On Evolvability in Tools for Data Analytics and Intelligence: the Power BI case

Geert Haerens
Antwerp Management School, Belgium
Engie nv, Belgium
Email: geert.haerens@engie.com

Herwig Mannaert
University of Antwerp, Belgium
Email: herwig.mannaert@uantwerpen.be

***Abstract*—Data analytics and business intelligence are essential to contemporary organizations. Tools, such as Microsoft Power BI are widely used and appreciated by businesses because they offer low-code/no-code features, putting data analytics capabilities in the hands of those who best know the data. A closer analysis of analytics products built with Power BI's low-code/no-code features reveals significant evolvability issues that can only be resolved by leaving the low-code/no-code path and adopting a coding approach. In this paper, we will analyze the evolvability issues using Normalized Systems theory and argue that the low-code/no-code features produce non-evolvable solutions.**

Keywords*—*Normalized Systems Theory; Evolvability; Low-Code/No-Code; Power BI.

I. Introduction

Data analytics, intelligence, and visualization used to be in the hands of IT, where businesses depended upon IT to provide them with views on their data, as businesses lacked the skills to exploit and use such solutions. Today, data analytics is made available to businesses via easy-to-use low-code/no-code solutions. You no longer need to be a technical expert to process, analyze and visualize your data. Click/configure gestures have replaced Structured Query Language (SQL) statements and other coding techniques, thereby democratizing Business Intelligence (BI) capabilities.

We note, however, that the "easy-to-learn Graphical User Interface (GUI)" (i.e., low-code/no-code) is not yielding evolvable solutions. We use the concept of evolvability as defined by Normalized Systems theory (NS) (see Section III) and observe that simple changes, such as adding or removing data from the source, can break the solution when the "easy-to-learn GUI" is used. Those issues can be resolved, but then we leave the low-code/no-code path and begin developing code.

Based on that observation, the main argument of this paper is that the low-code functionalities deliver non-evolvable solutions, unless you start to code. But what is the point of low-code/no-code in that case? As such, the paper highlights an anti-pattern.

In this paper, we use Microsoft Power BI as a case study to further explore our observations. This paper is structured as follows. We start by introducing the data analytics and intelligence tool, Power BI, in Section II that we will use to build our case, followed by an introduction to NS theory in Section III. Section IV presents an example that demonstrates evolvability issues in Power BI, and Section V reflects on their impact. The paper is wrapped up in Section VI.

To the best of our abilities, we have not found similar case studies, and we decided not to include a related work section. Instead, we have incorporated relevant literature where applicable across the sections.

II. Data Analytics and Intelligence: the Power BI Tool

Data analytics, intelligence, and visualisation are a billion-dollar market. An objective source stating the exact market value is difficult to find. The available data mostly comes from market research firms, published in pay-for-play analysis reports. A quick Google search, well aware of its low academic value, indicates a market value of approximately $10 billion and a projected increase of 50-100% by 2030 [1]- [3].

Power BI is a leader in Gartner's Magic Quadrant for Analytics and Business Intelligence Platforms [4]. In the same report, the data preparation capabilities of Power BI via Power Query are cited as its strongest feature. In [5], Gartner describes this capability as:

"Microsoft Power BI's strongest capability is data preparation. Its Power Query feature enables data analysts to blend disparate datasets via an easy-to-learn GUI and automatically detects data types and relationships. Columns are assigned a default data type during load. Business users can override the automatic relationship in the visual diagram model or relationships menu."

Other lauded capabilities are the creation of metics, the integration with Microsoft's data ecosystem, the inclusion of GenAI via Copilot, and its market presence and talent availability.

According to Microsoft [6]: "Power BI is Microsoft's business analytics platform that helps you turn data into actionable insights. Whether you're a business user, report creator, or developer, Power BI offers integrated tools and services to connect, visualize, and share data across your organization."

Power BI is part of the Microsoft Fabric data platform. The version of Power BI used by desktop users is called Power BI Desktop. Power BI Desktop allows you to connect to different data sources, transform the data as required, and visualize it on a dashboard using various visualization widgets (e.g., tables, bar charts, pie charts). A dashboard created in Power BI Desktop can be published and shared in Power BI Services. Power BI Desktop is used to create personal dashboards and publish them to Power BI Services. Power BI Desktop contains two subtools: Power Query and Power BI Desktop (confusingly having the same name as the main product).

The function of Power Query is to extract, transform and load (aka ETL [7]) data coming from different sources into a data format/store that can be used in Power BI Desktop to visualize and analyze the data. Data is read from a source and loaded into a tabular representation called a query. For that query, various transformations and filters can be applied via low-code functionality. Examples of transformations on the columns of a query include renaming, deleting, reordering, adding columns, and changing data types. Examples of filters include sorting and filtering on text, numerical patterns, or expressions. Queries can also be merged, pivoted and anti-pivoted. We refer to [8] for more elaborate analysis on how low-code can be used to perform ETL.

The function of Power BI Desktop is to visualize the data that is exposed by Power Query. The data is provided in tabular form and can be linked using the standard Entity-Relationship Model features, thereby creating a data model. The data can be sliced and diced via filters and filtering functionalities of the different visualization widgets. The dashboard can then be published in the Power BI Services environment and shared with other users.

Interaction with both tools is mediated by low-code capabilities. This is a key feature of Power BI and an important value proposition of the tool: anybody can do Business Intelligence - you don't need to be an IT expert. The user does not need to write code; they only need to configure transformation steps in Power Query and filters and widgets in Power BI Desktop. Both components have a programming language under the hood. In Power Query, this is M, a language for working with queries and lists; in Power BI Desktop, this is DAX, a language similar to the functions available in Excel.

III. Fundamentals of NS theory

Software should be able to evolve as business requirements change over time. In NS theory [9] [10], the lack of Combinatorial Effects (CE) measures evolvability. When the impact of a change depends not only on the type of change but also on the size of the system it affects, we refer to a CE. The NS theory assumes that software grows indefinitely and undergoes unlimited changes over time. Under those conditions, CEs harm software evolvability.

NS theory is built on the principles of classical system engineering and statistical entropy. In classical system engineering, a system is stable if it has bounded-input, bounded-output (BIBO) stability. NS theory applies this idea to software design: a limited change in functionality should produce a limited change in the software. In classic system engineering, stability is measured at infinity. NS theory considers infinitely large systems that undergo an infinite number of changes. A system is stable for NS if it does not have CE, meaning that the effect of change only depends on the kind of change and not on the system size.

NS theory posits four theorems and a method for enforcing them through software extension. The theorems are proven formally, yielding a set of required conditions that must be strictly followed to avoid CE.

A. NS Theorems

NS theory [9] is based on four theorems that dictate the necessary conditions for software to be free of CE.

- Separation of Concerns (SoC)
- Data Version Transparency (DvT)
- Action Version Transparency (AvT)
- Separation of States (SoS)

Violating any of these four theorems will lead to CE and, thus, non-evolvable software under change.

Consistently adhering to the four NS theorems is challenging for developers. NS theory proposes the use of software expansion mechanisms, where software templates, called elements, are manually crafted with close attention to NS theorems, allowing reusage by combining a generalized pattern and parameter-based input. The expanded elements can then be customized to meet the required specifications. All customizations can be collected via a process called harvesting, allowing the independent evolution of patterns and customizations via the rejuvenation process. More information on this harvesting and rejuvenation can be found in [11].

B. NS and Power BI

In the context of this paper's topic, we want to investigate the impact of changes to the source data that is consumed by Power BI, and see the effect this has on the configured ETL steps in Power Query, and the impact on data visualizations in Power BI Desktop. We also want to investigate the effect of making changes to the visualizations in Power BI Desktop.

More concretely, for the ETL part, we are interested in Power BI's compliance with DvT and AvT. Dvt and AvT are defined as follows:

- DvT: A data structure that is passed through the interface of a processing function needs to exhibit version transparency in order to achieve stability.
- AvT: A processing function that is called by another processing function needs to exhibit version transparency in order to achieve stability.

The processing function will be an ETL step, and the version transparency indicates that execution should continue to work and produce the expected result when the input data changes.

The simplest change is adding new data attributes to a data item. A processing step configured to operate on all data attributes of a data item should also process the new data attributes. Likewise, a processing function that should process only selected data attributes of a data item should still be able to process those attributes even if additional attributes are added to the data item.

Compliance with the NS theorems assures we don't introduce CE for changes that add something. It does not, directly, protect against the removal of something. If the source data contains less data than is functionally expected, this processing function will indeed break. Note the importance of "functionally expected". If the functionality is "I want to perform operation X on data attribute Y of data item Z", and data attribute Y is not in the source, it is normal for the task

to fail. If the functionality is ”I want to perform operation X on all data attributes of data item Z”, then the task should continue to work independently of the number of attributes of data item Z. Thus, the impact of a removal depends on the functionality it entails. Compliance with the NS theorems ensures that no CE are introduced when processing functions no longer require a certain data attribute. They should be able to ignore data attributes they do not perform actions on.

For the data visualization part of Power BI, we want to investigate the impact of adding data widgets to the dashboard canvas. As it is known that visualization widgets can interact with each other on the dashboard canvas, we will be looking more carefully at SoC.

- SoC: A processing function can only contain a single task in order to achieve stability.

The processing function will be a vizualization function in this case, and the tasks can be the filtering, configuration and interactions. As multiple tasks are linked to a visualization function, there is a high probability that CE will be introduced. In Section IV, we will test these scenarios to evaluate NS compliance of Power BI.

IV. Demonstrating Evolvability Issues in Power BI

As outlined in Section II, Power BI has two main components: Power Query (used for data transformations) and Power BI Desktop (used for data visualisation). In this section, we demonstrate that both components exhibit evolvability issues and argue that these issues could have been avoided. In Section IV-A, we will discuss a well-known evolvability issue in Power Query, and in Section IV-B, we will discuss a less recognized evolvability issue in Power BI Desktop.

A. Evolvability Issues in Power Query

In Power Query, a source $\mathbf{S}$ is transformed into $\mathbf{S_T}$ by means of a transformation function $\mathbf{T}$. $\mathbf{T}$ applies consecutive transformations $\mathbf{T_i}$ on $\mathbf{S}$, where the output of $\mathbf{T_i}$ becomes the input for $\mathbf{T_{i+1}}$.

$$\mathbf{S_T} = \mathbf{T}(\mathbf{S}), \qquad \mathbf{T} = \mathbf{T_n} \circ \mathbf{T_{n-1}} \circ \cdots \circ \mathbf{T_1}$$

The different transformations $\mathbf{T_i}$ are configured via Power Query’s low-code functionality: select a query/column/row, choose a transformation method from the menu, and provide additional transformation parameters. Each transformation automatically gets a name assigned and is visualized in the query properties plane. Power Query translates each configured transformation step into an M statement. The consecutive execution of the different M-statements constitutes the total query transformation $\mathbf{T}$. M is a proprietary Microsoft ETL programming language. Its features are rich and powerful, but limited to the Microsoft ecosystem.

Let us consider the following example: We want to create a basic ETL cleanup function $\mathbf{T}$ that takes all data from a source, puts it in a tableau representation, assigns column names based on the source data’s first row, treats all incoming data as text, and replaces empty and null values with a default value of **None**. We need our transformation function to anticipate changes to the source data, such as the addition or removal of columns, or the renaming of columns.

We use an Excel file as a source (see Figure 1). The file contains six columns, the first three are text, the fourth is a decimal number, the fifth is a date, and the last is a percentage. We use only Power Query’s low-code functionality to perform the necessary transformations. The amount of data is limited, as our objective is not to evaluate performance but evolvability.

The example is a relevant use case: a non-IT person who lacks coding skills wants all source files uniformly transformed, and they would use Power BI’s low-code functionality.

- Step 1: Select sheet 1 of the file = ”source.xlsx” as source data
- Step 2: Promote the first row of the source to column headers
- Step 3: Select all columns and transform the data type to ”Text”
- Step 4: Select all columns and replace the empty string with ”**None**”
- Step 5: Select all columns and replace the null character with ”**None**”

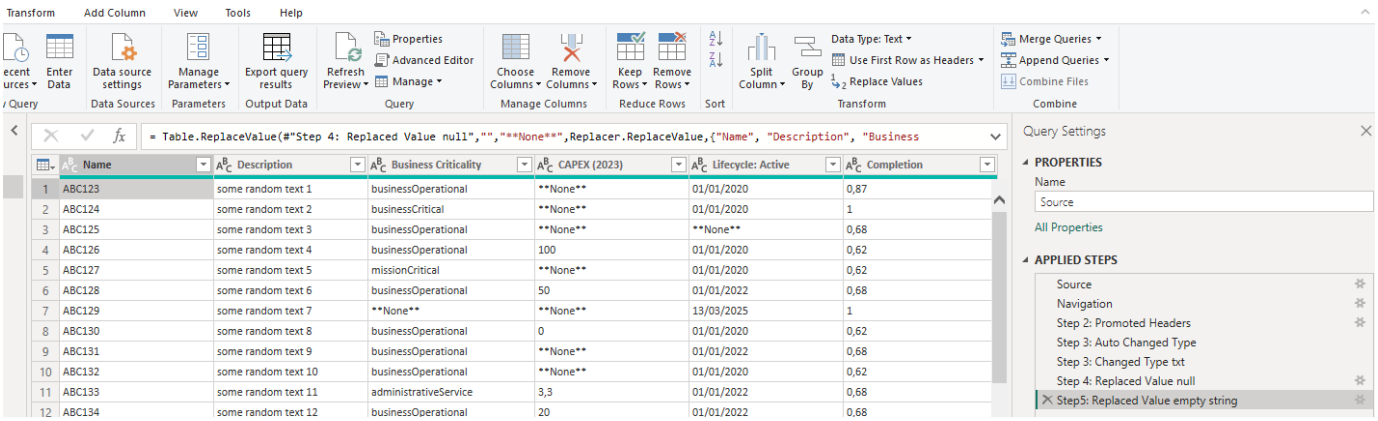
Name	Description	Business Criticality	CAPEX (2023)	Lifecycle: Active	Completion
ABC123	some random text 1	businessOperational		2020-01-01	87%
ABC124	some random text 2	businessCritical		2020-01-01	100%
ABC125	some random text 3	businessOperational			68%
ABC126	some random text 4	businessOperational	100	2020-01-01	62%
ABC127	some random text 5	missionCritical		2020-01-01	62%
ABC128	some random text 6	businessOperational	50	2022-01-01	68%
ABC129	some random text 7			2025-03-13	100%
ABC130	some random text 8	businessOperational	0	2020-01-01	62%
ABC131	some random text 9	businessOperational		2022-01-01	68%
ABC132	some random text 10	businessOperational		2020-01-01	62%
ABC133	some random text 11	administrativeService	3,3	2022-01-01	68%
ABC134	some random text 12	businessOperational	20	2022-01-01	68%

Fig. 1. Source file.

The applied steps and the result can be seen in the Query Properties plane (see Figure 2).

We now test if our transformation respects DvT and AvT. We add a new column to our source.xlsx file (a copy of the ”CAPEX (2023)” - see Figure 3) column, and we apply the same transformations. The result of the transformation is shown in Figure 4. We observe that, after the transformation, the new column is not the same data type as the others and contains empty strings.

We now rename a column in our source.xlsx file (rename ”CAPEX (2023)” to ”CAPEX (2025) - see Figure 5) and apply the same transformations. The result of the transformation is shown in Figure 6. We notice that the transformation produces an error.



	Name	Description	Business Criticality	CAPEX (2023)	Lifecycle: Active	Completion
1	ABC123	some random text 1	businessOperational	**None**	01/01/2020	0,87
2	ABC124	some random text 2	businessCritical	**None**	01/01/2020	1
3	ABC125	some random text 3	businessOperational	**None**	**None**	0,68
4	ABC126	some random text 4	businessOperational	100	01/01/2020	0,62
5	ABC127	some random text 5	missionCritical	**None**	01/01/2020	0,62
6	ABC128	some random text 6	businessOperational	50	01/01/2022	0,68
7	ABC129	some random text 7	**None**	**None**	13/03/2025	1
8	ABC130	some random text 8	businessOperational	0	01/01/2020	0,62
9	ABC131	some random text 9	businessOperational	**None**	01/01/2022	0,68
10	ABC132	some random text 10	businessOperational	**None**	01/01/2020	0,62
11	ABC133	some random text 11	administrativeService	3,3	01/01/2022	0,68
12	ABC134	some random text 12	businessOperational	20	01/01/2022	0,68

Fig. 2. Transformation Steps 1 to 5..

We conclude that the transformation function created in Power Query does not adhere to DvT and AvT, as changes to the data structure yield unexpected output.

Let us examine the M code generated by Power Query from our low-code configurations (see Figure 7). The M statement responsible for converting to the data type ”text” (line 6 in Figure 7) explicitly lists column names that match those in the initial data source. Logically, new columns are not accounted for, and if columns no longer exist or have been renamed, the transformation fails. The M statements responsible for the replacements of the empty string and null character (lines 7 and 8 in Figure 7) also contain an explicit list of column names that match the initial data source. Again, new columns are not accounted for, and if columns no longer exist or are renamed, the transformation fails.

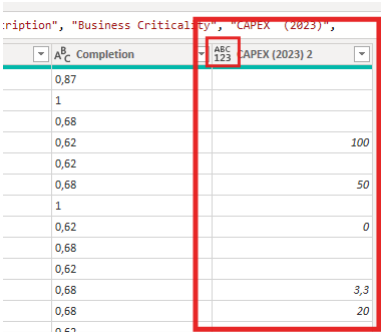
Name	Description	Business Criticality	CAPEX (2023)	Lifecycle: Active	Completion	CAPEX (2023) 2
ABC123	some random text 1	businessOperational		2020-01-01	87%	
ABC124	some random text 2	businessCritical		2020-01-01	100%	
ABC125	some random text 3	businessOperational			68%	
ABC126	some random text 4	businessOperational	100	2020-01-01	62%	100
ABC127	some random text 5	missionCritical		2020-01-01	62%	
ABC128	some random text 6	businessOperational	50	2022-01-01	68%	50
ABC129	some random text 7			2025-03-13	100%	
ABC130	some random text 8	businessOperational	0	2020-01-01	62%	0
ABC131	some random text 9	businessOperational		2022-01-01	68%	
ABC132	some random text 10	businessOperational		2020-01-01	62%	

Fig. 3. Adding extra column to source.

This issue is well-documented and recognized, even by Microsoft [12].

The M language provides a statement that returns the list of columns for a query. If the transformation function had first included this statement and used the result for Steps 3 to 5, we would have created a transformation function compliant with our requirements and thus data-version transparent. We will discuss this in more detail in Section V. The problem is, we can only do this by editing the M code directly. There is no low-code button/click/drag-drop or parameter available to configure the use of this statement.

Using Power Query’s low-code column transformation functionality, the initial source column names are preserved,



Completion	CAPEX (2023) 2
0,87	
1	
0,68	
0,62	100
0,62	
0,68	50
1	
0,62	0
0,68	
0,62	
0,68	3,3
0,68	20

Fig. 4. Result of the transformation after adding a column.

Name	Description	Business Criticality	CAPEX (2025)	Lifecycle: Active	Completion	CAPEX (2023) 2
ABC123	some random text 1	businessOperational		2020-01-01	87%	
ABC124	some random text 2	businessCritical		2020-01-01	100%	
ABC125	some random text 3	businessOperational			68%	
ABC126	some random text 4	businessOperational	100	2020-01-01	62%	100
ABC127	some random text 5	missionCritical		2020-01-01	62%	
ABC128	some random text 6	businessOperational	50	2022-01-01	68%	50
ABC129	some random text 7			2025-03-13	100%	
ABC130	some random text 8	businessOperational	0	2020-01-01	62%	0
ABC131	some random text 9	businessOperational		2022-01-01	68%	
ABC132	some random text 10	businessOperational		2020-01-01	62%	
ABC133	some random text 11	administrativeService	3,3	2022-01-01	68%	3,3

Fig. 5. Renaming a column in the source.

violating DvT and making the column transformation non-evolvable. However, the M language does include the required statements and functions to perform column transformations using DvT, but you need to code them explicitly. This leads us to conclude that Power Query’s low-code functionality, by default, produces non-evolvable transformation functions. To get evolvable transformation functions, you need to code,

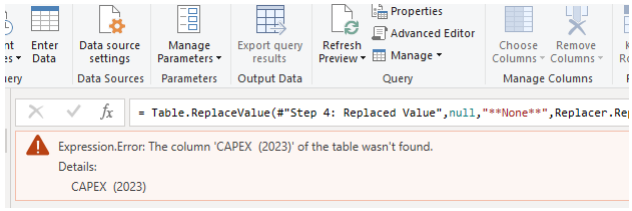


Fig. 6. Result of renaming a column in the source.

```
let
    Source = Excel.Workbook(File.Contents("C:\Users\ghaer\Downloads\Source.xlsx"), null, true),
    #"Sheet 1_Sheet" = Source{[Item="Sheet 1",Kind="Sheet"]}[Data],
    #"Step 2: Promoted Headers" = Table.PromoteHeaders(#"Sheet 1_Sheet", [PromoteAllScalars=true]),
    #"Step 3: Changed Type" = Table.TransformColumnTypes(#"Step 2: Promoted Headers",{{"ID", type text}, {"Name", type text}, {"Business Criticality", type text}, {"CAPEX  (2023)", Int64.Type}, {"Lifecycle: Active", type date}, {"Completion", type number}}),
    #"Step 3: Changed Type1" = Table.TransformColumnTypes(#"Step 3: Changed Type",{{"ID", type text}, {"Name", type text}, {"Business Criticality", type text}, {"CAPEX  (2023)", type text}, {"Lifecycle: Active", type text}, {"Completion", type text}}),
    #"Step 4: Replaced Value" = Table.ReplaceValue(#"Step 3: Changed Type1","","**None**",Replacer.ReplaceValue,{"ID", "Name", "Business Criticality", "CAPEX  (2023)", "Lifecycle: Active", "Completion"}),
    #"Step 5: Replaced Value1" = Table.ReplaceValue(#"Step 4: Replaced Value",null,"**None**",Replacer.ReplaceValue,{"ID", "Name", "Business Criticality", "CAPEX  (2023)", "Lifecycle: Active", "Completion"})
in
    #"Step 5: Replaced Value1"
```

Fig. 7. Generated M code of the transformation.

making Power Query a low-code platform, a contradictio in terminis.

B. Evolvability Issues in Power BI Desktop

When Power BI Desktop starts, it first makes the data transformed by Power Query available as tables. This data can be visualized on the dashboard canvas by selecting visual widgets and configuring them to represent the required data. The low-code aspect of Power BI Desktop lies in the configurability of the visuals, their filtering, and interactions. Via Filters, the data linked to the visuals is sliced. Via interactions, the data made available for visualisation on a visual can be influenced by the selection of data on another visual.

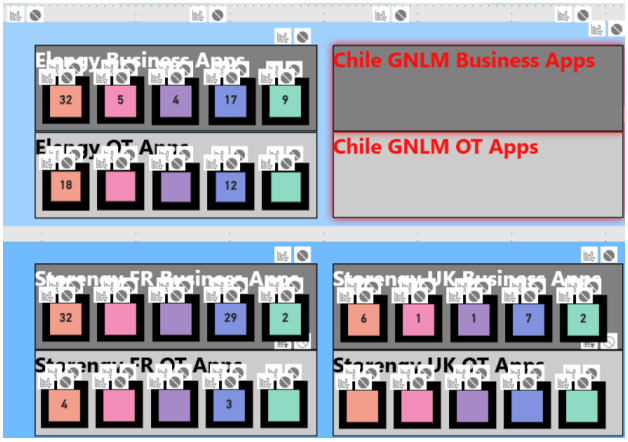


Fig. 8. Configuring Interactions: the small white/gray rectangles need to be clicked on for each activated or deactivated interaction with a newly added visual.

The interactions make the dashboard interactive and dynamic. By default, all visuals put on the modelling canvas interact with all existing visuals on the canvas. And herein lies the evolvability issues. It is not always desirable for visuals to interact. If you do not want selections on a visual to influence the other visuals on the canvas, you will have to disable the interactions on all other visuals manually. If you have ten visuals on your canvas, adding an eleventh means you need to disable interaction between the new and the existing visuals. The more visuals (i.e., the size of the system), the more effort is required to add a non-interacting visual. This is a clear example of a CE. The more visuals on the canvas, the more effort it takes. Figure 8 shows an example of a dashboard with over 100 visuals on it. Adding extra information to this visual is a tedious, error-prone task that can lead to unexpected dashboard behaviour and misinterpretation of data, the opposite of what a dashboard should be.

The root cause of this CE is a violation of the SoC. The introduction of a visual on a canvas should not be combined with configuring its interaction with existing visuals. Also, the default behaviour - interaction ON with existing visuals - results in the need for configuration. It would have been more logical to set this behaviour to default OFF and perform work proportional to the type of change: the need to interact with seven out of a hundred visuals requires an effort of seven, not a hundred.

V. Discussion

In this section, we consider the implications of the demonstrations' results from the previous Section. We begin with a framing of the demonstration and results, then discuss Power Query, followed by Power BI Desktop.

A. Framing of the Demonstration

It is important to frame the demonstration appropriately. The scenario we described assumes that Power BI is used for ETL and data visualization. There are, of course, scenarios in which a connection to a data source is made without any transformations occurring because the data is cleansed and transformed via other tools, and the end user connects only to the end result. But Power BI can do both, and over 30 years of personal experience in IT tooling leads us to believe that end users tend to exploit personal productivity tooling to the fullest, even if this results in highly unsustainable solutions.

This opinion warrants investigation using more rigorous methods. Additional research could be done on how frequently the type of issue we demonstrated appears in operational Power BI Dashboards. From the population of Power BI Dashboards used at the author's company, a sample could be drawn. In each sample, a change could be made to the data sources, similar to the one used in this paper: add a column to a source file and rename a column. The number of times this would result in a data load error would indicate how often the Power BI Dashboard creator has relied solely on low-code features to configure the ETL.

B. Implications of NS noncompliance in Power Query

When you do not know how to customize the M-code in Power Query, and thus enhance/replace what Power Query generates/expands, your ETLs will fail when there are changes in the source data. As you have no control over the source data, this impact is inevitable.

Microsoft may have intended Power BI as a tool for analyzing and reporting data, but the result is unsustainable if only the low-code/no-code features are used. We argue that such tools, seemingly designed for non-IT personnel, can

create tension between Business and IT. The tool is put in the hands of non-IT people, who will know better than IT how to interpret the data, but they are unaware of the tool's non-sustainability. When work is shared via publication and issues become visible to a large audience, IT is typically asked to resolve them, leading to rework and additional coding that cost more than initially expected and potentially contributing to Business-IT alignment tensions. The author (working for a multinational utilities company with over 98,000 employees) has witnessed such cases many times but recognises that this "anecdotal evidence" warrants further investigation. The research we propose would examine whether democratising IT capabilities (such as ETL and Data Visualisation) helps or adds tension to Business-IT alignment. After all, these tools are also socio-technical by nature [13].

In Section IV-A, we already touched on possible solutions. Why has Microsoft decided not to use the M language capabilities to expand low-code configuration into evolving code? The language is sufficiently rich to comply with AvT and DvT. A chain of ETL steps can be made SoS-compliant by implementing proper error handling, and because each transformation step addresses only one concern, SoC is also achievable. However, these features are available only to seasoned M-coders, not to low-code users.

C. Implications of NS noncompliance in Power BI Desktop

In Section IV-B, we saw that the default ON for interactions between visuals introduces a CE. Again, we ask why Microsoft made this design decision. In Power Query, the issues could still be fixed via M coding; in Power BI Desktop, there is no equivalent to steer the behaviour of the visualization widgets. Power BI Desktop provides the DAX language, but it is used to further process, filter, and extract information from the data. It does not allow control over the visualization widget's behaviour. There are other data visualization tools available, such as Graphana, in which the configuration of a visualization widget is stored in a JSON file. This opens the door for visualization widget expansion. In Power BI, this is not possible. We also suggest additional research in this area: which tools support expansion-based visualization widgets, and what should the structure of such a widget be to exhibit evolvability with respect to the addition/removal of widgets on a dashboard?

VI. Conclusion

The overarching argument we present in this paper is that using Power BI's low-code features yields a non-evolvable solution. We began by situating the topic in Section I, followed by an introduction to Power BI in Section II and to NS in Section III, as NS is the tool we use to evaluate evolvability. By means of demonstrating evolvability issues in Section IV, we present our arguments in Section V and point toward additional research.

References

[1] Apps run the world, [Online], Available: https://www.appsruntheworld.com/top-10-analytics-and-bi-software-vendors-and-market-forecast, [retrieved: March, 2026]

[2] Mordor Intelligence, [Online], Available: https://www.mordorintelligence.com/industry-reports/data-visualization-market, [retrieved: March, 2026]

[3] Grand View Research, [Online], Available: https://www.grandviewresearch.com/industry-analysis/data-visualization-tools-market-report, [retrieved: March, 2026]

[4] A. Ganeshan, E. Macari, J.O'Brien, K. Schlegel, and C. Long, "Magic Quadrant for Analytics and Business Intelligence Platforms," Gartner, IDG00822218, June 2025.

[5] A. Ganeshan, E. Macari, J.O'Brien, K. Schlegel, and C. Long, "Critical Capabilities for Analytics and Business Intelligence Platforms," Gartner, IDG00822221, June 2025

[6] Microsoft Learn, [Online], Available: http://learn.microsoft.com/en-us/power-bi/fundamentals/power-bi-overview, [retrieved: March, 2026]

[7] C. Ravi,"ETL (Extract, Transform & Load) Automation," International Journal of Emerging Trends in Computer Science and Information Technology, Volume 6, Issue 1, pp. 51-54, 2025

[8] T. Pellekoorne, "Building a Low-Code Platform for versatile Data Integration," Diss. Master's thesis, Technische Hochschule Mittelhessen (University of Applied Science), 2024

[9] H. Mannaert, J. Verelst, and P. De Bruyn, "Normalized Systems Theory: From Foundations for Evolvable Software Toward a General Theory for Evolvable Design," ISBN 978-90-77160-09-1, Koppa, 2016.

[10] H. Mannaert, J. Verelst, and K. Ven, "The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability," Science of Computer Programming, Volume 76, Issue 12, pp. 1210-1222, 2011.

[11] G. Haerens and H. Mannaert, "On the operationalization of composable architecture by means of NS theory," In PATTERNS 2025: The Seventeenth International Conference on Pervasive Patterns and Applications, pp 23-30, 2025.

[12] Microsoft Community, [Online], Available: https://community.fabric.microsoft.com/t5/Desktop/How-to-Add-a-New-Column-to-an-Existing-Power-Query-Without/td-p/4104466, [retrieved: March, 2026]

[13] N. Prinz, C. Rentrop, and M. Huber. "Low-Code Development Platforms-A Literature Review," AMCIS, 2021