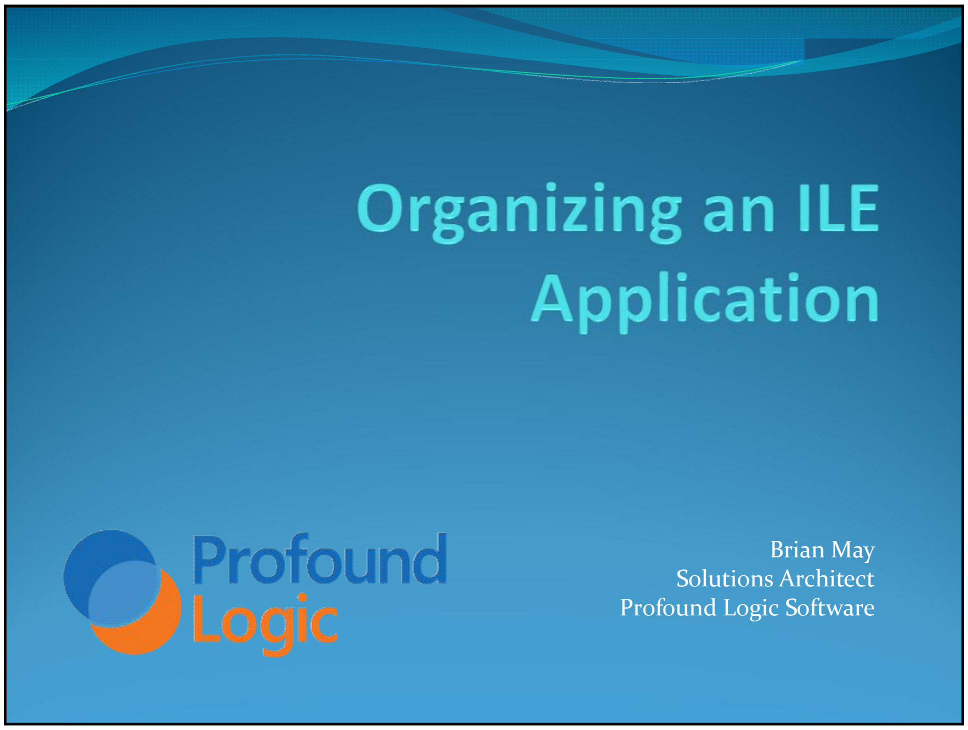
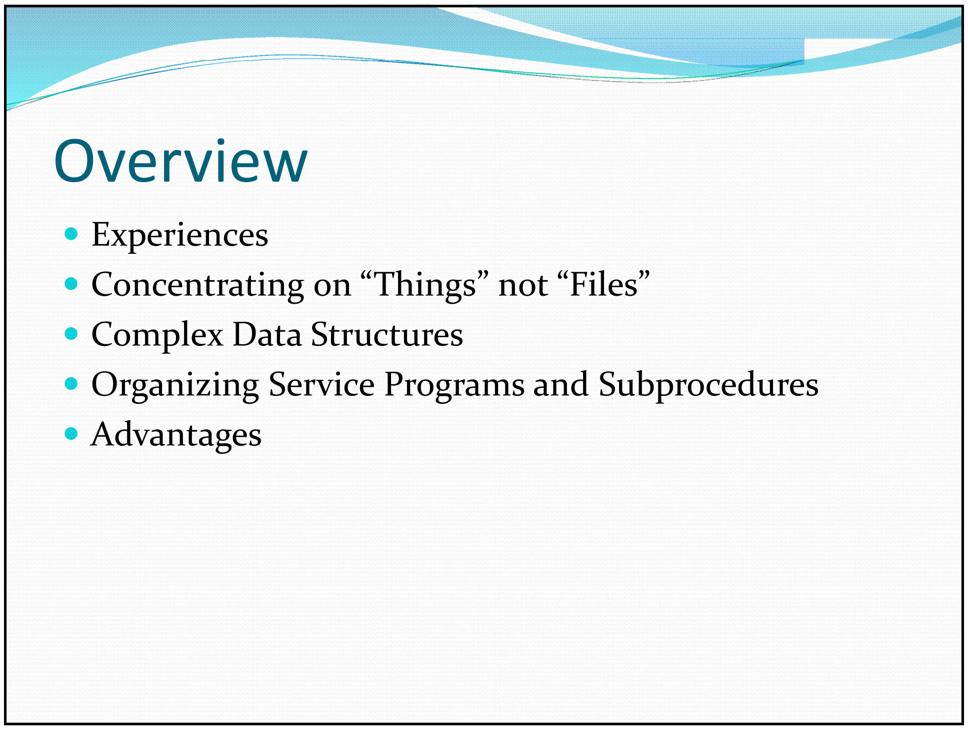


Organizing an ILE Application

Brian May
Solutions Architect
Profound Logic Software

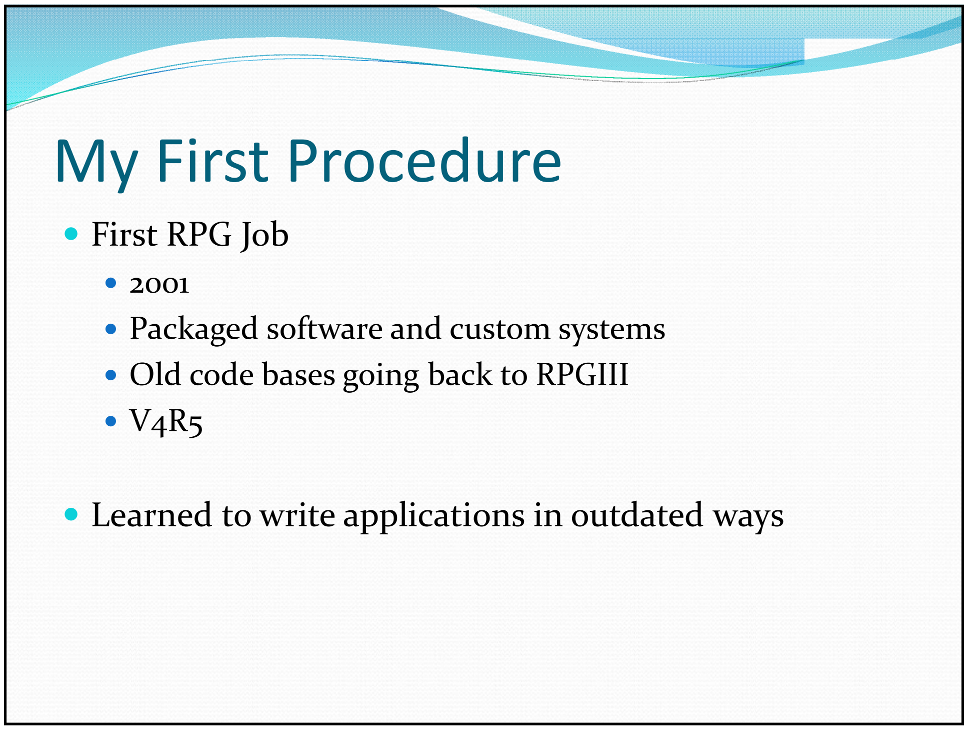


Overview

- Experiences
- Concentrating on “Things” not “Files”
- Complex Data Structures
- Organizing Service Programs and Subprocedures
- Advantages

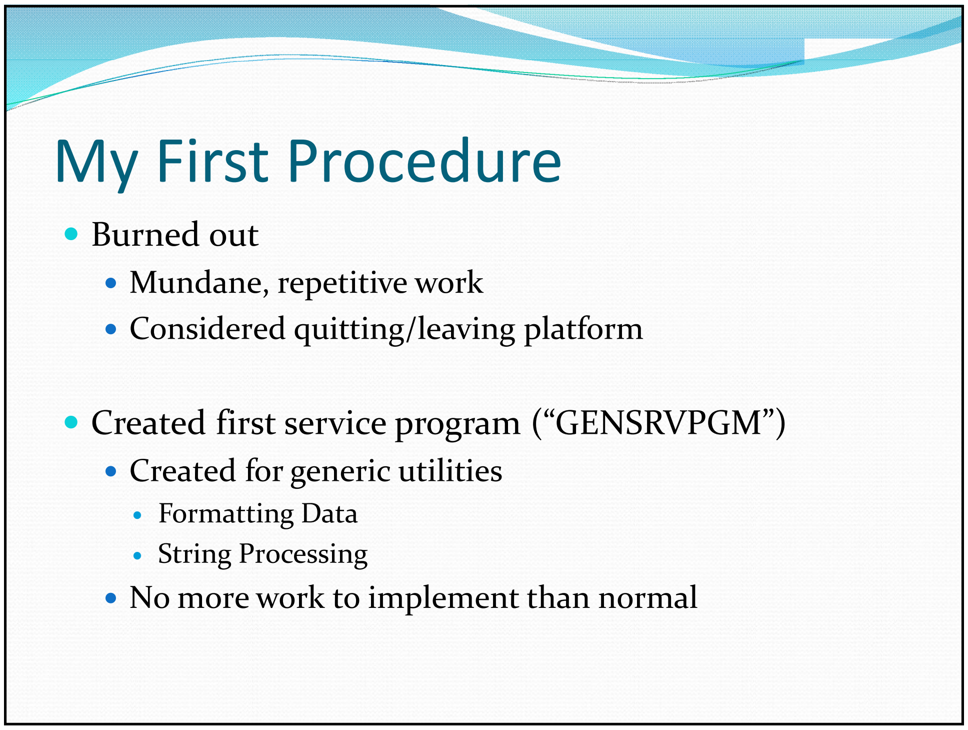
Lessons Learned

Learn from my mistakes



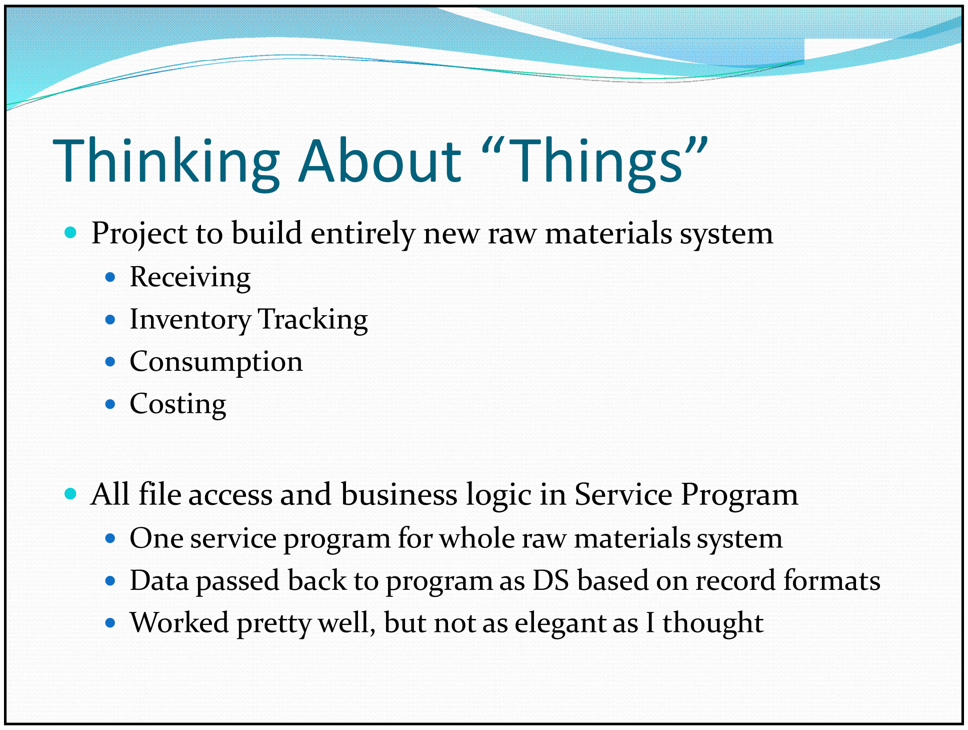
My First Procedure

- First RPG Job
  - 2001
  - Packaged software and custom systems
  - Old code bases going back to RPGIII
  - V4R5
- Learned to write applications in outdated ways



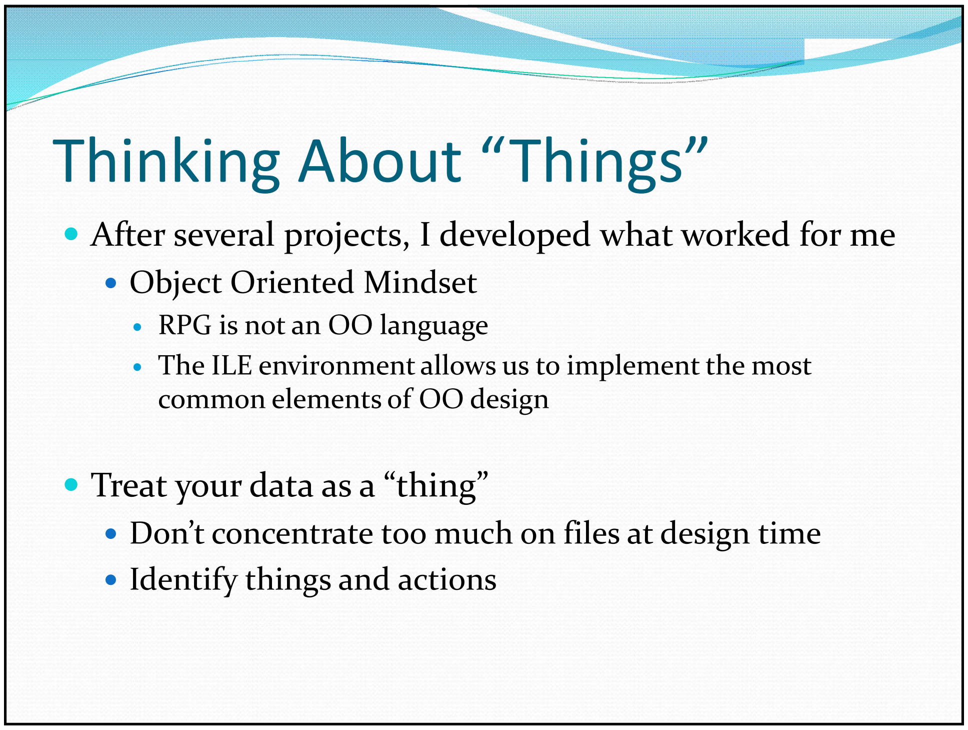
My First Procedure

- Burned out
  - Mundane, repetitive work
  - Considered quitting/leaving platform
- Created first service program (“GENSRVPGM”)
  - Created for generic utilities
    - Formatting Data
    - String Processing
  - No more work to implement than normal



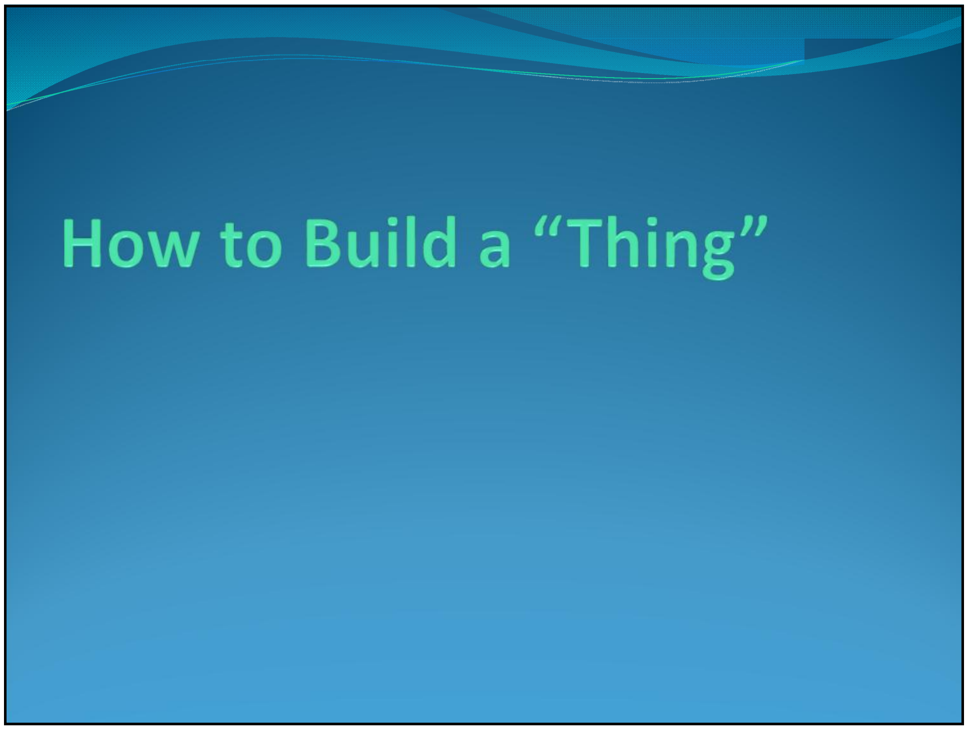
Thinking About “Things”

- Project to build entirely new raw materials system
  - Receiving
  - Inventory Tracking
  - Consumption
  - Costing
- All file access and business logic in Service Program
  - One service program for whole raw materials system
  - Data passed back to program as DS based on record formats
  - Worked pretty well, but not as elegant as I thought



Thinking About "Things"

- After several projects, I developed what worked for me
  - Object Oriented Mindset
    - RPG is not an OO language
    - The ILE environment allows us to implement the most common elements of OO design
- Treat your data as a "thing"
  - Don't concentrate too much on files at design time
  - Identify things and actions



How to Build a "Thing"

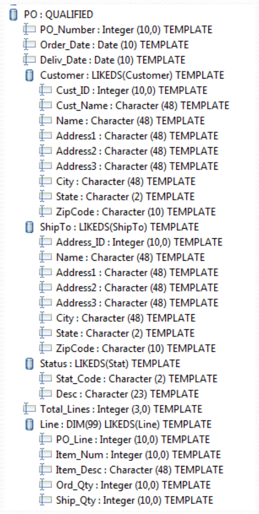
How to Build a "Thing"

- Example: Order Entry System
  - Thing: Purchase Order
- Create a Service Program
  - 1 Thing = 1 Service Program
    - Helps with organization
    - Prevents accessing "global" data as a shortcut
  - Use a binding directory to simplify compiles
  - Use a copybook for definitions and prototypes

How to Build a "Thing"

- Define what data your thing has
  - Purchase Order
    - Key info
    - Customer info
    - Order Lines
- Create a complex data structure to represent your "Thing"
  - Don't try to include everything!
    - Just include the most commonly used info
    - Create procedures to work with less used info separately

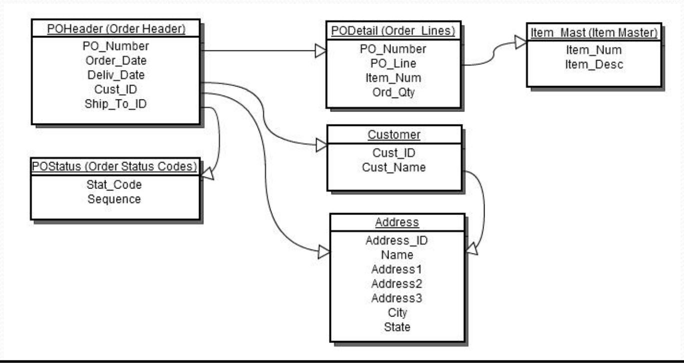
How to Build a “Thing”



```
PO : QUALIFIED
  PO_Number : Integer (10,0) TEMPLATE
  Order_Date : Date (10) TEMPLATE
  Deliv_Date : Date (10) TEMPLATE
  Customer : LIKEDS(Customer) TEMPLATE
    Cust_ID : Integer (10,0) TEMPLATE
    Cust_Name : Character (48) TEMPLATE
    Name : Character (48) TEMPLATE
    Address1 : Character (48) TEMPLATE
    Address2 : Character (48) TEMPLATE
    Address3 : Character (48) TEMPLATE
    City : Character (48) TEMPLATE
    State : Character (2) TEMPLATE
    ZipCode : Character (10) TEMPLATE
  ShipTo : LIKEDS(ShipTo) TEMPLATE
    Address_ID : Integer (10,0) TEMPLATE
    Name : Character (48) TEMPLATE
    Address1 : Character (48) TEMPLATE
    Address2 : Character (48) TEMPLATE
    Address3 : Character (48) TEMPLATE
    City : Character (48) TEMPLATE
    State : Character (2) TEMPLATE
    ZipCode : Character (10) TEMPLATE
  Status : LIKEDS(Stat) TEMPLATE
    Stat_Code : Character (2) TEMPLATE
    Desc : Character (23) TEMPLATE
  Total_Lines : Integer (3,0) TEMPLATE
  Line : DIM(99) LIKEDS(Line) TEMPLATE
    PO_Line : Integer (10,0) TEMPLATE
    Item_Num : Integer (10,0) TEMPLATE
    Item_Desc : Character (48) TEMPLATE
    Ord_Qty : Integer (10,0) TEMPLATE
    Ship_Qty : Integer (10,0) TEMPLATE
```

How to Build a “Thing”

- File structure
  - New or Existing
  - Map it out



How to Build a "Thing"

- Create subprocedures for your actions
  - Create procedures to build your "thing" and to save it
  - Pass your "Thing" Data Structure as a parameter to any procedures with business logic
    - Think of the Data Structure as your "instance" of the "thing"
  - If procedure will only use/modify one of the underlying data structures, it is ok to just pass it

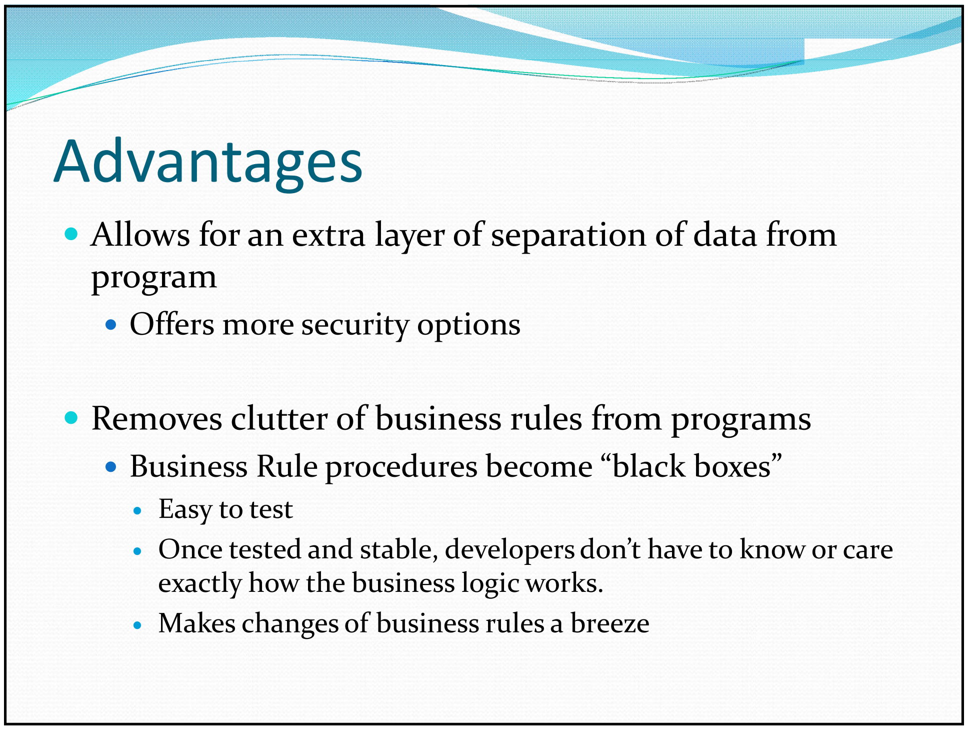
How to Use a "Thing"

- Use "action" subprocedures
  - DO NOT access the database files directly
    - Defeats the purpose
  - Program will actually have no file specs for tables
- Use same names in Displays / Print Files
  - Use data structures for all File I/O
  - Allows EVAL-CORR to reduce code
- If you call another program that needs the "Thing", pass it as a parameter. Don't build the object again in called program.

Advantages

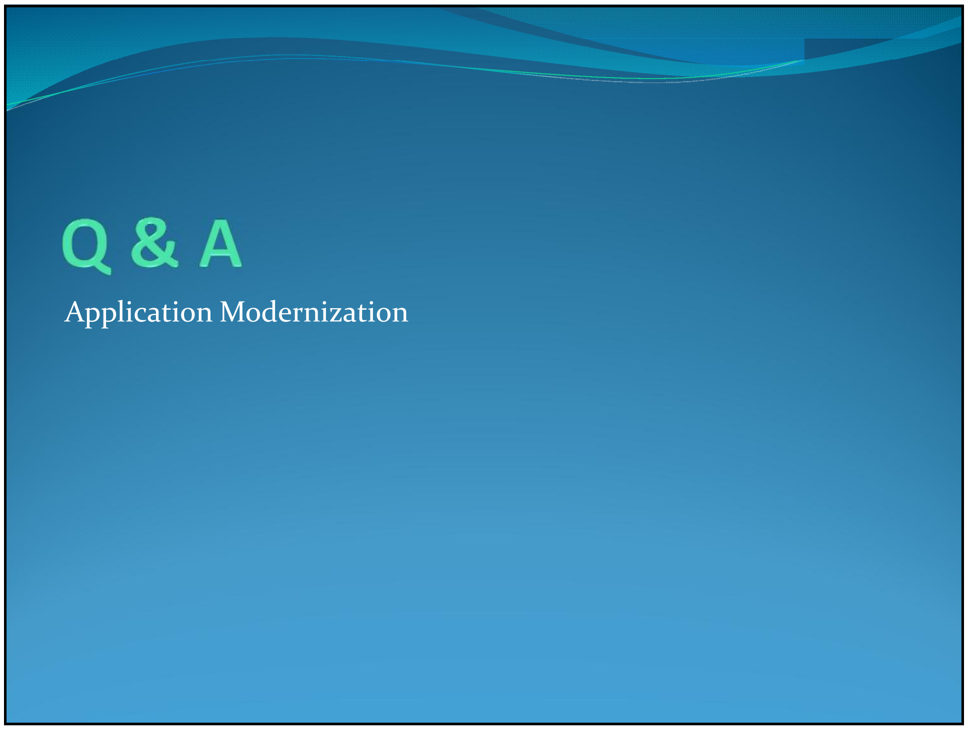
Advantages

- Using this “object” approach will:
  - Reduce File I/O
    - No need to retrieve the same data in every program or procedure
    - File I/O is one of the slowest parts of an application
  - Reduce Code
    - Removing file access from individual programs and centralizing it greatly reduces the amount of code
    - Much more digestible



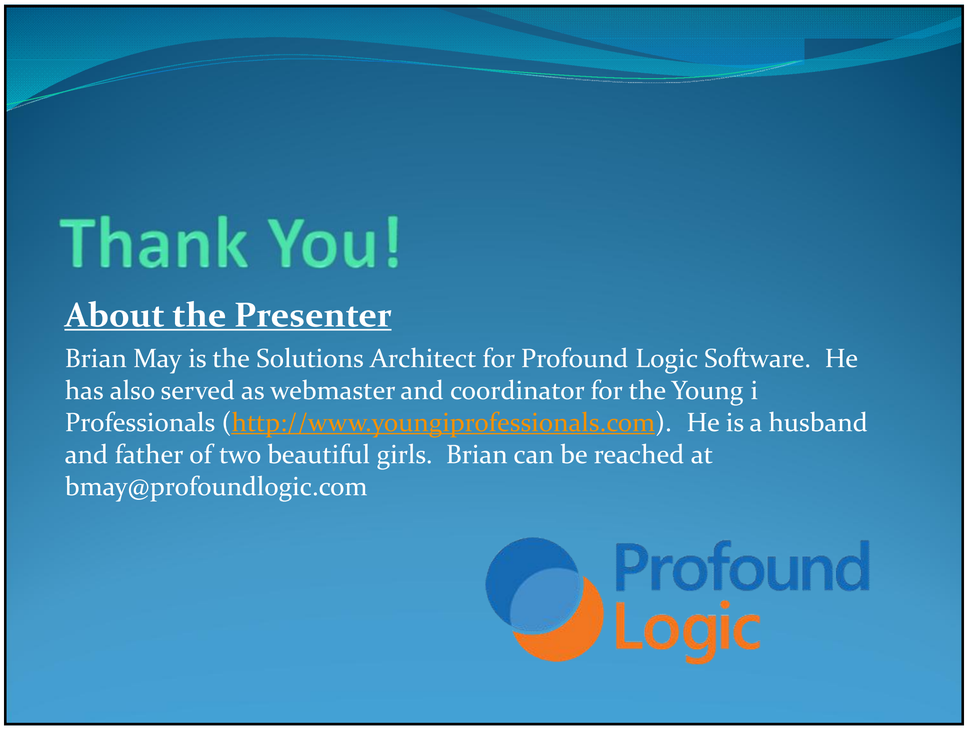
Advantages

- Allows for an extra layer of separation of data from program
  - Offers more security options
- Removes clutter of business rules from programs
  - Business Rule procedures become “black boxes”
    - Easy to test
    - Once tested and stable, developers don’t have to know or care exactly how the business logic works.
    - Makes changes of business rules a breeze



Q & A

Application Modernization



Thank You!

About the Presenter

Brian May is the Solutions Architect for Profound Logic Software. He has also served as webmaster and coordinator for the Young i Professionals (http://www.youngiprofessionals.com). He is a husband and father of two beautiful girls. Brian can be reached at bmay@profoundlogic.com

Profound Logic