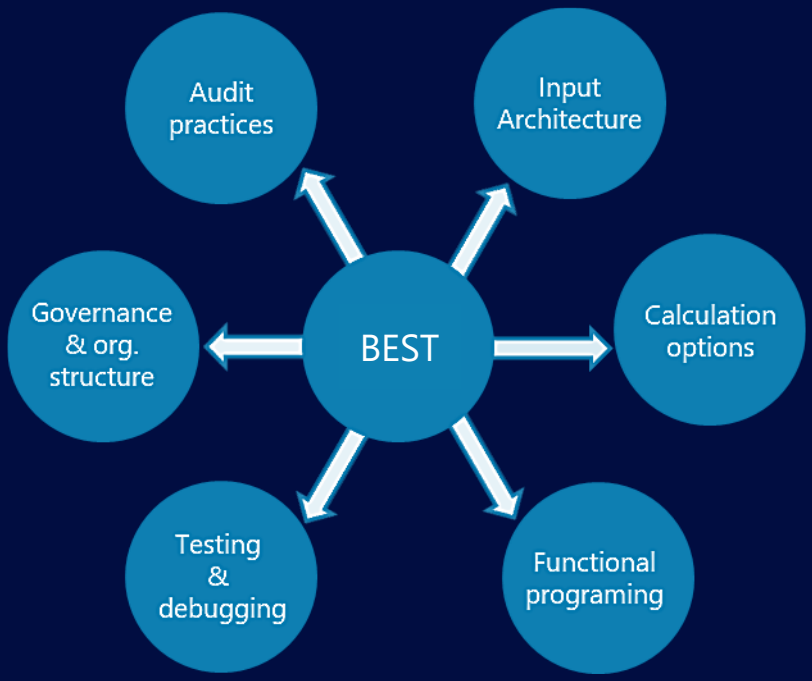
# The BEST Framework

## Comprehensive guide to financial modelling with modern spreadsheets




Contributor(s):

1. Viswanathan M B

info@profectus.in

https://profectus.in

*Version 1.0 – Published on August 21, 2025*



## *Acknowledgments*

*The author would like to acknowledge the financial modeling community, in general, that have actively participated in discussion related to modeling with dynamic arrays. And specifically, the author would like to acknowledge the following two people, whose active contributions to the discipline have inspired the author:*

*Craig Hatmaker, Founder of BXL (Craig Hatmaker | LinkedIn)*

*Peter Bartholomew, Technical Fellow at NAFEMS (Peter Bartholomew | LinkedIn)*

*Note that the acknowledgement does not mean that those who have been acknowledge have endorsed the author, publisher, or the contents of this material.*

# Preface

The introduction of Dynamic Arrays and the LAMBDA function mark a significant evolution in modern spreadsheets such as MS Excel, Zoho Sheets, and Google sheets.

These functionalities empower financial modelers to build more powerful financial models — models that are transparent, robust, scalable and work like an app. However, this new paradigm demands a different set of best practices compared to legacy scalar modeling techniques. And the absence of a clearly documented framework is a key barrier for its widespread adoption.

This guide provides that framework. It is built on four foundational pillars that we call the **BEST** principles:

**Balanced**: Financial modelers must find optimum balance between conflicting priorities.

**Efficient:** The model must be fast and responsive enough to aid proper decision making.

**Stable:** The model must remain functional and reliable when users interact with or update the data.

**Thorough:** The model should not have loose ends, which creates spreadsheet risk.

Each guidance in this first version of the document is a product of first principles thinking, established software design principles, and our extensive experience at Profectus in providing robust Dynamic Array based solutions to the clients over the last five years. Every guidance is accompanied by a clear rationale.

But the future version of this document is expected to be driven by feedback, deliberations and contributions from more professionals, worldwide.

We refer to this version as a guide and not a standard because it focuses on providing proper principles rather than imposing rigid rules.

To that extent, we distinguish our guidance into two parts:

1. **Prescriptions**: Critical practices that are essential to avoid tech debts and spreadsheet risks.
2. **Recommendations**: These practices enhance the user experience or performance of the financial models.

The guide also includes discussions of alternative views, presented in the Appendix, to encourage healthy debate and critical evaluation of alternatives.

This guide is intended for experienced financial modelers, who are in the best position to determine the appropriate approach for each modeling challenge. We neither insist that dynamic array is the only way forward for future financial modeling, nor do we prescribe when to use them.

Instead, for those who choose the modern paradigm, the guide delivers practical guidance on input architecture, design principles, validation and debugging, governance, and audit. Ultimately, it provides the framework needed to fully leverage dynamic arrays and LAMBDA, while addressing the real-world concerns that arise with this new paradigm.

# Additional resources

This guide is accompanied by sample financial models that illustrate the robust architecture and design principles discussed in this guide.

These can be accessed from The BEST corner in our website:

The BEST corner | Profectus

It will also have the latest version of the file. You can access the page either by clicking on the link above, or by scanning this QR code.

# Table of Contents

# Chapter 1: BEST framework – Foundation for modern financial modeling

Traditional financial modeling in spreadsheets has followed a cell-by-cell, copy-and-paste methodology for decades. While this approach has served the industry well, it comes with inherent limitations: models become unwieldy as they scale, formulas are prone to copy-paste errors, and maintenance becomes increasingly complex as business requirements evolve.

Dynamic arrays (DA) and the LAMBDA functionality fundamentally change this paradigm. Modelers can now create compact, powerful expressions that handle entire datasets as single units. This shift enables financial models to behave more like software applications—robust, scalable, and self-maintaining.

However, implementing reliable financial models with dynamic array and Lambdas (DA Model) requires a 360° overhaul of financial modeling practices that are widely followed.

## The need for a 360° approach

A powerful and scalable formula handling complex logic needs a strong – yet friendly – input architecture to work reliably. And to ensure there is consistency when the complex logic is reapplied, it must be encapsulated in a LAMBDA function.

However, since the LAMBDA can be argued to create a "black box," its quality and reliability must be assured. This requires applying strong functional programming design principles and ensuring that the Lambdas are thoroughly tested, validated, and bugs, if any, are identified and fixed before they are used.

But these responsibilities cannot be left to individual modelers. The financial modeling function itself must be reorganized with strong governance and organization structure, **transforming it from an individual craft into a disciplined organizational process**.

Finally, these changes will count for nothing if third parties cannot rely on the model. This requires that modellers still care about making their models auditable, and that auditors modify their approach to focus on validating functions over checking formulae.

This involves more than learning new functions. It requires a different approach that focuses on cohesion between all the elements to achieve the true potential of DA models with far lesser risk.

This guide provides guidance across these aspects to ensure such cohesion in each of the next few chapters.

## The BEST principles

This guide is built around four foundational principles that form the acronym BEST: Balanced, Efficient, Stable, and Thorough. These principles work synergistically to create models that are both powerful and practical.

The table below explains these principles in brief.

| | |
|---|---|
| **B**alanced | The Balanced principle recognizes that financial modeling with spreadsheet involves constant trade-offs, such as these:<br>• Pure technical optimization often conflicts with user experience.<br>• Complete transparency can compromise performance.<br>• Future proofing may increase the time to roll out models<br>The guidance in this document consistently seeks the optimal balance point for each such decision. |
| **E**fficient | Models must open quickly and perform calculations within acceptable timeframes to truly support decision-making. Therefore, this guide favours efficient use of human and computing resources.<br>It discourages practices that favours simplicity in calculations at the cost of creating bloated files with sluggish performance. |
| **S**table | Stability demands that models remain functional despite common user interactions or events, such as updating data, inserting or deleting rows, or sorting and filtering tables. This guide favours approaches that allow a fully built model to automatically accommodate these changes without breaking, preserving its integrity and reliability.<br>Stability in dynamic array models extends beyond formula resilience. It encompasses the stability of the underlying data architecture, the consistency of function libraries, and the predictability of model behaviour across different usage scenarios. |
| **T**horough | A scalable and stable financial model cannot have loose ends that depend on end user actions. Therefore, the financial modelers should be thorough with their approach. Financial modelers should not bypass complexity to merely to avoid effort and exploit the full capabilities of modern spreadsheets to handle such complexities. The guide clearly segregates a financial modeler from the end user. |

## How BEST Principles Compare to Other Standards

While standards like FAST, SMART, and the ICAEW Financial Modelling Code provide a crucial foundation for good modeling, they were largely formulated before the advent of Dynamic Arrays and LAMBDA. As a result, they focus on best practices for a legacy, cell-by-cell approach. The BEST framework, in contrast, is built exclusively for the modern DA paradigm.

While we align with the *spirit* of these standards—to ensure financial models are reliable—the context of DA modeling results in significant philosophical and practical deviations. Further, to keep the focus sharp on DA model, we have deliberately ignored topics like use of VBA where we find existing industry practices to be acceptable.

Here are the key areas where this guide deviates from others, either in philosophy or practice.

**1. Framework for the system and not the individual**

The key differentiator of BEST framework is that it takes a holistic view of the financial modeling *function*, treating it as an integrated system of people, processes, and technology. It puts the onus of making the financial model reliable on the system. This is a departure from other standards that focus primarily on the best practices of an individual modeler.

**2. Treats formula logic as sacrosanct and not malleable**

Legacy standards insist that formulas should be simple enough for any user to understand and modify. The BEST framework takes a starkly different view and insists that a complex formula, especially Lambdas, should not be modified, ever, once approved.

**3. Different philosophy to transparency**

Other standards place simplicity as the cornerstone of transparency. This guide views transparency as keeping the code accessible and in ensuring that the lambdas are pure i.e. not affected by any value other than what is passed as an argument to it.

**4. Emphasizes validation over comprehension**

This guide acknowledges that creating formula that satisfy the BEST principles would require writing logics that may be extremely difficult for standard users to comprehend. It emphasizes that standard users and auditors should focus on validating the Lambda by applying it on familiar data sets rather than trying to understand the model logic.

**5. It Replaces Ambiguous "Rules" with Clear Prescriptions and Recommendations**

Where some standards prescribe a single set of rules that can be broken with justification, the BEST framework provides a clearer, two-tiered framework:

(i) **Prescriptions:** Non-negotiable guidelines that needs to be followed in totality to avoid loose ends and accumulating tech debt.
(ii) **Recommendations:** A set of best practices where professionals can apply their judgment with a conscious choice that it can affect user experience or hamper performance.

## Summary

Dynamic array and Lambdas create a new paradigm in financial modeling. They enable creating models that are robust, scalable and work like an app.

But realising its true potential requires more than mastering a new set of spreadsheet functions. It requires a 360° overhaul of the entire financial modeling function covering working practices in spreadsheets, software development and testing practices, organization structure and even the audit process. Financial modeling will need to transform from being an individual craft to a disciplined organizational practice.

The BEST framework provides the philosophical foundation for establishing such a robust practice:

- Balanced: Navigate trade-off between user experience, transparency, performance, future-proofing and immediate business needs.
- Efficient: Prioritize practices that avoid slow loading and sluggish performance.
- Stable: Ensure models do not suffer because of common user actions
- Thorough: Place accountability on the modeler to do the heavy lifting to prevent any loose ends.

The chapter sets the tone and scope for the guide: subsequent chapters translate BEST framework into concrete practice.

In the next chapter, we turn to the first layer of DA models: The input architecture.

# Chapter 2: Architecting inputs for stability

Input architecture is an extremely critical part of DA models. The decisions made at this stage determine whether a model can scale efficiently, remain stable under user interactions, and support the advanced calculation techniques discussed in later chapters.

A lack of dynamic front end within spreadsheets makes it extremely difficult to satisfy the dual objective of a robust backend and very friendly front-end. This chapter tries to balance the two so that the compromises neither affect the stability of the model, nor be a major deterrent for user adoption.

## Scope

The guidance in this chapter focuses specifically on practices where inputs are directly entered into the spreadsheet. While this guide encourages importing data directly from source systems and remains open to use of forms and PowerApps, which enhance user experience and control, these topics fall outside the scope of this version.

Certain standard practices for input, including, keeping the input in a separate cell and distinguishing them with different colour codes, continue to be relevant while building financial models with dynamic arrays.

However, certain additional guidelines are relevant for DA models, which are outlined, here.

## Definitions and Terminologies

In the context of this guide, the input variables are classified into four categories. The terminology that is used, and their meaning are as under:

**(i)** **Scalars:** These variables have only one value and is kept in a single cell. For instance, the WACC variable in most financial models are static across time and is, thus, an example of a scalar input.

**(ii)** **Vectors or one-dimensional data:** These variables have multiple values and are presented as a list of values either in a single row or single column. The size of the list can increase (or, in fewer cases, decrease). For example, when projecting long-term financial data, analysts may apply distinct sales growth rates to each forecasted period. As the number of periods under consideration increases, the list will increase in size. The forecasted sales growth rates, in this case, is a vector variable. Vectors are very common in financial modelling.

*Exhibit 1: Example of vector data set*

| | 2XX5 | 2XX6 | 2XX7 | 2XX8 | 2XX9 |
|---|---|---|---|---|---|
| Sales growth rate | 11% | 10% | 8% | 8% | 8% |

**(iii)** **Multi-dimensional data:** These refers to variables that have two or more dimensions and the data can grow on any of the dimensions. For example, if an analyst needs to input bill of materials (BOM) data for a multi-product company, the input may spread across multiple rows and columns as shown in Exhibit 2

*Exhibit 2: BOM – an example of a multi-dimensional data*

| Product | Chem 1 | Chem 2 | Chem 3 | Chem 4 |
|---|---|---|---|---|
| Product A | | 150 | | 150 |
| Product C | 180 | 190 | 150 | |
| Product D | 120 | | 110 | |
| Product E | 170 | | | |
| Product F | | | 60 | |
| Product G | | | 170 | |
| Product I | 80 | | 110 | |
| Product K | | | 120 | |
| Product L | | | 70 | |
| Product N | 100 | | | |

**(iv)** **Single group, mixed nature data:** These are variables that belong to a same group or list, but the nature of one or more-line items in the group varies from the other items. For instance, in a funding schedule, all different source of capital can be visually presented as a list. But, often, the weight of one of the sources is kept as a balancing figure (100% - weight of all other source) while other sources might be entered as an input.

*Exhibit 3: Funding schedule – an example of single-group-mixed-nature data*

| Source | K | Weight |
|---|---|---|
| Debt | 9% | 40% |
| Preferred shares | 12% | 20% |
| Equity | 18% | 40% |
| **Total** | 13% | 100% |

In this chapter, and subsequent chapters we shall also be using the term "structured tables".

**Structured Tables:** It refers to a range that is marked as a table in the spreadsheet. In MS Excel it is done by marking the area as a **"Table"** from the insert menu or by using the **CTRL + T** shortcut key combination. These tables are not just formatted to appear as table for the human eye, but they are recognized by the software as such.

## 2.1 Architecture for scalar input variables

***Prescriptions***:

**P1.** **Keep scalar inputs in a separate cell, with proper labels and colour coding to make it easier to identify.**

*Rationale*

- **Ensures centralized updates**: Keeping scalar variables in a separate cell ensures that when the values of those variables are changed, such a change in consistently applied across the file without having to repeat the same action again. Further, keeping the values in a separate cell makes it possible to create data tables for performing sensitivity analysis.
- **Aligns with current practices:** This is in complete alignment with commonly recommended best practice documents such as FAST or SMART

***Recommendations:***

**R2. All scalar input should be kept in a separate sheet that should be protected and all cells other than the input cells should be locked for editing. The sheet should not allow deleting of rows or columns.**

*Rationale*

- **Reduces the chance of inadvertent deletion of cells**: The main technical risk with a scalar input cell is that it can get deleted and throw #REF! error, in linked cells. Preventing deletion of rows or columns reduces this risk.

Readers should note preventing deletion alone is not fail proof for #REF! error. When a cell that is cut from elsewhere is pasted in an unlocked cell, it causes the same problem.

**R3. Unit conversion factors such as number of months in a year, number of metres per kilometre, should be kept in a separate set of cells but the cells should be locked for editing.**

The other alternative is to directly store these values in a name under the Name Manager. Since the names are out of sight, there is less chance of inadvertent changes. But this reduces transparency of the model and does not fail proof any intentional manipulations of the variables.

*Rationale*

- **Reduces ambiguities:** Although unit conversion factors themselves do not vary, ambiguities can occur when these values are entered directly into a cell. For instance, multiplying a volume by 12 may not clearly indicate whether monthly figures are being annualised or quantities are being converted from dozens to units. By placing such constants in a separately labelled cell, the purpose becomes clear and transparent to the reader.
- **Prohibits unwarranted changes:** The purpose of keeping unit conversion factors in a separate protected sheet, away from standard input, is to ensure that people do not inadvertently change their values.

**R4. Cells with scalar variables should be named. Such names should be brief, yet descriptive. For instance, a cell containing the WACC value can be named as _WACC.**

*Rationale*

- **Easy access:** Cell names can be easily called with help of ***IntelliSense*** from any sheet by the modeler without having to navigate to the source sheet.
- **Improved readability:** When a formula uses a descriptive cell name, it makes it easy for users to understand what the cell represents without having to navigate away to the source sheet and read the labels.
    - For instance, let us say the WACC is in input sheet C8. An end user who is trying to understand the logic can easily understand what `=1/(1+_WACC)` is doing

without having to navigate to the input sheet and checking what does C8 represent.

- **Reduces vulnerability in programs:** When you are referring to a cell in VBA or Scripts, references to cell names are not vulnerable when rows/columns are inserted or deleted. This improves integrity of the model.

Naming cells has some drawbacks, too; users must follow clear naming conventions to prevent confusion, especially in models with many variables.

This version of the document does not cover naming best practices.

## 2.2 Architecture for vector input variables

***Prescriptions***:

**P2. Vector inputs must be organized as a structured table. In the case of time series data, this would involve the timelines to be placed as row heading, flowing from top-to-bottom, while the fields would be placed across columns.**

*Rationale*

- **Dynamic Expansion**: Financial models are often updated, increasing the number of items in a vector. Inputs in dynamic array formulas must update automatically as new items are added. Using a structured table ensures all related formulas update when the list expands.

Users, who are mostly accustomed to seeing timelines placed horizontally and line items placed vertically, would need to reorient in the case of table architecture. In Appendix 1, we discuss alternative arrangement involving placing timelines horizontally, along with its pros and cons.

**P3. Avoid storing vector input in traditional range of cells to be used with TRIMRANGE function or trim operators in formula.**

*Rationale*

- TRIM operators introduce higher level of spreadsheet risks that have been detailed in para P17.

**P4. Sheets with input tables must not be protected with "Protect sheet" feature**

*Rationale*

- **Prevents dynamic expansion of table:** As on the date of last update of this document, structured tables do not expand in protected sheets. Therefore, the sheets should be unprotected to ensure the model is dynamic.

**P5. Data sets that are not compatible across the key dimension should be kept in separate tables. Forceful grouping of incompatible data set in a single table can lead to confusion, inefficient referencing, and errors in dynamic array calculations.**

As an example, historical financials and the assumptions for forecasts differ from each other in terms of nature — historical data does not exist for future periods, while assumptions are irrelevant or non-existent for the past. As shown in Exhibit 4, this results in unwanted blank cells and inconsistent columns.

*Exhibit 4: Improper grouping of incompatible data in single table*

| Year | Sales | Expected growth rate | Gross Profit | Expected margins |
|---|---|---|---|---|
| 2XX3 | 300 | | 146 | |
| 2XX4 | 327 | | 152 | |
| 2XX5 | | 8% | | 48% |
| 2XX6 | | 7% | | 46% |

*Rationale*

- **Enhanced Clarity**: Each table has a clear purpose, making it easier for users to understand and update specific datasets without navigating unrelated data.
- **Improves scalability:** Since all columns belong to one group, adding more columns will not disrupt the existing structures or formula dependent on the table.

  For example, if a table is exclusively used to hold historical income statement data and analyst needs a formula that would return all historical income statement line items for year 2023, it can be written as follows:

  ```
  =Drop( FILTER(tbl_IS, tbl_IS[Year]=2023),, 1)
  ```

  Since all the columns would be pertain to historical income statement, users wouldn't need to manually select individual columns, which would be laborious.
- **Reduces spreadsheet risk:** When unrelated data are forced together, it would lead to many blank cells or reference errors, which may lead to incorrect computations. By avoiding such forced grouping, the risk of such error is reduced.

***Recommendations:***

**R5. Structured tables should be given brief and descriptive names replacing the default names given by MS Excel.**

*Rationale*

- **Improves readability:** The default names given by MS Excel are non-descriptive. A custom name that is descriptive, therefore, would make it easy to understand a formula. For instance, a cell reference such as `=IS_History[Gross Margin]` clearly tells us that we are extracting gross margin from historical period and not forecasts.
- **Easy access:** When the names are descriptive, it is easy to recollect and type the table name directly in a cell, further aided by ***IntelliSense***, without having to manually navigate to each table and select them.

**R6. Modelers are encouraged to use column(s) to perform row level validation of input to check whether all input columns are correctly filled.**

For instance, let us say there is an IS_Table with historical financials with nine fields, including one for the timeline and another eight for eight different line items, we may insert a 10th column to check whether all the first 9 columns are filled with a formula such as this:

```
=COUNT(IS_Table[@Year]:[@PAT])=(COLUMNS(IS_Table)-1)
```

This formula would return TRUE only if all the input columns are duly filled. Exhibit 5 shows one such implementation with a smaller example.

*Exhibit 5: Input table with a validation column*

| Year | Volume | Price | IsValid |
|---|---|---|---|
| 2XX6 | 30 | 280 | TRUE |
| 2XX7 | 33 | 290 | TRUE |
| 2XX8 | 36 | 300 | TRUE |
| | | | FALSE |

*Rationale*

- **Prevents stray rows from affecting calculation:** When working with tables it is common that an end user may add additional rows to the table either inadvertently, or intentionally to serve as a place holder. The formulae referring to the table can filter out such stray rows using the validation column.

**R7. Modelers are encouraged to use additional calculated columns to perform multi-table validations**

For instance, one may add a column in income statement to check whether all columns of balance sheet for the same period have been filled up.

*Rationale*

- **Allows synchronization of calculations:** When a calculation requires data from multiple tables (say asset turnover ratio), the multi-table validation columns allow us to filter out data sets when corresponding data is not available in another data set.

**R8. This guide does not discourage using any calculated columns, which may aid end users in the input process. However, such column shall not be used in core calculation sections (refer prescription P18)**

For instance, a calculated PAT column helps analysts check the accuracy of entered financials by comparing computed number with reported number. Likewise, using a moving average of sales growth can guide arriving at a projection assumption. Including these calculations alongside input data can be beneficial.

*Rationale*

- **Balances user experience with rigid structure:** Allowing calculated columns in input table to allow figures flowing elsewhere improves the end user experience while keying information despite the rigid architecture.

**R9. The guide discourages including homogenous data of multiple entities in a single input table.**

For instance, while technically it does not create an issue if all historical data including income statement, balance sheet, and cash flow items are kept in one table, modelers should consider keeping them in separate tables.

*Rationale*

- **Improves user experience during input:** When data of multiple entities are mashed up in a single table, it is likely to be overwhelming to enter the input data in the table.

This specific guidance pertains only to tables where input is fed. It does not concern itself with tables that are imported from external sources using any ETL tools.

## 2.3 Architecture for input of multi-dimensional data

Multi-dimensional data must also be organized as a structured table. And all the prescriptions and recommendations that applied to organizing vector as a structured table also apply to them. This section lays out additional guidelines that are exclusively relevant for multi-dimensional data.

***Prescription**:*

**P6. Variables that have three or more dimensions must be organized in the long form in a single table. Avoid creating multiple two-dimensional tables.**

For instance, a company may provide segmental report with revenue for multiple periods, broken down by geography, which is further broken down by business segments. In annual reports, companies may provide separate table for each geography. However, in dynamic array models, they should be presented in a long form format as shown in Exhibit 6.

*Exhibit 6: Illustration of long data format for multi-dimensional data*

| Year | Geography | Business | Sales | EBITDA |
|---|---|---|---|---|
| 2XX3 | Americas | Beverage | 300 | 80 |
| 2XX3 | Americas | Snacks | 280 | 75 |
| 2XX3 | APAC | Beverage | 280 | 75 |
| 2XX3 | APAC | Snacks | 220 | 40 |
| 2XX4 | Americas | Beverage | 315 | 78 |
| 2XX4 | Americas | Snacks | 305 | 79 |
| 2XX4 | APAC | Beverage | 240 | 55 |
| 2XX4 | APAC | Snacks | 240 | 55 |

*Rationale*

- **Scalability:** This format would be able to easily fit any additional item added to a given dimension without needing to create a new table. For instance, in the example provided, if the company were to foray into a new region, say Europe, in 2XX5, the segmental data can be easily added as extra rows in the same table for 2XX5
- **Enhanced analytical power:** This structure is optimized for data processing and analysis functions in spreadsheets. Numbers from the table can be easily crunched and queried across dimensions using auto filters, SUMIFS, FILTER and many other functions.
- **Only viable format for true DA models:** In pure DA models, formulae must not be modified once the models are built and published. In this format, since no additional

tables are required to be added when a new segment is added, such additions will seamlessly integrate into the file.

In the case of data sets that are not likely to be used in "what-if" analysis i.e. not likely to be changed after entering once, modelers can consider keeping the input in more human friendly format and transforming them into long form format using Power Query or any such transformation tools. However, parametric estimates should always be directly entered into such long form table.

**P7. Two dimensional variables must be in long form format if the grid size is likely to become too big to eyeball. Cross tab formats, however, should be preferred for data sets that are likely to remain small.**

For instance, the BOM table in Exhibit 2 is cross tab format, which is acceptable if the number of output products and input materials are expected to remain constant. However, if the company is like to add many more products need many more other materials, the format must be in long form.

*Rationale*

- **Human friendly structure:** Cross tab format is intuitive when the data size is small. However, they become less user friendly when their size gets larger. For instance, users may find it difficult to identify the correct inter-section cell to input the right value when the grid is large.
- **Allows easy eyeballing and in-situ analysis:** Long form data can be easily filtered and eyeballed for quick analysis and sanity checks as compared to a large grid.

***Recommendations:***

**R10. Normalize data to reduce redundant input.**

For instance, in the segmental data illustration in Exhibit 6, each combination of geographic and business can be given a unique name. Another table can be created which provides details of what geography and business segment are present in the unique name.

This is shown in Exhibit 7, where we have one "Segment table" that gives a unique name to segments along with its details. In the segmental data, the user would enter the unique segment name and have the system auto populate the geography and business segments using XLOOKUP.

Although the segmental data is not "Normalized" in the strictest sense as it does contain the redundant columns, this approach reduces the redundancy from input perspective.

*Exhibit 7: Illustration of normalizing data to avoid redundant input*

**Segmental data**

| Year | Segment | Sales | EBITDA | Geography | Business |
|---|---|---|---|---|---|
| 2XX3 | Am_Bev | 300 | 80 | Americas | Beverage |
| 2XX3 | Am_Snacks | 280 | 75 | Americas | Snacks |
| 2XX3 | APAC_Bev | 280 | 75 | APAC | Beverage |
| 2XX3 | APAC_Snacks | 220 | 40 | APAC | Snacks |
| 2XX4 | Am_Bev | 315 | 78 | Americas | Beverage |
| 2XX4 | Am_Snacks | 305 | 79 | Americas | Snacks |
| 2XX4 | APAC_Bev | 240 | 55 | APAC | Beverage |
| 2XX4 | APAC_Snacks | 240 | 55 | APAC | Snacks |

**Segment details**

| Segment | Geography | Business |
|---|---|---|
| Am_Bev | Americas | Beverage |
| Am_Snacks | Americas | Snacks |
| APAC_Bev | APAC | Beverage |
| APAC_Snacks | APAC | Snacks |

*Rationale*

- **Balance user experience with rigid structure:** Multi-dimensional data necessarily needs to be in long form format, which is laborious and needs redundant inputs. By normalizing the structure, we reduce the number of redundant input variables that need to be typed.

## 2.4 Architecture for single group of mixed nature data

This situation calls for complex balancing between technically strong design principles and human friendly approach and the possible options fall within a broad spectrum. Therefore, this guide does not prescribe any practice as a must follow practice for this. However, there are certain recommendations.

**Recommendations:**

**R11. When a set of variables belong to a single group but the nature or behaviour of one or more items vary from the others, treat all the individual items as scalars but format them to visually appear in tabulated form ("Tabulated Scalars").**

*Rationale*

- **Flexibility:** Structured Tables have rigid structures and mixed nature data may not fit in easily. Tabulated scalars allow flexibility
- **Allows Intuitive flow:** Other technically sound possibilities, such as organizing each nature of data in separate tables may affect intuitive flow for the end users.
  - For instance, in the example in Exhibit 3, debt and preference shares details could have been kept in a structured table, while the weight of equity is calculated elsewhere, outside of the table. However, such an approach would reduce the intuitiveness of the flow, especially if the analyst is trying to optimize weights to minimize WACC.

- **Easy to read:** Tabulating these scalars together as a group rather than treating them as independent scalar variables make it easy for the readers to understand the variables better.

Tabulated scalars do not allow dynamic referencing as well as a Structured table and is more vulnerable to user action such as inserting or deleting rows. The guide, therefore, recommends users to be very selective in choosing to use tabulated scalar structure over a Structured table and treat the structure as an exception than a norm.

## Summary

In this chapter, we discussed that a robust input architecture is a critical foundation for building stable, scalable, and efficient Dynamic Array (DA) model. The architectural choices for input directly affect the stability of subsequent calculations.

However, as spreadsheets lack an embedded dynamic front end feature, certain pragmatic compromises need to be made to balance robust architecture with human friendliness.

**Key guidelines**

- **Primary of structured tables:** Input data, other than scalars, must be necessarily organized as a structured table.
- **Ensure separation of concerns:** Data sets that do not share the same dimension (such as historical financials and forecast assumptions) must not be in the same table.
- **Fail proofing with validation columns:** Tables should have a calculated validation column to check row level validity and mark stray rows as invalid.
- **Use long form for multi-dimensional variables:** Variables with 3 or more dimensions must be necessarily in long form. Two dimensional inputs must be captured as a cross tab when the grid size is expected to be small; but they must be organized in long form if the grid size is likely to grow.
- **Practices for scalar values are unchanged:** Scalar variables should be kept in a separate cell and properly colour coded and preferably, named.
- **Mixed nature data should be treated as scalars:** Data that belong to single group but have different nature should be treated as a scalar data but they need to be grouped and formatted to appear as a table to human eye.

In the next chapter, we shall discuss how the calculations need to be performed and understand how the input architectures discussed in this chapter play key role in ensuring stable, scalable, and efficient calculation sections.

# Chapter 3: Making the calculations dynamic

Dynamic array and Lambda fundamentally change how calculations are performed in spreadsheets. The following outlines this change:

(i) The calculation approach moves from individual cell-by-cell calculations that are copy-pasted to dynamic array-based expressions.
(ii) Favors in-memory computations where a lot of intermediate steps can be stored in temporary variables rather than storing them in cells.
(iii) Allows creating custom user defined functions, using spreadsheets' own formula language and within the workbook.
(iv) Enables using iterative and recursive logics in functions.

These changes, besides removing the need for copy-pasting of formulas, also enable a lot more possibilities that were not possible earlier. For instance, circular reference problems can be solved with recursive or iterative Lambdas, and eliminates the need, and the associated risk, of enabling iterative calculations in the workbook.

However, these enhancements create an overwhelming number of choices for performing the same calculation including some that are inherently risky. This chapter offers the guidelines on the appropriate approach that satisfies the BEST principles.

## Terminologies and Definitions

In this chapter and other chapters, certain terms are used with specific meaning in the context of this guide.

**Lambda(s) vs LAMBDA:** In the context of this guide, the term "Lambdas" refers to any custom function created using the LAMBDA function in spreadsheet. Some part of the guide may also refer to them as custom functions.

The term "LAMBDA" written in all upper case refers to the built-in function in spreadsheet which is used to create the custom functions.

## 3.1 Array referencing practices

***Prescriptions***

**P8. When referring to a dynamic array range, always refer to it using the dynamic array format and avoid using fixed range references.**

For instance, if a dynamic array in cell B5 spills to B9, reference the result as `=B5#` (assuming the cell is not named) instead of `=B5:B9`. This is critical to ensure the model dynamically updates if the formula in B5 produces larger results.

Rationale

- **Dynamic expansion**: If the referred spilled array expands, dynamic references will automatically consider such additional values. Static references break this adaptability.

**P9. Specific cells or ranges within an array should be referred as an element of array and not through direct cell reference. Functions such as TAKE, DROP, FILTER, XLOOPUP, CHOOSECOL can be used for this purpose, depending on the context.**

For instance, let us say we have the previous 20 days stock prices, in ascending order of dates, as a single array spilling from cells B3:B22. The formula to compute the daily returns must be given as follows:

```
=DROP(B3#,1)/DROP(B3#,-1) - 1
```

It should not be computed as `=B4:B22/B3:B21 - 1`

Rationale

- **Stable under change**: When the size of the array expands or shrinks, array-based reference will automatically adapt, while static references will fail.

**P10. Any reference to a cell or range inside a LET function must be assigned to a variable and used. A particular cell or a range must not be referred more than once inside a given LET function.**

For instance, let us say cells C15, D15, and E15 have start date, end date, and current date and the let function needs to calculate two variables: (i) Total duration, which is D15 *minus* C15 and (ii) Lapsed time, which is MIN(E15-C15, D15-C15).

The appropriate way to use these cells is as follows:

```
=Let(StDt,       C15,
     EndDt,      D15,
     CurrentDt,  E15,
     TotalDur,   EndDt - StDt,
     LapsedPrd,  Min(CurrentDt - StDt, EndDt - StDt),
     ……..)
```

*Rationale*

- **Centralizes changes in cell reference:** If the cell reference needs to be changed — for instance when the same logic is copied to a different table — the change will need to be done only once. This reduces the risk of inconsistent changes.

**P11. "Sort proof" references to tables and arrays so that the formula returns correct answer even if the order of the underlying data changes**

For example, let us say we have *SalesTbl* with annual sales by year, with the sales for 2025 at the bottom. But sorting can move the year's data from its original position. To reliably get the sales figure for 2025, regardless of sorting, use a formula such as this:

```
=Xlookup(LookupYr,  SalesTbl[Year], SalesTbl[Amt])
```

*Where LookupYr is 2025*

Rationale

- **Increases integrity and stability of calculations:** User interactions such as sorting the underlying data are very common. Sort proofing the calculations ensures that output is not affected by such interactions.

P12. **Any reference to an input table should anticipate invalid or stray rows and filter them out.**

As explained in Chapter 2, blank or incomplete rows may get added to a Structured table. When a formula refers to a table column, it would include such stray rows, and this may lead to incorrect computation. Thus, any formula referring to the input table should anticipate such stray rows and filter them out.

Continuing with the previous example, let us say we need to get the latest year sales from the *SaleTbl, which also* has an *IsValid* column — that returns TRUE only when the row meets the validation criteria. A formula such as the one below would give correct answer even if end users add place holder rows for future periods.

```
=Let(LatestYr, Maxifs(SalesTbl[Year], SalesTbl[IsValid], TRUE),
     Return, Xlookup(LatestYr, SalesTbl[Year], SalesTbl[Amt]),
     Return)
```

Rationale

- **Avoids unwarranted dynamic expansion:** It safeguards the models getting affected because of common end user interactions such as including place holder rows or filling up partial data sets.

P13. **External references must always be handled using ETL tools, such as Power Query or SQL, and directly linking a cell to another workbook must be avoided.**

Sometimes, a model requires data from another model—for example, a Master Budget may need information from various functional budgets, or an industry analysis may need data from individual files for different companies.

In MS Excel, Power Query can integrate such data efficiently.

Rationale

- **Avoids risk of stale values or incorrect references:** If the data is brought into the model through copy-pasting, it could turn stale when the other model is updated and live workbook links are vulnerable to changes in the location of the cell. Importing the data from a structured export sheet (explained later in the guide) in the upstream model reduces both the risks.

***Recommendations***

**R12. Use structured referencing while referring to Structured tables.**

Rationale

- **Improves readability and access:** Structured reference with clear table and column names are easier to read and modelers can easily refer to the tables from any part of the workbook with the help of ***IntelliSense***.
- **Reduces chances of wrong references:** Structured reference ensures that table columns are always referred in their entirety and reference to wrong columns can be easily spotted by looking at the column name.

Structured references are absolute but act as relative references when dragged right. Users should be aware of this behaviour.

**R13. Name the cells that contain the spill formula.**

Rationale

- **Improves readability and accessibility:** As explained in many parts of the guide, when named references are used over relative references, the readability and accessibility of the formula increases.

## 3.2 Calculations and Calculation blocks

***Prescriptions***

**P14. Timelines in a financial model must be model specific and not schedule or line item specific.**

For instance, say a project has a life of 25 years, while the debt taken for it should be repaid in lesser duration of 20 years. If an analyst is projecting cash flows for the full 25 years, the debt schedule must also flow for the entire 25-year period, with a formula that makes all the values in the debt schedule going to zero for years 21 to 25.

**However, this guide does not discourage, modelling certain line items at a higher frequency — than others — in a separate schedule and aggregating them in a common schedule.**

For instance, an analyst may choose to model revenue and operating expenses for every quarter while modelling interest and taxes at an annual level and present all of them on annual level. Such a practice is acceptable provided the quarterly schedules are organized in a separate calculation section.

Rationale

- **Prevents fragmentation and errors:** Ensuring that all the arrays are of uniform length ensures that operates on more than one array, works seamlessly. If the arrays are of different length, DA formula may return error, and more complicated effort would be required to handle such errors in the formula.

**P15. All calculations that produce a vector or a matrix output must be modelled as a dynamic array and no such item shall be modelled using legacy scalar iterative approach.**

For instance, many multi-level corkscrew calculations — such as a complex cash sweep — are, perhaps, easier to build using legacy approach. However, if the model is built using DA approach, then such calculation blocks must also be built using DA approach, even if it appears more complicated.

Rationale

- **Ensures integrity of calculations**: Since the entire vector or matrix is driven by one formula, there are no risk of inconsistent formula within a range. For instance, in a legacy approach it is possible for an analyst to, intentionally or otherwise, modify a single cell inside a large range of copy-pasted formulae, which would throw a challenge for model reviewers. Such a risk vanishes with this approach.
- **Reduces dependency on end users for model maintenance:** Using only dynamic arrays eliminates manual tasks for the end user. Including legacy methods require copy-pasting and adjusting formulas, increasing the risk of errors.

The spreadsheet risk when legacy approach is mixed with dynamic array is far greater than in a pure legacy approach. The issue is discussed in detail in Appendix 3.

Therefore, this guide recommends using a pure legacy approach over the mixed approach, if an analyst finds it impossible to model any array using the dynamic approach.

**P16. Modelers must avoid enabling iterative calculations in DA models. They must opt for alternative closed form solutions, or iterative/recursive Lambdas.**

Circular reference issues in DA models can be handled by adopting a different set of equations (derived algebraically) that avoid circularity. If that is not possible, modelers can create iterative or recursive Lambda to solve such problems.

*Rationale*

- **Avoids unpredictable model behaviour:** If certain variables that drive the size of a spill array turns circular, by design or otherwise, they may cause the model to spill incorrectly leading to unpredictable results.
- **Extremely difficult to debug:** When external iterative calculations affect in-memory computation of DA models, bugs are far difficult to trace and debug. On the other hand, when iterative calculations are performed using an iterative Lambda that adheres to referential transparency, they are easier to resolve.

**P17. Modelers must <u>not</u> use TRIMRANGE functions or trim operators while referring to a cell as they increase the risk of being affected by stray values.**

*Rationale*

- **Inadvertent stray values should not affect calculation:** End users, who are unaware that a particular cell flows into a TRIMRANGE function or with trim operator, may type a stray value (like a rough calculation) in what appears to be a blank cell, and this will lead to all the dependent calculations automatically extending themselves upto such stray value and produce erroneous output.

**P18. Calculated columns in structured tables must not be used for model calculations except in the following circumstances:**

(i) **Validation columns, described in recommendation R6 and R7 can be used for filtering out stray or invalid data.**

(ii) **When the calculations need to be done iteratively across the rows in a table and it is impossible to do it as a dynamic array with acceptable level of performance.**

Ever since the introduction of SCAN and REDUCE, it has become easier to perform iterative calculations as dynamic arrays, and few cases exist where that cannot be done. However, this guide allows using calculated column in structure tables as a last resort keeping in mind that there can be tail cases where calculated column is the only option.

*Rationale*

- **Prevent model errors from polluted tables:** Although structured tables are also dynamic and expand, they are not fail-proof. For instance, cell in a calculated column can be overwritten, especially when the sheet is unprotected. It makes such calculated columns vulnerable for unauthorised editing.

**P19. Modelers must not stack multiple variables in one array except in the following cases:**

(i) **All the variables are inter-related and computed together iteratively, or recursively**

(ii) **The stack is used to present the final output of the model (for display or export)**

(iii) **The stacking is required as an intermediate step inside a formula, but the formula does not return such a stack**

For instance, theoretically, it is possible to create a custom function that will return entire term loan schedule, including the opening balances, interest expense, borrowings, repayments, and closing balance as a single schedule. However, since the six variables are separate variables, each of them should be presented as individual vectors and not stacked together.

On the other hand, in cash sweep modelling of a revolver, since cash flows, interest, taxes and borrowings are inter-related, all of them need to be modelled iteratively using REDUCE function and these variables will need to be stacked.

Further, this guide accepts stacking multiple schedules together as part of the output or in an export sheet if individual vectors of the stack are not directly used elsewhere in the model.

*Rationale*

- **Improves Accessibility:** When each variable is presented in an independent vector, it is easy to refer in all dependent calculations.
- **Maintains model stability when new line items are added:** When variables are stacked, functions like INDEX or CHOOSEROWS require specific row or column references. If the number of items or their order in the stack changes, these functions may return incorrect results.

**P20. When major computations are processed in-memory and intermediate steps are not directly observable, modelers must include validation utilities to verify the output.**

For instance, if a fund holds 100s of debentures, a scalable approach would be to model the entire balances, redemptions, and incomes for multiple periods together in a single schedule rather than preparing separate schedule for each of them and then aggregating them separately.

Although these are scalable, auditing or validating them would be extremely difficult.

To support the audit process, the model should have separate calculation section that allow reviewers to drill down the calculation at individual debt level. This calculation section will serve as a validation utility for the main model.

*Rationale*

- **Balances auditability and scalability:** In-memory computations are highly scalable and efficient. However, they significantly reduce the transparency of the model. Having a utility that can help validate the calculations, and which are easy to understand will improve the transparency, and hence, the trust on the financial model.

**P21. Calculations and calculation blocks must be positioned in a way that they do not potentially cause a spill error if the dynamic array expands.**

If a calculation block is expected to expand into more columns, do not place other calculation sections to its right. Similarly, if it is expected to expand across rows, avoid placing anything below it. Totals are best placed above or to the left of such blocks.

*Rationale*

- **Prevents spill errors:** Keeping other cells clear along the array's direction helps avoid spill errors. While these errors are usually fixed by adding rows or columns, they may affect downstream models importing data (until the error is resolved).

***Recommendations***

**R14. Modelers should show intermediate steps transparently in the model unless it significantly affects the scalability or performance of the model.**

Most spreadsheet-based financial models use only a few thousand calculations. Thus, splitting them into multiple steps across separate ranges does not significantly increase file size or affect performance.

*Rationale*

- **Balances transparency and scalability:** Breaking the calculation by intermediate steps makes it easy to follow the model. And if the problem is not too big, the advantage would be gained without practically compromising on performance or scale.

**R15. Complex in-memory computations should be done through approved Lambda rather than as direct formula in the cell. Writing complex logic directly in a cell should be opted only if the modeler wants to protect the IP.**

Chapter 6 prescribes governance strategies and guidelines for Lambdas to be developed and approved with the objective of ensuring the Lambdas that are used in production are highly reliable.

*Rationale*

- **Increases reliability:** Reviewers and auditors can easily validate an approved Lambda rather than complex multi-line logics that are directly written in a cell. This increases the reliability of the model.

**R16. Models should have well-structured and machine friendly set of export sheets that can be used by downstream models to import data.**

The export sheets are different from output sheet. They are not meant for presentation but for reading by machines. And a model may need more than one export sheet if the data to be exported are of different dimensions.

*Rationale*

- **Allows seamless integration with downstream models:** Downstream models can use SQL, Power Query or other such methods to easily extract data from a given file if the file has a well-structured export sheet.

Care must be taken on the design and behaviour of export sheets. The suggested practice for the export sheet is separately discussed in Appendix 5.

## Summary

In this chapter we had discussions on how calculation section should be set up and considerations that should be given in choosing a proper calculation approach so that models remain stable, scalable and efficient.

**Key guidelines**

- **Strict array reference principles must be followed:**
  - Dynamic array must always be referred using the # operator and static range reference must be avoided.
  - Specific cells or ranges within an array should be referred as an element of array and not through direct cell reference.
  - Cell references inside LET function must be assigned to a variable and same cell must not be referred more than once inside the LET function
  - Structured references and named cell references are preferred over relative cell references.
- **Calculation set up should follow these practices:**
  - Timeline must be model specific and not schedule specific
  - Multiple variables must not be stacked together but for certain exceptional cases mentioned in the guide
- **Fail proof formula against user interactions:**
  - Sort proof financial models
  - Prevent stray data and incomplete data updates from causing dynamic expansion.
- **Must strictly avoid certain legacy practices**
  - Legacy approach must not be mixed with dynamic array approach
  - Iterative calculation setting must stay disabled.
- **Choice between storing intermediate steps in memory versus in sheet must depend on performance factors**
  - Intermediate steps should be stored in sheets if it does not cause any performance issue.
  - If it causes performance issues, in memory calculations must be preferred but it should be done with an approved Lambda and must be accompanies by validation sections.

This chapter guided that complex logic should be handled using approved Lambda. In the next chapter, we shall discuss what process and practice should be followed so that approved Lambdas are more reliable than explicitly written formulae.

# Chapter 4: Building reliable, reusable Lambdas

LAMBDA, together with the LET function, in MS Excel enable functional programming in modern spreadsheets. These functions have made spreadsheets near-Turing-complete i.e. we can perform any calculations that can be described — with the only key limitation being the size of variables that the spreadsheets can handle.

Users working with LET and LAMBDA are essentially entering the functional programming paradigm. And modelers would need to follow most of the established software design principles such as the SOLID principles.

This guide adapts such software principles in the context of financial modelling. However, the guide does compromise somewhere on these principles for certain practical reasons.

One such compromise is to have a slightly different guidelines based on the scope of use of the Lambda. This guide classifies Lambda into the following two categories, based on their scope of use: (i) Global Lambda and (ii) Local Lambda

## Terminologies and Definitions

**Global Lambda**: These are custom functions that are used across models. This classification is not based on how many can access the Lambda, but it is based on whether the same function is expected to be used in more than one file.

**Local Lambda:** These are custom functions that are meant to be used in a specific file alone.

These are not expected to be used in another file. If a file is likely to be copied and used in its entirety for another purpose, the Local Lambda in the file should be considered as Global Lambda and follow the guidance given for the same.

This guide recognizes that Local Lambda may be often created with modeler convenience as its key objective. Thus, the guidance for Local Lambdas are less strict for the most part, except that a Local Lambda shall not be used as Global.

## 4.1 Foundation rules and design principles

***Prescriptions***

**P22. Create each Lambda to handle a single, well-defined task. For complex multi-step operations, create a wrapper function which calls each of the needed Lambdas to perform the set of tasks.**

For instance, a simplistic projection of sales may involve the following steps: (i) Compute historical growth rates (ii) Propagate the moving averages of the growth rates for future N periods (iii) Project sales using such moving averages. Each of the three actions should be performed by three separate Lambdas. A fourth, wrapper Lambda can be created to call the three in the right order to get the final output.

*Rationale*

This guidance stems from the application of the ***Separation of Concern*** principle and has several advantages. The most relevant from the point of financial modelers are as follows:

- **Enhances code reusability:** Logic stored in a single Lambda function can be used for multiple purposes. For example, the same moving average calculation can be applied to both sales projections and gross margins without rewriting the code.
- **Easy to test, debug, and maintain:** Since each Lambda does a specific task, it can be independently tested and can be debugged. Any correction in the logic of the Lambda will automatically be incorporated into all the wrapper LAMBDA that uses it.

**P23. Global Lambda logic must avoid direct references to cells, ranges, file locations, websites, or volatile functions like random numbers or current date/time. Instead, use parameters to pass these variables as arguments.**

For example, if we have dynamic daily share price data and need a Lambda to auto calculate the 20-day moving average (20 DMA) based on the current date, instead of embedding the TODAY() function directly inside the Lambda's logic, create a parameter for the current date to be passed in the Lambda by the user. End users can supply TODAY() as the argument for this parameter when calling the Lambda.

An exception to this rule is that a default value can be assigned for optional parameters that are omitted.

*Rationale*

- **Promotes transparency and simplifies debugging:** Making all variables explicit in the Lambda ensures that calculations depend solely on the arguments (referential transparency), making testing and troubleshooting easier.

Although referential transparency is critical in functional programming context, this guide does not mandate its strict adherence for Local Lambda, as it recognizes that such Local Lambda may be created with user convenience as its objective.

**P24. A Lambda that is expected to work with column vectors, should also work with row vectors (and scalar variables), and vice versa, unless the dimension of the vector is rigid in nature or practice.**

Most vectors, such as timelines, can be oriented horizontally or vertically, though some contexts require a specific direction. For example, fields in tables and eigenvectors are typically column vectors. Calculation requirements may also dictate vector orientation; if sales volume is a row vector, so must be the selling price.

*Rationale*

- **Supports flexible calculation layouts:** Users can choose to arrange variables horizontally or vertically, and Lambda functions will operate reliably with either orientation.

**P25. When creating a Lambda to address a problem with circular reference, priority must be given to closed form solutions, followed by iterative solutions. Recursive solution must be used only as a last resort.**

*Rationale*

- **Improves efficiency and easy to debug:** In term of performance, closed form solutions are the most efficient, followed by iterative solutions. Recursive Lambdas are the slowest among the three and may fail to perform when number of recursions exceed the limit. Ease of debugging follows the same order.

**P26. Variables names used in a LET function must not be the same as any of named ranges or functions and no range shall be given a name that is same as that of a Lambda**

*Rationale*

- **Prevents conflicts and errors:** If a variable name is same as a range name, then the range cannot be referred with the same name in the LET function. Further, when a range is given same name as a Lambda, the action will overwrite the Lambda leading to corruption in the model.

***Recommendations***

**R17. All Lambda names should end with a sign such as "λ" to make it easy to distinguish between custom functions created with Lambda and built-in MS Excel functions.**

For instance, a Lambda to calculate the moving average can be named as `MovingAvgλ`.

*Rationale*

- **Makes the user aware of the custom nature of the function:** The "λ" sign in the name makes it clear to the readers that these are custom functions and not prebuilt functions.
- **Reduces name conflicts:** Adding the "λ" symbol to Lambda functions keeps their names distinct from range names. For example, users can have a Lambda named WACCλ and a cell named WACC, without creating conflicts.

**R18. Variable names used in functions should be brief, yet descriptive. Modelers can also consider using commonly used mathematical notations such as μ and σ.**

*Rationale*

- **Makes it easier to understand the flow of logic:** When the variables are descriptive — such as "Loan", "Int" — it is easier to follow the logic in a lengthy function. However, if the variable names are too big, they may result in the cell exceeding the character limit.

**R19. The final step of the calculation in a LET function should be assigned to a variable name and the variable name should be given as the final return value.**

For instance, let us say a multi-step calculation is performed to calculate the interest expense for a stub-period. The final interest calculation should be assigned to a variable like "Return" or "Output" and these variable names should be mentioned at the end.

*Rationale*

- **Helps in debugging:** At the time of debugging, the function can be modified to return the intermediate value without having to make major code changes.

**R20. Local Lambda should be relatively simple with not more than a few steps.**

*Rationale*

- **Easy to audit:** Reviewer are likely to have only limited time to review Local Lambdas. Hence, it should be simple and handle only a few steps to facilitate quick and effective audits.

## 4.2 Development and validation

***Prescriptions***

**P27. All the Lambdas must be thoroughly documented, and in-line comments must be used to explain any complex line or block of calculations.**

Documentation must focus on the purpose, application and logic of the overall Lambda, the meaning of the input parameters, and explanation for the codes. The documentation must also explain the use case for the Lambda.

Advance Formula Environment (AFE) in MS Excel allows adding detailed comments. Modelers can also leverage the GitHub repositories that allows LAMBDAs to be stored with inline comments. Google sheets also has inbuilt functionalities to describe the parameters and provide sample input.

*Rationale*

- **Improves clarity, and thus, auditability:** Lambdas perform many calculations in-memory, where intermediate calculations will not be visible, and, thus, its workings are relatively difficult to understand. Proper documentation and in line comments offer clarity on what a code is meant to do. Such clarity also helps when auditing the function.
- **Reduces misapplication of function:** Documentation of the right use cases will help end users to correctly apply the appropriate function.

**P28. A Global Lambda must be thoroughly tested and validated before being approved for use in financial models**

*Rationale*

- **Makes the models reliable:** End users, often are likely to get the feeling of working with Black Box and may not be able to test its logic or accuracy. However, if the Lambda are pre-tested, it increases the reliability of the Lambda.

***Recommendations***

**R21. The documentation practices for Local Lambdas should be at par with Global Lambda, if not better.**

*Rationale*

- **Easy to inherit / handover:** Knowledge of Local Lambda is likely to be limited to the creator of the model. Having quality documentation ensures that other users who take over the file can develop good understanding of the same.

## 4.3 Deployment and maintenance

***Prescriptions***

**P29. All Global Lambda should be available in a common repository from which they can be imported.**

The AFE in MS Excel allows Lambdas to be imported from GitHub or users can also copy paste all available Lambda from a notepad file into a module. In the case of Google sheets, the repository can be in the form of another Google sheet from which it can be directly imported.

*Rationale*

- **Allows refreshing and importing new Lambda:** A central repository allows users to periodically import the updated list of Lambdas ensuring that the files have access to the latest set of approved Lambdas.

**P30. No modifications, including bug fix, must be done to a Global Lambda after it is published for production. However, erroneous Lambdas can be retired, after due process.**

If a Lambda's functionality needs to be improved, it must be published as a new Lambda. However, if a published Lambda is found to have serious bugs, it can be retired in its entirety following the due process by the model governance team (refer Chapter 6).

*Rationale*

- **Ensures consistency and backward compatibility:** Each Lambda maintains its functionality throughout all files, as any changes result in a new function. And reloading Global Lambdas from a central repository poses minimal risk to current calculations.
- **Reduces dependency on individuals:** New developers joining the team need not understand the inner workings of existing Lambdas, which ensures that future development will not suffer if original creators of the Lambda leave the organization.

In this context, thorough testing and validation of Global Lambdas before their publishing becomes extremely critical. Otherwise, every iteration of bug fix will lead to overload of too many similar LAMBDAs with minor modifications.

**P31. Users must avoid customizing a Global Lambda to suit the need of a specific model. Any customization should be done either by manipulating the argument passed or through a Local Lambda serving as a wrapper function.**

For instance, let us say an organization has created a custom function to calculate the number of weeks between a start and end date, with a code such as this:

```
NWeeksλ = LAMBDA(StDt, EndDt, [CurrentDt],//CurrentDt must be TODAY()
                Let(_EndDt, if(EndDt=0, CurrentDt, EndDt),
                    RoundDown((_EndDt-StDt+1)/7, 0)
                    )
               )
```

The Lambda, above, uses the current date as the EndDt in case the end date field is blank or zero.

In a separate instance, it is possible that we need the function to return "NA" as output if the end date field is blank. In this case, instead of changing the NWeeks Lambda, one option is to create a Local Lambda — especially, if we need to reuse the logic — as follows:

```
Local.NWeeksλ = LAMBDA(StDt, EndDt,
                  IF(EndDt = 0, "NA", NWeeksλ(StDt, EndDt) )
                  )
```

*Rationale*

- **Retains reliability:** Ensuring that a Global Lambda that has been validated is not modified retain the reliability of the said LAMBDA.

***Recommendations***

**R22. Global Lambda must be reloaded from the common repository whenever a user inherits a model or duplicates a file. Modelers should regularly reload Lambda for ongoing models and compare outputs to confirm consistency.**

*Rationale*

- **Minimises risk of Lambda pollution:** Spreadsheets offer limited safeguards for protecting a Lambda from unauthorised edits or overwriting. The risk gets significantly reduced when it is reloaded from a central repository.
- **Keeps the file updated with the latest Lambda:** Reloading the file regularly ensures that analysts can access all the latest Lambdas.

If a model's output changes after reloading, it should be promptly reported to the model governance committee (see Chapter 6). For individual or small-scale operations without such committees, modelers themselves must thoroughly analyse these cases.

## Summary

This chapter introduces a structured approach to functional programming in spreadsheets, leveraging the LAMBDA and LET functions. It adapts the robust software design principles and applies it in the context of financial modeling with a few pragmatic compromises reflecting certain limitations in spreadsheets.

**Key guidelines:**

- **Scope of Lambda:** Global Lambdas, that are applied in multiple models must be built using rigorous development standards. Local Lambdas, that are used in a single file can prioritize modeler convenience, but it must be simple, confined to a single file, and must not compromise on documentation.
- **Single-responsibility design**: Each Lambda should perform one well-defined task. For multi-step workflows, use a wrapper Lambda to orchestrate specialized Lambdas in sequence. This separation of concerns improves clarity, testing, and reuse.
- **Closed form > Iteration > Recursion**: Lambda logic should prioritise closed form solutions over iterative solutions. Recursive solutions should be used only as a last resort.
- **Immutability**: Once a Global Lambda is published to a production environment, it must not be altered. Any necessary improvements should be released as a new function, while buggy functions should be retired through a formal process. This prevents breaking existing models.
- **Strong testing and documentation**: All Global Lambdas should be thoroughly tested and documented before they are published.
- **Centralised distribution**: Global Lambdas should be stored in and distributed from a common repository. Modelers are responsible for regularly reloading these functions to ensure their models use the most current and consistent versions, especially when inheriting or duplicating files.

Following these guidelines help organisations to build a library of custom functions that are consistent, reliable, and easily reusable, enhancing the integrity of their financial models.

This chapter prescribed that Lambdas should be thoroughly tested. In the next chapter, we describe how to test and debug Lambdas and complex in-memory computations.

# Chapter 5: Testing and debugging the "Black Box"

While Dynamic Arrays and Lambdas offer immense power, they also demand a new discipline for quality assurance. Traditional validation methods, designed for cell-by-cell logic, are insufficient for models driven by single, complex array formulas. Auditing now involves scrutinizing compact, powerful expressions rather than tracing scattered precedents.

This chapter provides guidance on the strategies for testing, and debugging Lambda within this modern paradigm.

## 4.1 Testing the Lambda

The purpose of testing a Lambda is to ensure that it works correctly. The ultimate onus for testing lies with the team that is developing the Lambda. However, they may use the help of other teams.

The testing of a Lambda may not take as much time as testing a new system. However, it may still take more time than checking a normal formula. Given that a Global Lambda is likely an IP asset serving the organization in the long run, the additional time spent on testing is justified.

**Prescriptions**

**P32. Output of a Lambda must be compared with the same computations done using legacy approach and ensure that the outputs are consistent. Such testing must be performed with multiple data sets to increase the reliability.**

Organizations that have already built many models can leverage their model banks for such testing. For testing the accuracy of calculations, the input architecture in the scalar models need not be changed.

*Rationale*

- **Validates the output:** There may be some bugs or flaws in what appears to be a logically accurate function. Validating it with scalar iterative approach will help uncover or rule out any such bugs.

**P33. All the Lambda must be tested for edge cases.**

Edge cases in calculation can arise in plenty of ways including the following:

(i) Argument values being very high, low, or blank
(ii) Arguments in opposite sign i.e. positive values in place of negative and vice versa
(iii) Large size of underlying data
(iv) Parameters with different dimensions.

*Rationale*

- **Tests model stability:** This will help test whether the Lambda, and thus the overall model, can be stable under different situations.
- **Helps design error handling features:** Such an edge case testing will help identify scenarios where the Lambda fails. The understanding can then be used to create error handling procedures within the same Lambda.

- **Increases reliability:** When a Lambda is able to handle the edge cases, it increases its reliability.

In Appendix 4, we discuss argument in favour and against including customized error messages in the case of Lambda.

## 4.2 Debugging Lambda and calculations

Debugging formulae written with LET or LAMBDA function brings a different set of challenges for most financial modelers as most of the computations are handled in-memory.

This brings a challenge in terms of identifying and isolating the source of error for further investigation. This guide outlines a set of debugging strategies that financial modelers can use to identify and address the source of bugs.

***Recommendations***

**R23. Trace internal calculations: Make a variable in the intermediate step as the output variable and compare its value with manually calculated value or the value of that step in a scalar iterative model.**

Tracing is typically don by backtracking from the last step and moving one step backwards till we reach a step where there is no mistake ("last right step"). Then we correct the step that has a mistake and repeat the process until all errors are resolved.

*Rationale*

- **Faster to pinpoint the mistake step:** It allows to systematically locate the source of bug. Once the last right step is identified, we need not check the prior steps.
- **No Special Tools required**: It uses the inherent functionality of the modern spreadsheets

The modeler can also consider to directly test an intermediate variable if their intuition points to that as a possible problem step

**R24. Reduce the number of iterations to 1 and gradually increase: In the case of iterative Lambda, keep the number of iterations to 1 and check the result. If there are no mistakes, increase the number of iterations till an error is encountered. Then decompose the calculation between the two iteration counts.**

SCAN and REDUCE functions in modern spreadsheets allow creating iterative Lambdas. But, it is quiet challenging to backtrack the prior steps. When the number of iterations is set to 1, it behaves like a non-iterative Lambda, which makes it possible to backtrack the steps.

If the Lambda produces bug when iteration is more than 1, then decompose the result between the last correct iteration and next iteration to identify the source of bug.

*Rationale*

- **Allows tracing internal calculations:** It is not possible to backtrack steps in iterative Lambda. However, when number of iterations is set to 1, the function behaves like a non-iterative function, which makes it easy to backtrack.

**R25. Replace array parameters with scalar parameters to test (i) logic and the (ii) syntax. If the function works correctly with scalar parameters but fails to function with array parameters, it is likely the dimension of the parameter is incorrect.**

For instance, let us say we have written a function to calculate the total interest expense pertaining to a portfolio of loans. For the purpose of debugging, we shall test it with the data pertaining to single loan, instead of the entire portfolio.

*Rationale*

- **Easy to backtrack complex array calculations:** Backtracking steps, when the steps themselves are large arrays can be quite challenging. However, when array parameters / functions are converted to scalar parameters / functions, the size of the intermediate calculations will reduce and, thus, make it easy to check for errors.
- **Separate logical error from dimension error:** Certain error may arise because of a problem with the dimension. For instance, if the dimension of the various parameters passed within a MAP function are not uniform, the function will fail. When all parameters are made scalar, this issue will not arise.

**R26. Apply binary dissection method to narrow down records or items causing the error. Once an item / record causing an error is identified, isolate it and apply the formula/function for the individual item / record alone and follow the steps outlined earlier, in this section.**

For instance, let us say a function meant to calculate total interest expense on a portfolio with 20 loans does not match the calculation done using the legacy model. In this case, we should apply the function on one half of the portfolio (i.e. 10 loans in this case) and identify which half has the error. Once the half with the error is identified, halve it again and repeat the process till we identify the record with the error.

Once the loan with the error is identified, apply the function exclusively for that loan item and trace the internal steps.

*Rationale*

- **Easy to identify edge case items that cause trouble:** In large arrays, some edge case items may persist for which the logic that applies to all other items may not apply. This technique helps quickly zero down on such edge cases.

It is possible that there is more than one item in the array that causes trouble. Once we identify one error item and fix the logic, we may have to repeat the process again, if some other error persists.

## Summary

This chapter focused on robust techniques for testing and validating Lambda and in-memory computation.

**Key Guidelines**:

- **Validate Lambda**: Compare the results produced by Lambda with results computed using legacy approach.
- **Stress test**: Lambdas should be tested using edge cases to ensure that they work correctly.
- **Apply suitable debugging strategies**: The following steps should be taken for debugging Lambdas
    1. **Backward Tracing**: Make the model return each of the intermediate variable in reverse order and find the point at which the model breaks.
    2. **Reduce iterations to 1**: To backward trace iterative Lambdas, reduce the iteration count to 1.
    3. **Binary dissection:** When working with large array argument(s) systematically halve the arguments over-and-over to quickly narrow down items causing bugs
    4. **Convert vectors parameters to scalar:** Once an item causing error is identified, isolate the item and pass it as a scalar argument to the Lambda and subsequently backward trace the error.

The guidance given in this chapter should help financial modelers develop Lambda that are reliable and tackle errors and bugs, if any, effectively.

In the next chapter, we shall focus on how organizations should be structured and the governance practice they need to follow to ensure that responsibilities are properly assigned and human errors are avoided.

# Chapter 6: Governance and Organization structure

The power of dynamic array and Lambda bring in new challenges, if they are not handled responsibly: errors can propagate at scale, abstracted functions can become opaque "black boxes," and without discipline, complexity can grow unchecked.

To harness the power of these functions, it is critical to start treating financial models as an organizational task rather than as an individual task.

## Three-pillar organisation structure

This chapter outlines a robust, three-pillar organizational framework designed to instil discipline, ensure quality, and create a scalable, trustworthy financial modeling ecosystem.
This guide recommends splitting financial modeling team into three separate functions:

(i) Lambda creation and testing
(ii) Model building
(iii) Lambda and modeling governance

> While the modeling team is split into three functions, it may not necessarily increase the need for higher man power as pre-built Lambdas will significantly increase the speed at which individual team members would be able to build and update models.

**Lambda creation team**
This team will have end-to-end responsibility towards creation and maintenance of the central Lambda repository / library. This team's responsibility include (i) identifying reusable Lambda that need to be created (ii) create the function, (iii) test it, (iv) roll it out for production along with documentation, and (v) maintain the central repository / library.

The team may also engage in training end users on the custom functions.

Companies may also choose to outsource this function to an external consulting firm.

**Model building team**
This team is the core team that will be responsible for building financial model to support the business. This team will be responsible for identifying suitable model architectures and application of the custom functions to deliver financial model that meets the business needs.

The model team can also recommend new custom functions that can be created to the Lambda creation team.

**Model governance team**
The Governance team is the independent pillar that provides oversight and enforces the standards that hold the entire framework together. Their role is not to be a bureaucratic roadblock but to be the strategic enabler of quality and consistency.

The main objective of the team is to establish, maintain, and enforce a proper guidance system for the entire modeling framework. This includes defining naming conventions, setting documentation standards, and mandating architectural best practices. The prescriptions laid out in this guide regarding

these aspects should be considered the bare minimum; this team is empowered to make them more rigorous and to revise them over time as best practices evolve.

The model governance team will also be responsible for managing the library of approved model templates which can be reused.

***Summary of responsibilities across the financial modeling function***

| Task | Primary responsibility |
|---|---|
| Creation / adoption of DA modeling standards | Model Governance team |
| Monitoring the compliance with the standards | Model Governance team |
| Adhering to the standards | All the teams |
| Identifying the need for a particular Lambda | Lambda creation team |
| Creating, validating, and testing the Lambdas | Lambda creation team |
| Updating, monitoring, and managing the library | Jointly by Model Governance team and Lambda creation team |
| Escalation related to Lambda issues | All the teams |
| Retiring erroneous Lambda | Model Governance team |
| Building financial model that fits business purpose | Model building team |
| Identifying models that can be a template | Model building team |
| Approving the templates and managing the template library | Model Governance team |

This chapter outlines the guidance on how the teams should be staffed and function

One man organization or small teams are likely to be able to enforce discipline without necessarily splitting their team into three functions. They may neither have the necessary manpower for such segregation. However, within the team, they need to carry out the all functions outlined in this guide.

## 6.1 Core model governance principles

### *Prescriptions*

**P34. The model governance team must frame policies and practices including the following:**

(i) **Style guides and naming conventions,**
(ii) **Parameter order**
(iii) **Error handling**
(iv) **Documentation best practices**
(v) **Incidence reporting — in relation to Lambda or model governance**
(vi) **Retirement of Lambdas**

The standards set by the model governance must, at the least, include all the prescriptions in this guide.

*Rationale*

- **Increases coding consistency:** Consistency in naming conventions, parameter order and error handling practices make it convenient for financial modelers to work with variety of Lambda.
- **Formalises the best practices guidelines:** Policies of the model governance will set a clear binding guidelines on the best practices to follow for the organization.

**P35. All Global Lambdas and DA models must go through an approval process by the governance team before they are published for common use.**

The governance team's role is strictly restricted to ensure that the respective work adhered to the policies, guidelines and standards. They shall not interfere about the business logic applied in the file.

*Rationale*

- **Ensures compliance with approved practices:** The need for approval will reinforce the need to follow correct guidelines for the developer. Further, it also serves as a secondary check to detect and correct any gaps.

***Recommendations***

**R27. Model governance team should check that all Global Lambda from central repository are reloaded afresh before approving a model for external circulation.**

*Rationale*

- **Ensures that Global Lambdas are not polluted:** Reloading the Lambda under the supervision of the governance team reduces the risk of a tampered Lambda being used in the final output.

## 6.2 Segregation of responsibilities

***Prescriptions***

**P36. Testing and validation must be the responsibility of the Lambda creation team.**

The Lambda creation team can include model builders or members from other team for end user acceptance testing and to obtain feedback. But the ultimate accountability to ensure that Lambda is accurate shall reside with the Lambda creation team.

*Rationale*

- **Clear accountability:** Since creation and testing are with the same team, the Lambda creation team will have unambiguous accountability for producing correct Lambda.

**P37. The model building team must avoid creating or modifying any Global Lambda.**

If the model building team needs a Lambda where one does not exist, they should provide that as feedback to Lambda creation team and shall not create one on their own. However, they can create Local Lambda which are not complicated.

*Rationale*

- **Prevents proliferation of inconsistent or rogue functions:** By ensuring that Lambdas are created by one central team ensures that a given Lambda used by the entire organization is the same.

**P38. Sourcing Lambdas from external party and validating it must be the responsibility of Lambda creation team, and if such a team is absent, then it should be handled by the model governance team. Model building team should not directly procure Lambdas.**

Lambdas procured from external parties need to be tested and validated, which is primarily the responsibility of the Lambda creation team. However, if the entire creation process is outsourced, then the task should be handled by the model governance team, who can validate whether the Lambda meets their internal standards.

*Rationale*

- **Applies quality control for third party Lambdas:** Applying same level of quality checks and approval process for externally procured Lambda ensures that all the Lambda used in their models comply with the internal standards.

**P39. All the teams are responsible for incidence and bugs reporting and these should be reported to the model governance team.**

There may be instances where model governance systems have been bypassed, or a user identifies a bug in an approve Lambda or a model template. All such cases should be escalated to the model governance team, who can then take the necessary action.

*Rationale*

- **Increases the chances of finding bugs:** There is a higher chance of finding issues and bugs and finding them quickly if there are more eyes watching out for them.
- **Ensure single point responsibility for redressals and correction:** Assigning redressal responsibility to the model governance team ensures clear accountability for the same.

**P40. Model governance team must be exclusively responsible for retiring a published Lambda.**

*Rationale*

- **It needs careful execution:** Retiring an existing Lambda is a far more complicated process as it is likely to affect several existing models and may even affect daily operations. Only the governance team can ensure a proper execution of the task.

## 6.3 Operational safeguards

***Prescription***

**P41. Back up of the central repository must be periodically taken and secured.**

*Rationale*

- **Supports business continuity plan:** Secured backups allow firms to quickly restore the repository should it get corrupt or lost for any reason.

***Recommendations***

**R28. Central repository should have dual approval system: Any update or editing of the repository should require approval from the model governance team and the Lambda creation team.**

*Rationale*

- **Maintains sanctity of the repository:** Requiring the presence and approval of the two teams significantly reduces the chance of unauthorized editing of the repository and maintains its sanctity.

## 6.4 Staffing and training

***Recommendations***

**R29. Most of the Lambda creation team should comprise of financial modelers who are power users and capable of creating Lambdas.**

Power users in this context are expert financial modelers who have strong technical skills to develop custom functions using Lambda.

*Rationale*

- **Allows rapid development of context driven functions:** Power users with deep understanding of business context can develop functions that are more practical and relevant and without needing to spend a lot of time on requirement gathering and specifications.
- **Fixes accountability:** Power users can be made accountable for both the purpose, and the design / logic of the function. This ensures that the development process generates a higher return on investment.

**R30. Managerial staff in the team should be well versed with software development principles and frameworks.**

*Rationale*

- **Reduces use of poor design principles:** Managers with strong foundation in software design principles, can correct when Power users — not being well trained software developers — deviate from solid design principles.

**R31. All new joiners should be provided dedicated training on custom built Lambdas. Existing team members should also be given periodic refresher training on the library.**

*Rationale*

- **Makes teams aware of internal IP:** Unlike standard spreadsheet functions, the resources for learning the custom-built Lambdas are not likely to exist outside the organization. Training programs ensure that all team members are aware of such internal IPs.

## Summary

This chapter establishes guidelines emphasizing treating financial modeling as a disciplined organizational task, rather than as an individual craft, to mitigate risks of unchecked complexity and dilution of standards.

It guides for financial modeling function to be split into three functions:

1. **Lambda Creation and Testing**: This function holds end-to-end responsibility for identifying, creating, validating and publishing the Lambdas.
2. **Model building team:** This function comprises of the financial modelers who use the approved, pre-built Lambdas to construct models that serve business needs.
3. **Model governance team:** This the oversight function that sets and enforces modeling standards. Their key role is to serve as a gatekeeper, formally approving all Global Lambdas and models templates before they are released for production use.

One-man organisations and small teams need not have three separate teams, but all the three functions need to be present.

**Key guidelines:**

- **Clear segregation of duties**: Each function has specific role and should not step into the role of the others
    - Lambda creation and testing is the exclusive domain of the Lambda creation team. Model building team should not create or modify Global Lambdas.
    - Applying the right Lambda to the business need is the role of model building team and model governance cannot interferer in it.
    - Model governance team sets and implements the standards. Other teams will need to follow it without exception.
- **Mandatory approval and sign-off**: All Global Lambdas and financial models meant for external distribution must pass a formal approval process by the Model Governance Team.
- **Security and Integrity**: The central repository must be periodically backed up. It is also recommended to require approval from both the Governance and Creation teams before any modifications can be made to the Lambda library, ensuring its sanctity.
- **Training and Awareness**: All new and existing team members should be trained on the organization's custom-built Lambdas, treating them as valuable internal intellectual property

The guide so far focused on the practices that are focused on internal organizational and financial modeling practices to ensure that DA model satisfy all the BEST principles. In the next chapter, we close the loop by providing guidance on auditing practices to be followed by external parties for validating DA models.

# Chapter 7: Auditing dynamic array models

In the context of this guide, auditing refers to verification of the model's reliability by an independent third party.

One aspect of model audit that will undergo a paradigm change for DA models is testing of formula logic in spreadsheets.

Model auditors will need to shift their focus from checking whether the formula logic applied in each cell is accurate to validating the working and integrity of Lambda. This chapter specifically focuses on this aspect.

## Approach to model audit

The end objective of model audit is to ensure that the financial model is reliable. In terms of ensuring correct application of formulae and functions, the model auditor should ensure the following:

- ☐ The modeling team has strong model governance practices
- ☐ No array has been modelled using legacy approach
- ☐ All complex in-memory computations are performed using Lambda
- ☐ The Lambdas are pure functions with referential transparency
- ☐ The Lambda works as described
- ☐ The Lambda has been applied in the correct business context
- ☐ Iterative calculations have not been enabled.

Auditors must continue to follow their legacy approach for simpler expressions used in the cells.

## 7.1 Pre audit and audit planning

**P42. Auditors should obtain model governance practice and standards, if any, from the client and assess the robustness of the system in preventing rogue Lambdas.**

Often, the Lambdas and the models may not be built by the client directly but by an external consulting firm. In such cases, the audit firm should understand the Lambda governance practices of such external firms.

*Rationale*

- **Assess risk of faulty Lambdas:** Understanding the governance practices will help evaluate the risk of faulty Lambdas in the file. This will help plan the effort that needs to be applied for audit.

**P43. Obtain Lambda documentation to understand its purpose and applications.**

*Rationale*

- **Helps evaluate contextual fit of Lambda:** Documentation helps understand which Lambda should be used for which purpose, which will help understand whether application of a Lambda was correct in the given business context.

**P44. Obtain access to the Lambda repository of the client and reload all the Lambda afresh into the model that is being audited.**

*Rationale*

- **Ensures that there are no unauthorized modifications to the Lamda:** Reloading all the Lambda from the common repository eliminates the risk of any wilful, or otherwise, modification of the authorized Lambda at the modeler's end.

## 7.2 Lambda validation

***Prescriptions***

**P45. Inspect the Lambda code to ensure that it maintains referential transparency i.e. no external variables or values other than values of the parameters passed to the Lambda is used in its calculation.**

The check needs to be more rigorous if the client does not have proper model governance team or if the controls appear weak.

*Rationale*

- **Reduces the risk of rogue Lambdas:** Referential transparency ensures that the output of the Lambda is not getting manipulated by any external factors.

**P46. Ensure that all the optional parameters defined in the Lambda are relevant and that the default values assigned to them are appropriate.**

A Lambda that accepts an optional parameter will typically have a default value assigned inside the logic using the ISOMITTED function. If no such default value is applied, then spreadsheet would take their default value as zero.

*Rationale*

- **Prevents intentional manipulation:** Inappropriate default value for optional parameters is one of the ways to deceitfully manipulate the Lambda outcome.

  For instance, the Lambda below, to get average DSCR, uses a smoke screen and returns a DSCR of 2.5, irrespective of the actual number, by misusing the optional parameter.

```
AvgDSCRλ = LAMBDA(Cfads, Pmts, [guess],
              LET(_guess, IF(isomitted(guess),2.5,guess),
                    D_2, AVERAGE(Cfads/Pmts),
                    Return, D_2*Guess + _guess,
                  Return)
                )
```

An inappropriate optional parameter (that does not fit the business context of the Lambda) or an inappropriate default value assigned to them are major red flags. If auditors encounter such situation, they should significantly increase the level of scrutiny.

**P47. Apply the Lambda on proprietary data and compare its results with the existing values to ensure that the Lamda works. If there are variance in the results, client must be required to explain the same.**

Many of the audit firms are likely to have a repository of several financial models on a given domain. They can utilise such models to test and validate the Lambda. **The firms are also encouraged to think about edge case scenarios and design data sets for the same**.

The firms need not treat all variance as an error. It is possible that some variances are a result of improved calculation capabilities of DA models. Clients should be given an opportunity to respond.

*Rationale*

- **It is faster and more reliable to confirm the correctness:** Testing the Lambda on existing data allows audit firms to quickly establish whether the function works correctly as compared to reading through the entire logic and checking the working.

Many Lambdas that are applied directly in the spreadsheet are likely to be wrapper Lambdas that call other Lambdas. From an audit perspective, it is adequate to ensure that the wrapper Lambda, which is directly called in the spreadsheet, works correctly.

**P48. Require validation utilities to be included in the case of highly abstract models. And use the validation utilities to satisfy the accuracy of the calculations.**

As mentioned in para P20, functions that performs all computations in-memory should be accompanied by a validation utility. Auditors should ensure that such utility is available, and they should use those utilities to validate the calculations.

*Rationale*

- **Helps validate a Lambda when proprietary data sets don't exist:** A validation utility, in the form of a separate calculation sheet, that allows users to drive down the calculation will help the auditors to validation Lambda output even if they do not have a proprietary data set to test.

**R32. Ensure that the function that is called in the model is the appropriate function for a given module.**

For instance, a client may have one Lambda that computes amortized interest for a zero-coupon bond and another Lambda for computing amortized interest for a loan with annuity repayment. Auditors would need to ensure that the function that is called for computing the amortized interest is appropriate for the underlying loan type.

*Rationale*

- **Ensure contextual application of logic:** Lambdas are too abstract, and the Lambda name may not completely reveal the true purpose or the underlying logic. There is a risk that a Lambda meant for a different context is misapplied for another context. Thus, validating that the function called is indeed appropriate for the given context becomes critical.

***Recommendations***

**R33. Use LLMs and AI tools, albeit with caution, to understand and evaluate a Lambda.**

As on the date of this version, many of the commonly available LLM and reasoning models have evolved enough to understand and explain the logic of a complex Lambda function or spreadsheet formulae. Such a capability can be utilized to understand the Lambda written by other firms. However, given the probabilistic nature of these models, end users should be cautious while understanding and accepting the response.

*Rationale*

- **Get an additional non-binding perspective:** It reduces the chances of failing to spot a mistake, which is inherent in the audit process.

## Summary

In this chapter, we discussed that auditing spreadsheet formulae in dynamic array models involves a paradigm shift. Auditors need to shift their focus away from checking the accuracy of the formula logic cell-by-cell to validation of Lambdas.

**Key guidelines**

- **Evaluate governance practice:** Check whether the client has strong governance practice over Lambda, and in their absence, plan for a more stringent scrutiny.
- **Validation over comprehension:** Model auditors need not understand the logic built inside a Lambda. They should rather focus on the following:
    - The Lambda upholds referential transparency
    - Does not have misleading default value for optional parameters
    - Its results, applied on test data sets with edge cases, matches with the expected results.
- **Test contextual fit of Lambda:** Get the documentation of Lambda from the client to understand the purpose and application of Lambdas and check whether correct Lambda has been applied in the given business context.
- **Use legacy technique for simpler expressions:** Several expressions or formulae in DA models are likely to be simpler, albeit using newer functions, and directly typed in the cell. Auditors can review the logical accuracy of such expressions using existing methodologies.

---

With this chapter, we conclude the guidance on best practices for dynamic array models. In this guide, we established the BEST principles and detailed a 360° framework encompassing input architecture, model calculations, Lambda designs, testing and debugging, organization structures and governance and guidance on audit techniques for DA models.

The prescriptions and recommendations in this guidance, hopefully addresses concerns and questions of financial modelers in relation to adopting dynamic array practices in financial modeling.

In the appendices section, we discuss certain alternative views on some of the critical prescription in this guide along with their pros and cons and we also discuss the characteristics of a good export sheet.

If you have any views, feedback related to the document, or if you would like to contribute or collaborate in future to enhance this guidance note, you can email us on info@profectus.in.

# Appendices

## Appendix 1: Evaluating the case for horizontal input layout

This guide prescribes use of structured table for organizing time series input, which would required time series input to be placed vertically. This is likely to create a problem for end users, used to horizontal layout, when eyeballing the data. It also increases the chances of data entry error, as most source reports are likely to have time series placed horizontally.

Therefore, there are clear arguments in favour of keeping the inputs horizontally.

But we need to evaluate options that allow us to work with such layout in a DA model. We have identified three options:

1. **Use normal ranges and update formula when data is added:** This requires that when a new period is added, the dynamic array formula should be modified to include the new period.

   ***Disadvantage***

   👎🏾 This option clearly **fails the "Thorough" principle** as the modeler is leaving the maintenance task to the end user — which creates higher risk. Further, manually modifying DA formula is way more laborious than copy-pasting practice of legacy approach.

2. **Use normal ranges along with TRIMRANGE or OFFSET functions:** The reference to a normal range can be made to dynamically increase when a new data is added by using the new TRIMRANGE function or the erstwhile OFFSET function.

   ***Disadvantage***

   👎🏾 This option, as already discussed in the guide, has the risk of some stray data resulting in unwarranted expansion of the result array, and causing the model to fail. This option **fails the "Stability"** principle.

3. **Place the data horizontally in a structured table and reshape with new formulas:** As structured tables expand horizontally, a work around solution is to place the input horizontally in it. The input can then be transformed, if needed, using variety of inbuilt functions or a custom Lambda to be used with financial model.

   ***Advantage***

   👍🏾 The thorough principles favours making the financial modeler take the extra efforts in transforming the date to make it easy for the end users.

   ***Disadvantages***

   👎🏾 This option has too many failure points that require extensive fail proofing:

   ❌ An intuitive reordering of rows (as tables make is easy to sort rows) will result in calculations picking data from wrong rows. This will need to be prevented either by conscious removing

the filter button or by using complex double XLOOKUP or INDEX with double match to take care of the years as well as line items. This threatens the **Efficiency** principle.

❌ There is a significant level of risk that an end user types an expression in a new column, and the spreadsheet treats it as a calculated column. (i) The user may fail to delete the expression from some rows or (ii) accidentally restore the calculated column. It is difficult to fail proof this risk and thus it fails the **Thorough** principle

👎 It lacks a future road map. Once, DA modeling practices mature, a well-designed dynamic PowerApps front-end or forms can be integrated with structured tables in their unadulterated form. However, such a vision is perhaps not feasible for the horizontal layout.

**Verdict: Prohibited**

Option 3 is the only option that comes close to being viable within the BEST framework but given the potential number of failure points and lack of future road map for enhancement, the author believes that balance is better maintained by letting the user bear some minor inconvenience for the sake of model's integrity.

The future road map for enhancing the user experience should focus on using other tools that allow a dynamic front-end to be integrated with spreadsheets.

## Appendix 2: Evaluating "table of scalars" for scalar input

This guide prescribed that scalar inputs should be placed in cells, in line with legacy practices. An alternative case exists for keeping all scalar values in a table and referencing them through XLOOKUP.

For instance, scalar variables such as cost of equity, WACC, terminal growth rate etc., can be placed in a structured table with one column having the label and another the values. Then a variable such as WACC can be obtained using XLOOKUP with "WACC" as lookup value and the label column being the Lookup array.

***Advantages:***

👍 It makes it easy to add more scalar variables to the model by just adding them as row. XLOOKUP would be able to automatically pick up the variable.

👍 It brings architectural consistency by making table as common framework for all input variables.

***Disadvantages:***

👎 It is extremely vulnerable and fails the **Stability** principle as even a small edit to the label will cause dependent formulae to fail. And if the cells are protected to prevent such editing, then the table does not expand dynamically and negates the advantage.

👎 Table structures are rigid and prevents an intuitive flow and structuring for scalar variables

**Verdict: Prohibited**

This alternative has limited advantages, but significantly higher risk. Therefore, this guide prohibits it.

## Appendix 3: Evaluating the case for mixed approach

This guide takes a stance that the choice between DA model and legacy approach must be binary. Users will have to pick and choose one of the approaches. But there are arguments in favour of mixed approach that recommends using the dynamic for most cases but sticking to legacy approach for modules that are more complicated.

***Advantages:***

The main arguments made in its favour are these two.

- 👍 It makes the model easier to maintain compared to pure legacy approach and at the same time it is easier than a pure DA approach.
- 👍 The concession will allow many financial modelers to start adopting dynamic array approaches.

***Disadvantages:***

- 👎 It creates deceptive illusion of dynamism, and this can lead to severe model error. For instance, let us say most parts of a model are dynamic except a structured loan repayment schedule. When timelines are extended, the end user — especially who has inherited the model from another — may assume the entire model to be dynamic and miss to manually the loan schedule. This will lead significant overestimation of cash flows and solvency position. It, thus, **fails the Stability principle.**
- 👎 Extending from the previous argument, the model depends on careful rollover of the loan schedule and updating the dependent formula by the end user and, thus, **fails the Thoroughness principle.**
- 👎 It is laborious and has more riskier maintenance practice compared to legacy approach. In legacy approach, when timelines must be extended, one just needs to copy paste the last calculated column. In the mixed approach, the last column needs to be copy pasted for the legacy schedule. And every dependent formula needs to be identified, and the first cell of the array needs to be edited to extend the range referred in the formula.

  Thus, for the benefit of making it easy for the financial modelers, it makes the end users work more complicated and introduces high degree of spreadsheet risk. Thus, it gets the **balancing principle completely wrong.**

**Final Verdict: Strictly prohibited**

The mixed approach with deceptive illusion of dynamism is far riskier than legacy approach and significantly increases the complexity of maintenance, especially as the models are handed over to other users. The mixed approach is a recipe for disaster waiting to happen.

## Appendix 4: Evaluating the case for customized error messages

This guide recognizes that Lambda functions should have error handling procedures built into it. There is also a case made to have the Lambda return a customized error message — rather than standard spreadsheet error types — to the let users know the exact mistake, .

For instance, let us say we have a Lambda that returns the minimum value for each row or column of two vectors and written as follows:

```
MinArrayλ = Lambda(A,B, Map(A,B, MIN) )
```

For the above Lambda to work, both the vectors should be either a column vector or a row vector. The function will return a matrix of mostly #NA error, if one is a row vector and another is a column vector. This error is specific to the way the Lambda was designed to work, which the end user may not realise.

Hence, an argument is that the function should return a custom message, and could be created as follows:

```
MinArrayλ = Lambda(A,B,
                        If(//check that no. of rows &columns is same
                            (rows(A)=rows(B))*(columns(A)=columns(B)),
                            //If yes, calculate the minimum of both arrays
                                    Map(A,B, MIN),
                            //Else return error
                                    "Array Dimensions mismatch"
                        )
```

***Advantage:***

- In the case of a complex Lambda, the step causing the error would be buried inside the Lambda function and end users would not be able to trace that step. If the Lambdas are designed to anticipate these errors and provide a customized message, it would make it easier for the end users to fix it.

***Disadvantage:***

- While the message conveys the error, from a machine perspective, the function has produced a valid text output. Therefor, built-in error handling functions of spreadsheets will not recognize them as error. This may need us to have a big library of customized error handling functions that recognizes the error of each lambda — a near impossibility.
- Novice end users, or those who are not native speakers of the language, may not be even realise that the function is conveying an error, especially if the function was meant to produce a text output.

**Verdict: On Hold**

The argument in favour of the custom error message is very strong, especially since the guide says that end users do not have understand the formula logic. But we need to develop sufficient frameworks on how to tackle subsequent error handling steps, and how to communicate to non-native users.

## Appendix 5: Guidance principles for export sheets

An export sheet is a dedicated worksheet within a financial model that consolidates and presents key values (can be input, output, or an intermediate step) for consumption by a downstream model, analytics engine, or other external systems. In the context of this guide, they are very valuable when a downstream model needs to extract data from a given model using ETL tools (such as Power Query).

The design principles of an export sheet deserve a dedicated chapter. However, to keep the focus sharp on key DA modeling practices, we have excluded it from the main part of the guide.

This appendix outlines the key principles to be followed for a robust export sheet(s)

**Key guidelines**

1. **Design to be machine-friendly:** Export section should not have headings — sheet level or section level, and there should not be any blank row or column before or in-between the export values. User friendly formatting is unnecessary, and merge cell must be avoided. Sheet names must be used to provide indication of the sheet's content.
2. **Data with different dimensions should be in different export sheets:** Data that have different dimensions should not be forced fit into a single sheet. For instance, a standard equity valuation scalar data points such as WACC, Beta, equity value etc., should be in a separate export sheet while time series data should be in a separate export sheet. Vectors of other dimensions — such as production capacity across geographies — should be in a different sheet.

   This allows downstream models to load the entire sheet together and work seamlessly even if contents of sheet get reordered or relocated within the same sheet.
3. **Export sheet should be auto populated with a thorough formula and should not require any user interference:** Every direct interaction of end user with the export sheet is a risk to integrity of the sheet. Therefore, these sheets need to be auto populated without need user interference. Hiding the export sheet from the main view to prevent accidental alteration will not be a wrong choice.
4. **Its content can increase but not decrease:** Removing the content of an export sheet can have severe cascading effect on downstream models that have been using such content. Therefore, existing content cannot be removed.
5. **Content should be driven by raw data and not user interface choices:** Unlike the output sheets, which can have interactive elements to let users modify the view, export sheets should have rigid contents. For instance, the start and end period of export sheet should be driven by earliest and latest dates in raw data rather than what user chooses in an interface. This builds on the previous point and ensures that contents of the export sheet do not vanish based on user inputs.

## Invitation for feedback and contributions

Our goal is to make this guide a community mission with feedback and contributions from financial modeling professionals, worldwide.

If you have thoughts on the material or would like to contribute to the future versions, please email us in the following address:

✉ **info@profectus.in**

## About Profectus

Profectus is a consulting and training firm dedicated to delivering niche solutions that help companies navigate complex financial landscape with confidence and clarity.

The company, and its predecessor, pioneered financial modeling with dynamic arrays as early as in 2020 and has delivered effective solutions to clients across industries including real estate, infrastructure, consulting, private equity, financial services and IT services, over the last five years.

Visit our website to know more about us.

🌐 https://profectus.in

PROFECTUS