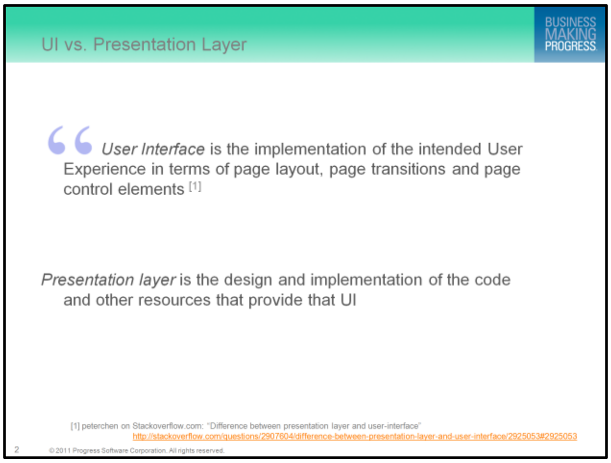
Moving from UI to a Presentation Layer

BUSINESS
MAKING
PROGRESS

Peter Judge
pjudge@progress.com

PROGRESS
software



UI vs. Presentation Layer

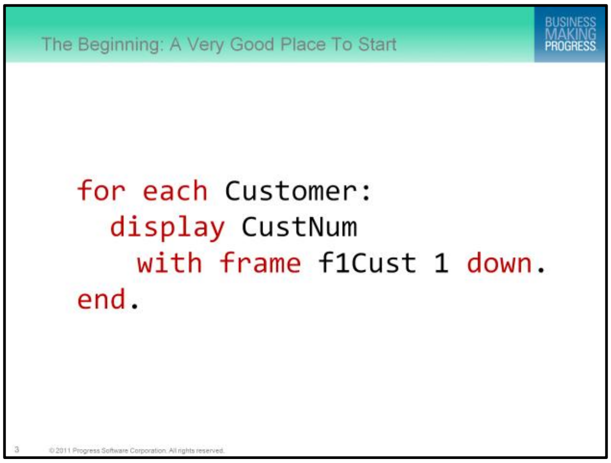
> “*User Interface* is the implementation of the intended User Experience in terms of page layout, page transitions and page control elements [1]

Presentation layer is the design and implementation of the code and other resources that provide that UI

[1] peterchen on Stackoverflow.com: “Difference between presentation layer and user-interface” http://stackoverflow.com/questions/2907604/difference-between-presentation-layer-and-user-interface/2925053#2925053

What’s the difference? Aren’t they the same thing?

Differently put, UI is what you see, PL is what shows it to you.



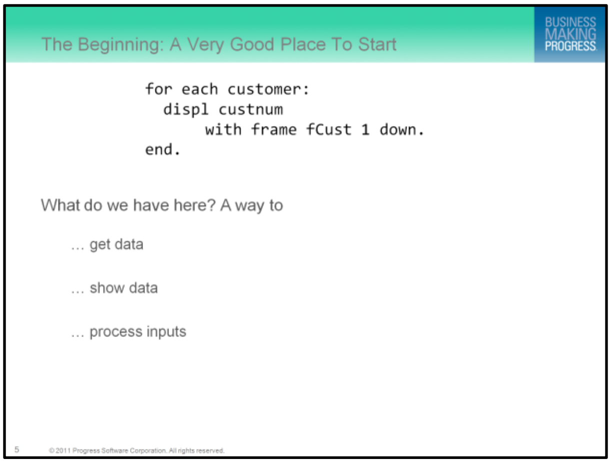
The Beginning: A Very Good Place To Start

```
for each Customer:
  display CustNum
    with frame f1Cust 1 down.
end.
```

How many of you recognise this? Most of you ☺

You can argue that FOR EACH CUSTOMER is a presentation layer in itself. There's code, there's interaction with a human or 2. All set, right?

So I guess to clarify what I mean by presentation layer in the context of this session. Today we'll look a little deeper into the **design** aspect of the presentation layer, although we'll do so largely through code.



The Beginning: A Very Good Place To Start

```
for each customer:
  displ custnum
       with frame fCust 1 down.
end.
```

What do we have here? A way to

… get data

… show data

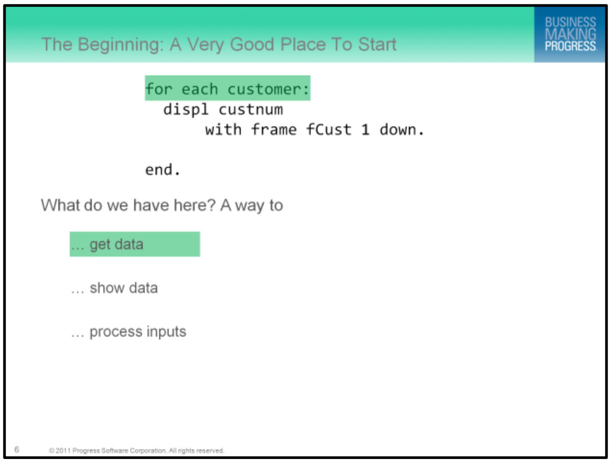
… process inputs

5

It’s quite a powerful few lines of code …

This code contains all of the concepts and code that we need/want to work on

It gets data, shows data and handles user input.



The Beginning: A Very Good Place To Start

```
for each customer:
  displ custnum
       with frame fCust 1 down.

end.
```

What do we have here? A way to

… get data

… show data

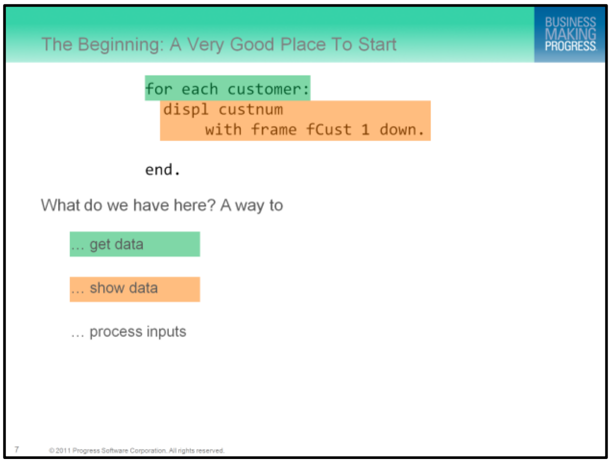
… process inputs

6 © 2011 Progress Software Corporation. All rights reserved.

No way to process inputs? Think about the ‘press spacebar to continue’ message .

This is not very nice code for a number of reasons

1. Direct DB access
2. No way to intercept user input in ABL



The Beginning: A Very Good Place To Start

```
for each customer:
  displ custnum
       with frame fCust 1 down.

end.
```

What do we have here? A way to

… get data

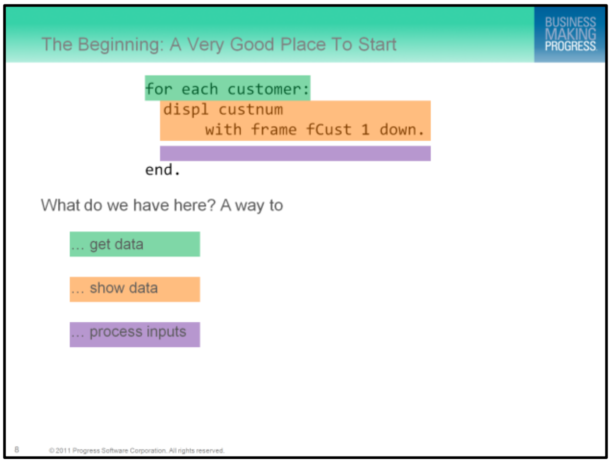
… show data

… process inputs

No way to process inputs? Think about the ‘press spacebar to continue’ message .

This is not very nice code for a number of reasons

1. Direct DB access
2. No way to intercept user input in ABL



The Beginning: A Very Good Place To Start

```
for each customer:
  displ custnum
      with frame fCust 1 down.

end.
```

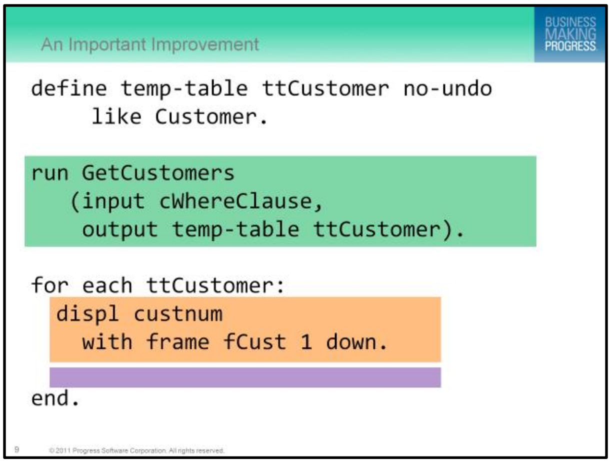
What do we have here? A way to

- … get data
- … show data
- … process inputs

No way to process inputs? Think about the ‘press spacebar to continue’ message .

This is not very nice code for a number of reasons

1. Direct DB access
2. No way to intercept user input in ABL



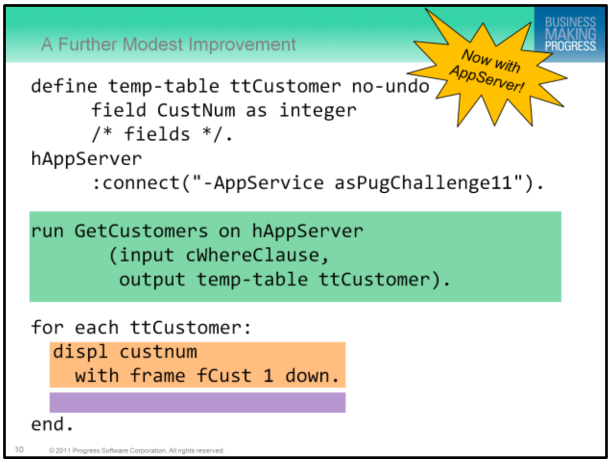
An Important Improvement

```
define temp-table ttCustomer no-undo
     like Customer.

run GetCustomers
   (input cWhereClause,
    output temp-table ttCustomer).

for each ttCustomer:
  displ custnum
    with frame fCust 1 down.

end.
```

Much better.

Having something like GetCustomers() is very important since it allows us to populate ttCustomer any way we want to. We can filter, sort , slice and dish, mashup and futz with our data any way we want, WITHOUT having to change the display. We have a contract (of sorts) contained in the temp-table definition, and if we mess things up – like remove the CustNum field – the compiler lets us know all about it.

Direct DB connection broken in this program, but still UI issues remain. Still local.



A Further Modest Improvement

Now with AppServer!

BUSINESS MAKING PROGRESS

```
define temp-table ttCustomer no-undo
      field CustNum as integer
      /* fields */.
hAppServer
      :connect("-AppService asPugChallenge11").

run GetCustomers on hAppServer
       (input cWhereClause,
        output temp-table ttCustomer).

for each ttCustomer:
  displ custnum
    with frame fCust 1 down.

end.
```

Notice that we’ve changed the temp-table definition to define the fields, rather than using “Like Customer”. We’ve now completely broken the dependency on the database in our UI code. getCustomers() obviously still needs to know where to get the data from, but the UI has no clue.

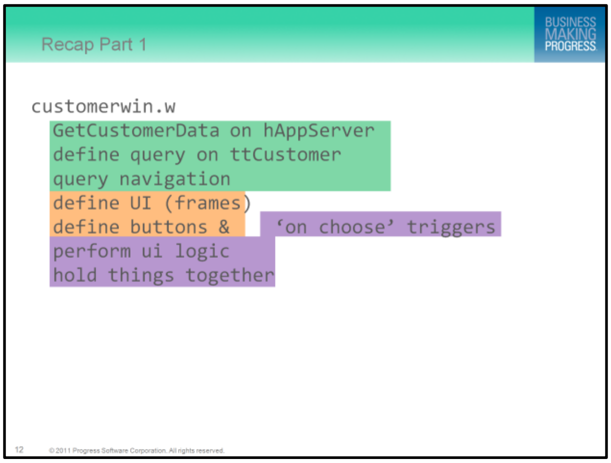
Ok, now we have a decent starting point. We run on the AppServer, which means we get performance benefits: more machine, typically. We can also ask for a subset of data by passing filter criteria.

This is a very simple starting point, but it’s intended to illustrate the concepts and separate concerns that we’re dealing with.

Time to look at real* code

* relatively real-ish, at any rate

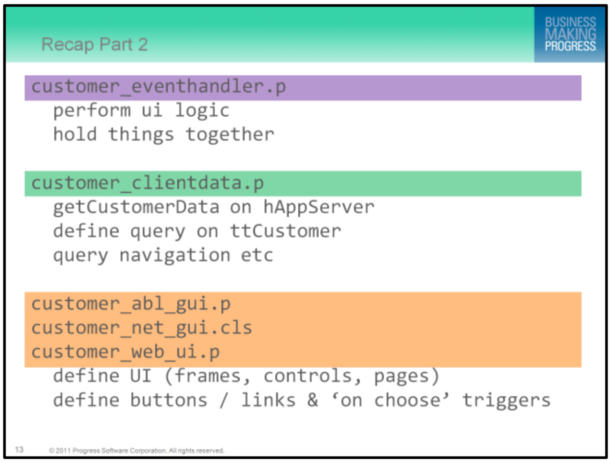
Client/customerwin.w



Recap Part 1

```
customerwin.w
  GetCustomerData on hAppServer
  define query on ttCustomer
  query navigation
  define UI (frames)
  define buttons &   'on choose' triggers
  perform ui logic
  hold things together
```

Let’s put this in terms of our earlier categorisations



Recap Part 2

BUSINESS MAKING PROGRESS

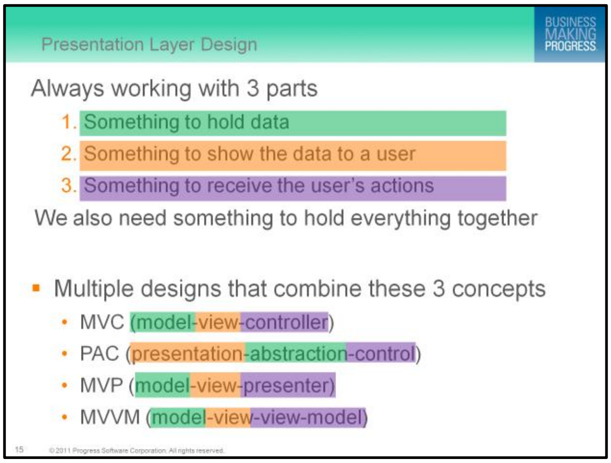
```
customer_eventhandler.p
  perform ui logic
  hold things together

customer_clientdata.p
  getCustomerData on hAppServer
  define query on ttCustomer
  query navigation etc

customer_abl_gui.p
customer_net_gui.cls
customer_web_ui.p
  define UI (frames, controls, pages)
  define buttons / links & ‘on choose’ triggers
```

13 © 2011 Progress Software Corporation. All rights reserved.

Now each action has it’s own program / component

Tying the pieces together

Client/customerwin.w



Presentation Layer Design

Always working with 3 parts

1. Something to hold data
2. Something to show the data to a user
3. Something to receive the user's actions

We also need something to hold everything together

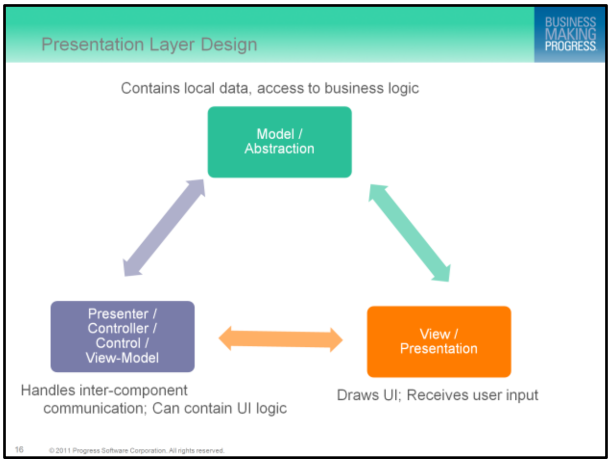
- Multiple designs that combine these 3 concepts
  - MVC (model-view-controller)
  - PAC (presentation-abstraction-control)
  - MVP (model-view-presenter)
  - MVVM (model-view-view-model)

15 © 2011 Progress Software Corporation. All rights reserved.

Data: Not a DAO but a client-side proxy of business entity.
Ok so more like 4 parts.

There are many named designs - or patterns - for implementing this stuff. MVC, MVP, …
Note that they all have 3 parts. Even MVVM is really only 3 parts.

MVC – controller takes user input directly
PAC – controller takes user input directly
MVP – lets view take some input
MVVM – view-model is effectively a controller. Introduced for Silverlight



Presentation Layer Design

Contains local data, access to business logic

Model / Abstraction

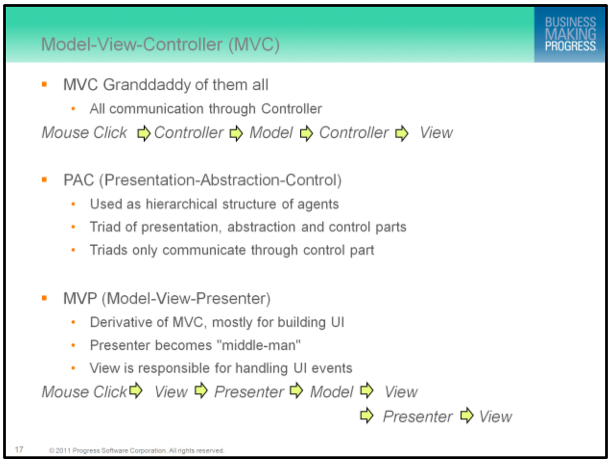
Presenter / Controller / Control / View-Model

View / Presentation

Handles inter-component communication; Can contain UI logic

Draws UI; Receives user input

The directions and strength of the connections also differentiate the patterns. Conceptually, they’re all quite similar, and hence a lot of confusion about what differentiates them. I’ll try to explain.



Model-View-Controller (MVC)

- MVC Granddaddy of them all
  - All communication through Controller

Mouse Click ⇨ Controller ⇨ Model ⇨ Controller ⇨ View

- PAC (Presentation-Abstraction-Control)
  - Used as hierarchical structure of agents
  - Triad of presentation, abstraction and control parts
  - Triads only communicate through control part

- MVP (Model-View-Presenter)
  - Derivative of MVC, mostly for building UI
  - Presenter becomes "middle-man"
  - View is responsible for handling UI events

Mouse Click ⇨ View ⇨ Presenter ⇨ Model ⇨ View
⇨ Presenter ⇨ View

"Grandaddy" MVC typically associated with frameworks, it is essentially an architecture.
Controlflow: Many flavoursControl flow is generally as follows:

The user interacts with the user interface in some way (for example, by pressing a mouse button).

The controller handles the input event from the user interface, often via a registered handler or callback, and converts the event into an appropriate user action, understandable for the model.

The controller notifies the model of the user action, possibly resulting in a change in the model's state.

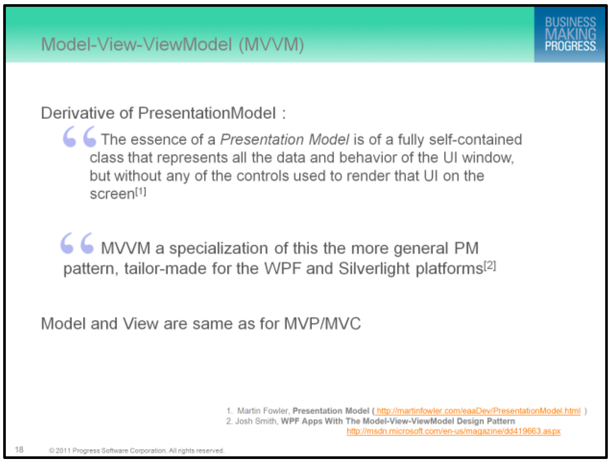
A view queries the model in order to generate an appropriate user interface (for example the view lists the shopping cart's contents). The view gets its own data from the model. In some implementations, the controller may issue a general instruction to the view to render itself. In others, the view is automatically notified by the model of changes in state that require a screen update.

The user interface waits for further user interactions, which restarts the control flow cycle.

PAC has same flow as MVC : presentation -> control -> abstraction -> control -> presentation

The model-view-presenter software pattern originated in the early 1990s at Taligent, a joint venture of Apple, IBM, and HP, a

Other versions of model-view-presenter allow some latitude with respect to which class handles a particular interaction, event, or command. This is often more suitable for web-based architectures, where the view, which executes on a client's browser, may be the best place to handle a particular interaction or command.



Model-View-ViewModel (MVVM)

BUSINESS
MAKING
PROGRESS

Derivative of PresentationModel :

> The essence of a *Presentation Model* is of a fully self-contained class that represents all the data and behavior of the UI window, but without any of the controls used to render that UI on the screen[1]

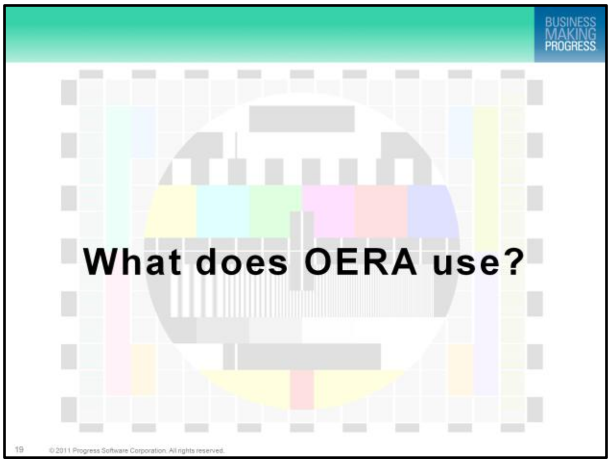
> MVVM a specialization of this the more general PM pattern, tailor-made for the WPF and Silverlight platforms[2]

Model and View are same as for MVP/MVC

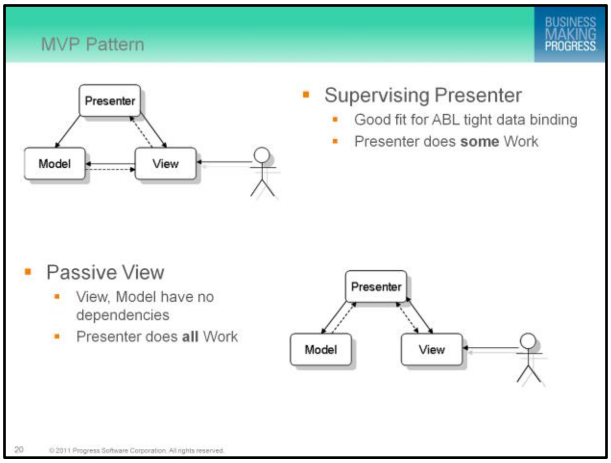
1. Martin Fowler, **Presentation Model** (http://martinfowler.com/eaaDev/PresentationModel.html)
2. Josh Smith, **WPF Apps With The Model-View-ViewModel Design Pattern** http://msdn.microsoft.com/en-us/magazine/dd419663.aspx

PresentationModel/ViewModel means that even without seeing the UI we know what the UI looks like (as it were).

WVVM is pretty MS-specific (XAML etc).



What does OERA use?



MVP Pattern

- Supervising Presenter
  - Good fit for ABL tight data binding
  - Presenter does **some** Work
- Passive View
  - View, Model have no dependencies
  - Presenter does **all** Work

Using Model-View-Presenter (you might have seen that coming :)

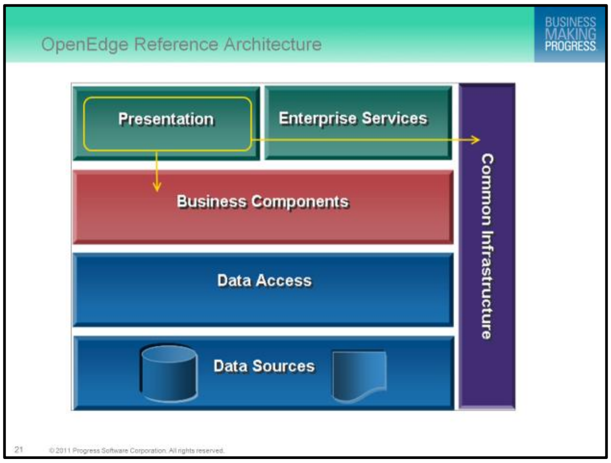
Allows Views to do some work – necessary with multiple UI techs (.cls, .p etc)

Supervising Presenter Variant/Pattern allows View / Model communication

Fits well within OpenEdge Reference Architecture

Though can be used outside OERA

While the decision of design patterns to use should be technology-agnostic as far as possible, the ABL's heritage of providing strong or tight data binding constructs in the language – going back to the DISPLAY and UPDATE statements and continued in the OpenEdge ProBindingSource object – mean that the Supervising Presenter is a pattern well-suited to ABL applications' presentation layers.



OpenEdge Reference Architecture

Presentation

Enterprise Services

Business Components

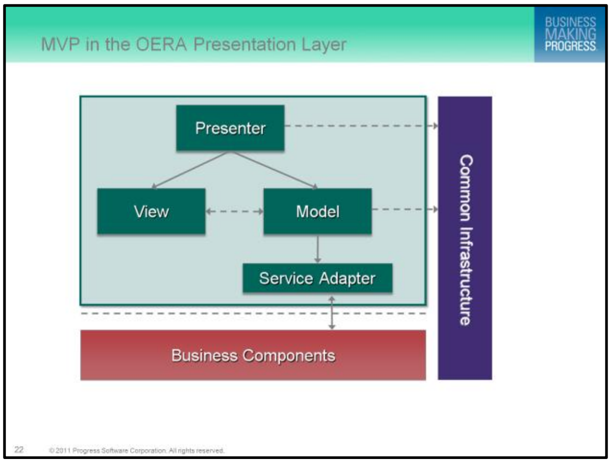
Data Access

Data Sources

Common Infrastructure

21 © 2011 Progress Software Corporation. All rights reserved.

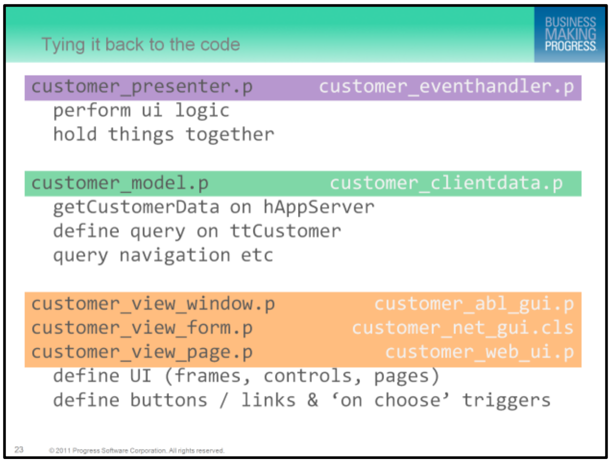
As you'd expect from the title of this talk, it fits into Presentation Layer. Talks to Business Components, Common Infrastructure



MVP in the OERA Presentation Layer

BUSINESS MAKING PROGRESS

Note that the Model becomes a conduit to the data layer (business components). So in OERA the Model typically talks to a Business Entity



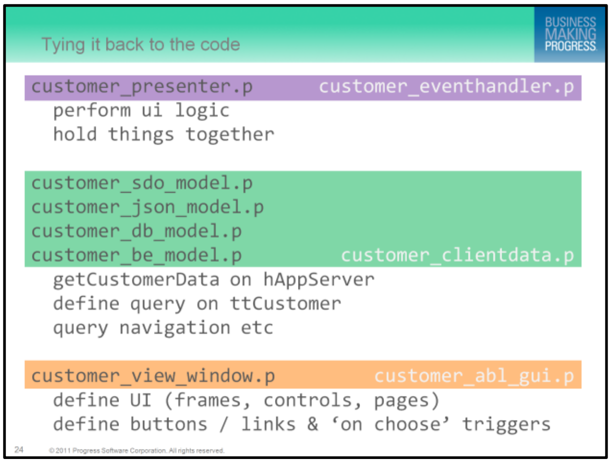
BUSINESS
MAKING
PROGRESS

Tying it back to the code

```
customer_presenter.p         customer_eventhandler.p
  perform ui logic
  hold things together

customer_model.p              customer_clientdata.p
  getCustomerData on hAppServer
  define query on ttCustomer
  query navigation etc

customer_view_window.p              customer_abl_gui.p
customer_view_form.p              customer_net_gui.cls
customer_view_page.p                 customer_web_ui.p
  define UI (frames, controls, pages)
  define buttons / links & ‘on choose’ triggers
```

23 © 2011 Progress Software Corporation. All rights reserved.

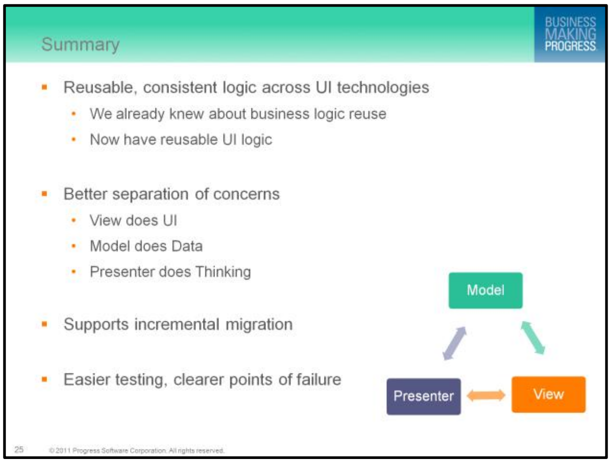
Now each action has it’s own program / component



Tying it back to the code

```
customer_presenter.p         customer_eventhandler.p
  perform ui logic
  hold things together

customer_sdo_model.p
customer_json_model.p
customer_db_model.p
customer_be_model.p            customer_clientdata.p
  getCustomerData on hAppServer
  define query on ttCustomer
  query navigation etc

customer_view_window.p            customer_abl_gui.p
  define UI (frames, controls, pages)
  define buttons / links & ‘on choose’ triggers
```

This pattern lets us swap out the model-side code for different models, without the presenter/views caring



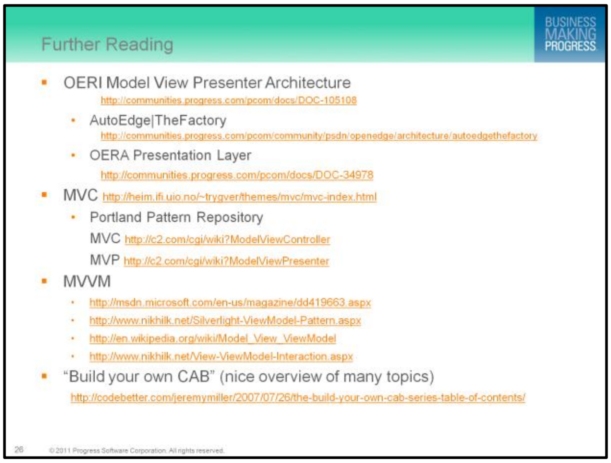
Summary

- Reusable, consistent logic across UI technologies
  - We already knew about business logic reuse
  - Now have reusable UI logic
- Better separation of concerns
  - View does UI
  - Model does Data
  - Presenter does Thinking
- Supports incremental migration
- Easier testing, clearer points of failure

Model
Presenter
View

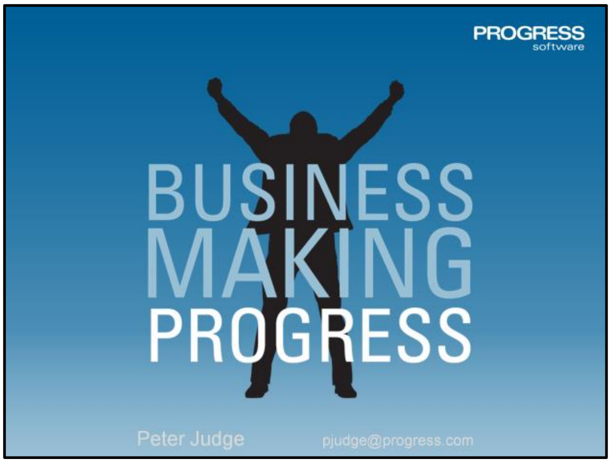
Separation of concerns:
•View … easier to build new Uis
• Means that we can swap out things like Models too, without affecting the other components.

Testing: Maybe fewer bugs. More like fewer places for bugs to appear



Further Reading

- OERI Model View Presenter Architecture
  http://communities.progress.com/pcom/docs/DOC-105108
  - AutoEdge|TheFactory
    http://communities.progress.com/pcom/community/psdn/openedge/architecture/autoedgethefactory
  - OERA Presentation Layer
    http://communities.progress.com/pcom/docs/DOC-34978
- MVC http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html
  - Portland Pattern Repository
    MVC http://c2.com/cgi/wiki?ModelViewController
    MVP http://c2.com/cgi/wiki?ModelViewPresenter
- MVVM
  - http://msdn.microsoft.com/en-us/magazine/dd419663.aspx
  - http://www.nikhilk.net/Silverlight-ViewModel-Pattern.aspx
  - http://en.wikipedia.org/wiki/Model_View_ViewModel
  - http://www.nikhilk.net/View-ViewModel-Interaction.aspx
- “Build your own CAB” (nice overview of many topics)
  http://codebetter.com/jeremymiller/2007/07/26/the-build-your-own-cab-series-table-of-contents/



PROGRESS
software

BUSINESS
MAKING
PROGRESS

Peter Judge pjudge@progress.com

PROGRESS REVOLUTION

BUSINESS MAKING PROGRESS™

Sept. 19 – 22, 2011
Boston Westin Waterfront Hotel
and Boston Convention & Exhibition Center

PROGRESS software