Low-code vs traditional code

Application of low-code solution in ERP systems

Viking Säfsten
Abstract

Low-code vs traditional code

Viking Säfsten

Low-code is an emerging method for software development that promises faster and cheaper development by allowing non-developers to develop software using graphical interface tools. In this thesis, to evaluate low-codes claim of cheaper and quicker development I will create a solution in low-code and then compare it to an existing solution created using traditional code. For this comparison I will use Power Platform as the low-code solution. Power Platform is Microsoft's suite of low-code solutions focusing on business processes and automation. Microsoft also provides an enterprise resource planning system that allows for add-ons created using traditional code. The original use case using only low-code was found to lack the functionality to duplicate the add-on. When extending the use case to require some coding the add-on could be duplicated. My analysis of the comparison between the traditional and low-code solutions found that development time is shorter for low-code but maintenance and updates costs is lower for traditional code. The limited UI further hinders a low-code solution. Power Platform was found to be not suitable to replace the selected add-on.
Contents

1 Introduction .......................... 4
   1.1 Problem formulation ................. 4

2 Background ........................ 4
   2.1 Low-code ................................ 5
   2.1.1 Testing in low-code ............... 5
   2.2 Microsoft .......................... 6
      2.2.1 Connectors ......................... 6
      2.2.2 Dataverse ......................... 7
      2.2.3 Power Apps ....................... 7
      2.2.4 Power Automate ................... 7
      2.2.5 Power BI .......................... 8
      2.2.6 Testing in Power Platform ...... 9
      2.2.7 Learning the Power Platform .... 9
      2.2.8 Licensing ....................... 9
   2.3 Related work ....................... 10

3 Comparison ........................ 11
   3.1 Existing system ...................... 11
      3.1.1 The AL language .................. 11
      3.1.2 eCom Documents .................... 11
   3.2 Previously know limitations ........ 12
      3.2.1 Limitations of scope ............... 12
      3.2.2 Limitations of extensions created in Power Platform .... 12
      3.2.3 The Business Central connector .... 12
   3.3 System Requirements ................. 12
   3.4 Implementation ...................... 15
      3.4.1 Data storage ....................... 15
      3.4.2 Logic ............................. 15
      3.4.3 Sending logic ...................... 19
      3.4.4 UI ............................... 20
      3.4.5 Implementation summary ........... 22
   3.5 Licensing .......................... 22

4 Discussion ........................ 22
   4.1 Conclusions ........................ 22
   4.2 Lessons learned ..................... 23
   4.3 Limitations ........................ 24
   4.4 Future work ......................... 24

References .......................... 26
1 Introduction

This project strives to test the usability of a specific low-code development platform (LCDP) family in a real world use case. The project is entirely based in Microsoft’s different tools, trying to test if the interoperability between services promised by Microsoft is usable for a specific but common use case.

Microsoft offers an enterprise resource planning (ERP) system for small to medium sized businesses under the name Dynamics 365 Business Central (BC) [1]. BC is a modular system consisting of several services such as sales, inventory and relationship management. In addition to the provided services BC has support for custom extensions [2] that can add functionality, change behaviors or integrate BC with other services. These extensions are written in Microsoft’s programming language AL [3].

The Microsoft partner InBiz has developed an extension for BC called eCom Documents. This extension is an automated system for sending out invoices and other documents generated by BC.

Microsoft also provides a suite of low-code development platforms called Power Platform [4] which has the ability to perform the create, read, update and delete (CRUD) operations on BC’s underlying databases as well as trigger BC functions. It should therefore be possible to replicate the behavior of BC extensions written in AL with Power Platform’s low-code solutions.

Low-code development strives to be easy and accessible, allowing non-developers to produce software that solves their own needs. If the design goal of allowing non-developers to produce software has been achieved, without compromising the functionality of the end product, applications could be developed at a lower cost. Microsoft’s Power Platform is promoted as being suitable for business processes but its usability for process automation in BC is not as developed. The abilities of Power Platform is therefore not fully explored and its suitability for usage in relationship to BC should be evaluated.

In this report I will cover the eCom Document extension written in AL and my work implementing an equivalent system within the Power Platform. The development of the low-code system should, as much as possible, use the graphical interface of the Power Platform instead of coding. This in order to facilitate analysis of the Power Platform’s capabilities in the hands of its intended users.

1.1 Problem formulation

The two different ways of developing additional functionality for the BC platform shall be compared. To achieve such a comparison the functionality of the extension eCom Documents, created by the Microsoft partner InBiz, shall be duplicated using the Power Platform. The two systems shall then be compared to determine if and where low-code solutions could be used to replace traditional code solutions. Specific questions to be addressed are the differences in:

- the required training for development
- the cost in terms of development
- the cost in terms of maintenance
- the cost in terms of licensing
- routines for updates of the solution
- routines for handling updates related to BC

Since the idea of low-code is to enable non-developers to produce software an overarching theme is the transfer of work from the Microsoft partner to their customer, the discussion of the questions will also include who will carry each cost or responsibility.

2 Background

This section is divided into three main topics to provide a technical background to both the problem and to the tools that are being evaluated. Section 2.1 and its subsection covers the origin of low-code and important terminology. Section 2.2 covers the Microsoft products used in this paper, they are BC, the current way BC is modified and the Power Platform. The Power Platform subsections cover the relevant concepts and components of Microsoft’s low-code solution. Finally section 2.3 covers related work and places this work in relation to it.
2.1 Low-code

Ever since the introduction of programmable computers, steps have been taken to increase the abstraction of the code and thereby decrease the complexity of creating software [5]. Assembly languages were developed to allow programmers a more intuitive way of writing software than machine code. This trend continues for example with the C-family languages that were designed to be even more intuitive than Assembly languages.

The increasing digitalization creates a higher demand for software services, specifically customer-facing applications [6]. In 2014 the market research company Forrester identified an emerging set of tools to accelerate the creation of such applications which Forrester refer to as “Low-code”. The main philosophy of low-code is that instead of having a programmer that knows how to code having to learn the problem, have a user that knows the problem learn to code. The developers of low-code claim that the tools are the next step in software development [7], [8]. However, such languages and development environments are not new, since the 1970s they have been part of what is called “Fourth-generation programming languages” (4GL). 4GL is a vague term as it sometimes refers to anything not COBOL and other times what Forrester labels as low-code. To avoid taking any stance on the definition of 4GL this text will only use low-code as the term describing tools developed for non-programmers to create software.

The users of low-code solutions are referred to as citizen developers [9] to differentiate them from developers with a programming background. The choice of term is also an important part of the marketing of the Low-code services as the providers refer to the Low-code development method as “democratizing” software development [7]. A suitable citizen developer for a given problem is the owner of the problem, for example if an employee always has to input data in a specific way, that employee would be a suitable citizen developer for a system developed to automate that input.

The low-code solutions are tools designed to allow citizen developers to produce software more quickly. Low-code solutions are technically not programming languages but systems of low-code development platforms (LCPDs) [10], these are characterized by a set of graphical user interface tools used to create software. Different LCPDs are focused on specific domains of software development, with common domains being: databases, business processes and web-based applications. The graphical appearance and the tools within a LCPD is determined by its domain, meaning that two LCPDs acting in different domains may have very different UI and still both be LCPDs.

Since 2014 when Forrester coined the term low-code a multitude of LCPDs has been created and some LCPDs have already been discontinued. One of the more notable discontinuations was that Google in January 2021 discontinued their platform AppSheet [11]. This was however at the same time Google acquired the company AppSheet [12] which provides another LCPD. Since 2016 Forrester has released several ratings of what they regard as the top LCPDs for different purposes, the list for Q2 2021 features their top 14 platforms for “Professional Developers” [13] and when combined since 2016 the lists have featured 45 unique companies. There are many companies providing different LCPDs and the market is constantly evolving with new LCPDs being introduced and old ones being discontinued.

2.1.1 Testing in low-code

Citizen developers are by their nature not developers and thus are not familiar with software testing, Khorram, et al. [14] identifies this problem as what the role the citizen developers should be in testing. As they are the experts on the requirements of the system, citizen developers have a natural and important role in evaluating if an application achieves its requirements. But in what way a citizen developer can run or participate in testing depends both on the individual and on the LCPD.

The way in which tests can be implemented is dependent on the LCPD, support for testing may not be included in a LCPD. If a platform supports tests, how are tests created. Are they created within the same LCPD, in another LCPD or in traditional code. If a LCPD allows tests to be created within the same tools used for application development, the citizen developers themselves could create tests. But if these citizen developers lack a framework or understanding for systematic software testing, traditional developers will have to provide the testing framework or write the test themselves. If the development of tests are done in a different environment, low-code or not, citizen developers may lack the ability to create tests. These cases require a traditional developer to completely manage the testing phase of the development.

The limitations imposed on the LCPDs to make them attractive for citizen developers may result in cases where automated testing is hindered or hidden. For example there may not exist a command line alternative to the graphical UI or only tools included in the LCPD may be used, preventing external testing solutions from being used.
2.2 Microsoft

BC is a continuation of Microsoft's previous small business ERP Dynamics NAV, the re-branding took place during the fall of 2018 [15]. This update promoted the Software-as-a-service (SaaS) model [16] as the primary model for future BC deployments. As a system BC consists of multiple modules based around different aspects of a business, for example: sales, inventory, human resources and service management [17].

Microsoft promotes the BC system as being highly adaptable [18], the adaptability is achieved in several different ways. The easiest customization is the ability for each customer to choose what modules of BC that should be used, this can then be narrowed to a per user level where each user has access to different modules or presentations of the modules. The customization relevant for this text is the more advanced customization called extensions, they are applications written to change the behavior of BC [2]. Examples of extensions can be an integration with a payment solution such as PayPal or localization extensions such as providing the user with regional reports. Extensions can change almost every aspect of BC, to achieve such customization extensions can add or change the logic in BC, such code is referred to as codeunits. Extensions can also alter or create new database tables and alter or create new pages within the system. These extensions are written in the BC specific language AL [3].

Highly relevant to this project is that Microsoft’s 2018 re-branding also promoted an improved integration with the Power Platform, Microsoft’s low-code platform. The Power Platform consists of Power BI, Power Apps, Power Automate, Power Virtual Agents and Dataverse. From the BC 2018 release notes:

“New Power platform capabilities combine Power BI, PowerApps, Microsoft Flow [renamed Power Automate], the Common Data Service for Apps [renamed Dataverse], and Power BI dataflows into an unmatched palette of tools to extend, customize, and integrate Dynamics 365 and Office 365 into your environment.” [19]

Connecting back to the domains of LCDPs the Power Platform is focused on business processes, web-based applications and automation.

![Microsoft Power Platform Diagram]

Figure 1: Structure of Microsoft’s Power Platform [20]

Figure 1 shows the current composition of the Power Platform. The following subsections will cover Power BI, Power Apps, Power Automate, Data connectors and Dataverse. AI Builder and Power Virtual Agents will not be covered as they are not relevant for this report.

2.2.1 Connectors

Connectors, in Figure 1 called Data connectors, are Microsoft’s low-code system for handling APIs [21]. A connector wraps around any API and gives the user a low-code standardized way of using that API. Connectors are used in Power Apps and Power Automate. In Power Apps they are used to access data provided by the API and allow the user to operate on that data, for Automate in addition to accessing data they can also be used to trigger automation.

An example of a large connector is the Office 365 Outlook connector [22], Microsoft’s email service, that for Automate has multiple triggers and actions. Examples of its triggers are: a new email has arrived
or a calendar event has been changed. Actions are things that can be done to the data, here we generally
mean the CRUD operations, in relation to the Outlook connector this can be create new emails, get
emails, manage contacts or remove calendar events.

Connectors have three different versions: standard, premium and custom. Standard and premium
connectors are the same thing except that premium connectors require additional subscriptions. Connectors
are primarily provided by Microsoft but other companies can also provide connectors for their non-Microsoft
services. As companies that provides their own connector can use their own payment solution the cost
for premium connectors can be hard to overview. For Microsoft services the premium connector is often
included with the license for that specific service.

Custom connectors are user defined connectors. A user that has access to an API can use their access
to create their own connector. The process of creating custom connectors falls outside of the scope of
low-code as it requires the understanding of OpenAPI definitions [23], Postman collections [24] or JSON
formatting and the relevant API’s definitions [25]. Despite that custom connectors are outside the scope
low-code, they can still play an important part for low-code development as developers can provide citizen
developers with custom connectors relevant and tailored to their needs.

2.2.2 Dataverse

The old name for Dataverse was “Common Data Service for Apps” which more accurately described
what it is, a data storage solution for the Power Platform [26]. The Dataverse service is a database and a
database management tool with integrated APIs both as normal APIs and in the form of the Dataverse
connector. The web interface for Dataverse can display the data in an Excel-like format, allowing the
user to delete data; to add or change the date there exist other views which can be customized thought
Power Apps. In addition to storing data, Dataverse can also provide virtual tables of any data stored in
Microsoft Azure, Microsoft’s main database service that hosts all of Microsoft’s SaaS solutions, among
them BC. This means that through Dataverse the services within the Power Platform can access any
Azure data in real-time.

2.2.3 Power Apps

Power Apps is the service in Power Platform that provide different UIs, they come in three different
versions: canvas apps, model-drive apps and portals. As the name suggests portals are external web-pages
that offer external users access to curated business data [27]. Model-driven apps are tools for structured
data input [28], they define a process that the user follows to input data, their logic allows for dynamically
changing what data is required depending on previously entered data. The final version of Power Apps is
canvas apps, which Microsoft describes as mobile applications developed as PowerPoint with an Excel-like
expressions language called Power FX [29]. The canvas apps are the version of Power Apps that will be
used in this project.

The underlying data used by Power Apps come from connectors with the main ones being Dataverse
and Excel, for canvas apps any connector that provides data can be used. When creating a canvas app
from an Excel or Dataverse table, an automatically generated app displaying the data is created.

Creating a canvas app from scratch is done by placing graphical objects on a screen, very similar to
that of a PowerPoint slide. The objects on the screen or the screen itself can then have their attributes
determined using the Power FX language. Examples of these attributes are: position, viability and on
click. The Power FX language allows for navigation between different screens and then populating those
screens based on what action the user used to enter them.

The canvas app can then be deployed both through the Power Apps web page or through the Power
Apps mobile application. Canvas apps can easily be deployed to all or a subsection of users within the
same organizational license. Apps can also be shared by the built it share function, similar to document
sharing in OneDrive.

2.2.4 Power Automate

Power Automate is the service within the Power Platform concerned with automating different processes
[30]. Power Automate comes in different versions: business process flows, cloud flows and Power Automate
for desktop [31], each concerning different types of automation. Business process flows is a system that
streamline data input and it will not be covered here. Power Automate for desktop will also not be
covered as it is a locally installed system for robotic process automation (RPA) which in its simplest
form is a system for recording desktop actions and then repeating them.
Cloud flows (flows) are the part of Power Automate that we will focus on in this paper. Flows are logic processes that automates something, each flow starts with a trigger from a connector [32]. These triggers can be divided into three groups: scheduled, automated and instant. Scheduled triggers are as the name implies started at specific intervals. Automated triggers are instead started by a connector that listens for an event, an Outlook example would be an arriving email.

Instant flows are triggered by pressing a button on the Power Automate web-page or in the in Power Automate mobile app, this function will not be covered and should not be confused with the triggering buttons that can be added in other Power Platform services. The buttons placed in Power Apps and Power BI are instead handled as automated triggers by Power Automate.

Flows are constructed of blocks of one trigger and multiple actions. The logic of a flow is that blocks are placed below each other thus creating a logical sequence of actions. The output of each block is, with one exception, known to each following block.

![Sample flow](image)

**Figure 2: Sample flow [33]**

In Figure 2 we see a sample flow that triggers when an email arrives, then the action block “Apply to each” which loops through each attachment of that email and the OneDrive (Microsoft’s cloud storage) connector’s action “Create file” saves each file.

In the figure we see one of the control blocks in “Apply to each”, control block acts as a special connector that changes the behavior of a flow [34]. The provided control blocks are if-else statements, switch statements, for-loops, while loops, a scope statement and a terminate statement. The scope statement is different from the other blocks as it hides all the outputs of the blocks within it from the rest of the flow. Outside of this there is also one control function that isn’t a block but rather a connection between blocks. Normally after one block has successfully finished the next block starts, however any block can be configured to run after the previous block has failed, been skipped or timed out.

Flows also support variables through the variable blocks, the data in these variables are set but can be of any data type using the catch all type Object. The variable blocks also support arrays, primarily to be used with the for-loop. Variable declaration is restricted to the global level; outside of scopes, conditions, and loops. Variables can however be changed through an entire flow. These control functions and variables allow for the creation of complex logical programs with error handling.

### 2.2.5 Power BI

Power BI is a service that is used to visualize business data in dynamic reports that intend to give the user an understanding of business metrics [35]. Common use cases for BI are viewing sales numbers or customer retention, but BI can also hold canvas apps as a report object and trigger flows by adding the Power Automate button to a report. The full range of BI’s functionality exceed what is described here but the full range of Power BI’s features are not relevant for this report and are thus omitted.
2.2.6 Testing in Power Platform

To test flows Microsoft [36] suggests that the developer should use a spreadsheet and then manually steps through possible inputs to find undesired behavior. Microsoft also lists three tools for testing [37], of these three only the Flow checker is an actual tool. The Flow checker provides a checker for syntax errors and issues with the flow. The Flow checker acts as compiler's error and warning messages and is thus not a tool for testing but for syntax errors. The next tool that Microsoft's list is the built in notification system that notifies the owner of a flow when it fails. The final tool is to configure the send email block to "run-after-failed" to send an email with an error message. The last two tools both requires that the flow is in active use and will thus only catch errors when they occur during live deployment.

There is one more tool for testing flows included in the LCPD, the "Test" button [38]. Depending on the type of trigger the flow uses button has different behaviors, the simplest behavior is if the trigger has no input then the test button simply acts as if that trigger has occurred and starts the flow. For triggers with input the button can either put the flow in a state waiting for the trigger to occur or run the flow with the same input as one of the latest runs. For certain triggers the feature is expanded further so that it can use example data of that trigger's type.

2.2.7 Learning the Power Platform

Microsoft offers two official ways of learning how to work with the Power Platform, the Power Platform documentation which has been much of the source for the preceding sections and the Microsoft Learn platform [39]. The Learn platform offers online courses in several of Microsoft's services with hundreds of modules for the Power Platform grouped into many different learning paths. In addition to these resources the Power Platform forums, one for each part of the Power Platform, and unofficial video material on sites like YouTube are available for citizen developers interested in learning how to use the Power Platform.

2.2.8 Licensing

To interact with BC a user needs one of three licenses: Team Member, Essentials or Premium [40]. Of these the Team Member license offers the least functionality and it is used by people not working in BC but rather viewing BC. Such users have full read access to BC but very limited write access, a user may input to your timesheet, approve tasks and work on sales and purchasing orders. For the users of BC who enter and change data in BC an Essentials or a Premium license is required. A BC environment cannot have both Essentials and Premium licenses at the same time and so all non-Team Member licenses need to be of the same type. The Essentials licenses gives the holder access to most of the features of BC, the differences from Premium is that the Premium license gives its holder access to the service management and manufacturing modules. The practical implications of this is that if a company works with production or service they would need Premium for most other companies the Essentials license is sufficient.

For all licenses of BC the user is also given licenses to other Microsoft services, relevant for this report Power Automate and Power Apps licenses [41]. For these services the licenses are divided into "Dynamic 365 Applications" (Team Members) and "Dynamic 365 Enterprise Applications" (Essentials and Premium), with Enterprise offering more functionality. Within Power Apps the non-enterprise user can't use custom apps and effectively doesn't have a license but are granted access to all connectors relevant to their BC license. Enterprise users on the other hand can run and create custom apps within the same environment as the BC license and with all relevant connectors. For Power Automate the licenses are almost the same and can run flows using premium and custom connectors to services available in their BC license, the only difference is that a non-enterprise user is limited to how many Dataverse tables they can use.

To make things slightly more confusing, many users of BC could also be expected to have a Office 365 license, this is the license that gives enterprise users access to services such as Word, Excel and Outlook [42]. This is relevant for Power Apps as it gives the holder of any Office 365 license access to canvas apps with the limitation that only standard connectors are available [41]. The Office licenses also provide limited access to Power Automate by only allowing the usage of standard connectors within flows. By combining these licenses a holder of both a BC Team Member license and a Office license should then be able to use and develop canvas apps with the only hard limitation being the amount of Dataverse storage available.

Both Power Apps and Power Automate also offer individual licenses for their services [41]. Power Automate offer both per user and per flow licenses and canvas apps offers a "per user, per app" and a per user license. The Automate per user license allows a user to create and run any number of flows whereas
the per flow license allows the creation of one flow per license and an unlimited amount of users may use the flow. For Power Apps the individual licenses cover both canvas or model-driven apps, the per user, per app license allows the holder to create and use one app and the per user license allows one user to create and use any number of apps.

The final relevant license is the licensing for Power BI, which is not included in any BC licenses. Power BI has three licenses: Free, Pro and Premium [43]. The Free license is as the name suggests free of charge but is then limited to only allowing the user to interact with their own report “My Workspace” or reports shared with them from a Premium licensed account within the premium users tenant. Pro licenses holders have the ability to share their reports with other Pro license holders. The Pro license was until September 2020 [44] the full version of Power BI. The last license version is Premium per capacity which instead of a per-user plan is an organizational plan that gives the organization their own cloud instance of Power BI. These organizational instances can then share reports with users regardless of their Power BI license but to be able to provide reports through the Premium license the user that share content needed a Pro license. With the 2020 announcement of Power BI Premium Gen2, in addition to the organizational license a new Premium per user license was added. This license grants the user all the ability of a Pro license but the user also gains access to the control features of a Premium license as well as a faster update rate.

Since the embedded BI report in BC is not a “My Workspace” report the user needs either a Pro or Premium per user license to be able to view it.

<table>
<thead>
<tr>
<th>License</th>
<th>Price/month</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Central Essentials</td>
<td>$70</td>
<td>[45]</td>
</tr>
<tr>
<td>Business Central Premium</td>
<td>$100</td>
<td>[45]</td>
</tr>
<tr>
<td>Customer Service Professional (BC Team Member)</td>
<td>$50/$20</td>
<td>[45]</td>
</tr>
<tr>
<td>Sales Professional (BC Team Member)</td>
<td>$65/$20</td>
<td>[45]</td>
</tr>
<tr>
<td>Office 365 E1</td>
<td>$8</td>
<td>[46]</td>
</tr>
<tr>
<td>Office 365 E3</td>
<td>$20</td>
<td>[46]</td>
</tr>
<tr>
<td>Office 365 E5</td>
<td>$35</td>
<td>[46]</td>
</tr>
<tr>
<td>Power Automate per user</td>
<td>$15</td>
<td>[47]</td>
</tr>
<tr>
<td>Power Automate per flow</td>
<td>$100</td>
<td>[47]</td>
</tr>
<tr>
<td>Power Apps per app/per user</td>
<td>$5</td>
<td>[48]</td>
</tr>
<tr>
<td>Power Apps per user</td>
<td>$20</td>
<td>[48]</td>
</tr>
<tr>
<td>Power BI Pro</td>
<td>$9.99</td>
<td>[49]</td>
</tr>
<tr>
<td>Power BI Premium per user</td>
<td>$20</td>
<td>[49]</td>
</tr>
<tr>
<td>Power BI Premium per capacity</td>
<td>$4,995</td>
<td>[49]</td>
</tr>
</tbody>
</table>

Table 1: Table of listed licensing prices
1. First price if the license is the first Dynamics 365 license for the user, second price for subsequent licenses
2. The per flow price is $100 but a customer needs to purchase at least 5 license

2.3 Related work
Waszkowski [10] analyzes the requirements and implementation of a LCDP for manufacturing. He also provides a clear definition of LCDPs and their proposed benefits over traditional software solutions. He describes them as “This is, so far, the fastest and probably also the cheapest method of developing software”, this claim is part of a more widespread claim promoted specifically by the providers of LCDPs.

Sabay et al. [9] further describes the taxonomy of LCDPs and primarily compares the functionality of eight major LCDPs, among them Microsoft’s Power Apps, finding four main areas of limitations: Interoperability, Extensibility, Learning curve and Scalability. Most of which play an important role for this project, the interoperability between BC and the Power Platform being the main question for this work. The learning curve is featured as a question when trying to determine what training is required for using the platform. The excitability of the platform is also discussed as a possible solution to problems discovered during the project.
3 Comparison

The following parts are structured as follows, first an overview of the eCom Documents solution is provided followed by a section detailing the limitations of requirements on the low-code solution. After that a table providing an overview of how the two implementations archive the requirements. Following the overview the implementation of the requirements will be a more detailed account on how and why certain implementations were done.

3.1 Existing system

3.1.1 The AL language

The AL language used for writing BC extensions is specifically developed for usage with BC, as such the documentation is limited to Microsoft sources. AL is an object-oriented language with all parts of BC are described as objects, database tables are represented by table objects that describe the table schema and the page object describes the user interface of each page [50]. An important property of the language is “extend object” that allows for existing tables and pages to be expanded with additional columns or UI objects.

Microsoft suggest that a developer should use Visual Studio Code with the AL language extension to develop AL extensions [50]. Besides providing an IDE with the expected support of key-word highlights and a compiler, the AL language extension also includes templates and integration for publishing extensions directly from the IDE.

To learn AL Microsoft offers digital courses on their Learn platform, for example [51] and for Microsoft partners through multi-day courses. The Learn platform’s introduction path to AL takes roughly the same time (8h 25min) [51] as the introduction to the Power Platform (8h 50min) [52] with the main difference being that the AL learning path expects the user to know basic programming.

Within the Microsoft partner InBiz most developers had learned AL by attending at least parts of the multi-day course and one or more workshops, in addition to that some of the developers had used the Learn platform and others had studied AL by consuming material on YouTube. The general feeling was that AL is easy to learn as the languages capabilities are narrow in scope, but that the process for certain thing like reports creation was unnecessarily programmer focused where they could have been more graphical.

3.1.2 eCom Documents

eCom Documents is an extension created by InBiz, it is designed to automate the sending of invoices, credit notes, order acknowledgments, purchasing orders and quotations. The automation is based on the premise that each night all completed documents of the supported types that are not already sent should be sent. This is done through the included BC functionality of Job Queues, which allows for scheduled starts of tasks [53]. The eCom Documents extension utilizes all aspects of BC extensions: codeunits, page extensions, custom pages, table extensions and custom tables.

eCom Documents extends the pages and tables related to the supported document types and those of customers and vendors. The pages are extended with additional fields to accept more inputs which then corresponds to table extensions. The pages are also extended with buttons to trigger the actions in the codeunits. The custom pages and tables are used to hold an outbox of scheduled documents, error logs and sent documents. Finally a custom page and table is used to hold a settings page for the eCom Documents system.

The required updating of eCom Documents differs depending on what type of BC installation is deployed. For customers using an on-premise installation of BC the need for updating eCom Documents can only occur when updating the BC version. Even when updated BC eCom Documents only needs to be updated if the BC update changes something that would affect the extension. InBiz decides the update schedule together with their customer and many customers prefer not to update their BC installation due to the risk that the update would break one or more customization. For SaaS installations the control over BC updates are outside of the control of both the customer and the Microsoft partner selling the service as Microsoft contiguously will update the installation. Therefore each new patch could break an extension, drastically changing the updating requirements of the extensions. The cost of maintaining extensions is thus increased for SaaS installations.
3.2 Previously know limitations

The following subsections covers the known and decided limitations of the low-code solution. The first section covers the limitations decided on together with InBiz, the second section covers the known ways in which the Power Platform limits what can be done. The final section covers the knowledge of the BC connector before starting the project.

3.2.1 Limitations of scope

The low-code implementation of an eCom Documents equivalent system has one major limitation of scope, instead of supporting several different document types the low-code system will only support invoices. Invoices are the primary use case for eCom Documents and offer the biggest challenges. The reason why invoices are harder to implement that other document types is that they have multiple stages and only invoice that are posted but not yet sent shall be processes by the system.

3.2.2 Limitations of extensions created in Power Platform

In relation to AL extensions any extension created through the Power Platform has several limitations that need to be taken into consideration, the two main ones are that the Power Platform cannot add or extend tables and it cannot add UI elements onto BC pages.

The lack of ability to add or extend tables is not a major limitation as Microsoft offers multiple other ways of storing data in database form. As previously discussed the Power Platform’s preferred method is to use Dataverse, Excel tables also work but with much slower read and write access. Thus instead of extending a BC table, a new Dataverse or Excel table can be created with an ID per row to link it with a BC table entry.

The limitation of not being able to add UI elements such as buttons, files etc. is a stricter limitation as there is no obvious or equally good solution. Power Platform does offer ways to visualize things with the Power Apps suite but they are then limited to an external canvas app or locked to the BC home page by integrating a canvas app into the home page Power BI report. For the low-code implementation a UI will have to be developed but its design must use different methods than those of the existing extension.

3.2.3 The Business Central connector

The Power Platform offers two connectors for BC, one for on-premise installations [54] and one for SaaS installations [55], both of these connectors are flagged as preview. The preview status means that Microsoft is more likely to change the functionality of the connector. Despite the preview status both connectors are in use since the 2018 BC update and are promoted by Microsoft.

In general the SaaS connector is the more supported version with a larger set of operations. Both connectors offer the following actions with variants to the names: Create, Delete, Get, Get types and Update. In addition to these, the SaaS connector also has the “Execute action” action, which allows a flow to trigger predetermined actions in BC, most important for this project the action of sending out invoices. The on-premise version has no triggers, meaning that it cannot be used to trigger flows, the SaaS version on the other hand offers several triggers. These triggers can be divided into two groups; approvals and table changes. The approval triggers cover all approval requests that can be made in BC and the table changes triggers cover addition, removals and updates of records in the database.

To interact with data in BC both the SaaS and on-premise connector use the systemID as the lookup key, meaning that to interact with any data in BC an extension needs to know the systemID. For the SaaS version the system each trigger returns the systemID of the record that prompted the trigger, to then access the rest of the record the systemID can be used in the get action. Due to the lack of the triggers and the “Execute action” action in the on-premise connector the low-code version will only support the SaaS versions of BC.

3.3 System Requirements

The system requirements for the low-code implementation are based on the system requirements of the existing eCom Documents solution. With the limitation in scope, the known limitations of Power Platform and the Business Central connector the requirements were then revised. The revised requirements were developed together with InBiz and in particular the developer of eCom Documents. The requirement elicitation process was conducted by inquiring the desired behavior of the product, using the existing eCom Documents solution and code review of the existing solution together with the developer.
Table 2 shows the system requirements for the low-code solution. A short summary of how they are implemented in the existing system and the suggested low-code solution are provided. The last column are references to which section that covers each requirement.
<table>
<thead>
<tr>
<th>System Requirements</th>
<th>Existing solution</th>
<th>Suggested Low-code solution</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The system shall automatically send out email invoices that are posted on a schedule.</td>
<td>Utilizing BCs job queue two custom functions are triggered on a schedule. One function iterates over all invoices and adds them to a custom table and the other one iterates through the custom table and sends them out.</td>
<td>A flow listening for changes in the invoice table and then checks invoices that haven’t been sent and adds the to a external excel file. Another scheduled flow iterates through the excel file and triggers the BC function for sending invoices.</td>
<td>3.4.2, 3.4.3</td>
</tr>
<tr>
<td>2. The system shall only automatically send out invoices that have not already been sent.</td>
<td>The automated add to queue checks if invoice has already been sent.</td>
<td>The listening flow checks for duplicates before adding invoices to the excel file.</td>
<td>3.4.2</td>
</tr>
<tr>
<td>3. The system shall be able to send out invoices that have already been sent.</td>
<td>Manual addition to queue by custom button on invoice.</td>
<td>Button added to invoices in the canvas app triggering a flow that stores them to a manual queue that is sent at the same time as the automatic queue.</td>
<td>3.4.4</td>
</tr>
<tr>
<td>4. System administrators shall be able to change the scheduling.</td>
<td>The scheduling of the process can be changed on the job queue settings page.</td>
<td>An administrator can change the automated flows scheduling by editing the flow.</td>
<td>3.4.3</td>
</tr>
<tr>
<td>5. Users shall be able to manually send all relevant documents.</td>
<td>Button on custom outbox page.</td>
<td>Button in canvas app that triggers a flow that sends out all emails in the queue.</td>
<td>3.4.4</td>
</tr>
<tr>
<td>6. Users shall be able to add invoices to the automatic sending system.</td>
<td>Custom button added to standard invoice page.</td>
<td>A button added to canvas app triggering a flow that adds the invoice to a manual queue.</td>
<td>3.4.4</td>
</tr>
<tr>
<td>7. Users shall be able to see which documents have been sent.</td>
<td>Custom page reachable by custom cue card from main menu.</td>
<td>Graphical representation in canvas app.</td>
<td>3.4.4</td>
</tr>
<tr>
<td>8. Users shall be able to see which documents are scheduled to be sent.</td>
<td>Custom page reachable by custom cue card from main menu.</td>
<td>Graphical representation in canvas app.</td>
<td>3.4.4</td>
</tr>
<tr>
<td>9. Users shall be able to see which documents, if any, has failed to be sent.</td>
<td>Custom page reachable by custom cue card from main menu.</td>
<td>Graphical representation in canvas app.</td>
<td>3.4.4</td>
</tr>
<tr>
<td>10. Administrator shall be able to set up the addresses to send document from.</td>
<td>Custom settings page created to hold such email addresses.</td>
<td>Skipped since the function is already supported by Microsoft.</td>
<td>3.4.4</td>
</tr>
</tbody>
</table>

Table 2: The system requirements and its solutions
3.4 Implementation

The following sections cover the process of implementing the low-code version of eCom Documents. The general structure will be that each section starts by covering how the issue is solved in the existing eCom Document solutions. Then each section will describe the design process that lead to the final implementation and cover the final implementation. For a short summary of how the system requirements are solved see table 2.

3.4.1 Data storage

In the existing solution the additionally required data is stored within the BC database, this is done by using table extensions and custom tables which augments the BC database tables. The table extensions to customers and vendors are used to store extra email addresses for each document type. As the low-code version only needs to support invoices this requirement was removed and will not need to be duplicated. The storage part that needs to be duplicated is the outbox, error log and sent table which in the existing solution are stored in a custom table.

To implement storage in a low-code solution an external storage solution is required, the recommended solution by Microsoft is Dataverse but Excel is also an option. From the perspective of a flow both the Dataverse and Excel connector looks the same and offers the same functionality, they can thus be swapped without changing the behavior of the low-code system.

Excel suffers from a rather substantial worse read and write time compared to Dataverse, this is due to Excel Online only allowing one reader or writer at the time. The reason why Excel is attractive for this implementation is that it is easier in Excel to perform tasks such as changing table schema and manually edit data in the tables. During the development process the schema on the tables were changed multiple times which was easy to facilitate in Excel. Due to these easier prototyping capabilities of Excel the decision was made to use Excel Online instead of Dataverse for the implementation. The final version of the tables schemes could be duplicated in Dataverse to increase the performance of the solution.

The storage of data in Excel by the Excel connector does not work on individual cells but on tables. These are formatted parts of Excel sheets where each row acts as a database row, the header of each column names that column. The connector can then interact with each table based on its key value. Due to the design of the BC connector the systemID was chose as the key value. What additional data that needs to be stored depends on the implementation of the rest of the system and will be covered together with those implementations in the following sections.

From the existing solution there were three views, scheduled, error and sent, to duplicate. In the low-code solution this could then be done in three tables or by creating one table with a column to note what type of entry each row contained. To allow for expansions without having to change previously implemented logic and minimize the complexity of the storage the decision was to use different tables for each type of entry.

3.4.2 Logic

The existing system relies on two main functions and BC’s job queue to achieve the ability to automatically send out invoices. Both functions are configured to run on a schedule in a job queue.

The first function will will add documents to the outbox queue. This function can be configured to run as often as desired but will, when run, decrease the performance of the BC installation. The function will iterate through the table of posted invoices and find those not yet sent and add them to the outbox queue unless they are already in the queue.

The second function will instead iterate over the queue of found documents, generate the appropriate document and then send it as an email attachment. The function also logs the results of its attempts in either a log of sent documents or an error log. The scheduling of the document is what enables the automatic sending of invoices and this scheduling can be changed in BC’s job queue if the scheduling requirements are changed.

The first suggested low-code solution was to create a scheduled flow that used the BC connector to filter out all unsent invoices and then use the BC action to send them out, see figure 3. This solution is simple but would require additional logic to satisfy the requirements of data visualization. The attempt to implement such a flow immediately ran into the problem that no action for getting multiple entries existed within the BC connector. Each entry would instead have to be requested by its systemID, something not accessible to the flow.
The next proposed solution was to use the BC trigger “When a record is changed” to listen to the invoice table for any changes, see Figure 4. If the change could be determined to be that an unsent invoice had been posted, its systemID could be saved and then used later by another flow to send the invoice. This solution also ran into major problems, the first one being which tables the connector actually provided access to.

In BC invoices are stored in different tables depending on their status, either they are in the table Sales Invoices or after being posted they are in the Posted Sales Invoices table. The BC connector doesn’t provide access to each of these, instead it provides access to a virtual table consisting of both of these tables. For the implementation this wouldn’t have been a problem if it still was possible to determine from which table each entry originated. Unfortunately no such field exists as the property of being posted
is derived from the entry being in the posted invoices table. The lack of this information could not be solved in a flow but since the invoice number was available a strict numbering convention could allow the system to still identify which invoices that where posted.

The lack of posted status led to an investigation into what data the connector actually provided and it was discovered that the connector only provides access to a subset of the combined tables columns. The most important omission is the amount of times a invoice had been sent. Without this information any implementation would result in sending out the same invoices multiple times. A change from “When a record is changed” to “When a record is created” could limit the error to only affect invoices that were manually sent. Such a deviation from specification would still render the implementation useless as sending a customer the same invoice multiple times is unacceptable.

This final limitation lead to the conclusion that the task of implementing automated sending of posted invoices is not possible within the Power Platform using the provided connectors. In the interest of seeing if the task could be achieved if the BC connector had worked as expected the decision was taken to create a custom connector, that would have access to all data fields. As the creation of a custom connector was deemed to be outside of the ability of a citizen developer it was previously excluded from the scope of this project.

As previously stated a custom connector requires its creator to have access to the API they want to use. BC offers two methods for making API requests that are supported for any table within BC, OData and SOAP. OData is a protocol used for REST APIs. Using the OData version of BC’s built in API function a custom connector was set up. The process of creating a custom connector was deemed possible for a citizen developer except for a important step. When formatting the API response so that a flow could use the data JSON formatting was required which likely would be outside the knowledge of a citizen developer.

From this point two GET actions were created for the custom connector, a filtered get and a get individual invoice. The creation of the get-filtered would in theory allow for a version of figure 3 to be implemented by using the custom connector to get all invoices. Such solution would be even simpler than that solution since instead of filtering the response within the flow the filtering could be done by the get command, see Figure 5. However when trying to implement this solution a problem with BC’s APIs was discovered.

![Diagram of custom connector solution](image)

Figure 5: Mock-up of first custom connector solution in Power Automate

When using the integrated API system for BC to extract posted invoices the API does not provide the user with the systemID. This issue arises from the differences between SaaS and on-premise installations. In on-premise installations the underlying tables can be viewed by developers but for SaaS installations only virtual representations of the tables are available. This leads to that sometimes the content of a table is unknown for both the extension developer and the end-user as tables viewed in SaaS can be merges of data from multiple actual tables. With the structure of a SaaS installation database unavailable, the
reason for the omission of the systemID field is not known, one possible explanation is that the systemID is stored in a master table that is then merged with the invoice tables. Another possible explanation is that the systemID is part of the invoice table but Microsoft has for some reason decided to exclude it from the API call of Saas installations.

This omission of the systemID from the API call could however be solved within Power Automate by extending the record change solution with the additional get record action from the custom connector. The additional get could solve the two problems posed by the solution in Figure 4. By using the custom connectors get individual invoice to get number of times the document was sent. By only running the API against the Posted Sales Invoices table the flow could determine if the invoice that triggered the flow was from the Sales Invoices table or the Posted Sales Invoices table. If the flow tries to get an invoice that isn’t present in the Posted Sales Invoices table the action will return a no such record failure, the flow can then use that failure to determine that the record change didn’t concern a posted invoice.

![Diagram](image)

Figure 6: Mock-up of second custom connector solution in Power Automate

Figure 6 shows an updated version of the record change solution that works for the case that an invoice is posted but has not yet been sent. The flow in Figure 6 will the add that invoice to the Excel table. But this solution also has shortcomings, none of which are unsolvable but they all require additional logic to be implemented.

The first problem occurs if an invoice is already in the Excel table when the same invoice causes the flow to trigger, if this would happen multiple instances of the same invoice would be added to the Excel table. This can occur if the invoice is changed, for example being sent manually or having its address updated. To solve this before checking if the invoice has been sent a lookup of the triggering invoice is done to the Excel. If this lookup fails we know that the invoice is not in the table and can then check if we should add it or not by checking the number of times sent. If it is in the table we still need to check if it has been sent because that means it should not be sent out by the automated system and must thus instead be removed from the table.

The next shortcoming is due to the ability of flows to be run in parallel. Automated triggers are not immediately alerted to changes in a table, instead they poll the data source at some frequency. The polling frequency is generally determined by the license of the flows owner/creator and ranges from 1 to 60 seconds. The license used to implement this solution having a polling frequency of 30 seconds. If the flow is triggered by the same invoice multiple times within the polling frequency multiple instances of the flow will be triggered to run concurrently. The occurrence of multiple changes exacerbated by the behavior of BC. When an invoice is first posted BC will first create a new entry in the Posted Sales Invoices table, that entry is then immediately changes by adding all the data to the entry. This means that every time an invoice is posted the flow will trigger twice within the same polling cycle. When this happens both instances of the flow will check if the invoice is in the Excel table and see that the invoice is not there. Both instances will then add that invoice to the Excel table, resulting in two instance of the same invoice in the table. To solve this issue the simple solution would be to check the Excel table for duplicates before sending out invoices, however doing so would result in the table normally containing duplicates which then complicates the next shortcoming.

If one invoice is manually sent it should be removed from the Excel table, this can be solved by using the custom connectors ability to check of the invoice is send and if that is the case remove it from the
Excel table. Note that this removal would also catch invoices that are sent by any automate method and also remove them from the table. Connecting back to the duplication issue, if the invoice that has been updated is duplicated within the table we need to remove not only one but every instance of that invoice from the table. This means that we need to run a duplication check on the Excel table for every posted invoice that trigger the flow again. Therefore we cannot outsource the duplication check to the sending flow. By running a duplication removal after each run of the get flow we can solve this issue.

The final addition to the get invoices flow was not due to a problem with the flow but to a non-specified requirement of maintaining a log of when data was entered into the table. To achieve this the current execution time of the flow is retrieved and then stored together with the record in the Excel table. This results in a function that achieves the same result as the existing AL implemented function that adds documents to the queue. The main difference being that the flow is not a scheduled event, instead it constantly waits for changes in the invoices table and then determines if the entry should be store or not.

![Diagram](image)

Figure 7: Implemented solution for getting the invoices to be sent in Power Automate

### 3.4.3 Sending logic

Since the logic for extracting and sending the invoices could not be combined, a separate flow for the sending had to be developed. In relation to the get invoices this flow is much simpler and would in its simplest form only require a schedule start, getting all the invoice to be sent and then for each of them send them out, see Figure 8. The usage of a scheduled start of the flow satisfies requirement 4 as the entire flow can be changed by an administrator. The act of changing the schedule is then done by changing the recurrence trigger block.

Such an implementation would almost satisfy requirements 1 and 2 given proper input but would cause issues with the visualization requirements 7 and 9. To satisfy those requirements the result of each send action must be logged in either the sent table or in the error table. As stated at the end of the get invoice implementation the get invoices flow would remove all invoices that were successfully sent from the scheduled Excel table but to maintain a structured scheduled table the send flow can remove them after sending them. The important date to log for the error log besides the information stored about the invoice is the error message provided by the send invoice action. As the error message is not an expected output of the send invoice action there doesn’t exists a standard error output to be used by the error logger. Instead to retrieve the error message a simple expression has to be written into the error logger that fetches the error message. This once again stretching the expected knowledge of a citizen developer, as the expression code may be short but doesn’t rely on the graphical interface of the LCPD. The resulting for-loop can be seen in Figure 9.

There is one more addition that is needed for the sending flow to always satisfy requirement 1 and 2 and that is the previous mentioned duplication. The get invoices flow removes duplicates every time it runs but in the rare chance that the last run of the get invoice flow was the cause of such a duplication the get flow cannot catch it. Therefore the same duplication check needs to be run before sending the invoices.
To satisfy requirement 3 the system needs a way of re-sending invoices that has been sent by the scheduled sending flow. Since the automated system for getting invoices cleans and maintains the Excel table from already sent invoices such invoices, if added manually to the Excel tables, would be removed by the get invoices flow. An additional table could be used to store such invoices. The send flow would then require an additional sending loop to process that table. The result of these improvements can be seen in Figure 10.

The addition of the terminate statement after the sending loops is due to how flows handle errors within loops, even though the error has been handled by adding the invoice that caused the error to the error log the loop will exit with a failure flag after all other invoices have been handled. When a flow terminates with the status flag set to failure the run will be marked as failed and an email will be sent to the owner of the flow notifying them that a flow has failed. This behavior is undesirable with the implemented error handling so by forcing the flow to terminate in success after the sending process will eliminate that behavior.

3.4.4 UI

To be able to satisfy requirements 5, 6, 7, 8 and 9 some form of user interface is required. In the existing solution this is done by page extensions and the creation of custom pages. This means that the UI for eCom Documents is spread out through BC on each relevant page for the document types. Since the
low-code solution doesn’t support multiple documents types most of these UI elements don’t need to be replicated.

The main eCom Documents UI is the Outbox, a custom page that shows all documents that are present in the eCom Documents system. From the home page of BC the Outbox is reachable by three cue cards, each offering a different filtration of the documents. The three views are: documents to be sent, sent documents and documents that failed to be sent; all of these must be replicated to satisfy requirements 7, 8 and 9.

From the Outbox the user can send all documents as well as access eCom Documents settings page. On the settings page the user can set up the email addresses and email texts that should be used for each document type. To provide the same functionality, even if it is only for one document type, is outside of the capabilities of the Power Platform, however while investigating the issue an opt-in functionally for BC was discovered. The opt-in functionally allows for the same features as eCom Documents provides as each document type can be assigned to be sent from a different address. Therefore instead of having to implement the solution using low-code the functionally developed by InBiz for eCom Documents was found to be superfluous within the current SaaS version, making requirement 10 satisfied by default.

To develop a UI to satisfy requirements 7, 8 and 9 a canvas app was chosen. Since the UI only needs to display the data in the Excel table, the only required connector is the Excel connector. As the data source is Excel the automated generation of a canvas app to display the table could be used. Using the automated generation a two page app was generated for the scheduled invoices table. This app consists of one list view of the table rows and a detailed view of a selected table row. The automated app can then be tailored by changing which information is displayed in the list preview and what data should be displayed in the detailed view. These two views can then be duplicated for the other tables; manually scheduled invoices, the sent invoices and error log. The manually scheduled invoices and the sent invoices are identical save for headings. For the error log the detailed view is augmented with an additional field displaying the error message.

To allow for navigation between the different table a home page view was created with navigation buttons to each tables view. Each of these views were then updated with a button to allow the user to return to the main view. At this point requirements 7, 8 and 9 are satisfied as the user can see documents for each step of the process.

The two remaining requirements are then 5 and 6 which require both UI and logic elements. Requirement 5 could be solved by creating a copy of the scheduled sending flow where the trigger was changed to a Power Apps button. That button could then be linked to a button on the home view of the canvas app.

For requirement 6 new flows had to be developed; these flows request the **systemID** from the Power App, then retrieves the additional information using the BC connector and the current time using the time connector and finally store that date in the manual queue table. A copy of that flow also has to be developed for invoices in the error log view as they should be removed from the error log table at this point. Figure 11 shows the error log version with the last step being the extra removal action.
3.4.5 Implementation summary

For the low-code implementation of the eCom Documents system Power Automate and Power Apps was used. The two main parts developed in Power Automate was the get invoices flow and the send invoices flow. The send invoices flow was then copied so it also could be initialized from a canvas app. Two flow to add invoices to a manual queue and a flow to remove invoices from the queues was also created. Finally a canvas app was created to allow the user to see which invoices that was being handled by the system and any error messages.

3.5 Licensing

To sort out the required licenses and thereby the cost of using the implement solution we first need to recap what parts of the Power Platform are used. The created Power Automate flows interact with Excel, Power Apps and BC. The canvas app interacts with Excel and Power Automate, an extended version of the canvas app could also interact with BC to enrich the presented data. If the app should be visible within a BI report to view it on BC’s homepage, one connection to BI is also needed.

The common setup for SaaS installations of BC is that all the users that actually use BC in any capacity are either holding the Essential or Premium licensing plan, depending on the nature of the company’s business. The Team Member licenses may be used by executives to view BC or by employees using BC to register their time cards.

With this setup all users of BC that would be required to use the automation and its UI are those with Essential or Premium BC licenses, meaning that no additional licenses would be required for the usage of the automation. This scenario would not lead to any increased licensing cost for the end customer. However if a user wishes to view the canvas app within BC they would need at least a Power BI Pro license.

If the automation should be independent of any individual user an additional set of licenses would be required. In this scenario only the flows need their own licenses as the canvas app isn’t necessary for the automation but only for the UI. Due to the pricing difference between additional BC licenses and per flow licenses, the new most economically viable solution is to have an additional BC license. This resulting in a requirement of one additional BC license and one Power BI Pro license per user of the system.

4 Discussion

4.1 Conclusions

To build a low-code version of eCom Documents using only the Power Platform without relying on any code or custom connector proved to be impossible. The previously unknown limitations of the BC connector that made vital data un-accessible through the connector being the cause of this. But with the slight widening of the scope, achieved by creating a custom connector the task became solvable, but
still required several workarounds. The knowledge required to set up a proper multifunctional connector that could completely replace the standard BC connector would greatly exceed the ability of any citizen developer.

With the custom connector created using the BC OData API a workable prototype was created. The prototype could manage to find all relevant invoice and send them out with on a schedule meaning that the logical part of the solution achieves its system requirements. The canvas app UI together with the attached flows also provides a bare bone visualization and control of the system however, this system is not as intuitive or as user friendly as the eCom Documents UI. The biggest downside is that the low-code UI is an external system and not integrated throughout BC, even if the low-code UI was visible from the BC home screen a user could not interact with it from all relevant screens. This makes low-code versions not suitable for extensions that requires integrated UI elements and thus make the usage of a low-code solution unsuitable for replacing eCom Documents.

If a connector that worked as the BC connector was expected to work was available the barrier for implementing automation in BC would be significantly lowered and extension development could be handed over to citizen developers. To complement such a connector a traditional extension would probably have to be created to optimize usability. One such optimization, that is identified, is the ability to include the systemID on all relevant API calls. With an optimized connector extension pair low-code solutions such as the suggested implementation in figure 5 could be created by citizen developers. Due to first having to create the connector some of the initial advertised cost savings of low-code would be negated. However if the custom connector could be reused, those cost savings would once more be relevant.

The maintenance of a low-code system does not have to consider data migration and changing the hardware due to the SaaS model that the LCDP is built upon. However, as with the risk for SaaS installations of BC, any software developed in a low-code solution could break from an update to one or more of its components, leading to a sudden acute need of updating the solution or in the worst case re-writing it. As long as such software breaking updates are located within the Power Platform the maintenance can be done by the citizen developer but if such an update would break the connector major work with updating it would be required.

Handling updates within BC would not be any different for the low-code solution compared to the existing solutions as a custom connector and its accompanying extension would be susceptible to the same issues as any other BC extension.

For users with a programming background the time it takes to learn how to use the Power Platform or write programs in AL is both roughly the same. For a citizen developer the time to learn how to use the Power Platform is probably slightly longer than for a developer but significantly shorter than the time it would take to learn how to use AL. To get started with the Power Platform, a citizen developer could take the basic introduction learning path of 9 hours on the Learn platform and then roughly spend the same amount of time on some of the more focused learning paths to be able to develop within the Power Platform. With some proficiency in using search engines and reading documentation such a user would be able to begin creating complex solutions within a week. The major limitation being that if they try to implement something outside of the built-in functionality of the Power Platform, the material for implementing custom connectors or in other ways expand the Power Platform functionality, this is not as well documented.

When citizen developers use the Power Platform to develop applications, testing will be an underdeveloped part of the development process as the techniques of testing is not highlighted are the Power Platform. Where such tools exist they are hidden under menus such as “Advanced Tools” which increases the barrier to use them for citizen developers staring out. To achieve a suitable testing of Power Platform developments the testing routines must be maintained outside of the Power Platform and taught to the citizen developers.

4.2 Lessons learned

Working with Microsofts low-code tools have been both rewarding and frustrating, they are very inviting and lead to many ideas of what could be, but when looking closer all the limitations appear. When trying to evaluate how the tools perform using any sort of measurement, all the relevant information is hidden from the user, “How large is the program?”, “How many lines of code?”, “What is its CPU usage?”. For flows none of these questions can be answered, the only information that a flow can share is its execution time and if it succeeded or failed. For other project comparing low-code solution with either traditional solutions or other LCDPs the first question should have be “What can be measured?”. Regardless of this,
many questions can be answered concerning the usage of LCDPs but all analysis of them would benefit from knowledge of what can be measured.

To learn how to use the Power Platform before implementing the solution I used the Learn platform. Since I didn't time the actual time spent on the platform the only accurate measurement is by the Learn platforms suggested completions time. According to that measurement I participated in 14 hours and 42 minutes of relevant material. The material I consumed was a mixture of video, text and guided exercises.

When I started working with the implementation the best resource for finding solutions was within the Power Platform forums, the one problem I experienced while using these were that since the platform is continually updated old solutions, mostly "clever" workarounds, were sometimes found to be removed or changed. For more general questions YouTube was an invaluable resource as the video format is suitable to show the graphical interface of the Power Platform.

The one point where the online resources failed to provide the required guidance was when I was trying to implement the custom connector. The process involved using tools and languages not developed by Microsoft, these tools naturally aren't documented within the Microsoft documentation. Unfortunately the Microsoft documentation often didn't even contain information what type of input was expected for the non-Microsoft services, thus making the task of finding solutions unnecessarily hard.

To summarize my total time learning the Power Platform was likely under 20 hours and during the implementation I used the documentation and YouTube videos to solve issues I encountered. It was only for the custom connector that I was required to seek additional resources then first thought external videos and then by asking for help from InBiz.

4.3 Limitations

One of the biggest limitations of this work is the difficulty in measuring the performance of the low-code solution as the service does not provide any metrics that can be related to the existing implementation. For InBiz this isn't to much of an issue as the act of attempting to implement the low-code version provides data on which aspects of the Power Platform will require developer input to be usable. For the scientific analysis this is more of a problem as the result of the implementation can only be measured by if it can satisfy the functional requirements and not by how it satisfies the non-functional product requirements. Additional problem with dependability and up-time is also out of the developers hand due to the SaaS business model used by the Power Platform.

A related issue to this stems from how the implementation work was conducted. As the time consumption for development was not part of the original research question no accurate time logging was used resulting in that no such data was available.

As the report only covers one platform and one implementation of a low-code solution the narrow scope of the research question is mirrored in narrow conclusions. The constant updates of both BC and the Power Platform also limits the lifespan of the results as updates may change the systems to a degree that the previous work is rendered irrelevant.

As the focus of this implementation was on the flows and not on canvas app, the majority of testing should have been done on the flows. But as discussed in 2.2.6, the testing of flows has a very low degree of automation. The used method was to run the flows after each change with an expected outcome. What coverage of possible cases this has is hard to assess especially as the flows are integrations between services where all possible outputs are not known.

4.4 Future work

This work raises questions for both academically interesting future work and business focused future work. From an academic point of view the field of low-code testing is very interesting as the users are unfamiliar with software testing and the tools are limited in what types of test can be done, therefore routines and methods to enable useful and systematic testing for software developed with low-code should be evaluated further. In the same way methods for performance measurements of low-code implementations could be explored further, and an especially interesting question is how to compare implementations from different LCDPs in a meaningful way.

Another question that would be interesting to investigate is what challenges appear when creating more complex applications within low-code tools. Previous research is often based around simpler applications, this to facilitate comparison between a wider range of LCDPs but this then leaves more issues unexplored.

For the business perspective an interesting area of future work would concern the business uses for Power Automate. The main problem is to elicit the system requirements for a well developed custom
connector. Such a process could be done both to find all the limitations of the standard connector and to find what customer needs such a connector would solve.
References


