JavaScript Presentation Notes

* Who are you?
* Why are you an expert? Faced maintenance issues with JavaScript.
* Why is this important?
* What is maintainable code? Easy to change. Obvious. Self documenting. Easy to read.
* What we'll cover? Tricky parts of Javascript, code organization and finally some design patterns.

# What is maintainable code?

* Code that is easy to read.
* Easy to modify
* Expressive

# Measurements of complexity

* Lines of code
* Cyclomatic complexity – The number of distinct paths through a code block. Ten or more split into modules.

# First step, understand the rules

# Falsy - False isn't just false in JavaScript.

* Show TRIPLE EQUALITY operator
* Truthy values: “0”, “false”, empty functions, empty arrays, empty objects.

# Equality -Be Careful With Numbers

* Examples
  + 1 === 1
  + 100 === 100
  + 10000000000000000 === 10000000000000000
  + 10000000000000000 === 10000000000000001
  + 1.0000000000000001 === 1
* In JS, all numbers are floating numbers and are prone to floating point approximation.
* You can’t iterate on super large numbers.
* Imagine using a “Long” for an ID and passing as a number.

# Feature Testing

* Already know undefined === false
* Missing features are undefined
* Use jQuery

# Variable Scope – How can we encapsulate variables?

* Function scope
* Variables on the global object.
  + Only difference is first can be deleted and the second can't.

# Hoisting - How is this code unclear?

# Optional Params - How do we make it easier to read the calling of a method?

* Example 1
  + Some optional values
  + Long to type, hard to understand if only the caller is seen
* Example 2
  + Perfect use of JSON. Only use JSON for passing data.
  + Easy to see what parameters this object takes.
* Example 3
  + Coalescing operator and handling undefined

# Objects - JavaScript doesn't have classes but that doesn't mean we can't have encapsulation.

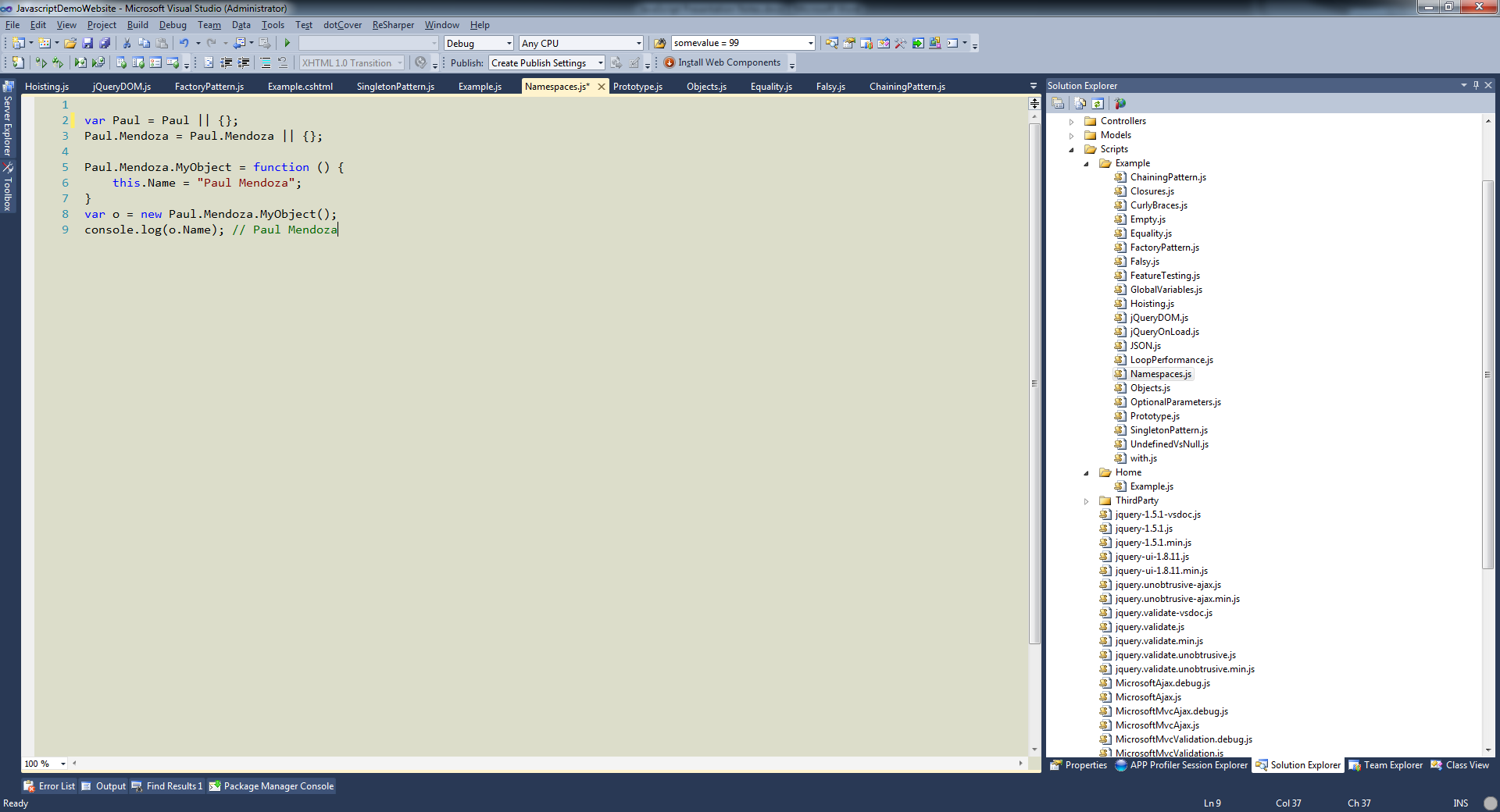
* Variables (public, private)
* Functions (public, private)
* Return statements

# New Constructor Syntax - What is the problem with the new constructor?

* Using “new”
  + Danger is someone forgets it
  + Test for it using if(!(this instanceof Person)) { }

# Curly Braces - Why does curly brace placement matter?

# Namespaces - Don't pollute the global object.



# With operator - Useful but unclear what is happening.

* Avoid
* Slow, interpreter has to look at a bunch of different scopes.
* Impossible to know what is being modified
* console.log(window.someValue);
* SOLUTION: Put the with statement into a variable and just use that.

# Prototype - How can we extend objects?

* Pros
  + Reduce memory consumption
  + Allow extensibility later (similar to C# extension methods)
* Cons
  + for-in loops
    - Only touch objects you own.
  + Can overwrite existing methods on the object.
  + No private variables (extension method)

# Why are design patterns good? Common point of reference. Solve common problems.

# Singleton - Easy pattern.

* No classes in JavaScript.
* Invoking constructor should always return the same object.
* Kind of an antipattern since unit testing with dependency injection becomes hard.

# Module Pattern

* Minifies well

# Strict Mode

* Catches many common errors
* Scoped to either file or function level.
* Will save you many frustrating times.

# Prevent Extension

* ECMAScript 5 only

# Put <Reference path=”Something.js”> at top of files for intellisense

# Microsoft ASP.NET Web Optimization Framework

* Minification and bundling. Better than Google Closure for performance.
* Already setup in ASP.NET MVC 4. Can be installed on prior versions though.
* Go to the **Demo** page on the demo site and how the JS references
* Tools -> Manager Nuget Packages -> Online -> Search “Optimizations”
* **Global.asax** file drag in **Bundling – 1.1** to the Application\_Start
* **Global.asax** drag in **Bundling – 1.2** after Application\_Start
* Go to **Views/Demo/Index.cshtml** and replace the JS references with @Scripts.Render("~/Bundles/Demo")
* Refresh the demo page and see the references are still there.
* Navigate to the **Web.Config** and change **system.web/compilation debug=”true”** to **debug=”false”**
* **Refresh** the demo page. Notice how many less scripts there are now.

# Template Engine

You should be using a templating engine. KnockoutJS is great. Includes some great tutorials on their site. I’ve only used KnockoutJS but Backbone is another popular one.

# Biggest problems when I’m writing JS

* What type is the object I’m dealing with? Auto doc comments sometimes solve this.
* Is this parameter required?
* What are the parameters on the JSON object? What if I don’t supply them?

# Wrap Up

* What did we learn? (Tricky javascript bits to avoid, variable scope and encapsulation, design patterns)
* Further reading: JavaScript: The Good Parts
* Other stuff: Avoid eval, minimize your code, Google Closure Compiler