Canvas apps and/or cloud flows with Power Platform. As long as other users are not relying on those applications and flows, there are no issues at all.

However, as soon as that application or flow gets shared with the other users, a lot of additional questions start showing up around future development, maintenance, support, security, data integration, data migration, business rules enforcement, etc.

It would not be realistic to expect a business user who has no software development experience to methodically answer all those questions, document the answers, and implement the solutions appropriately.

Of course this is where the concept of "fusion teams" shows up. Now there is a business user who can help prototype the app, and there are pro developers / architects who can ensure the whole development lifecycle is properly implemented.

However, with the fusion teams, we are essentially going back to the classic development process. Of course the idea is that pro-developers would only be involved "as needed", but, in reality, there is still a lot of reliance on the professional developers even when and if they are using citizen development tools. Have a look at the table below, and, perhaps, you'll see why:

	Business Users	Citizen Developers	Pro Developers
<b>Power FX</b>	Low	High	High
<b>Power Automate</b>	Low	High	High

<b>Canvas Apps</b>	Low	High	High
<b>Model-Driven Apps</b>	Low	High	High
<b>Application Lifecycle Management</b>	Low	Medium	High
<b>Support process / availability</b>	Low	Medium	High
<b>Pro-code</b>	None	Low	High
<b>Other (documentation , Scrum, etc)</b>	None/Low	Low/Medium	High
<b>Power Pages, Liquid, Javascript</b>	Low	Medium	High

This table, even though it's an approximation and you might argue it could be done a little differently, shows the level of familiarity / comfort different groups of users / developers would have towards the tools / methodologies / techniques.

And, apparently, Business Users would normally have "low" familiarity, Citizen Developers would have it somewhere between Low and High, and Pro Developers, for the most part, will have it at "high" (in comparison).

The more complex and the more mission-critical your project is, the more mature you need your team to be. For example, you can't have low-quality Power FX code even if you have a pro-developer covering application lifecycle management, so that means you need at least a Citizen Developer, not a business user, to take care of the Power FX formulas.

Take that logic to the extreme, and you'll end up with the same pro-developers team composition as on any other software development project. Yes, you may be able to engage business users a bit more, but, ultimately, it's going to be way less than you might have expected. Besides, business users have their own responsibilities, and, even if they are willing to take on some of the application development, it's not their main concern at all.

Which usually means you still need to find those developers, and, actually, you usually need to find developers familiar with the Power Platform. Which means they need to be familiar with Canvas apps, Power FX, Power Automate, Model-Driven apps, Dataverse, security model in Dataverse, etc.

It's interesting how Power Platform projects are, often, lacking resources simply because the benefits of citizen development can easily be exaggerated. It all depends, though, so your project may still benefit greatly from the ability of non-professional developers to contribute;

however, it does not look, at least not yet, as if pro-developers and/or solutions architects could just be let go.

## 7.8. User Interface

There are two somewhat contradicting aspects of the user interface in general, and this only gets exaggerated in the Power Platform world:

- User interface does not define the requirements
- Every user group wants application user group to be geared towards their specific needs

The first item there is sort of obvious, but there is, also, an always assumed requirement that, once the application is implemented, it will be as easy to use as possible. That undermines the whole point of the user interface being somewhat less important, and it does feed into the second item above. We can have all the functionality in the world, but, if the application feels awkward, if the users can't find their way through the application, they are not going to like it.

Which means the adoption rate will drop, we will get negative feedback, and, ultimately, the whole project may fail.

With the Power Platform, we have at least these 3 types of user interface:

- Canvas applications
- Model-driven applications
- Power Pages

Plus it may be worth mentioning Power BI and Virtual Agents, but, for now, let's leave them alone.

Here is another table:

	Canvas	Model-Driven	Power Pages
<b>Pre-built UI</b>	Low	High	High
<b>Customizable UI</b>	High	Medium	Medium
<b>Citizen development on the UI side</b>	High	Medium	Low
<b>Pro-development on the UI side</b>	Low	Medium	High
<b>UI from scratch availability</b>	High	Low	Medium

The first four lines there are just setting the stage. With the Canvas apps, we tend to build our UI interfaces if not from scratch, then, at least, from the ground up. With the MDA applications, there is a UI framework we would better stick to, since having that pre-built UI is one of the benefits of the MDA. And Power Pages fit somewhere in between - there is a lot of pre-built UI, but there is, also, a lot of UI customization opportunities (which are geared towards pro-dev, though).

Quite often, we would need to deal with a relatively advanced data model and a lot of functionality around it, and, while doing the initial assessment, we would decide to go with a model-driven application. Which is absolutely reasonable, since model-driven applications can be set up quickly, they offer a lot of data management functionality, we can add business rules in no time, there is advanced filtering and search. Almost everything is covered more or less out of the box once the data model is in.

It's the way it is covered, that's where the problem starts.

We can't:

- Significantly alter the way application navigation works in the MDA. This includes left-side navigation links, command bars, application selector, etc.
- Modify a lot of UI-related component properties. As in, we cannot modify text color or weight for an individual label or input box. We just don't work at that level of UI granularity when developing a model-driven application.
- Customize the way business-process errors are displayed
- Modify the behaviour of the search / advanced find

It matters quite a bit more often than it probably should. Which makes sense, since different users have different personal preferences, and, when UI personalization is quite limited, it should not be that surprising to see some of them expressing their concern. It still surprises me every time, mind you, but that's exactly why we have to deal with this early.

There are some ways to customize user interface in the model-driven applications, of course. We can start developing custom components, for example, but that's, definitely, a pro-dev activity, and, also, the more we do that, the less beneficial MDA application becomes for our project. Ultimately, if we were to develop the user interface from scratch, why would we go with a model-driven app to begin with.

What about the Canvas applications, though?

The way I see it, Canvas applications are great when it comes to using various data sources, and they do offer a lot of data-integrated components. We also have very granular control over the UI properties of those components. However, that kind of flexibility comes with the need to spend a lot more time on the UI development, and that means at least some of the following (plus, probably, some more):

- Creating standard themes and component libraries
- Making sure your app is accessible
- Implementing translations

- Making it responsive to support different screen sizes

Yet there is no advanced search or filtering in the canvas apps, there is no standard navigation, etc. Those apps are called Canvas for a reason - you have a clean canvas to start drawing on it. That works great for very targeted applications, but it tends to become a lot more complicated as more and more functionality is added.

You could almost argue Canvas apps development is not that scalable yet, since some of the standard software development concepts are missing. Even the named formulas are still experimental at the time of writing this chapter.

Although, the other way to look at it is to say that we now have named formulas, so it depends. I guess the main problem with experimental features is that you can easily use them in the personal productivity applications, but it would be a no-go for all those more widely-shared applications where you need to ensure you don't end up with tens or hundreds of users being unable to do their work in the app once and if that experimental feature is gone:

The screenshot shows a settings interface. On the left is a sidebar with menu items: 'Settings', 'General', 'Display', 'Upcoming features', 'Support', and 'Git version control'. The 'Upcoming features' section is active. It has a search bar containing 'formula' and a close button. Below the search bar are tabs for 'Preview', 'Experimental' (which is selected), and 'Retired'. A warning message states: 'These features might change, break, or disappear at any time.' Underneath, there is a section titled 'Named formulas' with a description: 'Use named formulas instead of global variables and collections for faster app load time and logic that is easier to understand and maintain.' At the bottom of this section is a toggle switch that is currently turned off, labeled 'Off'.

As for the Power Pages, typically they wouldn't represent your "main" application. In other words, I believe the main purpose of Power Pages is to give external users some access to your data. You would still need the data model to be set up in Dataverse, you would likely need to set up your views and forms there, and you would likely need to develop some model-driven app to let your "internal" users work with the data.

You can, then, use Power Pages to let other users in your organization access that data from a very specific perspective, or, perhaps, you can open your Power Pages to the public. In either case, Power Pages are not going to replace your model-driven and/or Canvas applications - instead, they are going to be yet another application you'd be developing for a different auditory. And, so, you would have to look at the additional set of user interface challenges, which is a bit of a combination of what you have in the MDA and in Canvas.



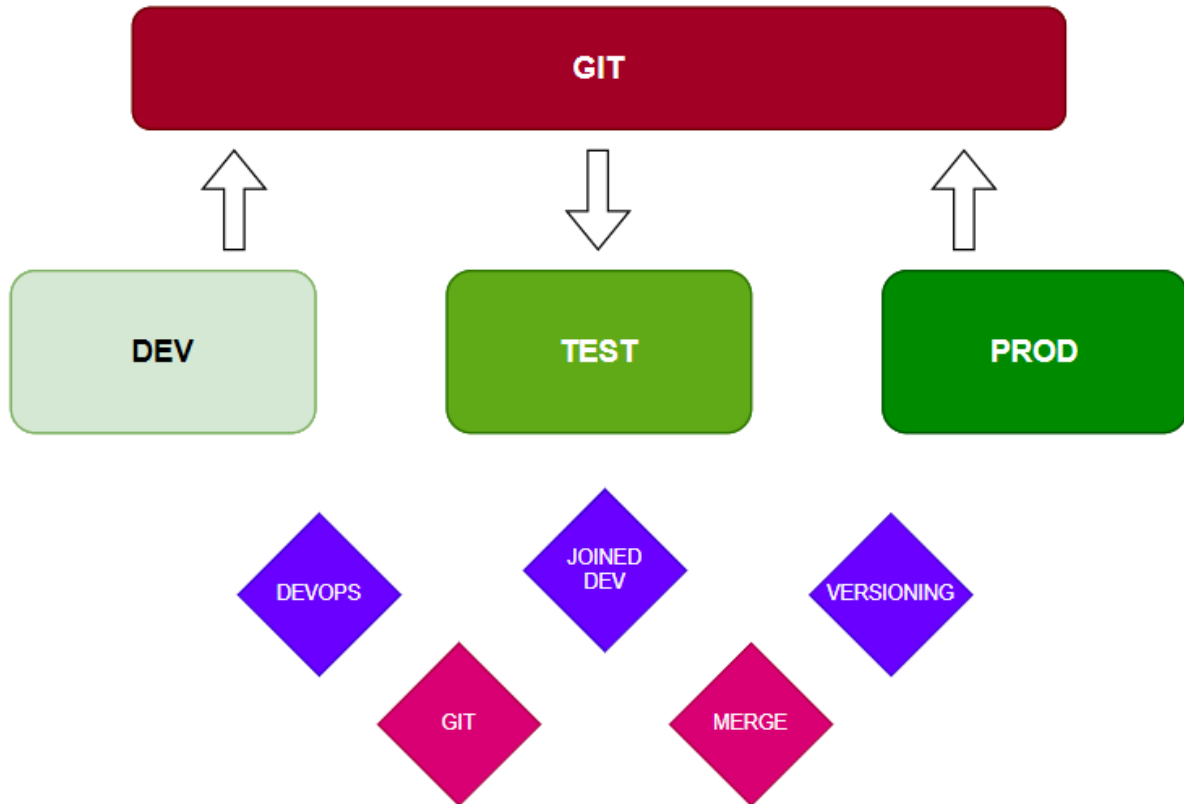
Per-built forms will have a somewhat rigid look and feel, but you can customize with Liquid and Javascript. That would require pro-devs to be involved, though, so, as soon as you start walking down that path, the benefits of low-code development might start diminishing, too.

It seems important to consider potential user interface and related user experience issues early to plan around some obvious gaps. UI customization tasks will often be time-consuming, they may land on the pro-code side of the team, and some of them may even be impossible to implement (especially in the model-driven applications). It's a potential issue only, but, as it is the case with all other potential issues, you'd probably want to uncover them earlier than later to start thinking of the possible workarounds, and, perhaps, to start educating your users on how to deal with them without having to invest too much into heavy development.

Because, after all, if you can't provide that notorious "one click" experience and it turns into ten clicks, this is going to be a problem.

## 7.9. Application Lifecycle Management

Ok, what can possibly go wrong here:



It depends on who you ask.

To begin with, there is a common misconception, which is that ALM for Power Platform is identical to the ALM for a regular software development project. It's not, and it's for a few reasons, here are some of those:

- It takes extra effort to set up an individual environment for each developer. More likely than not, your project is not going to be an exemplary exception, and you will have more than one developer working in the same environment. Don't get me wrong, we are all in the same boat, it just rarely goes the other way. Which actually means all the changes done in that environment will be difficult to separate from each other, and you will likely have to keep deploying all of them together. There are ways to mitigate that, to some extent, but it's almost impossible to get around it completely. So, then, if all the changes you make in dev have to be deployed together to test, that's great, but... how do you actually deploy hotfixes, how do you ensure granular testing, etc? There is a bit of a problem there.
- But let's say you have multiple environments, as many as one development environment per developer. Then you have another problem, since there is no easy way of merging changes from all of those environments into the test environment. It's one thing to merge C# or Javascript code, but it's quite different from having to join XML or json files. Besides, there is no Visual Studio, it's not as if you can push a

build button, get the code compiled and deployed... it's going to be quite a bit more involved.

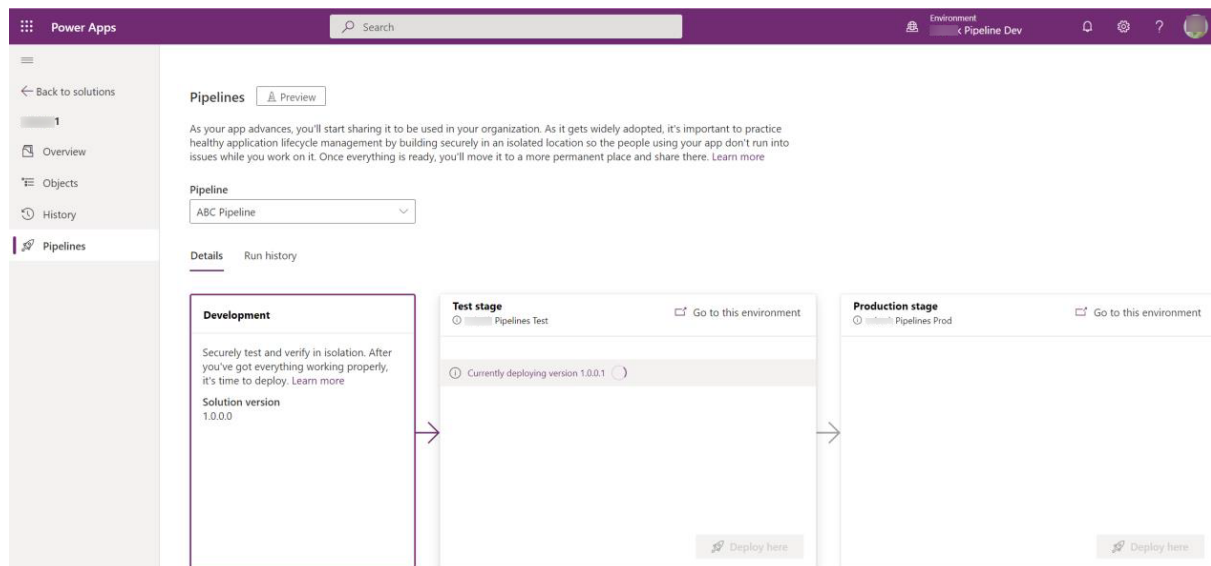
- How about the versioning? Technically, you can reset your environments - just hit that "reset" button in the PowerPlatform admin portal, wait for it, deploy the solution, get the reference data loaded.

It's all doable, but it's quite a few steps, so it's not as simple as just getting the files copied somewhere. You may need to set this up properly, you may need to allocate time for this, and you may need to start by making sure you have pro-dev resources who can do it all.

These are all the reasons property ALM is, often, ignored. Which, of course, saves some time, makes it easier to assemble the team, but, ultimately, it creates a problem which starts manifesting itself in the worst possible moment. Which is when the project gets closer to the finish line, and it does need to follow a predictable deployment / testing pattern which does not introduce regression errors on each and every deployment.

Microsoft has been working improving ALM experience, there have been interesting developments such Power Platform Pipelines:

<https://learn.microsoft.com/en-us/power-platform/alm/pipelines>



However, there is always something... kind of a little thing that makes it different from what you would expect. With the Power Platform Pipelines, at least as of now, there is no git integration:

## Can pipelines be used with Azure DevOps, GitHub, or the ALM Accelerator?

Not currently. We aim for these to work together more seamlessly in the future.

All things considered, you will likely have to compromise here and there when setting up ALM on your project. Will you sacrifice git integration to benefit from the simplicity of those Power Platform Pipelines? Will you set up multiple development environments, thus eliminating even the possibility of using Power Platform Pipelines? Will you use manual process for deploying your reference data, will you use a configuration migration tool, SSIS, ADF, Power Automate, or, perhaps, the dataflows?

There is more than one way to skin this cat, but, just like with anything else in this book, it would be better to find the time and resources to at least have it planned out before your project gets to the point where it does need some form of ALM to be implemented.

## 7.10. Infrastructure, Tools, Resources

Here is something that is often a “second thought”. When we hear “Power Platform project”, normally we start thinking about the usual Power Platform components:

- Power Apps
- Power BI
- Virtual Agents
- Power Pages
- Perhaps first-party applications

However, that list tends to ignore the fact that Power Platform lives in the Microsoft cloud ecosystem, and, so, with the exception of relatively small and contained projects, we can’t, really, ignore some of the related questions:

- How are we going to implement data integrations? Are we going to need Azure Data Factory? Will we have to secure Azure Data Lake resources? Do we need Sharepoint? Do we need Azure Functions to run .NET code? Perhaps Azure Service Bus would be required?
- Then there is a question of on-prem connectivity. Do we need it? Is there some data we still have on-premise that will need to be integrated into our Power Apps? Are we going to use virtual tables to surface that data, are we going to expose API, do we need to open ports on the firewalls, do we need to create API keys/secrets, etc.
- With all the questions above, there is a related question which often proves to be the hardest of all, especially when data protection and/or security in general has to be taken seriously. How do we get all that done in those cases where we have no required permissions, and where we need assistance from the tenant / on-prem administrators? They will have all the policies in place, and, out of a sudden, here comes a Power Platform project which requires some additional traffic routes to be configured, maybe some user accounts to be shared, etc. Not to mention that there are some features of Power Platform which require tenant admins to get directly involved, such as M365 audit logs, email approvals in Dataverse, etc.

This often leads to the situation where we know there is a better option, but, since we can’t get access to the required resources/tools/permissions, we still have to compromise, and, in the long term, those compromises always lead to more problems down the road.

For example, instead of using Azure Data Factory, we may have to use Power Automate for data migrations. Technically, it’s doable, but it’s much less efficient, it’s less reliable, and, ultimately, it may easily become a project risk from the implementation and maintenance perspective.

Then again, what if we were allowed to use Azure Data Factory in the example above? That would be great, but we would still need to add ADF expertise to the team, which, in many cases, may also be a problem.

How do you balance all those expectations, hidden resource requirements, yet how do you ensure they are all met not only in the short term, but, also, for the whole life of your project? After all, you need to think of it from the maintenance perspective, too.

From the organisation perspective, it usually makes a lot of sense to standardise on the processes, tools, and technologies, since that's how each organisation can keep operating in the predictable manner, including from the human resources perspective. However, Power Platform often comes with some additional expectations, so those practices, tools, and technologies need to be adjusted, and those organisations need to adapt.

It does not necessarily happen right away, and, while it's happening, we may keep compromising, creating "development debt" which will have to be dealt with, etc. It's not that we can't keep moving forward in such cases, but we need to keep track of that debt, and, since it is definitely going to affect our plans, we need to keep maintaining some level of awareness between the stakeholders so that it does not end up in a surprising realisation at some point later.

## 7.11. Licensing

Power Platform licensing is a topic on its own to be honest. There is no better way to confirm licensing requirements than to talk to a Microsoft representative, and there is absolutely no way I could cover licensing in a small chapter, but I still wanted to emphasise the importance of this topic.

There is no reason at all to start using Power Platform if you are unable to secure required licences. And there is definitely a cost associated with them, so there are all the usual questions of what you get in return which you may have to answer when and if those questions are asked.

Normally, you will probably be focusing on the PowerApps and PowerAutomate licensing, and, also, on the Dynamics 365 licensing.

To start with, use the link below:

<https://learn.microsoft.com/en-us/power-platform/admin/pricing-billing-skus>

Once you are there, make sure to download the licensing guides - they are very detailed and, for the most part, will answer all of your licensing-related questions.

Still, there are a few key points to keep in mind:

- M365 licences come with some PowerApps/PowerAutomate allowance, though it's limited
- PowerApps licences is what you need to unlock the full potential of Power Platform
- D365 licences are a bit special in the sense that they give your users access to the first-party applications AND allow them to create and utilise custom Power Apps in the context of those first-party applications. This may sound clear as mud, but, conceptually, the idea is quite simple: if your users had a Customer Service licence, they should be able to extend Customer Service capabilities with custom applications. But they should not be using those licences to create completely independent applications. Which might be difficult for Microsoft to enforce, of course, but it's not that relevant
- In the reverse scenario, a Power App-licensed user will usually be able to read data from the D365 restricted tables (but won't be able to write into those, unless properly licensed for D365)
- Power BI, Virtual Agents, and Power Pages have their own licensing models. Well, technically Power Pages usage is included into the Power Apps licences, but it only makes sense for the internal users, and, more likely than not, you'll be creating Power Pages portals for external users
- There are, also, pay as you go licences. Typically, they cost about twice as much as the regular ones, at least before volume discounts but you can use them to licence users "on demand"
- There can be volume and other discounts - it's not for 1 or 2 licences, of course, but, if you are looking to purchase hundreds or thousands, it would be a good idea to find out more about the volume discounts

And what about the actual costs? Keeping in mind how the licences can be mixed, how volume discounts can be applied, how pay-as-you go can complicate those calculations even more... it's, probably, hard to actually predict the costs. However, in general, the numbers can start adding up - Power Platform is not a free tool.

Also, there can be additional licensing/ subscription requirements around all the other tools involved. Azure Data Factory, Data Lake Storage, Sharepoint, etc - there can be associated costs in all those areas.

Hence, watch out for those situations where your stakeholders may start making assumptions about the availability of the cloud resources. For example, sometimes a business group within the organisation would not recognize that there is an expectation they'd be paying for their own licensing within that organisation, since they might be more used to the traditional IT model where they may need to "pay" for development, but not for the ongoing licensing.

As usual, it's almost always better to set correct expectations sooner than later, and it's no different with the licensing.



## 8. Feeling overwhelmed? Here is what may help



I believe there is, often, a disconnect between what is expected of the Power Platform and what it can actually deliver.

It's one thing to work in the environment where all the tools are, already, in place, where all required Azure subscriptions have already been set up, where there are people who can set up devops pipelines for your project, and, in general, where Power Platform has already taken roots, has been well adopted, and, in that

sense, you only need to deal with the specific project requirements.

It's a completely different story when some or all of that has to be set up, configured, enabled, unblocked, etc.

The main reason I often start feeling overwhelmed is not that Power Platform may be lacking some capabilities - it's almost always all about the wrong expectations. As in, someone would say "Canvas apps can be done by the business users", and you'd be left to deal with no ALM at all, since this is the last thing business users would be thinking about.

The only piece of advice I can suggest here is to remember that "It's not your fault".

It's quite possible that you are feeling overwhelmed because you have to deal with the unrealistic expectations, so, well, there is not a lot you can do other than to start adjusting those expectations. Start educating the stakeholders, perhaps start small, maybe even let them read this book at some point. Yes, there is, actually, a chance some of them will say that Power Platform is not the right choice for that particular organisation, and, perhaps, that's the case?

However, don't just give up easily either. Show off Power Platform capabilities, share the success stories (<https://powerapps.microsoft.com/en-ca/customer-stories/>), just make sure not to take it too personally. Yes, the level of effort required to implement Power Platform tends to be underestimated most of the time. And no, it's not, necessarily, what you are personally responsible for, so, then, don't take it too personally.

## 9. Conclusion

The purpose of Power Platform is to democratise software development, to let anyone create their own applications, to make this process faster and easier. This is a great and honourable goal, and Microsoft has done a lot to ensure Power Platform gets as close to that goal as possible.

I think there is an element of perfectionism there, and, once confronted by reality, this perfect picture becomes a little more complicated.

There are resourcing, infrastructure, functionality, security, and all sorts of other issues which can get in your way when you are working on a Power Platform project, and those might not be easy to tackle.

However, now that you know they are there (or, perhaps, now that you have confirmed your own suspicions), you may be able to navigate around them a little better, and, so, your next project may end up being even more successful than the previous one. Which will help all of us in the Power Platform community, since, ultimately, every success story matters.

After all, those success stories are what we need for the Power Platform to live long and prosper!

*Best of luck,  
Alex Shlega*

*PS. Don't forget to connect: <https://www.linkedin.com/in/alexandershlega/>*

## Appendix A - What's not included and will be added later

- Center of Excellence - why it matters
- Learning Curve
- Citizen Developers attrition issue