Breaking A Monolith

Modularizing Large Legacy RPG Programs

Profound Logic

Your Presenter

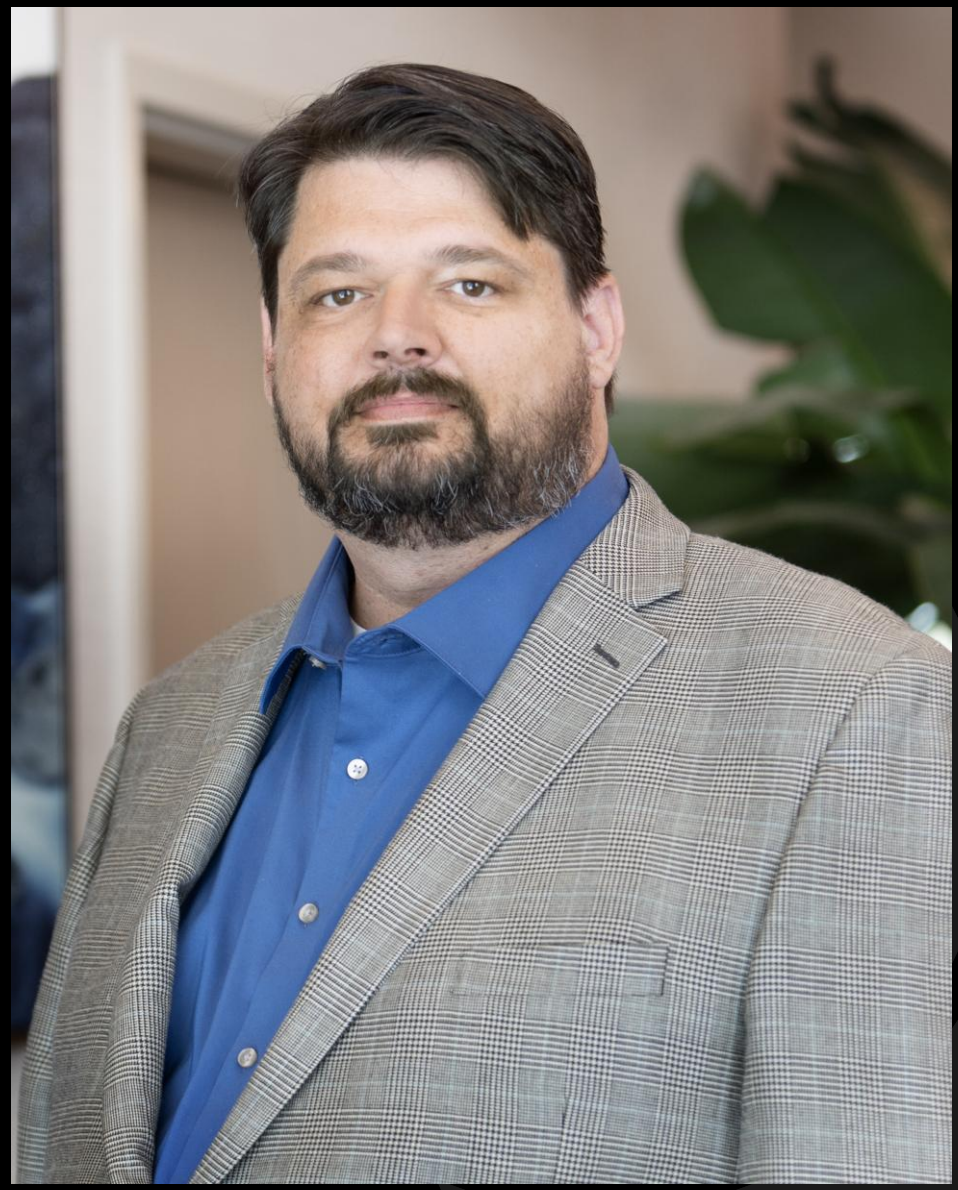
Brian May

Profound Logic, Principal Technology Evangelist

COMMON Board Member

CEF Board Member

Brian is the former product owner for Profound AI and Profound API, as well as an author, industry expert, and award-winning speaker.



Profound Logic

The Agenda

1

The Problem

Discuss Technical Debt and the Issues That Come With It

2

Clean Up Phase

Preparing Your Program For Modularization

3

Introducing Scope

Breaking the Application into Subprocedures and Dealing With Global Definitions

4

Reusing Functions

Externalizing Subprocedures to be Reused

5

Lessons Learned

Examine Results and Process to Improve Future Projects

Profound Logic

The Problem

Profound Logic

Technical Debt

Risk of Bug Introduction

- Large monoliths with only globally scoped files and variables are very risky to maintain
- Keeping track of the flow of the logic is difficult

No Code Reuse

- The same logic or business rule is often repeated in multiple programs
- Even multiple times in the same program

Older Code

- Most monoliths are very old code using outdated techniques and language features

Profound Logic

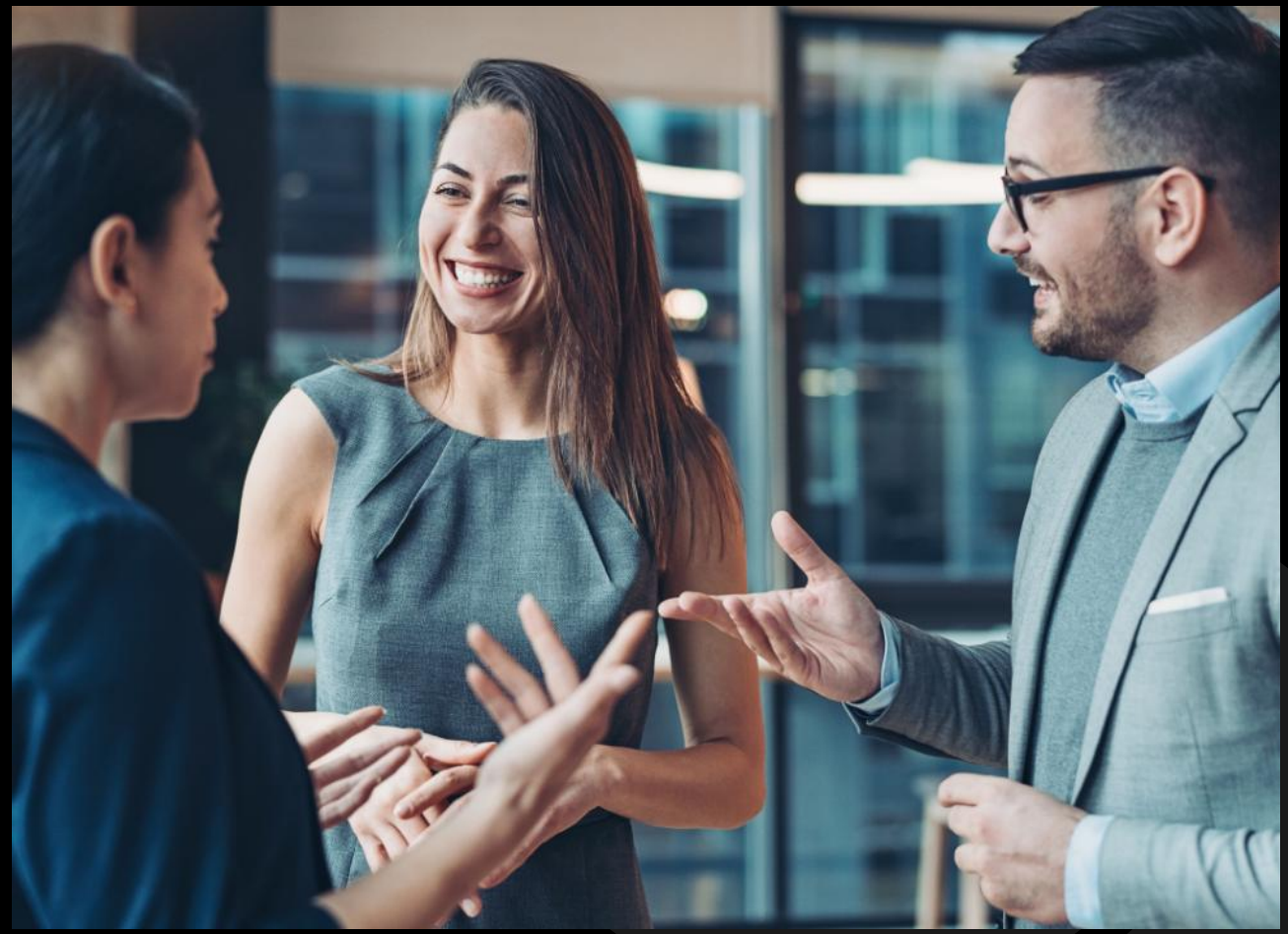
Lack of Understanding

Complexity

- The size and nature of monolithic programs make them extremely difficult to fully understand

Ownership

- Developers try to avoid changes to these programs. This leads to a lack of ownership or leadership to push for modularization and innovation



Profound Logic

Clean Up

Profound Logic

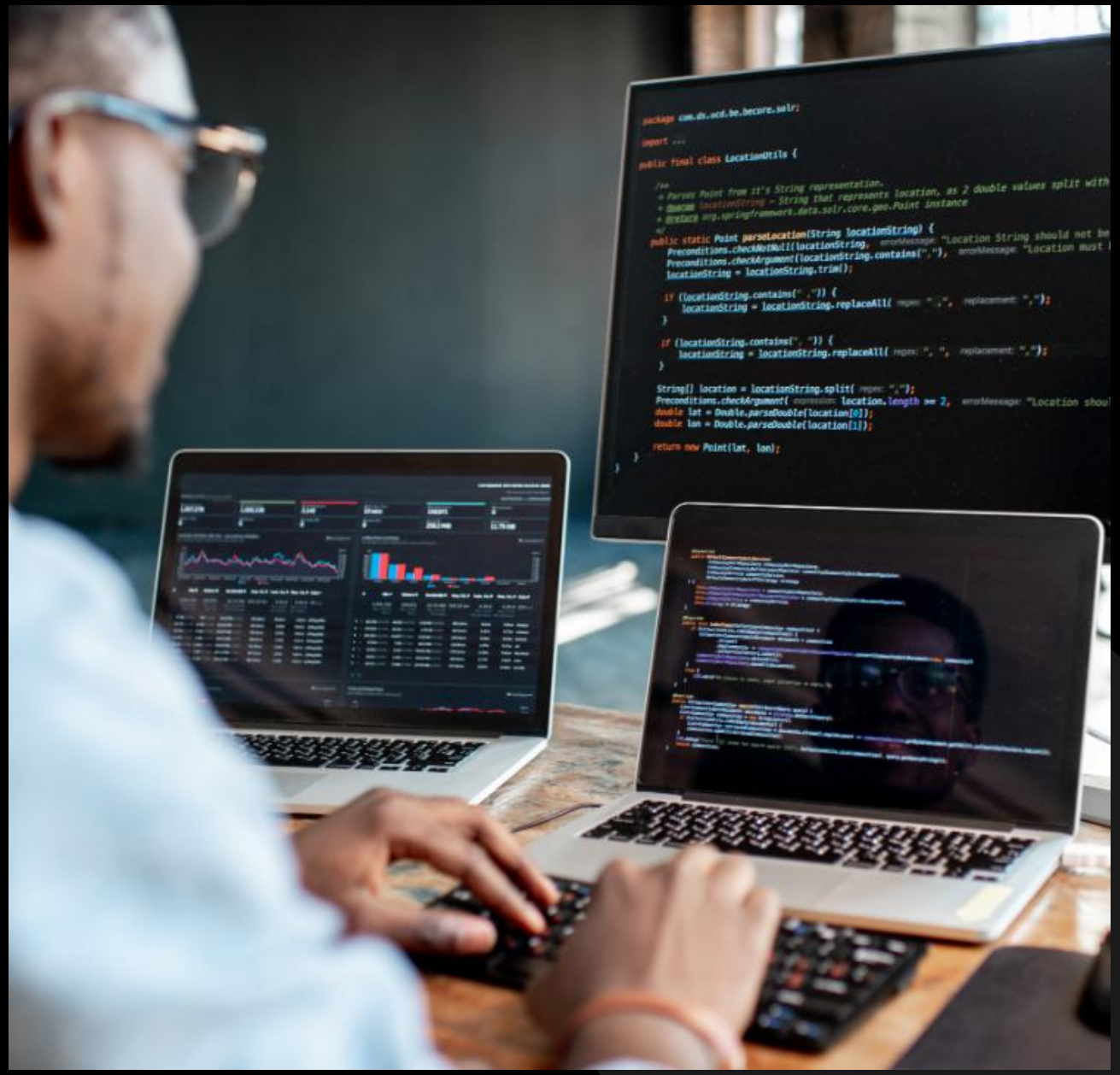
Where do I start?

Profound Logic

Move to RPGLE

It’s time to move forward

The first step to take is to move the program into the latest version of RPG. It’s not time for free form (yet), but to begin using ILE and modern features of the language, you can’t be running RPG III.



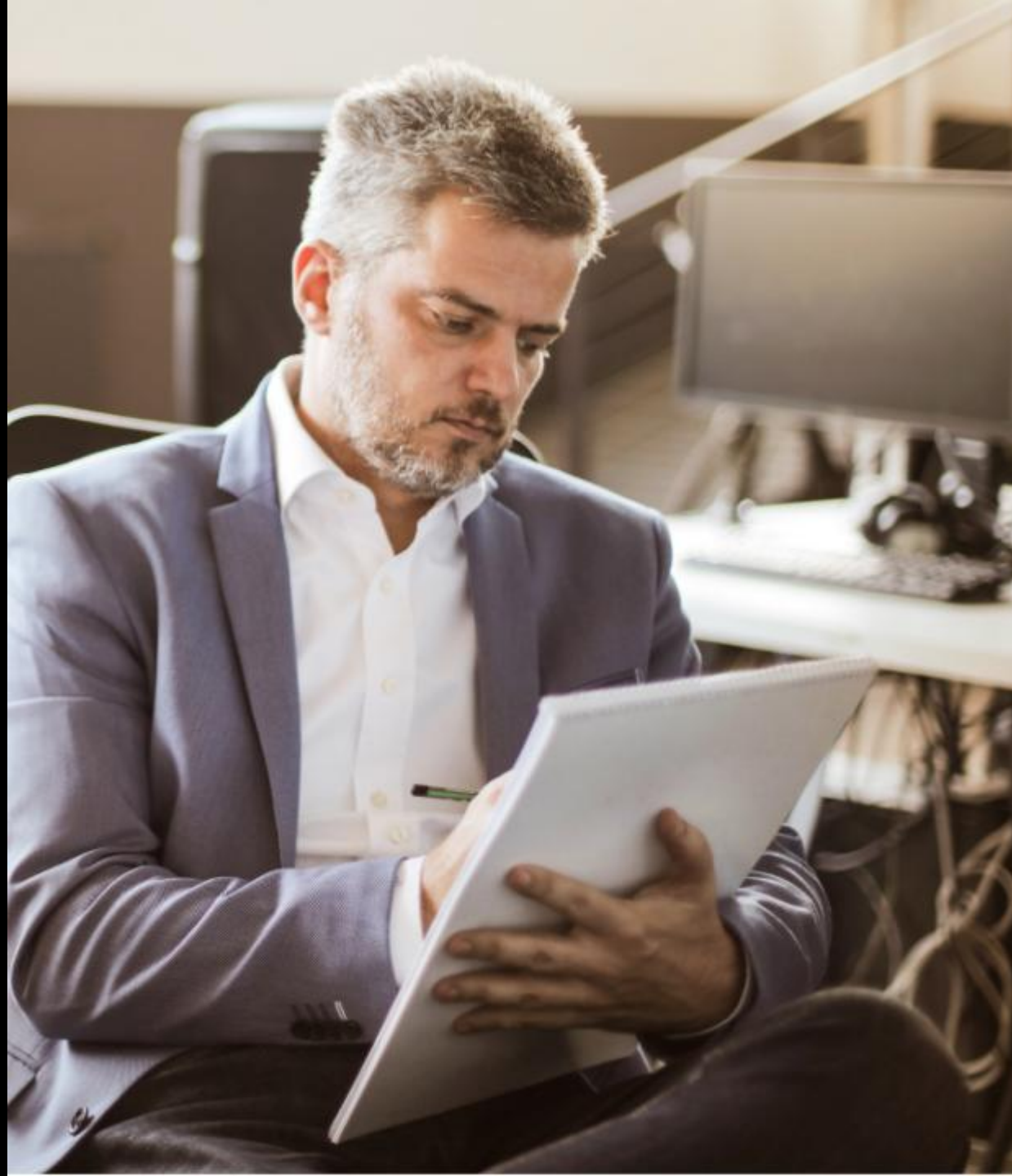
Profound Logic

The Logic Cycle

Replace with mainline logic

The next step in preparing and application to be modernized and modularized is to remove dependence on the RPG logic cycle.

Input primary file processing as well as just default looping are outdated techniques and will not be available in modern applications using linear main procedures.



Profound Logic

Structured Programming

GOTO

Eliminate all GOTOs in the program

In order to properly modernize and modularize a program, GOTO and TAG statements must be eliminated

CABxx

CABxx is just another GOTO

Get rid of compare and branch just like GOTO

Profound Logic

Input & Output

Internally Described Files

- All files accessed need to be externally described
- You will have subprocedures accessing those files and external definitions will be required
- No I specs or O specs should remain in your application

Profound Logic

Free Format

Free format calculations are a must. I recommend going fully free format. It will make your code easier to read and maintain for you and future developers.

No multi-step calculations

When moving to free format, old opcodes like ADD, Z-ADD, SUB, MULT, and DIV will need to be changed to EVAL.

You should also look at opportunities to consolidate lines into a clear calculation with all elements in one EVAL.

Profound Logic

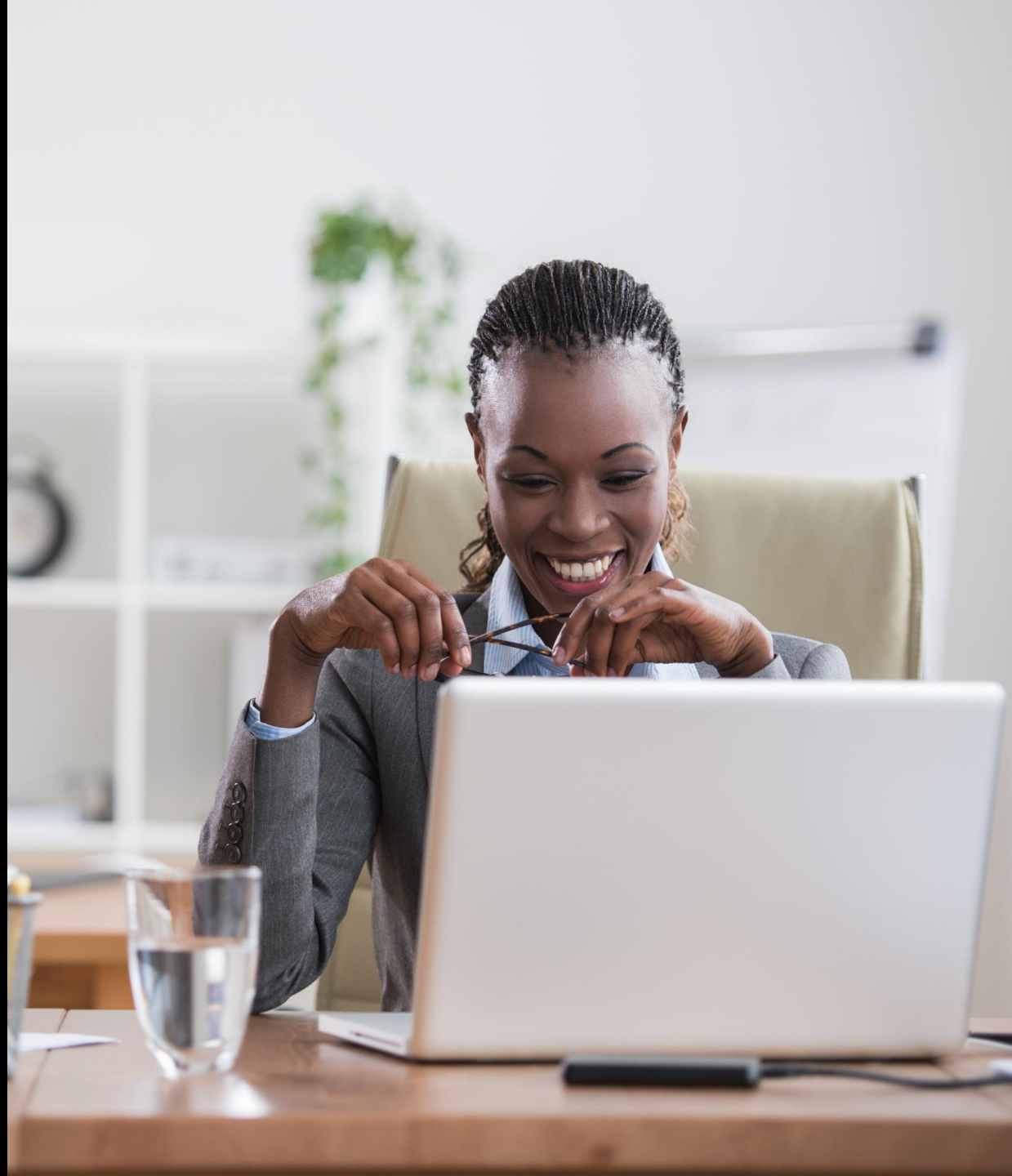
No Numeric Indicators

Use Built-In Functions

- %EOF()
- %FOUND()
- %EQUAL()

Indicator DS

- Use INDDS for display and printer files
- Meaningful names
- Unique set of 99 indicators for each file



Profound Logic

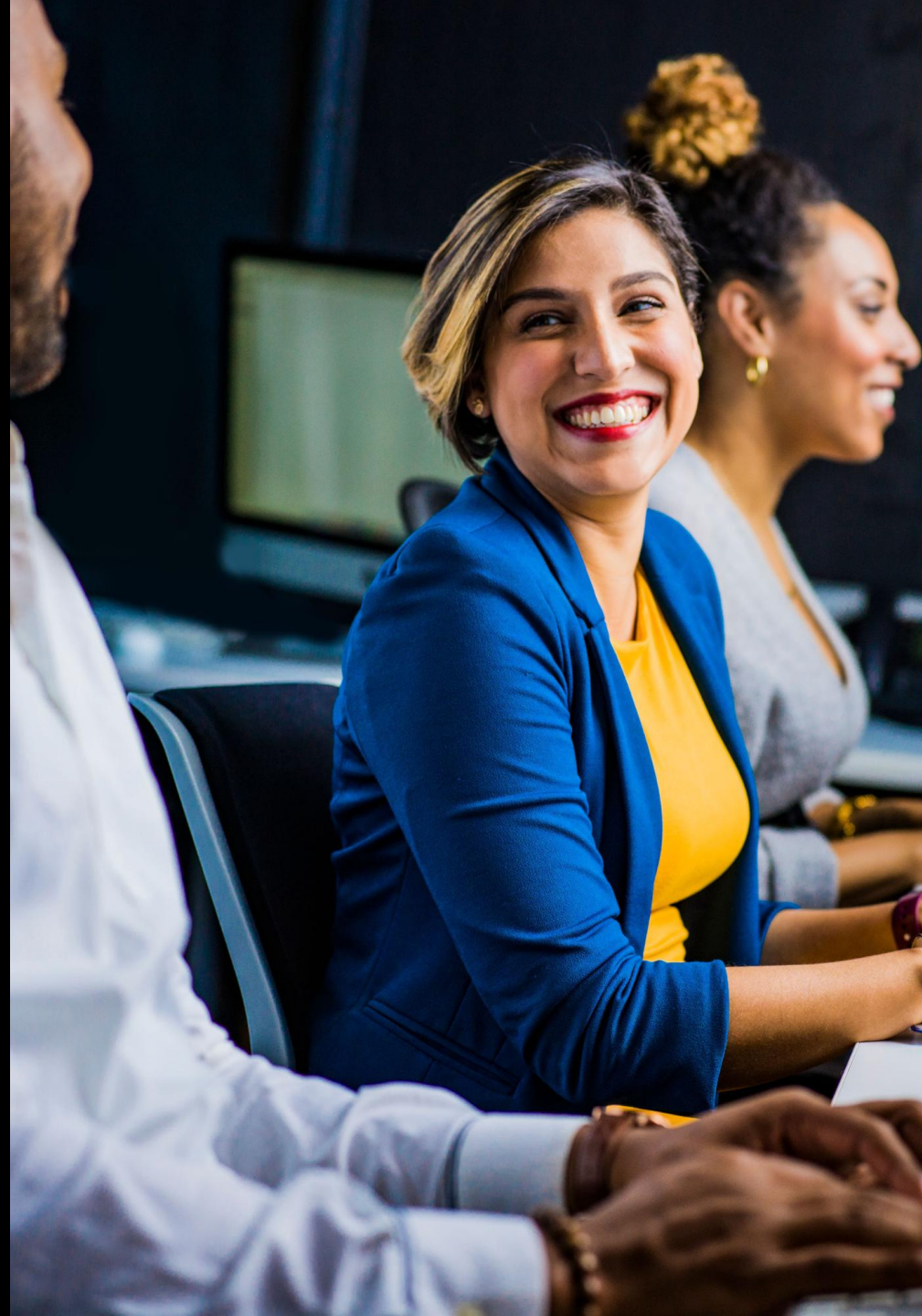
Introducing Scope

Profound Logic

Time to Begin Modularizing

Change Subroutines to Subprocedures

- This is a quick first step
- This change doesn’t add any particular risk or value in and of itself
- It is necessary to give each function its own local scope



Profound Logic

Make Subprocedures Independent

Create Parameter Interface

- Define a parameter interface that passes only necessary data
- Protect parameters with CONST and VALUE as appropriate

Eliminate Access to Global Variables

- Any variables used in the subprocedure should be defined locally or on the parameter interface

What About Files?

- Move to SQL
- Define local if only used here
- Files can be passed as parameters (LIKEFILE)

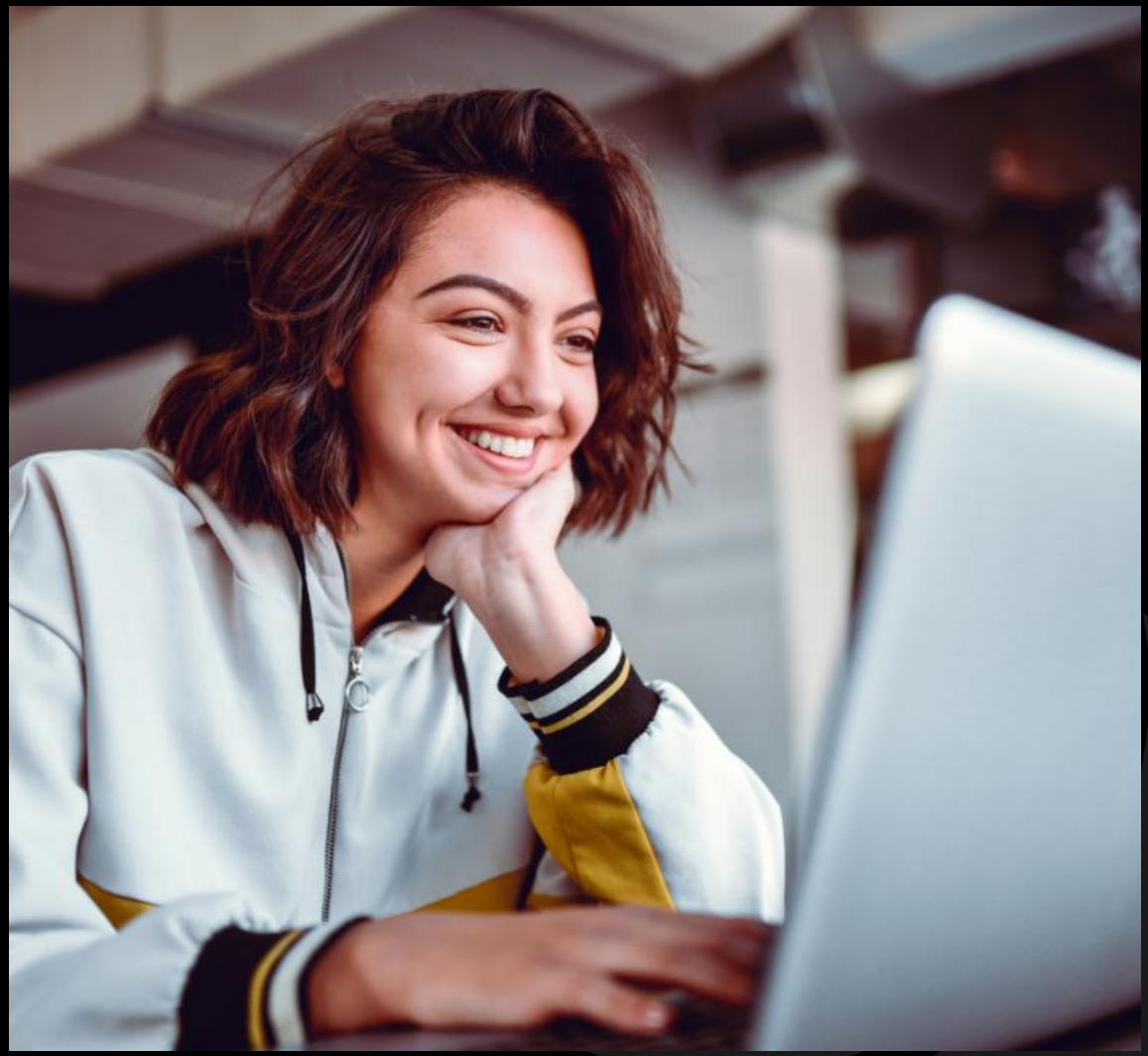
Profound Logic

Pro Tip

This is the perfect time to begin planning your test cases for each subprocedure

Plan tests that...

- Test expected input
- Test unexpected inputs
- Test repeated calls



Profound Logic

Removing Global Scope

Move to Linear Main

- Place main line logic into a new subprocedure
- Specify that subprocedure on the MAIN keyword

Remove Globally Scoped Definitions

- Move remaining globally scoped variables into main subprocedure
- If you are passing files as parameters, move your file specs into main subprocedure

TEST

- Fully test your program making sure every subprocedure is called
- Code coverage tools are great here

Profound Logic

Break it Down More

Make Subprocedures Bite-sized

- Rule of thumb is actual logic should fit on one screen
- Particularly complex sections should be isolated
- Anything that is duplicated or may be reused should be its own subprocedure

One Function per Subprocedure

- Each subprocedure should have one task
- New screen, new subprocedure (and maybe program)

Add Clarity

- Change variable names, procedure names, etc to be descriptive
- Add comments as needed

Profound Logic

Reusing Functions

Profound Logic

Breaking it Up

Decide What Subprocedures to Externalize

- Make a list of subprocedures
- Decide for each one if it's function will be useful outside of the original program

Organize Subprocedures

- Move subprocedures into service programs
- Put service programs in binding directory
- Use binding directory when compiling original program
- I prefer business object organization

Create Tests

- Write a program to implement test cases from previous phase
- Run test cases after changes going forward

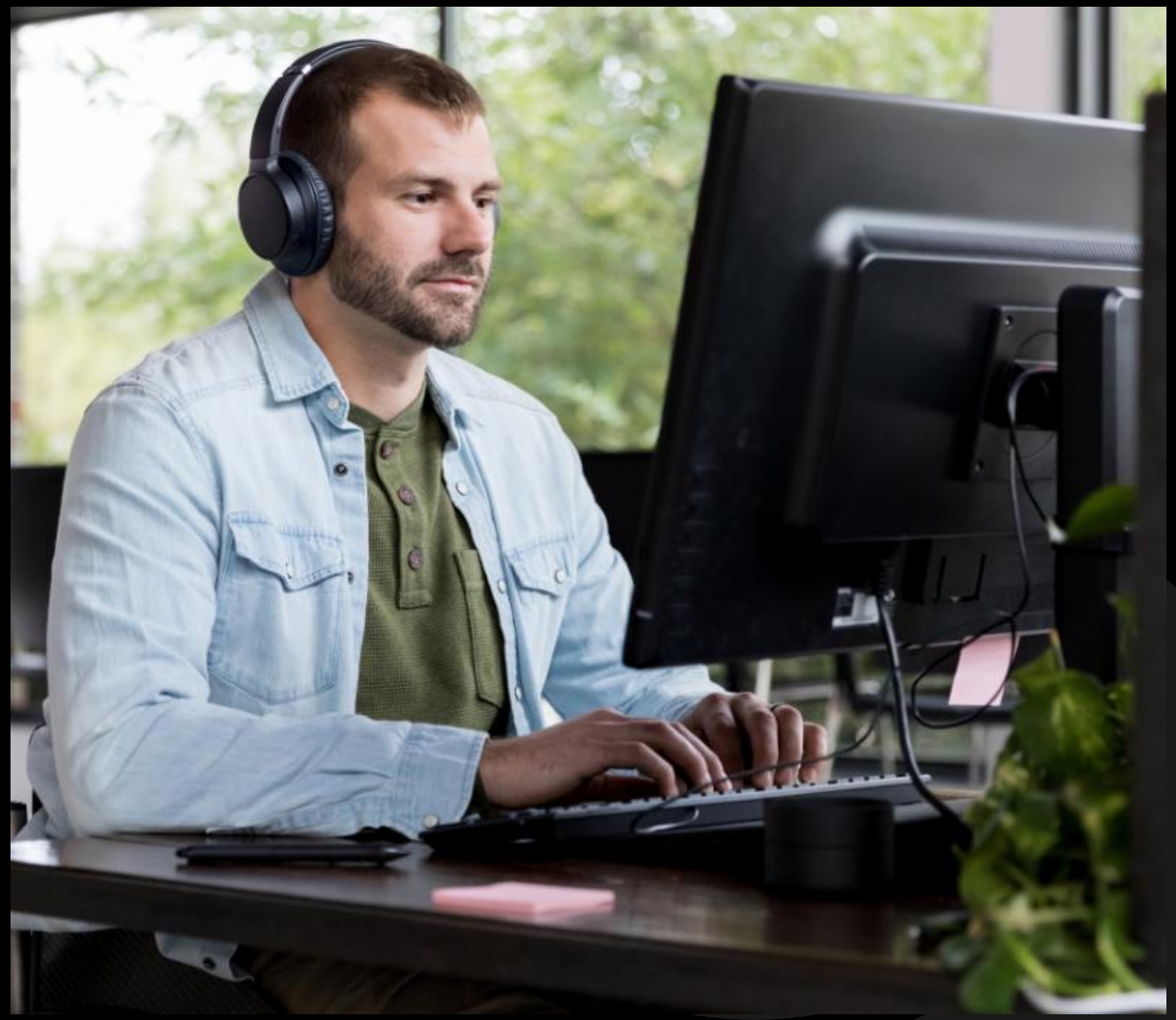
Use Your New Subprocedures

Original Programs

- Look for other programs where the same logic may be duplicated
- Remove duplicated code and replace with subprocedure call

New Use Cases

- APIs
- Stored procedures
- New applications



Profound Logic

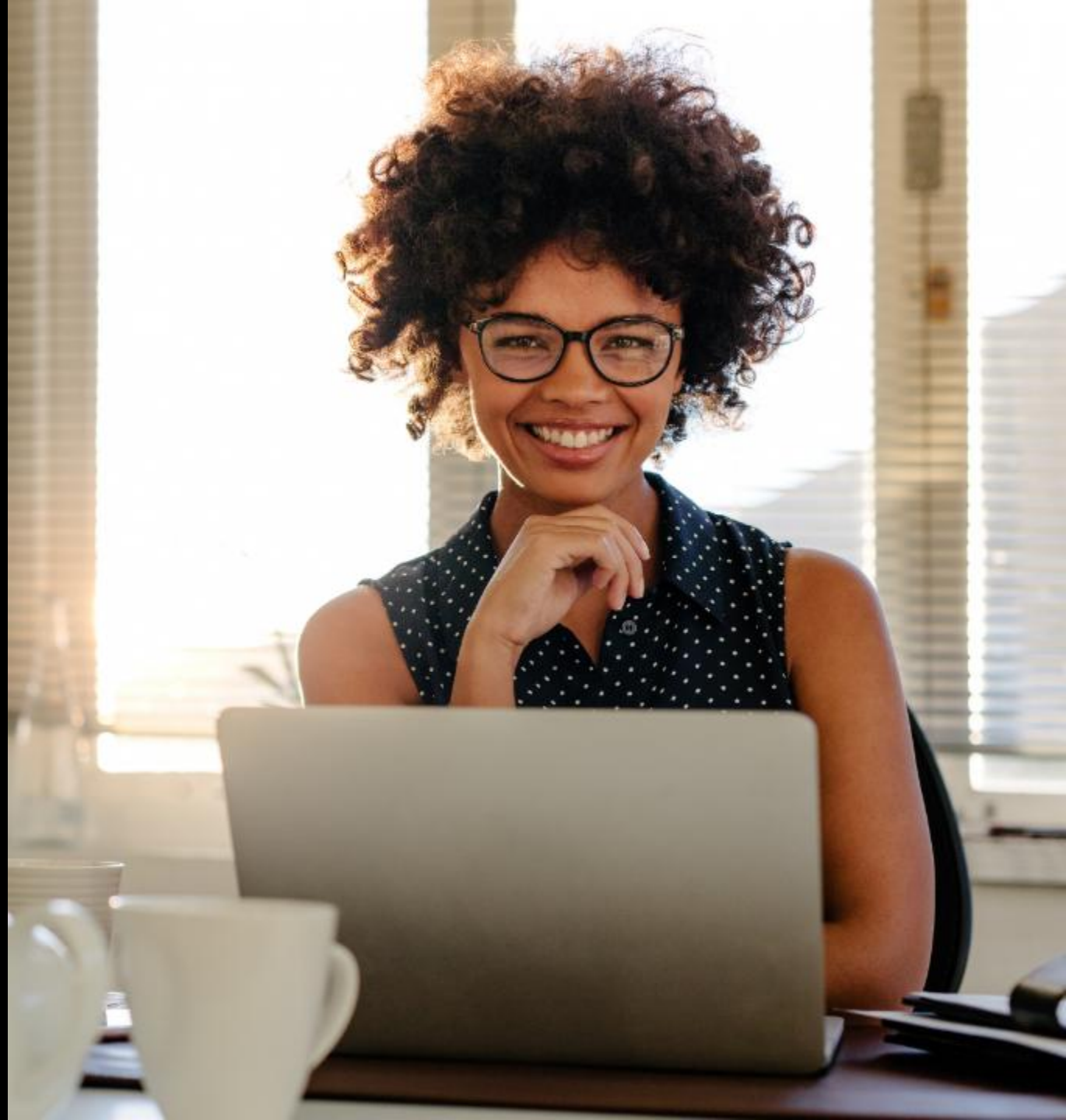
Lessons Learned

Profound Logic

Take Time to Reflect

It is Critical to Learn from Each Project

- What worked well/not so well?
- What skills did you learn?
- Can you streamline any phases?
- What new opportunities/projects did this enable?



Profound Logic

Questions?

Profound Logic

THANK YOU!

Profound Logic