

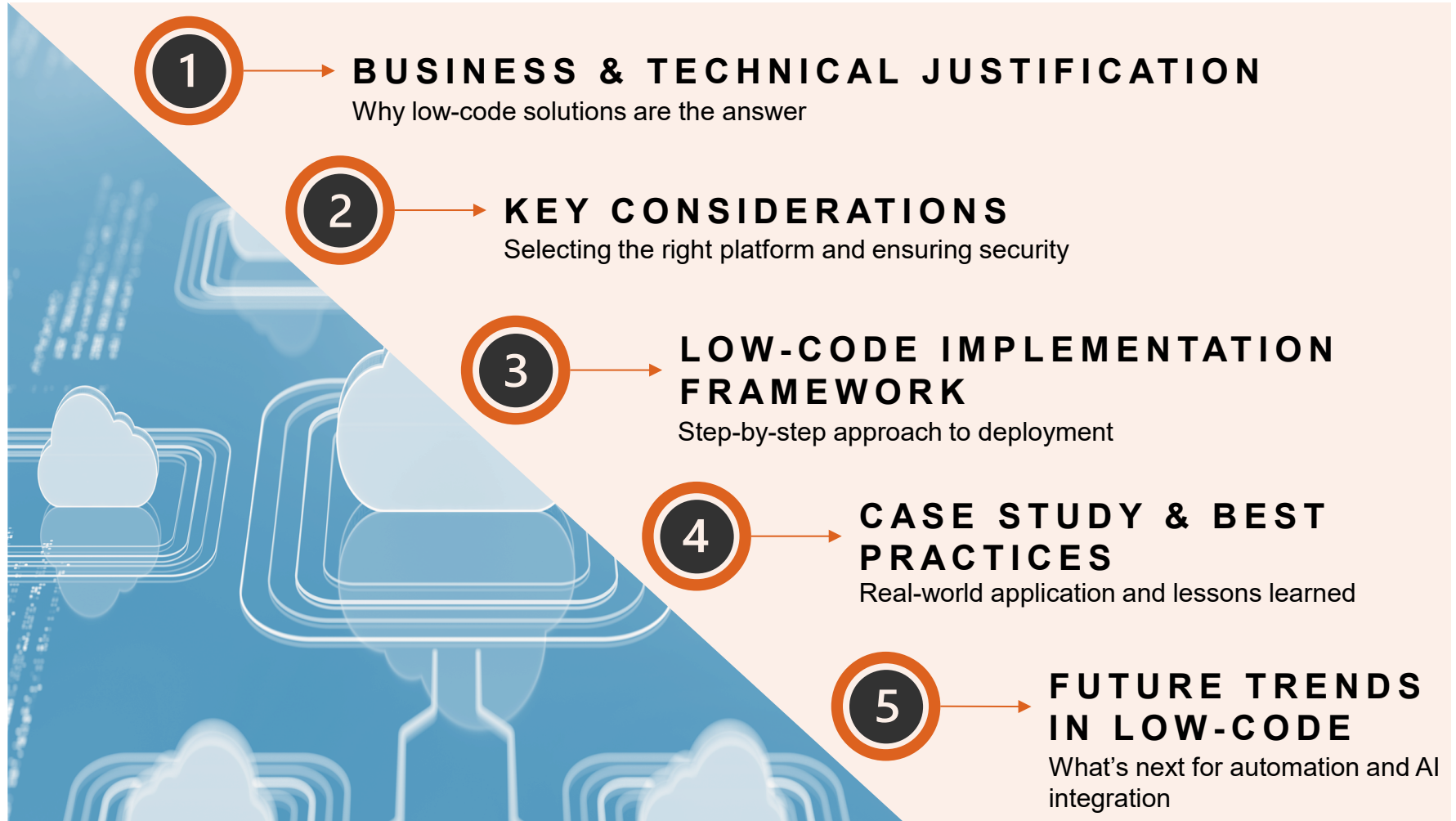


Implementing Low Code Solutions for Financial Surveillance and Reporting

Ryan Nicholos, Connor Maloney, and Sebastian Rodriguez
Traconis

What We Will Cover

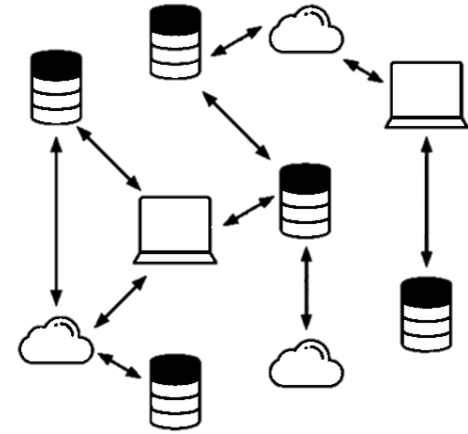
A Roadmap to Understanding Low-Code Implementation for Financial Management



Introduction

Siloed Data Limits Financial Decision-Making & Efficiency

Stakeholders in the private, commercial, and government landscapes lack visibility and access to critical data, leading to inefficiencies and wasted resources.



FACTORS:



DATA SILOS

- Data Availability vs. Data Accessibility – While organizations have vast amounts of data, much of it remains siloed in disparate locations, limiting its utility for analytics and decision-making.



LIMITED VISIBILITY

- Challenges of Multi-ERP Integration – Connecting data across multiple ERP sites is a common challenge, requiring seamless integration to gain a holistic view of enterprise data.



OPERATIONAL IMPACT

- Traditional vs. Low-Code Approach – Previously, custom-made integration required highly skilled programmers, whereas low-code solutions now enable faster, more efficient data connectivity with minimal coding.

RESULT:

These inefficiencies slow decision-making, inflate costs, and compromise mission readiness.

Technical and Business Justification

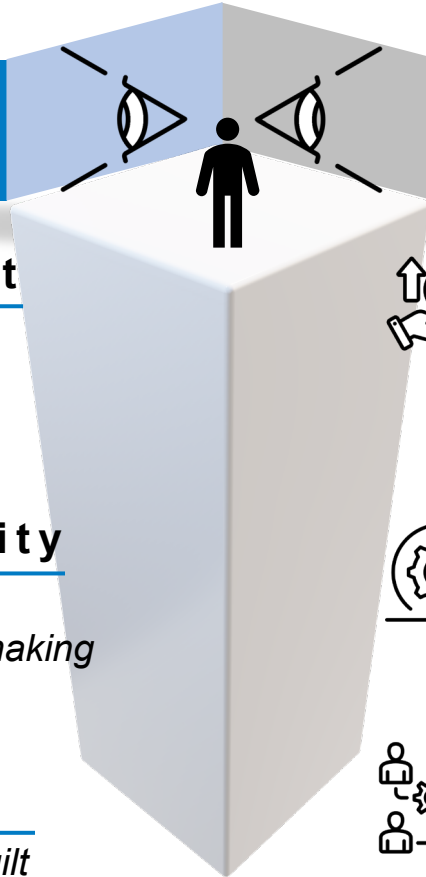
Low-Code Accelerates Development, Reduces Costs, and Enhances Agility

TECHNICAL

Feasibility & Appropriateness Focus

BUSINESS

Value & Strategic Alignment Focus



Faster Time to Market

Rapid development and iteration, accelerating delivery timelines compared to traditional coding



Cost Efficiency

Reduce development costs by enabling citizen developers and automating repetitive tasks, lowering reliance on expensive custom coding



Scalability & Flexibility

Scale with demand and integrate seamlessly with existing systems, making them adaptable to evolving needs



Improved Agility

Quick adaptation to changing business requirements, supporting continuous innovation and market responsiveness



Reduced Complexity

Drag-and-drop interface and pre-built components simplify development, reducing the need for specialized technical expertise



Enhanced Collaboration

Empowering business users to develop their own solutions fosters collaboration across teams, bridging the gap between technical and non-technical stakeholders

Key Considerations

Choosing the Right Low-Code Platform is Critical for Success

DIFFERENT LOW-CODE PLATFORMS HAVE MANY COMMON FEATURES



COST:

- Licensing is typically required for every user of apps and workflows. Sometimes licenses can be procured on a per-app or per-workflow basis.



INTEGRATION:

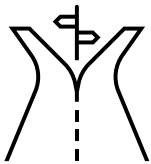
- Adoption is typically easier when tools natively integrate with existing software. APIs create efficiencies in data management and analysis.



EXPERIENCE:

- Developers can quickly develop solutions with software they've used before. There will be a small learning curve to use a new platform.

Do a business case analysis...



An organization must consider these factors when procuring a low-code platform. Lower-cost solutions will not always be cheaper in the long run, as lack of adoption, lack of integration, and developer learning curves can halt implementation efforts.

Platform selection should be a technical and financial decision.

Key Considerations for Implementation

Choosing the Right Low-Code Platform is Critical for Success

Evaluating Low-Code Platforms

Considerations like cost, integration, developer experience, and security are essential in selecting the right platform

Data Governance

Ensuring proper access control and security measures, especially with sensitive data, is critical to maintaining integrity and compliance

Understanding Client Needs

Establishing clear problem statements and business requirements is crucial

Integration with Existing Systems

Pre-built integrations and APIs can streamline development and reduce complexity

This ensures that the solution aligns with both the technical and business objectives of the client while maintaining security and efficiency.

Best Practices for Successful Implementation

Presented at the ICEAA 2025 Professional Development & Training Workshop - www.iceaaonline.com/at12025

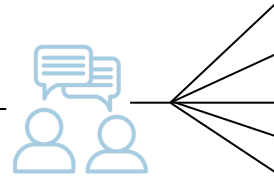


Systems Engineering paired with strong communication drives success

BEST PRACTICE

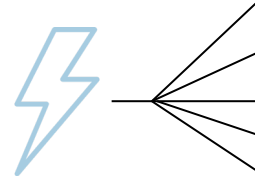
KEY DRIVERS

Meaningful stakeholder end user communication and engagement



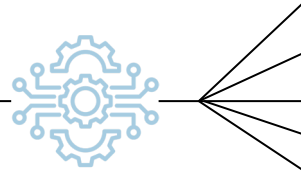
- Clarify requirements
- Enhance adoption
- Ensure governance & security compliance
- Improve agility

Staying sharp on new low-code capability and industry leading practices



- Utilizing modular design principles
- Incorporating data analytics
- Building for accessibility

Systems engineering approach



- Improved Requirements Management:
- Enhanced Scalability & Integration
- Risk Mitigation
- Lifecycle Management & Sustainability

Our Implementation Framework

A Structured Approach Ensures Sustainable Low-Code Deployment

APPLICATION LIFECYCLE MANAGEMENT (ALM)



➔ **Define problem, develop requirements, model data, select tools**

- Defining the problem and developing the requirements with the customer are crucial steps in the development cycle
- Properly modelling data and relationships allows for proper data storage and leads to efficiencies later on
- Selecting the right tools requires a breadth of technical expertise in the platform being used

➔ **Junior developers begin work under senior developer oversight**

- Typically 1-2 junior developers do most of the development, with assistance from a senior developer when needed
- Staying up-to-date on the latest features and trends of the platform being deployed is important to always provide the best possible product
- The use of development-test-production environment frameworks keeps issues out of the final product and provides a place for testing

Our Implementation Framework

A Structured Approach Ensures Sustainable Low-Code Deployment

APPLICATION LIFECYCLE MANAGEMENT (ALM)



➔ **Peer review and user acceptance test**

- The project team will test features against requirements internally to discover bugs early
- End users can then test the solution and request bug fixes or additional features



➔ **Publish to production environment**

- Publish updates to the end-user environment
- Distribute any SOPs or video demonstrations to users, if desired



➔ **Repeat the development cycle for bugs and new features**

- Regularly meet and discuss issues with process owner(s)
- Track version history and features developed to maintain configuration management



Our Implementation Framework

Our team developed a tool to manage our ALM process, tracking all products and their releases.





DIGITAL PRODUCT MANAGEMENT

- *2 sprints are used to 'time-box' our development, allowing us to pivot to urgent requirements as needed.*

- *Features are assigned and tracked through planning, development, test, and release.*

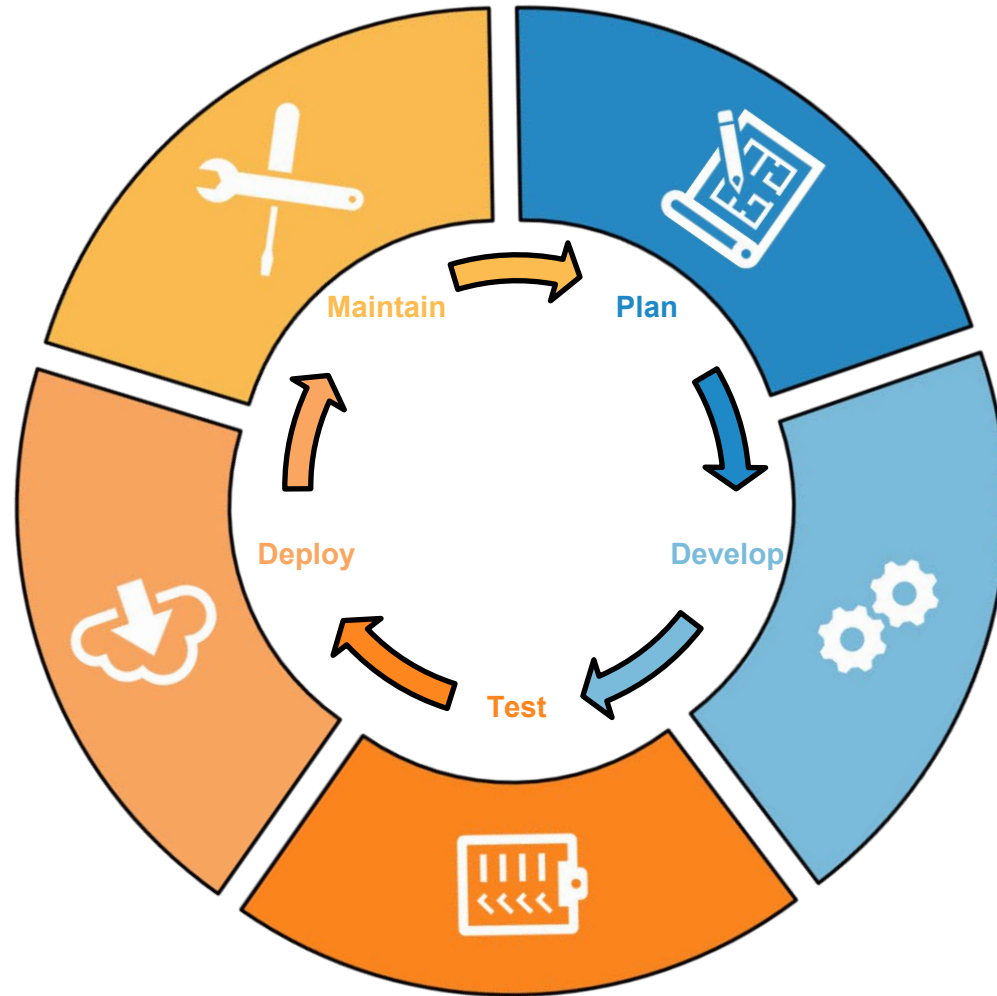
Case Study: Financial Surveillance Tool

Delivering insights that ignite action

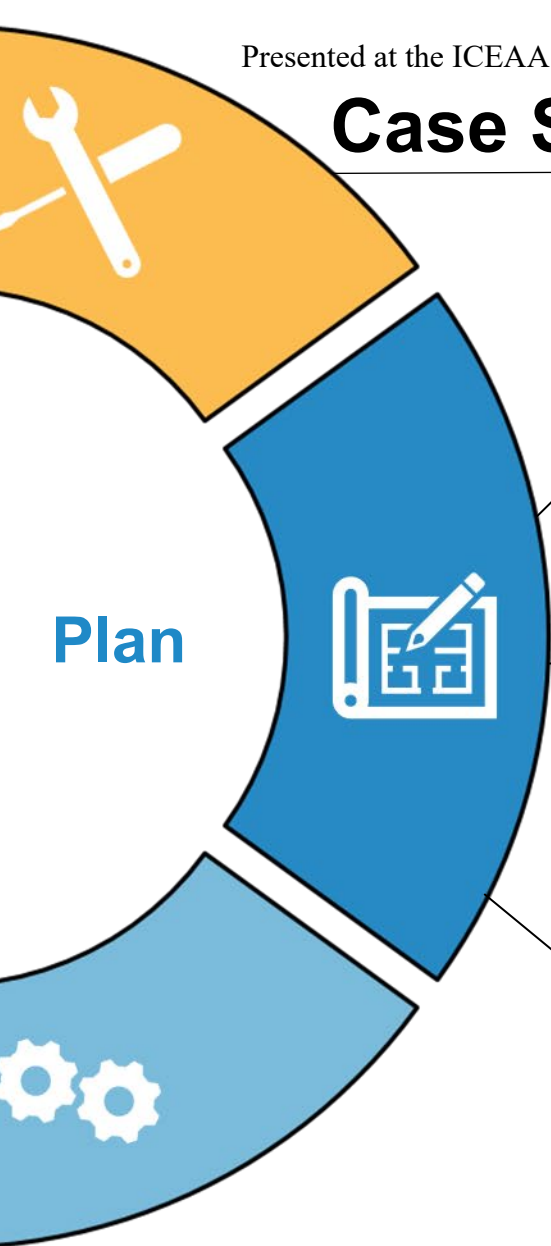
	Better Understand Financial Status	Connect Legacy Data Sources	Open Data Platform for Financial Report Development	
Objective	Enable clients to easily reference financial data for Planned Benchmarks vs Actual Execution	Seamlessly ingest, store, evaluate and optimize data management from siloed authoritative data sources	Build databases, document catalogs, and curate custom data visuals and reports for briefing to leadership	
	 <p>Program Selection</p>	 <p>Architecture Mapping</p>	 <p>Maturity Assessment</p>	 <p>Database, Reports, and Dashboard Generation</p>
Approach	<ol style="list-style-type: none"> Determine business requirements and success criteria Map data architecture for Program data Develop project plan to deliver low-code application solution 	<ol style="list-style-type: none"> Utilize APIs to connect legacy ERP systems Assess maturity of data management for required data <ul style="list-style-type: none"> Properly formatted In Excel or native database? Put existing database in accessible location or create business application/workflows to replace 	<ol style="list-style-type: none"> Work with data owners on case-by-case basis to set up process and database for managing data; document data 'catalog' Develop desired reports and dashboards 	

Our Implementation Framework

Application Lifecycle Management (ALM)



Case Study: Plan



COORDINATE WITH PROCESS OWNERS

- *Budget Financial Managers (BFMS)*
- *Principal/Assistant Project Managers (PAPMs)*

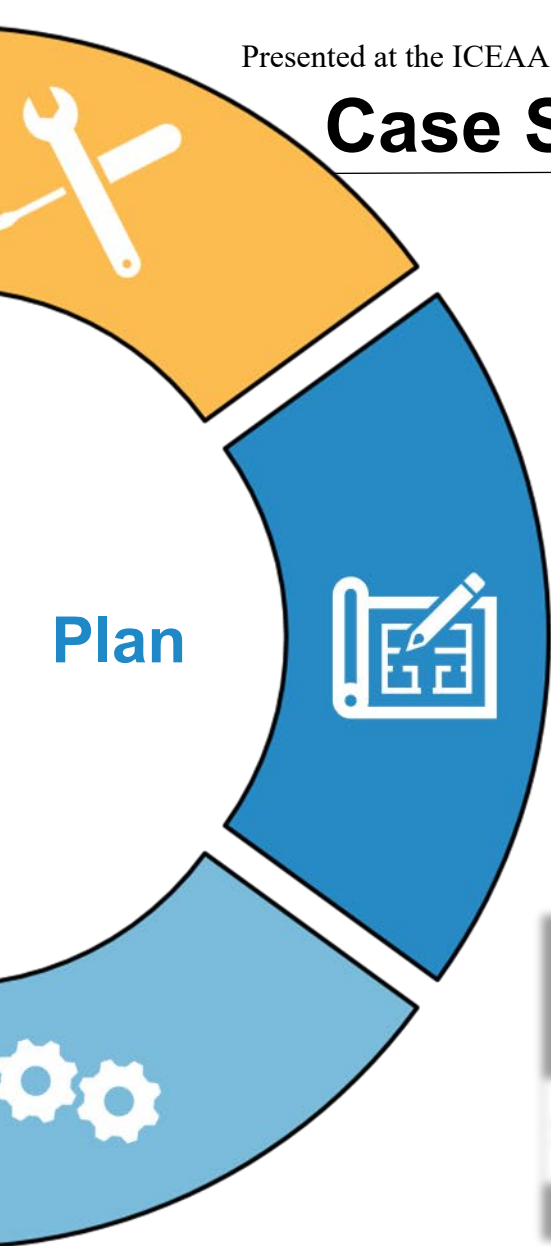
DEFINE REQUIRMENTS

- *Track Obligations & Expenditures*
- *Evaluate against Benchmarks*
- *Compare Plan vs Actuals*

IDENTIFY DATA SOURCES

- *ERP, NEPS*

Case Study: Plan (Identify Data Sources – ERP)

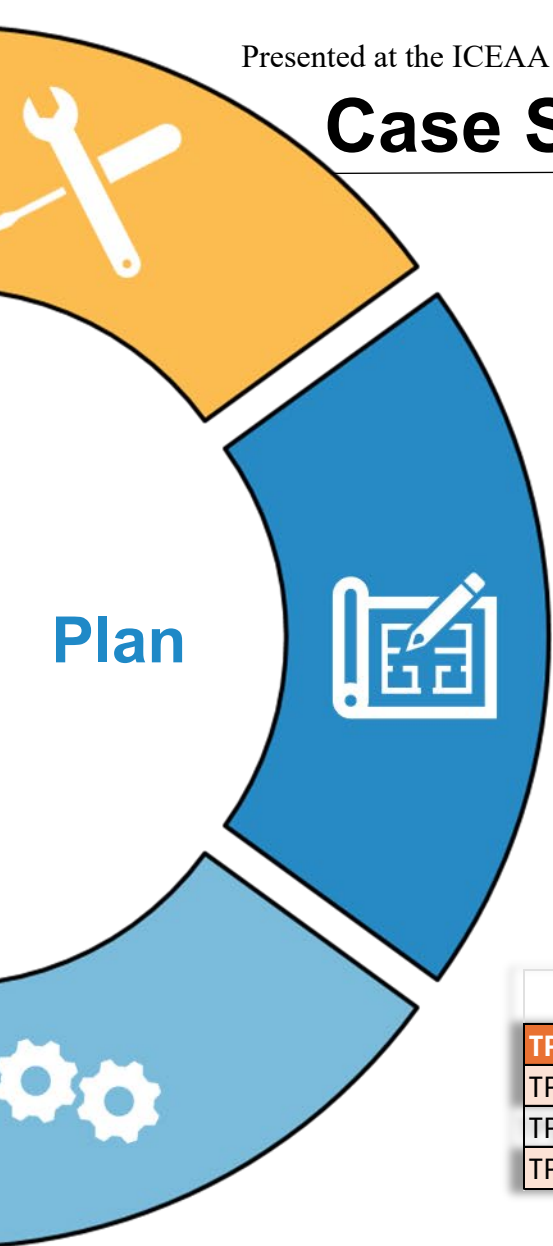


ERP (Actuals)

- ZRQs Run Excel Sheet
- Tracks Funding Released, Obligations and Expenditures
- Data is a snapshot in time
- Hierarchical Format using **Project Structures**

Project Structure	Text	Budget	Obligations	Actual Costs
1	Overhaul Project	\$10M	\$8M	\$4M
1/1.1	Engine Maintenance	\$4M	\$3M	\$3M
1/1.2	Hull Repair	\$6M	\$5M	\$1M

Case Study: Plan (Identify Data Sources – NEPS)

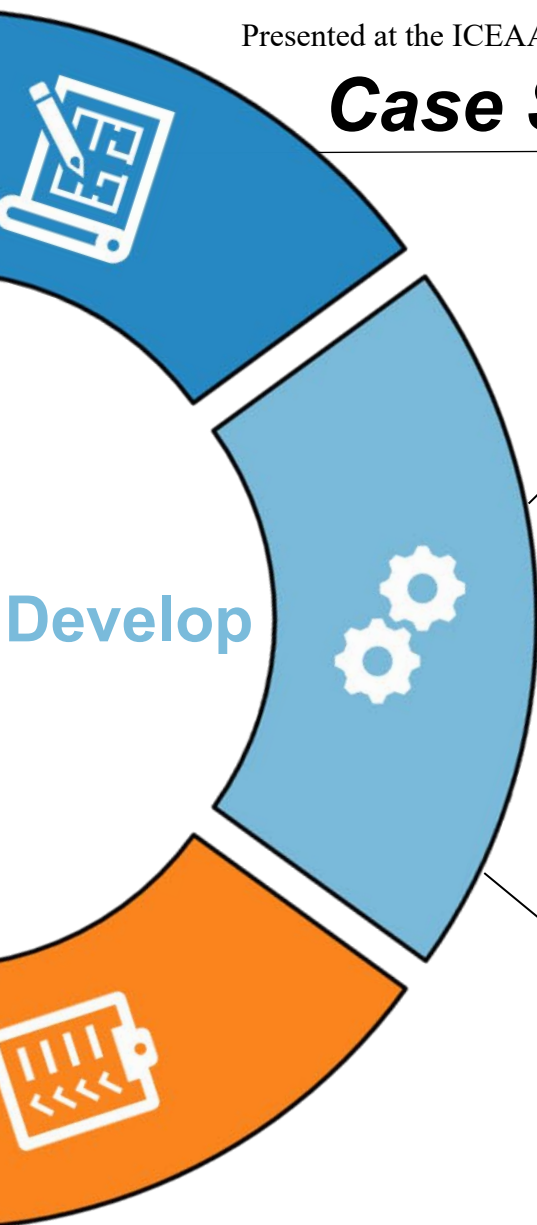


NEPS (Plan)

- CAR Report Excel File
- Phases Obligations and Expenditures through the Fiscal Year
- Broken down using Task Planning Sheets (TPS) Numbers

	Obligations											
TPS Number	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
TPS-20001	\$3M	\$3M	\$2M			\$1.5M		\$1.5M			\$2M	
TPS-20002		\$5M	\$5M	\$4M					\$3M			\$2M
TPS-20003	\$1M			\$1M	\$2.5M				\$3M	\$3M		

Case Study: Develop (ETL Process)



EXTRACT

- *Connect to Raw Data Source*
 - *Excel, SQL Server, JSON, PDFs, etc.*

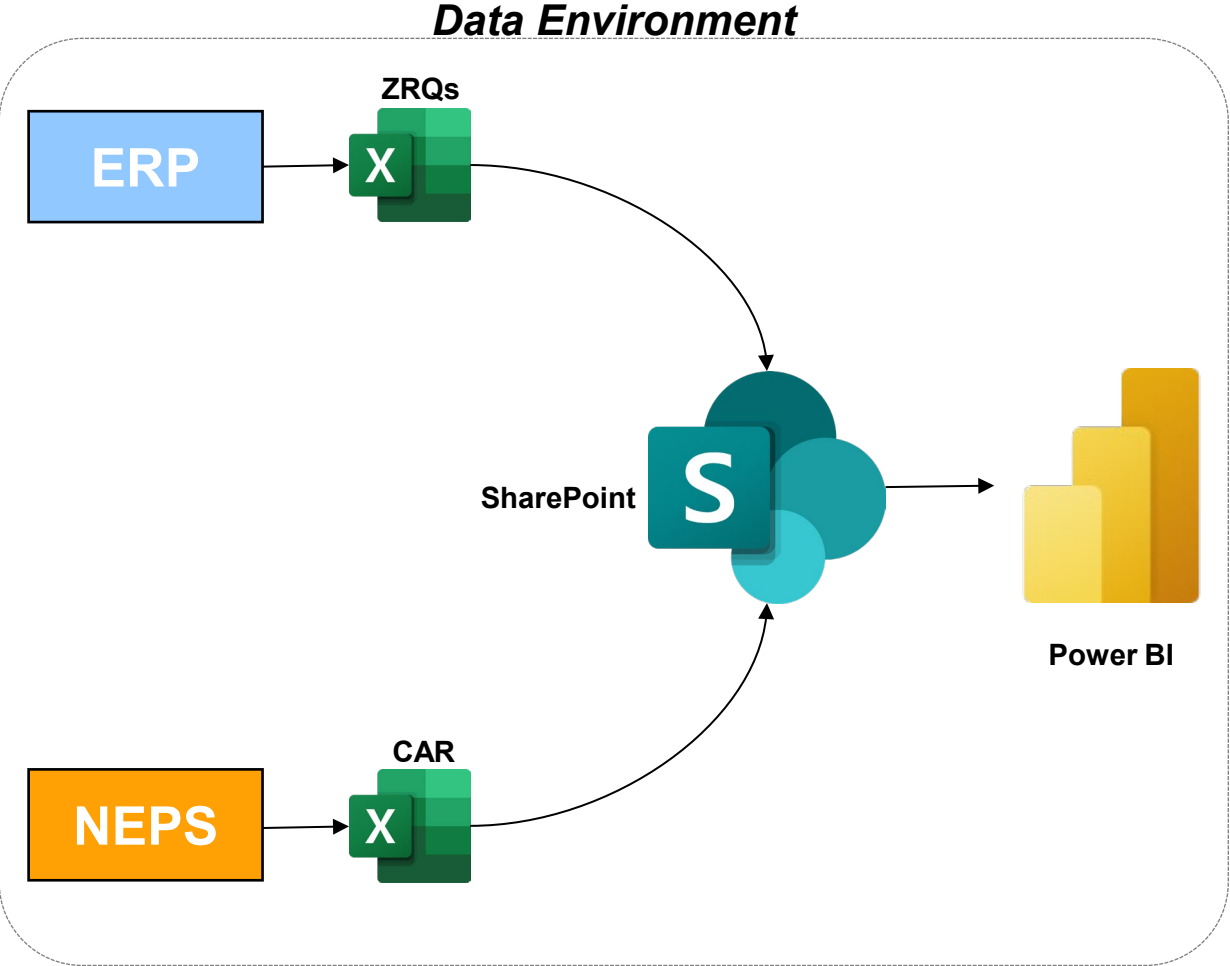
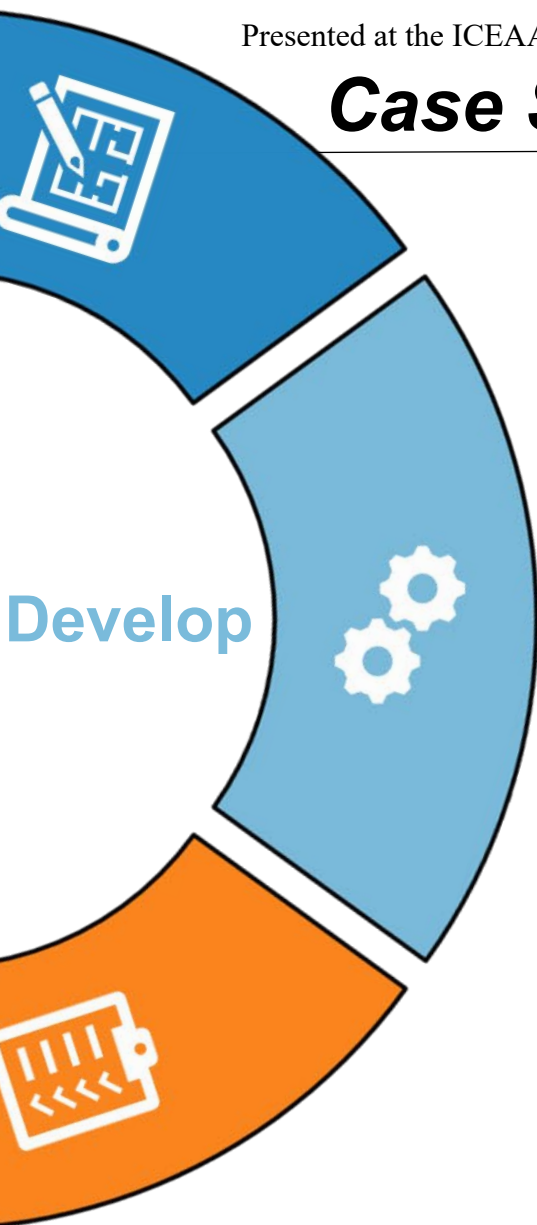
TRANSFORM

- *Clean Data*
 - *Remove Columns, Filter, Format, Splitting, etc.*
- *Restructure Data*
 - *Pivot, Transpose, Group, Merge, Append, etc.*

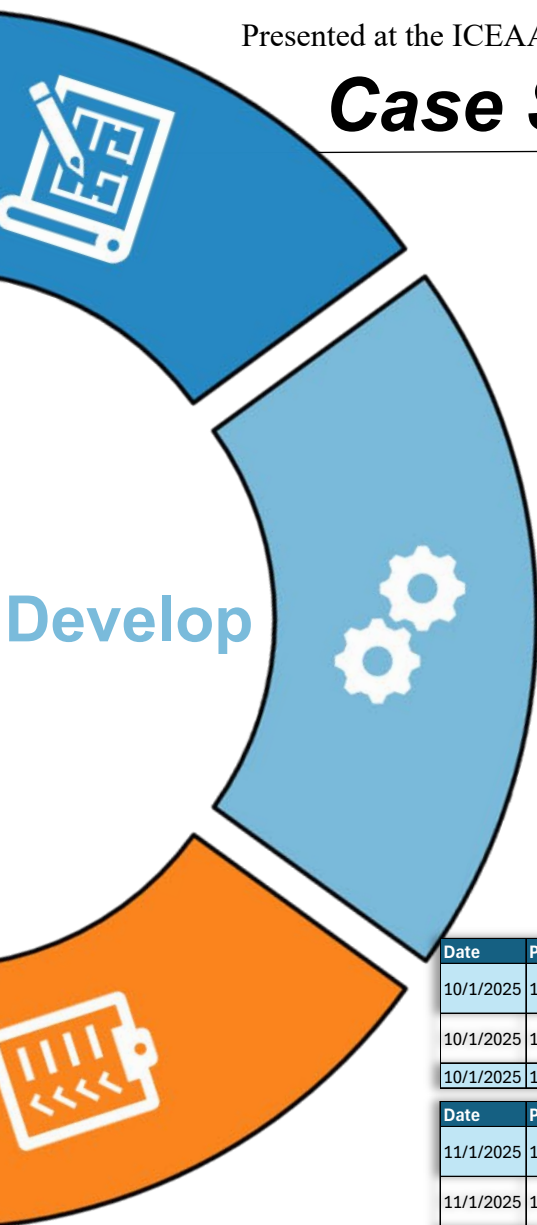
LOAD

- *Establish Relationships*
- *Create Calculated Columns/Measures*
- *Conditionally Format Visuals*

Case Study: Develop (Extract)



Case Study: Develop (Transform - ERP)



APPLIED STEPS

- Source
- Filtered Rows1
- Filtered Rows2
- Added Custom
- Expanded Custom
- Split Column by Delimiter
- Changed Type
- Split Column by Delimiter1
- Changed Type1
- Added Custom1
- Removed Other Columns
- Expanded Custom.Data
- Added Conditional Column
- Removed Columns
- Promoted Headers
- Filtered Rows
- Split Column by Delimiter2
- Added Conditional Column1
- Split Column by Delimiter3
- Changed Type2
- Added Conditional Column2
- Removed Columns1
- Added Custom2
- Renamed Columns
- Removed Columns2
- Changed Type3
- Trimmed Text

- Split File Name to get Date
 - “ZRQs_10_1_2025
- Consolidate ZRQs Files
- Create Parent and Child Project Structure Columns

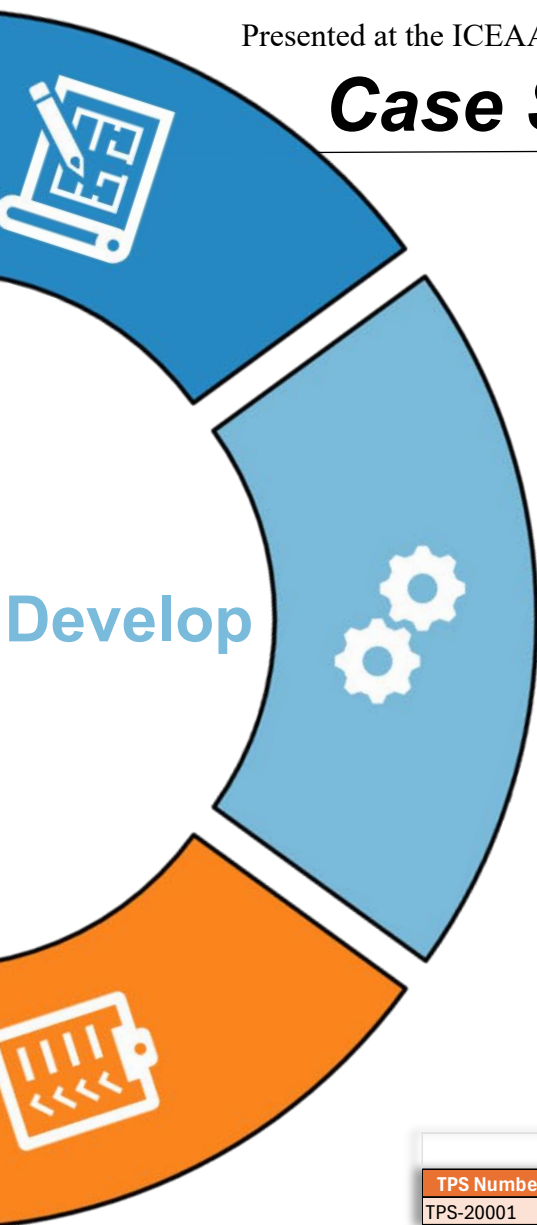
BEFORE

Date	Project Structure	Text	Budget	Obligations	Actual Costs
10/1/2025	1	Overhaul Project	\$10M	\$8M	\$4M
10/1/2025	1/1.1	Engine Maintenance	\$4M	\$3M	\$3M
10/1/2025	1/1.2	Hull Repair	\$6M	\$5M	\$1M
11/1/2025	1/1/1900	Overhaul Project	\$10M	\$9M	\$5M
11/1/2025	1/1.1	Engine Maintenance	\$4M	\$4M	\$3M
11/1/2025	1/1.2	Hull Repair	\$6M	\$5M	\$2M

AFTER

Date	Project Structure	Child	Parent	Text	Budget	Obligations	Actual Costs
10/1/2025	1	1	null	Overhaul Project	\$10M	\$8M	\$4M
10/1/2025	1/1.1	1.1	1	Engine Maintenance	\$4M	\$3M	\$3M
10/1/2025	1/1.2	1.2	1	Hull Repair	\$6M	\$5M	\$1M
11/1/2025	1	1	null	Overhaul Project	\$10M	\$9M	\$5M
11/1/2025	1/1.1	1.1	1	Engine Maintenance	\$4M	\$4M	\$3M
11/1/2025	1/1.2	1.2	1	Hull Repair	\$6M	\$5M	\$2M

Case Study: Develop (Transform - NEPS)



APPLIED STEPS

- Source
- Filtered Rows
- Split Column by Delimiter
- Filtered Rows5
- Changed Type
- Split Column by Delimiter1
- Changed Type1
- Added Custom
- Renamed Columns
- Removed Other Columns
- Sorted Rows
- Grouped Rows
- Custom1
- Expanded Custom
- Custom2
- Expanded Custom1
- Filtered Rows1
- Removed Other Columns1
- Expanded Data
- Added Conditional Column
- Removed Columns
- Promoted Headers
- Filtered Rows2
- Unpivoted Only Selected Columns
- Replaced Value
- Replaced Value1
- Split Column by Delimiter2
- Changed Type3
- Added Conditional Column1
- Filtered Rows4
- Removed Columns1
- Pivoted Column
- Changed Type2
- Added Custom1
- Trimmed Text
- Appended Query
- Removed Columns2

- Filter to most recent CAR Reports
- Unpivot Month Columns
- Remove Unnecessary Columns
- Pivot Values to create Obligations and Expenditure Columns

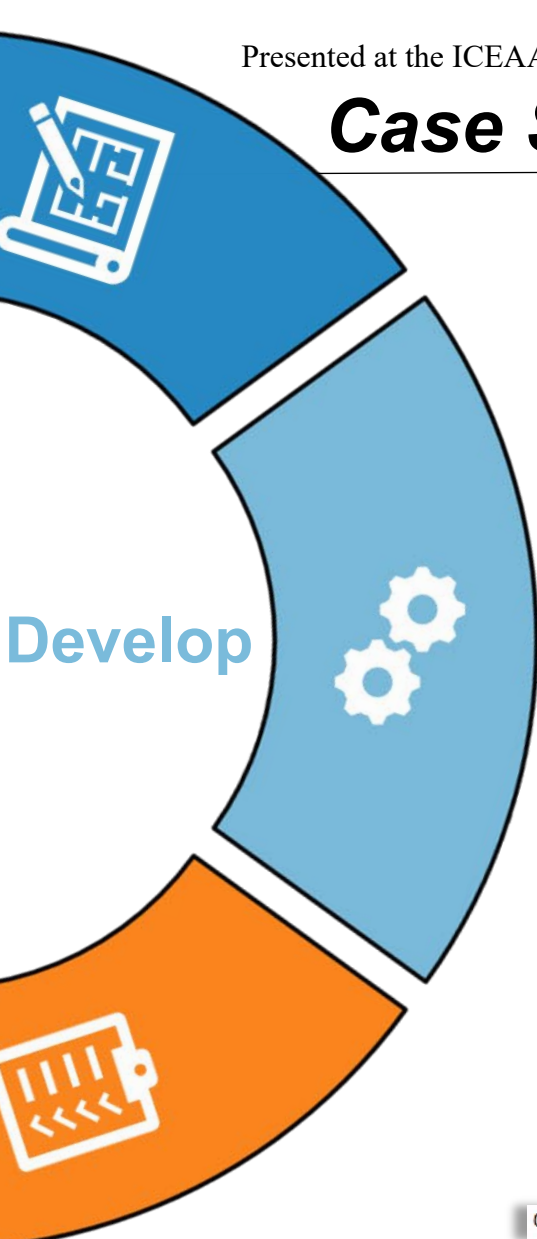
BEFORE

	Obligations											
TPS Number	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
TPS-20001	\$3M	\$3M	\$2M			\$1.5M		\$1.5M			\$2M	

AFTER

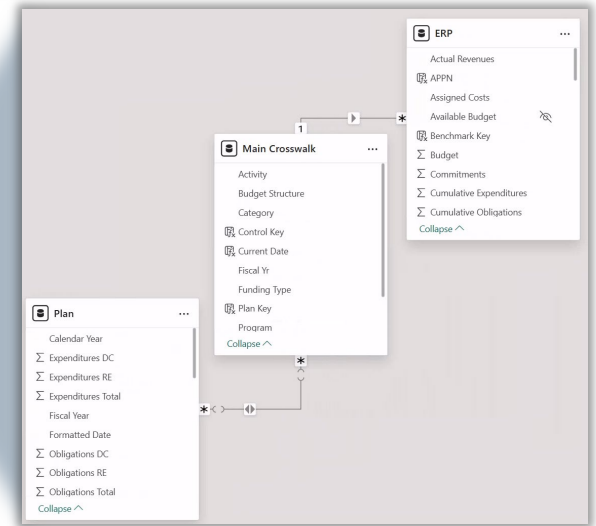
TPS Number	Month	Obs
TPS-20001	Oct	\$3M
TPS-20001	Nov	\$3M
TPS-20001	Dec	\$2M
TPS-20001	Jan	
TPS-20001	Feb	
TPS-20001	Mar	\$1.5M
TPS-20001	Apr	
TPS-20001	May	\$1.5M
TPS-20001	Jun	
TPS-20001	Jul	
TPS-20001	Aug	
TPS-20001	Sep	\$2M

Case Study: Develop (Load – Data Modeling)



ESTABLISHING RELATIONSHIPS

A crosswalk table that matches TPS Numbers to their corresponding Project Structure is used to relate the two sources



CREATE CALCULATED COLUMNS & MEASURES

Common Dax Functions

SUM – Adds up all values in a column.

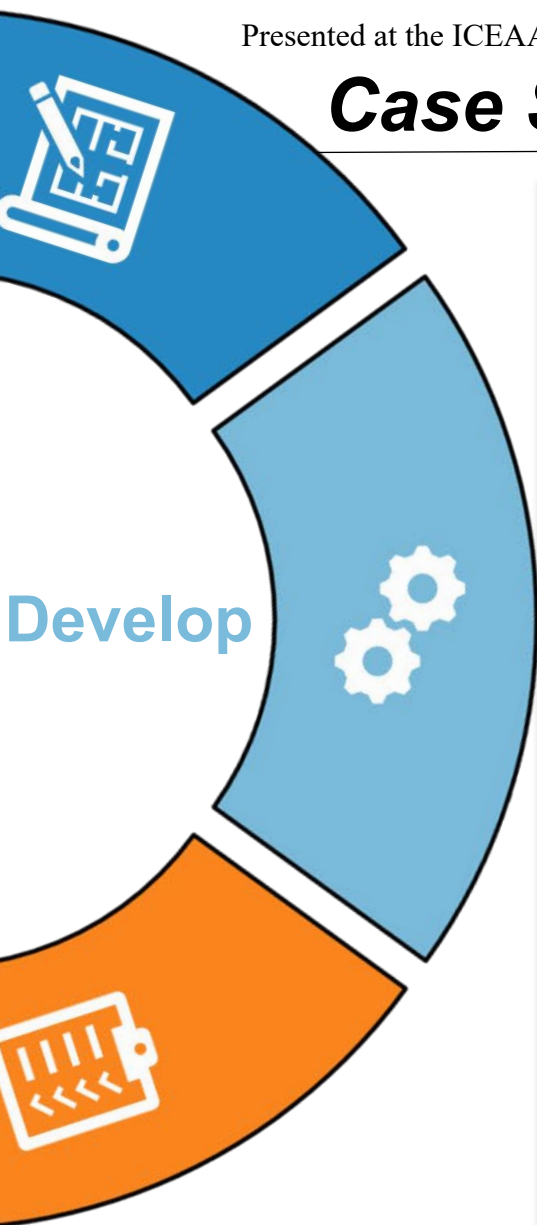
AVERAGE – Calculates the mean of values in a column

COUNT – Returns the number of values in a column

CALCULATE – Applies filters to modify calculations

```
Current Plan Obs/Exp = SWITCH([Selected Obs/Exp], "Obligations", [Current Plan Obligation], "Expenditures", [Current Plan Expenditure])
```

Case Study: Develop (Load - Visualizations)



Case Study: Test



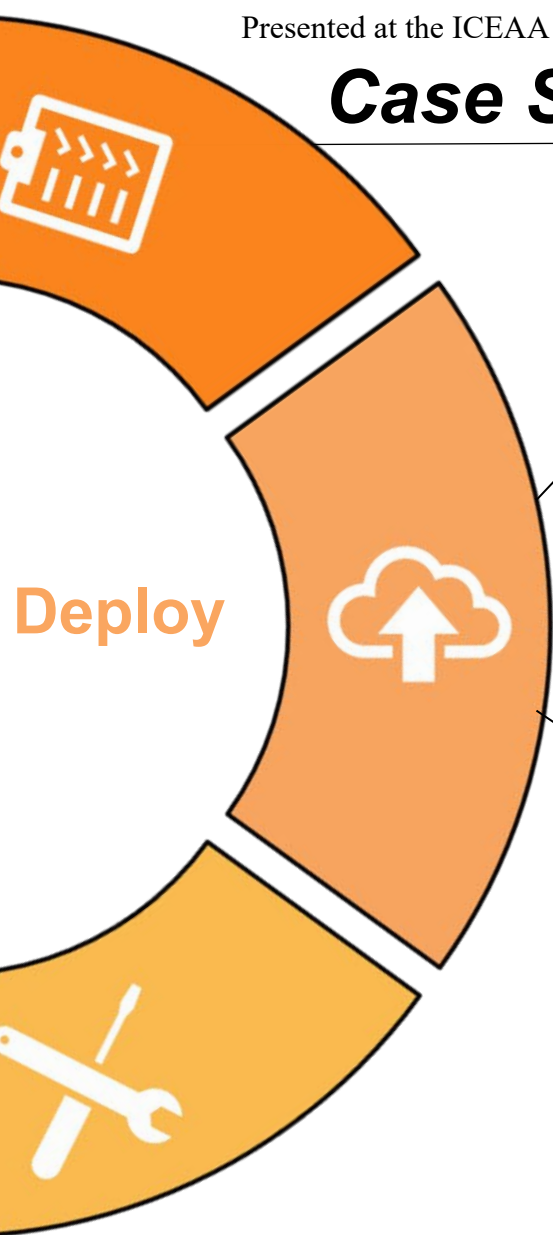
DATA VALIDATION

- *Perform Manual Crosschecks*
- *Confirm with Process Owners*
- *Troubleshoot errors*

PERFORMANCE ANALYSIS

- *Configure data connections*
- *Identify null/blank values*
- *Count Rows and Columns*
 - *Number of Distinct Values*

Case Study: Deploy



SHARE REPORTS



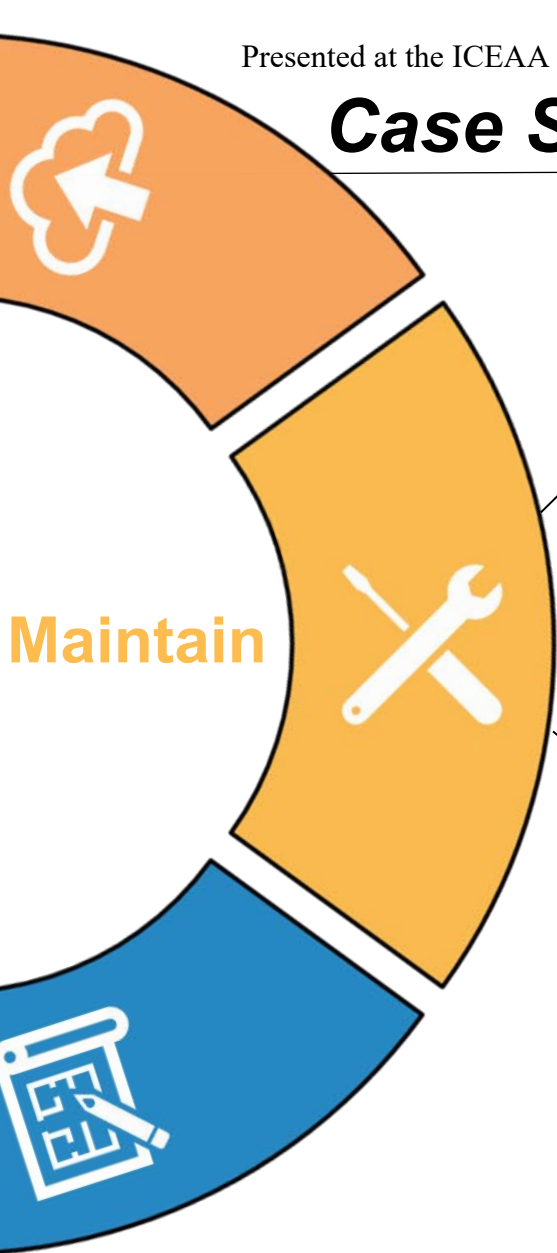
- *Host in Shared Environment*
 - *Teams*
- *Create Dashboards and Apps*

SECURITY



- *Row Level Security (RLS)*
- *Configure Permissions*
- *Control Accessibility*

Case Study: Maintain



DATA REFRESH



- *Set Scheduled Data Refreshes*
- *Automate File Management*
- *Consider Direct Connections*

NOTIFICATIONS



- *Set Alerts when KPIs met*
- *Paginated Reports*
- *Automate Notification System*
 - *Email, Teams Message, etc.*

Future Trends

Low-code platforms accelerate innovation, automation, and agility across all industries

SHORT TERM

LONG TERM



AI- Powered Development & Automation

**Advanced Integration with Legacy
Systems & APIs**

Low-Code for DevSecOps & Enterprise-Scale Applications

Hyperautomation & End-to-End Process Orchestration

**Industry-Specific Low-Code
Solutions**

**Democratization of Software
Development**

Conclusion and Parting Thoughts

Embrace low-code for a smarter, data-driven future!

1 Data Literacy is Essential

Accurate and timely data is the foundation of effective financial monitoring and decision-making.

2 Low-Code Enables Smart Decision-Making

By integrating various data sources, transforming insights, and enhancing accessibility, low-code platforms help organizations make informed decisions faster.

3 Breaking Down Silos

Standalone systems create inefficiencies; a unified low-code approach fosters collaboration and seamless data flow across authoritative sources.

4 The Power of Implementation

Applying systems engineering techniques ensures low-code solutions drive efficiency, accelerate development, and fuel innovation.
