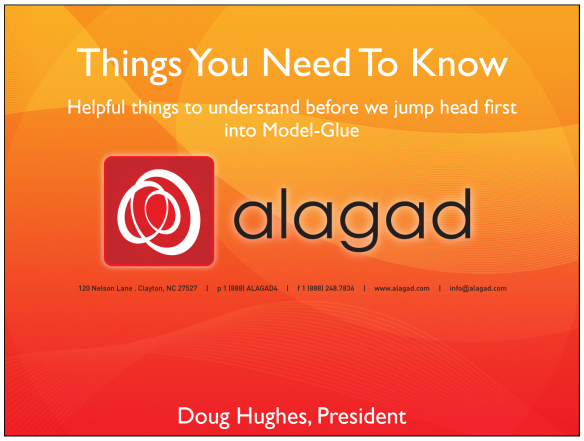


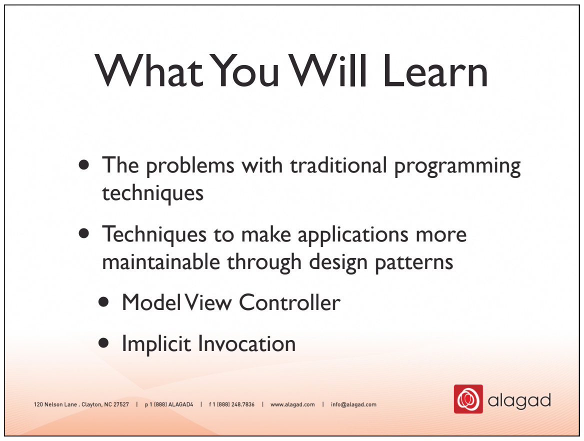
Things You Need To Know

Helpful things to understand before we jump head first into Model-Glue

alagad

120 Nelson Lane . Clayton, NC 27527 | p 1 (888) ALAGAD4 | f 1 (888) 248.7836 | www.alagad.com | info@alagad.com

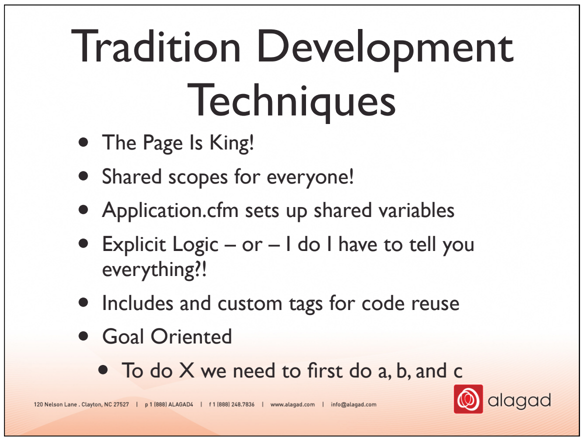
Doug Hughes, President



What You Will Learn

- The problems with traditional programming techniques
- Techniques to make applications more maintainable through design patterns
  - Model View Controller
  - Implicit Invocation

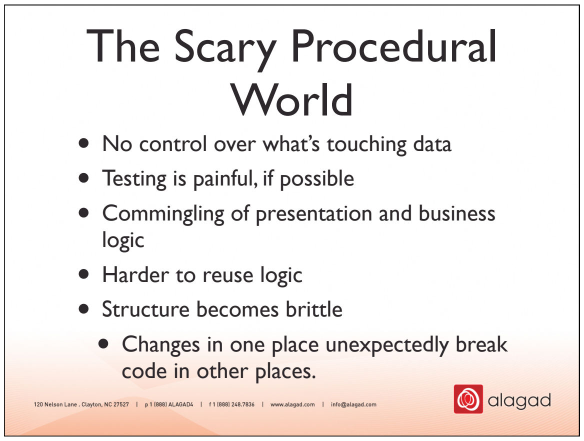
alagad



Tradition Development Techniques

- The Page Is King!
- Shared scopes for everyone!
- Application.cfm sets up shared variables
- Explicit Logic – or – I do I have to tell you everything?!
- Includes and custom tags for code reuse
- Goal Oriented
  - To do X we need to first do a, b, and c

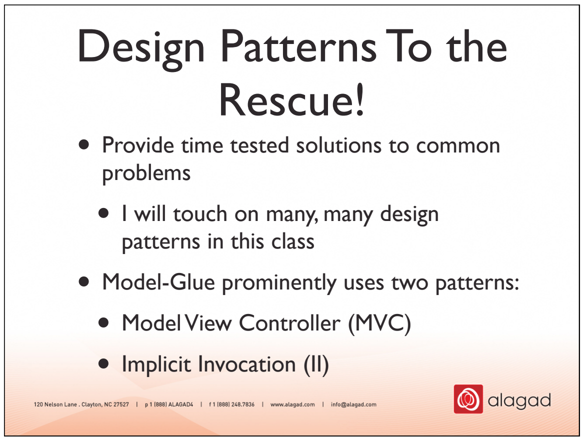
alagad



The Scary Procedural World

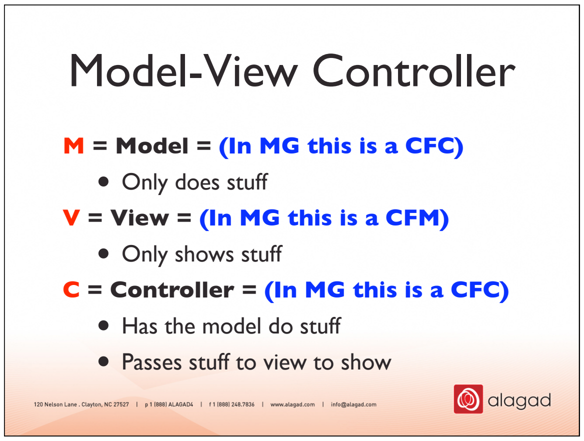
- No control over what's touching data
- Testing is painful, if possible
- Commingling of presentation and business logic
- Harder to reuse logic
- Structure becomes brittle
  - Changes in one place unexpectedly break code in other places.

alagad



Design Patterns To the Rescue!

- Provide time tested solutions to common problems
  - I will touch on many, many design patterns in this class
- Model-Glue prominently uses two patterns:
  - Model View Controller (MVC)
  - Implicit Invocation (II)



Model-View Controller

M = Model = (In MG this is a CFC)

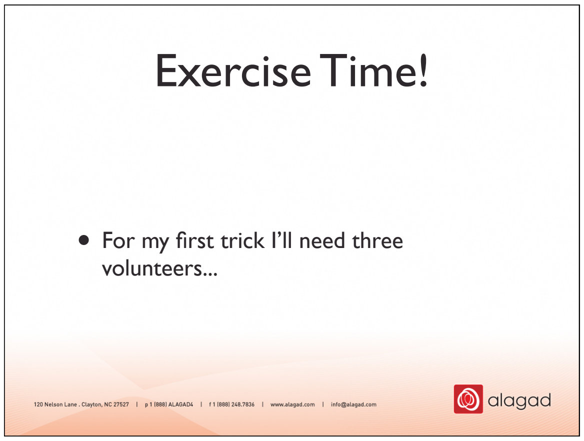
- Only does stuff

V = View = (In MG this is a CFM)

- Only shows stuff

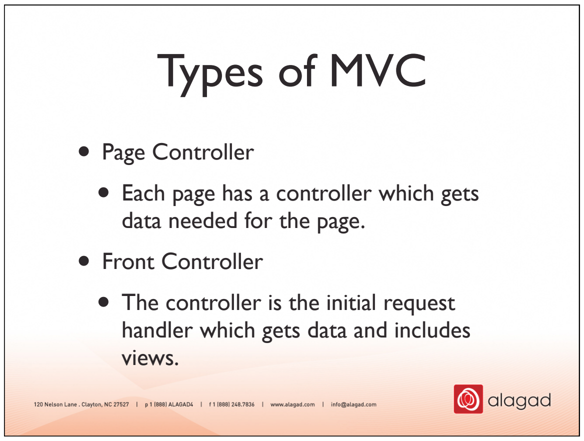
C = Controller = (In MG this is a CFC)

- Has the model do stuff
- Passes stuff to view to show



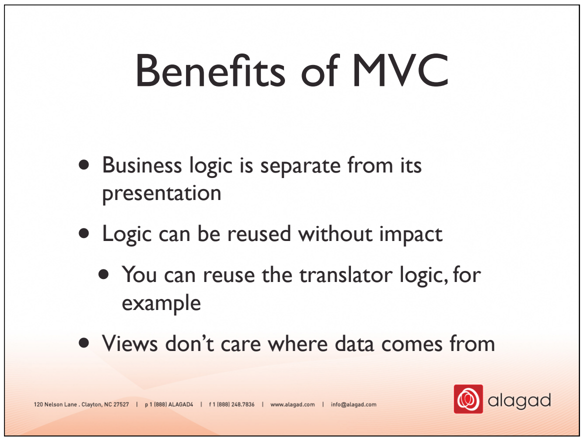
Exercise Time!

- For my first trick I'll need three volunteers...



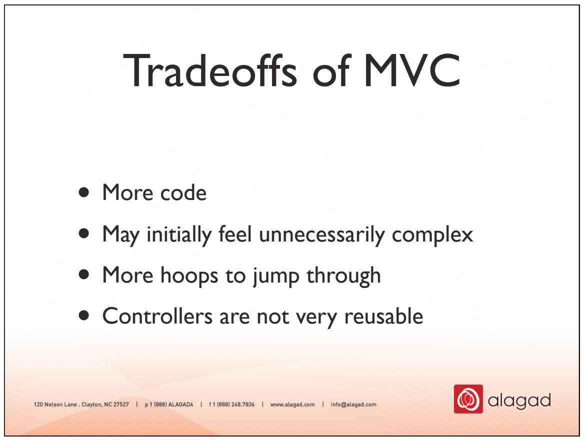
Types of MVC

- Page Controller
  - Each page has a controller which gets data needed for the page.
- Front Controller
  - The controller is the initial request handler which gets data and includes views.



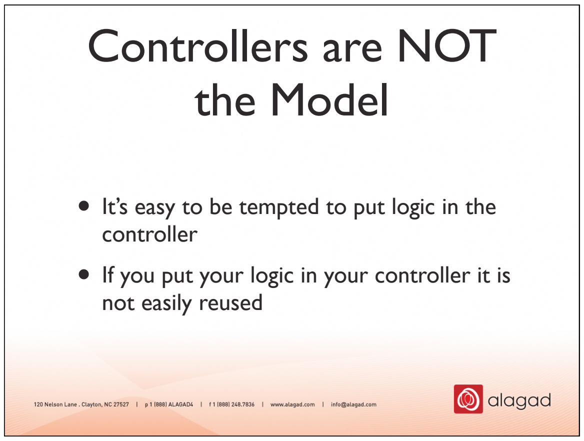
Benefits of MVC

- Business logic is separate from its presentation
- Logic can be reused without impact
  - You can reuse the translator logic, for example
- Views don't care where data comes from



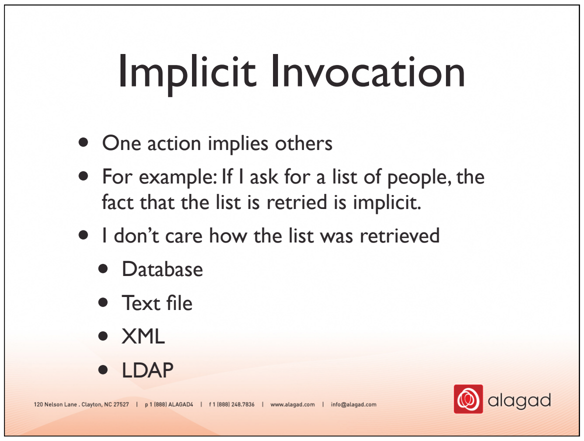
Tradeoffs of MVC

- More code
- May initially feel unnecessarily complex
- More hoops to jump through
- Controllers are not very reusable



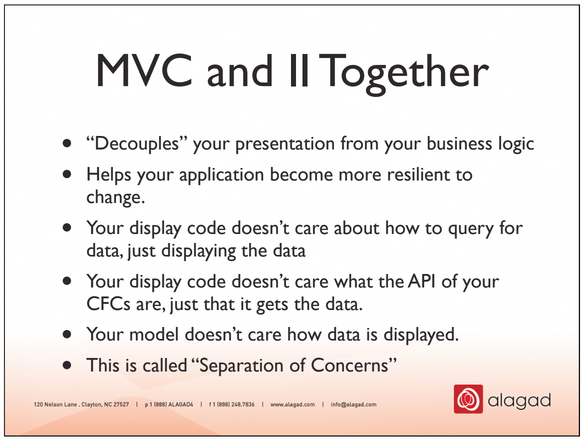
Controllers are NOT the Model

- It's easy to be tempted to put logic in the controller
- If you put your logic in your controller it is not easily reused



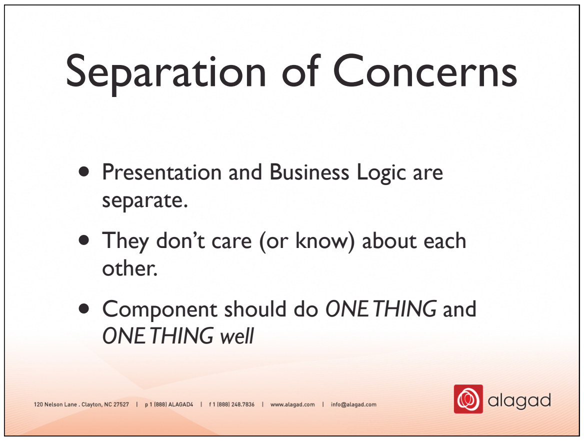
Implicit Invocation

- One action implies others
- For example: If I ask for a list of people, the fact that the list is retried is implicit.
- I don't care how the list was retrieved
  - Database
  - Text file
  - XML
  - LDAP



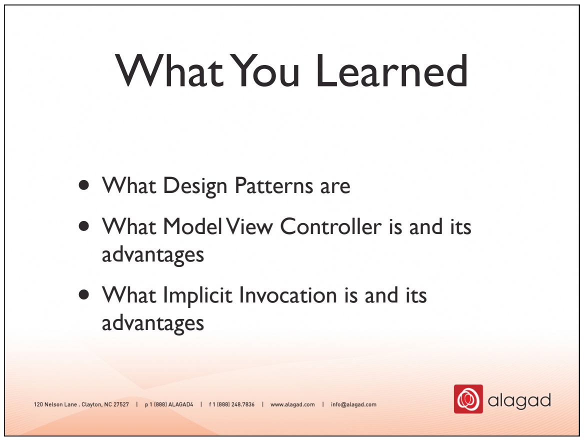
MVC and II Together

- “Decouples” your presentation from your business logic
- Helps your application become more resilient to change.
- Your display code doesn’t care about how to query for data, just displaying the data
- Your display code doesn’t care what the API of your CFCs are, just that it gets the data.
- Your model doesn’t care how data is displayed.
- This is called “Separation of Concerns”



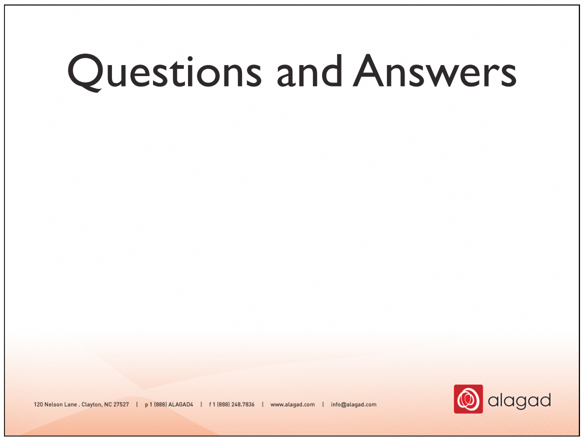
Separation of Concerns

- Presentation and Business Logic are separate.
- They don’t care (or know) about each other.
- Component should do *ONE THING* and *ONE THING well*

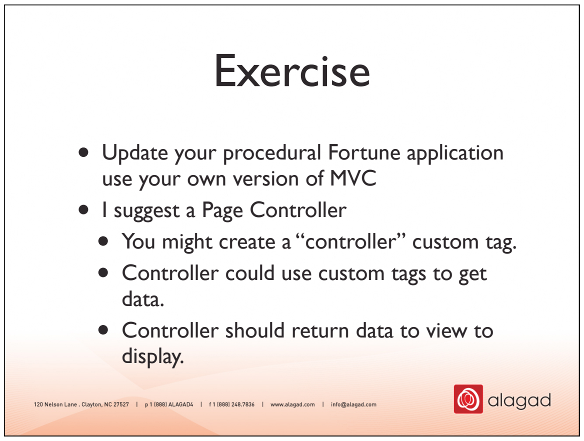


What You Learned

- What Design Patterns are
- What Model View Controller is and its advantages
- What Implicit Invocation is and its advantages

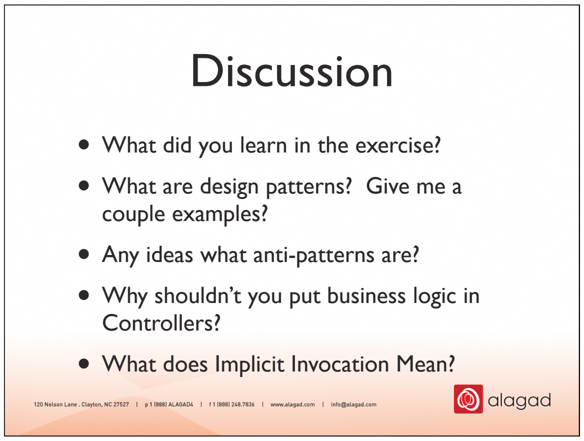


Questions and Answers



Exercise

- Update your procedural Fortune application use your own version of MVC
- I suggest a Page Controller
  - You might create a “controller” custom tag.
  - Controller could use custom tags to get data.
  - Controller should return data to view to display.



Discussion

- What did you learn in the exercise?
- What are design patterns? Give me a couple examples?
- Any ideas what anti-patterns are?
- Why shouldn’t you put business logic in Controllers?
- What does Implicit Invocation Mean?