Abstraction and Architecture:

A steady descent into madness

@pseale

What you need to know about me

1. @pseale on twitter
2. You can make this talk better

This is a low-level, detail-oriented talk about

~ ~ Architecture ~ ~

Because I'm working with a specific example, I may forget to talk about the architecture itself. Remind me.

As with all architecture, this talk contains

~ ~ **Assumptions** ~ ~

Long-term developer speed is my priority. Ignore performance*.

Today only, we are banning the following **~ ~ Architecture swear words ~ ~**

- Bad
- Good
- The right way
- Testable
- Maintainable
- Robust
- Extensible
- Cohesion/coupling
- SOLID
- DRY
- Spaghetti code
- “I can have you fired”

~ Each abstraction must justify itself ~

MACBETH

A architectural tragedy in five* acts

similar to Hamlet

Act 1: **Bliss**

Code is **exactly and only** what is **minimally necessary** to make our program work

~ demo ~

ACT I: Bliss

```
protected override void Update(GameTime gameTime) {
  if (GamePad.GetState(PlayerIndex.One).Buttons.Back
      == ButtonState.Pressed || Keyboard.GetState()
      .IsKeyDown(Keys.Escape))
    Exit();

  var keyboardState = Keyboard.GetState();
  _facingDirection = new Vector2(0f, 0f);

  _moveDirection = new Point();
  if (keyboardState.IsKeyDown(Keys.Up))
    _moveDirection.Y--;
  if (keyboardState.IsKeyDown(Keys.Down))
    _moveDirection.Y++;
  if (keyboardState.IsKeyDown(Keys.Left))
    _moveDirection.X--;
  if (keyboardState.IsKeyDown(Keys.Right))
```

ACT I: Bliss

```
   _playerPosition = _playerPosition + _moveDirection;

   var mouseState = Mouse.GetState();
   _firing = mouseState.LeftButton == ButtonState.Press

   var x = Math.Max(Math.Min(mouseState.Position.X, Scr
   var y = Math.Max(Math.Min(mouseState.Position.Y, Scr

   _facingDirection = new Vector2(0f, 0f);
   int xPositionOnScreen = (WidthMidpoint + (_playerPos
   int yPositionOnScreen = (HeightMidpoint + (_playerPo
   _facingDirection.X = ((float)(x - xPositionOnScreen)
   _facingDirection.Y = ((float)(y - yPositionOnScreen)
   float div = 1f/(float)Math.Sqrt(_facingDirection.X*_
_facingDirection.Y*_facingDirection.Y);
   _facingDirection = new Vector2(_facingDirection.X *
   _angle = (float)Math.Atan2(_facingDirection.Y, _faci
```

ACT I: Bliss

```
if (_cameraPosition.Y - _playerPosition.Y > NoFlexZo
  y2 += _moveDirection.Y;

if (_cameraPosition.Y - _playerPosition.Y < -NoFlexZ
  y2 += _moveDirection.Y;
_cameraPosition = new Point(x2, y2);

UpdateEnemies();
UpdateBullets();

DetectCollisions();
KillEnemies();
UpdateSplashes();
UpdateLevel();
UpdateExplosions();

base.Update(gameTime);
```

CHEATING ALERT: This is where I gave up and started making methods

ACT I: Bliss

SCORECARD:

Green: ok
Yellow: caution
Red: abort

In review:

- ✓ Easy to trace execution (just read from top-to-bottom)

- ϴ Duplication, which causes bugs
- ϴ Duplication also makes deep restructuring difficult
- ϴ Classic spaghetti code

Aside: **When is it okay** to write "blissfully ignorant" code?

Questions?

Questions about how the application works?

Act 2: **Procedural Programming**

Code is grouped into **procedures** until there is no duplication. Also, group cohesive logic

~ ~ Lightning-fast Monogame tutorial ~ ~

Update()

Draw()

ACT II: Procedural Programming

```
protected override void Update(GameTime gameTime)
{
  if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed || Keyboard.GetState().IsKeyDown(Keys.Escape))
  Exit();

  var keyboardState = Keyboard.GetState();
  _facingDirection = new Vector2(0f, 0f);

  _moveDirection = new Point();
  if (keyboardState.IsKeyDown(Keys.Up))
  {
  _moveDirection.Y--;
  }
  if (keyboardState.IsKeyDown(Keys.Down))
  {
  _moveDirection.Y++;
  }
  if (keyboardState.IsKeyDown(Keys.Left))
  {
  _moveDirection.X--;
  }
  if (keyboardState.IsKeyDown(Keys.Right))
  {
  _moveDirection.X++;
  }

  if (keyboardState.IsKeyDown(Keys.W))
  {
  _moveDirection.Y--;
  }
  if (keyboardState.IsKeyDown(Keys.A))
  {
  _moveDirection.X--;
  }
  if (keyboardState.IsKeyDown(Keys.S))
  {
  _moveDirection.Y++;
  }
  if (keyboardState.IsKeyDown(Keys.D))
  {
  _moveDirection.X++;
  }

  _playerPosition = _playerPosition + _moveDirection;

  var mouseState = Mouse.GetState();
  _firing = mouseState.LeftButton == ButtonState.Pressed;

  var x = Math.Max(Math.Min(mouseState.Position.X, ScreenWidth), -ScreenWidth);
  var y = Math.Max(Math.Min(mouseState.Position.Y, ScreenHeight), -ScreenHeight);

  _facingDirection = new Vector2(0f, 0f);
  int xPositionOnScreen = (WidthMidpoint + (_playerPosition.X - _cameraPosition.X));
  int yPositionOnScreen = (HeightMidpoint + (_playerPosition.Y - _cameraPosition.Y));
  _facingDirection.X = ((float)(x - xPositionOnScreen) );
  _facingDirection.Y = ((float)(y - yPositionOnScreen) );
  float div = 1f/(float)Math.Sqrt(_facingDirection.X*_facingDirection.X + _facingDirectio
  _facingDirection = new Vector2(_facingDirection.X * div, _facingDirection.Y * div);
  _angle = (float)Math.Atan2(_facingDirection.Y, _facingDirection.X);

  int x2 = _cameraPosition.X;
  int y2 = _cameraPosition.Y;
  if (_cameraPosition.X - _playerPosition.X > NoFlexZone)
  {
  x2 += _moveDirection.X;
  }

  if (_cameraPosition.X - _playerPosition.X < -NoFlexZone)
  {
  x2 += _moveDirection.X;
  }

  if (_cameraPosition.Y - _playerPosition.Y > NoFlexZone)
  {
  y2 += _moveDirection.Y;
  }

  if (_cameraPosition.Y - _playerPosition.Y < -NoFlexZone)
  {
  y2 += _moveDirection.Y;
  }
  _cameraPosition = new Point(x2, y2);

  foreach (var enemy in _enemies)
  {
  if (enemy.IsDoingNothing)
  {
    enemy.TicksUntilDoneDoingNothing--;
    if (enemy.TicksUntilDoneDoingNothing == 0)
    {
    enemy.IsDoingNothing = false;
    enemy.IsMoving = true;
    enemy.TicksUntilDoneMoving = 240;
    }
  } else if (enemy.IsMoving)
    {
      enemy.TicksUntilDoneMoving--;
      enemy.Position = enemy.Position + enemy.Direction;

      if (enemy.TicksUntilDoneMoving == 0)
      {
        enemy.IsMoving = false;
        enemy.IsTurning = true;
        enemy.TicksUntilDoneTurning = 90;
      }
    } else if (enemy.IsTurning)
    {
      enemy.TicksUntilDoneTurning--;
      enemy.Direction = enemy.Direction.Rotate(1);
      if (enemy.TicksUntilDoneTurning == 0)
      {
        enemy.IsTurning = false;
        enemy.IsDoingNothing = true;
        enemy.TicksUntilDoneDoingNothing = 60;
      }
    }
  }

  _bullets.ForEach(p => { p.Position = new Vector2(p.Position.X + p.Direction.X, p.Position.Y + p.Direction.Y); });

  var bulletsToDelete = _bullets.Where(x1 => Math.Abs(x1.Position.X) > GameBorder || Math.Abs(x1.Position.Y) > GameBorder)
    .ToArray();
  foreach (var bulletToDelte in bulletsToDelete)
    _bullets.Remove(bulletToDelte);

  if (_firing)
  {
    var xDelta = _facingDirection.X*10f;
    var yDelta = _facingDirection.Y*10f;
    foreach (var gunAngle in _gunAngles)
    {
      var angle = (int)Math.Sqrt(_random.Next(0, 2*2*gunAngle*gunAngle)) - gunAngle;
      var direction = new Vector2(xDelta, yDelta).Rotate(angle);

      var bullet = new BulletStruct()
      {
        Position = new Vector2(_playerPosition.X + 16 * _facingDirection.X, _playerPosition.Y + 16 * _facingDirection.Y),
        Direction = direction
```

```
      {
        Position = new Vector2(_playerPosition.X + 16 * _facingDirection.X, _playerPosition.Y + 16 * _facingDirection.Y),
        Direction = direction
      };

      _bullets.Add(bullet);
    }
  }

  foreach (var bullet1 in _bullets.ToArray())
  {
    foreach (var enemy1 in _enemies)
    {
      Vector2 position1 = bullet1.Position;
      Vector2 position2 = enemy1.Position;
      if (((position1.X + 2 > position2.X - 16 && position1.X + 2 < position2.X + 16)
          || (position1.X - 2 < position2.X + 16 && position1.X - 2 > position2.X - 16)) &&
        ((position1.Y + 2 > position2.Y - 16 && position1.Y + 2 < position2.Y + 16)
          || (position1.Y - 2 < position2.Y + 16 && position1.Y - 2 > position2.Y - 16)))
      {
        _bullets.Remove(bullet1);
        enemy1.Health--;
        _collisionSplashes.Add(new CollisionSplashStruct()
        {
          Position = bullet1.Position,
          Direction = new Vector2() - bullet1.Direction,
          SplashCounter = 0
        });
      }
    }
  }

  foreach (var enemy2 in _enemies.ToArray())
  {
    if (enemy2.Health <= 0)
    {
      _enemies.Remove(enemy2);
      var explosionStruct = new ExplosionStruct(){ Position = enemy2.Position, Ticks = 0 };
      explosionStruct.Fragments = new List<Vector2>();
      for (int i = 0; i < 36; i++)
      {
        explosionStruct.Fragments.Add(new Vector2(1, 0).Rotate(_random.Next(0, 360)) * _random.Next(0, 10));
      }
      _explosions.Add(explosionStruct);
      _playerXp++;
    }
  }

  foreach (var splash in _collisionSplashes.ToArray())
  {
    splash.SplashCounter++;
    if (splash.SplashCounter > 10)
      _collisionSplashes.Remove(splash);
  } _facingDirection.Y);

  if (_triggerPowerUpText)
  {
    _powerUpCounter--;
    if (_powerUpCounter <= 0)
    {
      _triggerPowerUpText = false;
    }
  }
  if ((_playerLevel * _playerLevel + 1) / 3 < _playerXp)
  {
    _playerLevel++;
    _gunAngles.Add((int)Math.Sqrt(_random.Next(2, 250)));
    _triggerPowerUpText = true;
    _powerUpCounter = 90;
  }

  foreach (var explosion in _explosions.ToArray())
  {
    explosion.Ticks++;
    if (explosion.Ticks > 120)
    {
      _explosions.Remove(explosion);
    }
  }

  base.Update(gameTime);
}
```

```
protected override void Upda
   var keyboardInput = Proces
   var mouseInput = ProcessMo
   _moveDirection = keyboardI
   _firing = mouseInput.IsFir
   _facingDirection = mouseIn
   MovePlayer();
   MoveCamera();
   UpdateEnemies();
   UpdateBullets();
   DetectCollisions();
   KillEnemies();
   UpdateSplashes();
   CheckLevel();
   UpdateExplosions();
   base.Update(gameTime);
}
```

ACT II: Procedural Programming

```
_font = Content.Load<SpriteFont>("Font");
_texture = Content.Load<Texture2D>("a.png");
_enemyTexture = Content.Load<Texture2D>("b.png");

_bulletTexture = new Texture2D(GraphicsDevice, 4, 4);
_collisionSplashTexture = new Texture2D(GraphicsDevice, 3, 3);
_shrubberyTexture = Content.Load<Texture2D>("shrubbery.png");
var magenta = new Color(Color.Magenta, 1f);
var yellow = new Color(Color.Yellow, 1f);
var red = new Color(Color.Red, 1f);
_bulletTexture.SetData(new Color[16] { magenta, magenta, magenta,
magenta, magenta, magenta, magenta, magenta, magenta, magenta, magenta,
magenta, magenta, magenta, magenta, magenta });
_collisionSplashTexture.SetData(new Color[9] { red, red, red, red,
yellow, red, red, red, red });
_explosionTexture = new Texture2D(GraphicsDevice, 8, 8);
_explosionTexture.SetData(new Color[64] { red, red, red, red, red, red,
red, red, red, red, red, red, red, red, red, red, red, red, red, red,
red, red, red, red, red, red, red, red, red, red, red, red, red, red,
red, red, red, red, red, red, red, red, red, red, red, red, red, red,
red, red, red, red, red, red, red, red, red, red, red, red, red, red,
red, red });
```

ACT II: Procedural Programming

```
_font = Content.Load<SpriteFont>("Font");
_texture = Content.Load<Texture2D>("a.png");
_enemyTexture = Content.Load<Texture2D>("b.png");

_bulletTexture = new Texture2D(GraphicsDevice, 4, 4);
_collisionSplashTexture = new Texture2D(GraphicsDevice, 3, 3);
_shrubberyTexture = Content.Load<Texture2D>("shrubbery.png");
var magenta = new Color(Color.Magenta, 1f);
var yellow = new Color(Color.Yellow, 1f);
var red = new Color(Color.Red, 1f);
_bulletTexture.SetData(new Color[16] { magenta, magenta, magenta,
 _collisionSplashTexture.SetData(new Color[9] { red, red, red, red,
 _explosionTexture = new Texture2D(GraphicsDevice, 8, 8);
_explosionTexture.SetData(new Color[64] { red, red, red, red, red,
```

```
LoadFont();
LoadTexturesFromFile();
LoadTexturesFromArray();
```

```
private void LoadFont() {
 _font = LoadFontByName("Font");
}

private void LoadTexturesFromFile() {
 _texture = LoadTextureFromFile("a.png");
 _enemyTexture = LoadTextureFromFile("b.png");
 _shrubberyTexture = LoadTextureFromFile("shrubbery.png");
}

private void LoadTexturesFromArray() {
 _bulletTexture = CreateSquareTexture(Color.Magenta, BulletSize);
 _collisionSplashTexture = CreateSquareTexture(Color.Red, CollisionSplas
 _explosionTexture = CreateSquareTexture(Color.Red, ExplosionFragmentSiz
}
```

ACT II: Procedural Programming

```
_bulletTexture = new Texture2D(GraphicsDevice, 4, 4);
_collisionSplashTexture = new Texture2D(GraphicsDevice, 3, 3);
var magenta = new Color(Color.Magenta, 1f);
var yellow = new Color(Color.Yellow, 1f);
var red = new Color(Color.Red, 1f);
_bulletTexture.SetData(new Color[16] { magenta, magenta, magenta,
 _collisionSplashTexture.SetData(new Color[9] { red, red, red, red,
 _explosionTexture = new Texture2D(GraphicsDevice, 8, 8);
_explosionTexture.SetData(new Color[64] { red, red, red, red, red,
```

```
private Texture2D CreateSquareTexture(Color color, int size) {
  var texture = new Texture2D(GraphicsDevice, size, size);

  texture.SetData(
    Enumerable.Range(0, size * size)
    .Select(cell => color)
    .ToArray());

  return texture;
}
```

ACT II: Procedural Programming

```
y = random.Next(0, 2);
if (y == 0)
  y = -1;
```

```
int GenerateRandomNegativeOrPositiveOne(Random random) {
  return GetRandomBool(random) ? 1 : -1;
}
bool GetRandomBool(Random random) {
  return NextRandomNumber(random, 1) == 1;
}
int NextRandomNumber(Random random, int maxValue) {
  return NextRandomNumber(random, 0, maxValue);
}
int NextRandomNumber(Random random, int minValue, int maxVa
    return random.Next(minValue, maxValue + 1);
}
```

ACT II: Procedural Programming

In review:

- ✓Easier to reason about code that is grouped by function (this means easier troubleshooting)
- ✓Eliminates duplication, which means fewer bugs and fewer things to remember
- ✓(limited) Encapsulation

- ϴ Harder to read from top-to-bottom
- ϴ Some friction moving the code for two reasons:
  - ϴ Mechanically difficult
  - ϴ Not sure where to put the abstracted code

SCORECARD:

Green: ok
Yellow: caution
Red: abort

Aside: if we all agree duplication is bad and easy to fix, why is there **so much duplication** in my codebase? Can we solve this problem?

Questions?

Act 3: **Objects**

It is **impossible to describe object-oriented programming in C#**. With that said, we collect cohesive behavior into classes in order to promote **encapsulation** and **composability**

ACT III: Objects

```
public static class MathHelper {
  public static Vector2 ShrinkVectorTo1Magnitude(Vector2 vector) {
    var magnitude = 1f / (float)Math.Sqrt(vector.X * vector.X + vector.Y
    return  vector * magnitude;
  }

  public static float ConvertToAngleInRadians(Vector2 direction) {
    return (float)Math.Atan2(direction.Y, direction.X);
  }

  public static Vector2 Rotate(this Vector2 v, float degrees) {
    float Deg2Rad = ((float)(2 * Math.PI)/ 360f);
    float sin = (float)Math.Sin(degrees * Deg2Rad);
    float cos = (float)Math.Cos(degrees * Deg2Rad);

    float tx = v.X;
    float ty = v.Y;
    v.X = (cos * tx) - (sin * ty);
    v.Y = (sin * tx) + (cos * ty);
    return v;
  }
```

ACT III: Objects

```
public class RandomNumberService : IRandomNumberService {
    private readonly Random _random;
    public RandomNumberService() {
        _random = new Random();
    }
    public RandomNumberService(int seed) {
        _random = new Random(seed);
    }

    public int NextRandomNumberBetweenPositiveAndNegative(int value) {
        return NextRandomNumber(value);
    }
    public bool GetRandomBool() {
        return NextRandomNumber(1) == 1;
    }
    public double GenerateRandomNumberClusteredTowardZero(int max) {
        return Math.Sqrt(NextRandomNumber(max * max));
    }

    public int NextRandomNumber(int minValue, int maxValue) {
        return _random.Next(minValue, maxValue + 1);
    }
}
```

ACT III: Objects

```
public class DrawService : IDrawService {
  public DrawService(SpriteBatch spriteBatch, GraphicsDevice graphicsDev
    _spriteBatch = spriteBatch; _graphicsDevice = graphicsDevice;
  }

  public void DrawEntityWithRotation(Texture2D texture, Vector2 position
    _spriteBatch.Draw(texture, position, new Rectangle(0, 0, playerSize,
      new Color(Color.White, 1f), MathHelper.ConvertToAngleInRadians(dir
      new Vector2(playerSize/2, playerSize/2), 1.0f, SpriteEffects.None,
  }

  public void InitializeFrame(Point cameraPosition, int widthMidpoint, i
    _graphicsDevice.Clear(backgroundColor);

    //http://www.david-amador.com/2009/10/xna-camera-2d-with-zoom-and-ro
    var transform = Matrix.CreateTranslation(new Vector3(-cameraPosition
                    Matrix.CreateRotationZ(0)*
                    Matrix.CreateScale(new Vector3(1, 1, 1))*
                    Matrix.CreateTranslation(new Vector3(widthMidpoint,
    _spriteBatch.Begin(SpriteSortMode.Deferred, null, null, null, null,
  }
}
```

ACT III: Objects

```
ice) {




, Vector2 direction, int playerSize) {
 playerSize),
ection),
 1);




 heightMidpoint, Color backgroundColor) {



tation/
.X, -cameraPosition.Y, 0))*



heightMidpoint, 0));
null, transform);
```

ACT III: Objects

```
public class Bullet {
  public Bullet(Vector2 position, Vector2 direction) {
    Position = position;
    Direction = direction;
  }

  public Vector2 Position { get; private set; }
  public Vector2 Direction { get; private set; }

  public bool ShouldBeDeleted(IBoundaryService boundaryService) {
    return boundaryService.OutOfBounds(Position.X)
        || boundaryService.OutOfBounds(Position.Y);
  }

  public void Move() {
    Position = Position + Direction;
  }
}
```

ACT III: Objects

In review:

SCORECARD:

Green: ok
Yellow: caution
Red: abort

✓Encapsulation
✓Composability

ϴ Wrongly-abstracted objects are worse than spaghetti

ϴ Object design is an art, and requires practice and study to become comfortable

Aside: how do you know your design is correct?

How long after do you feel your object designs ‘settle’?

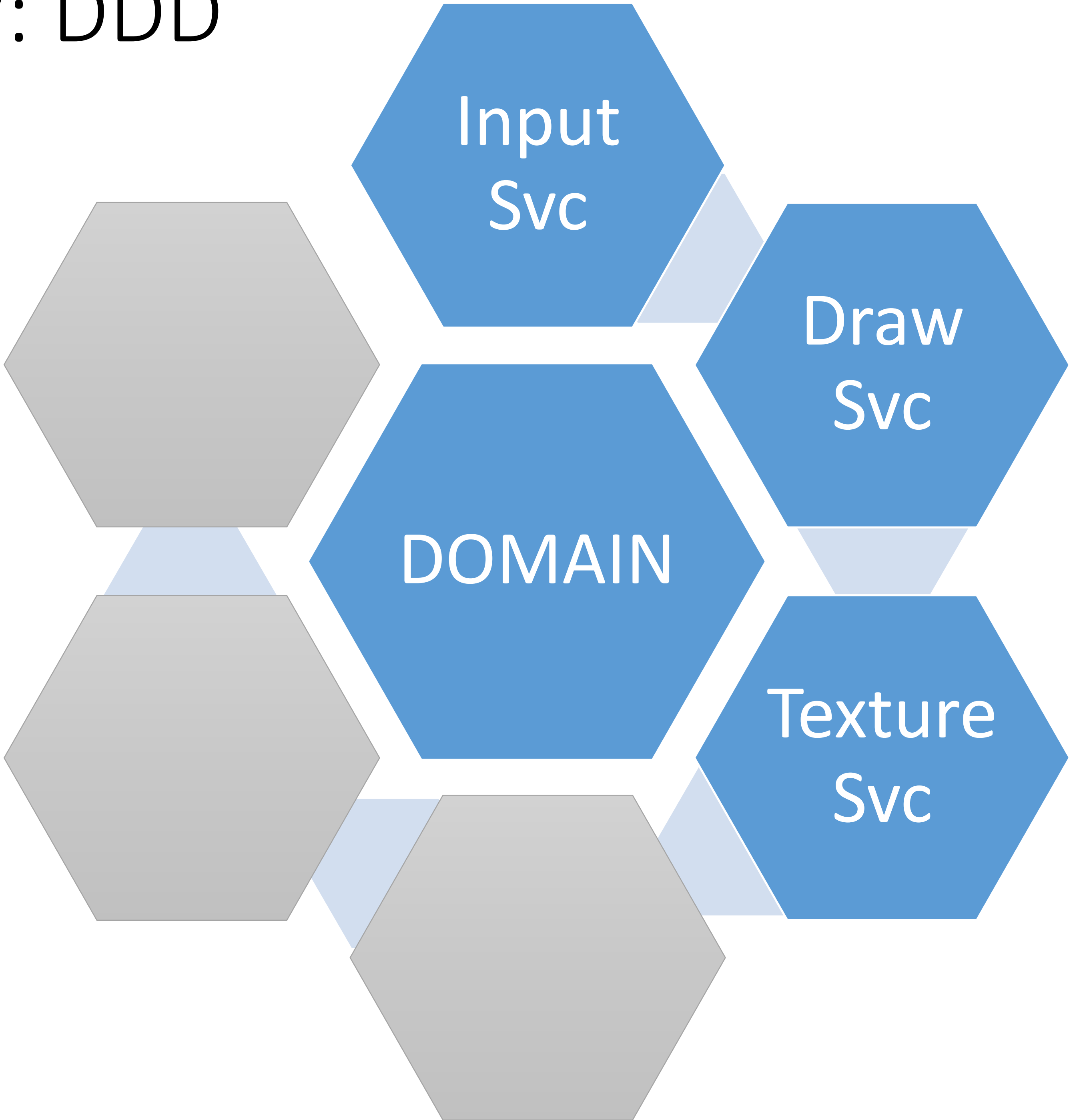
Questions?

Every question welcome, **except from functional programmers**

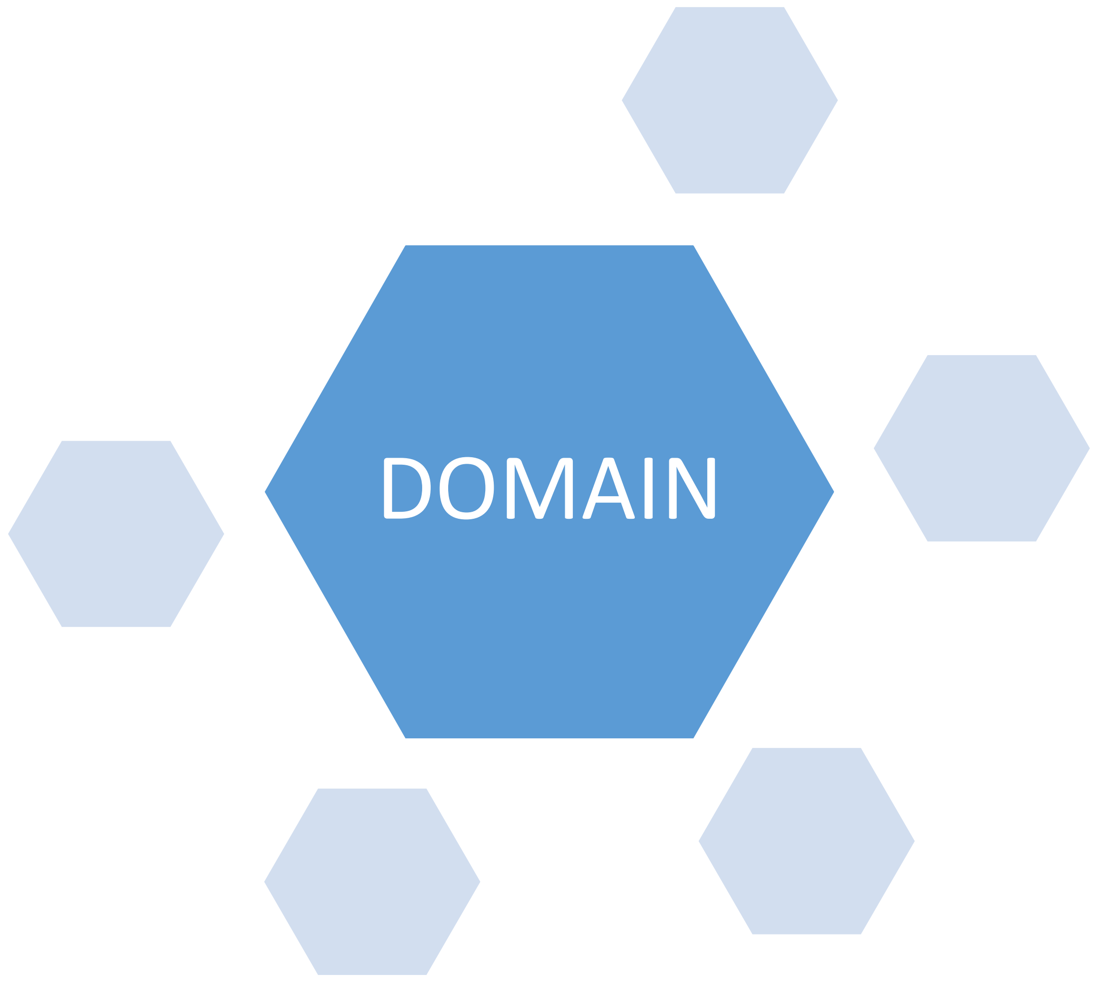
Act 4: **DDD**

Read the entire DDD book by Evans*, plus a thousand blog posts, then **apply DDD principles to codebase**.

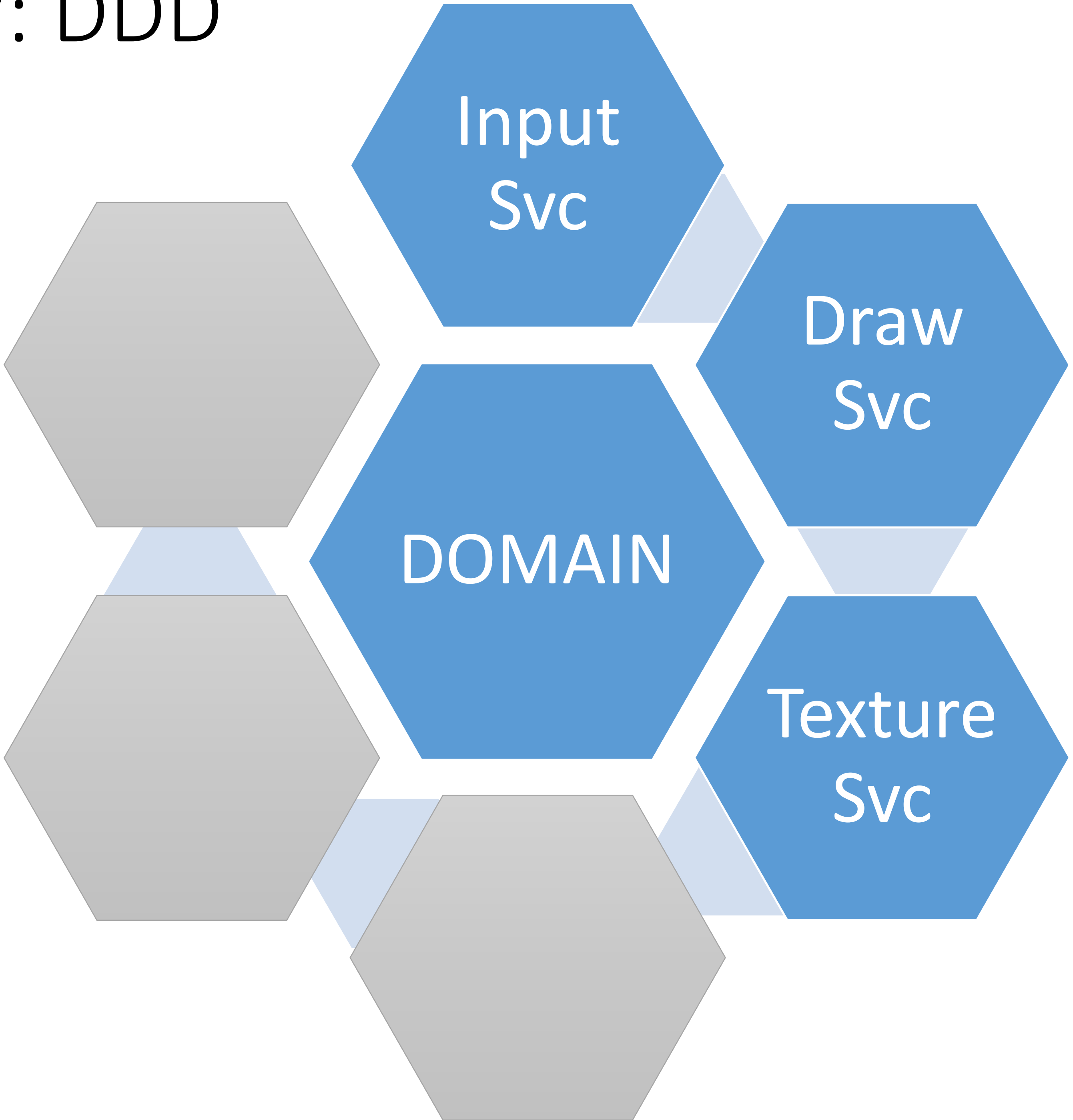
ACT IV: DDD



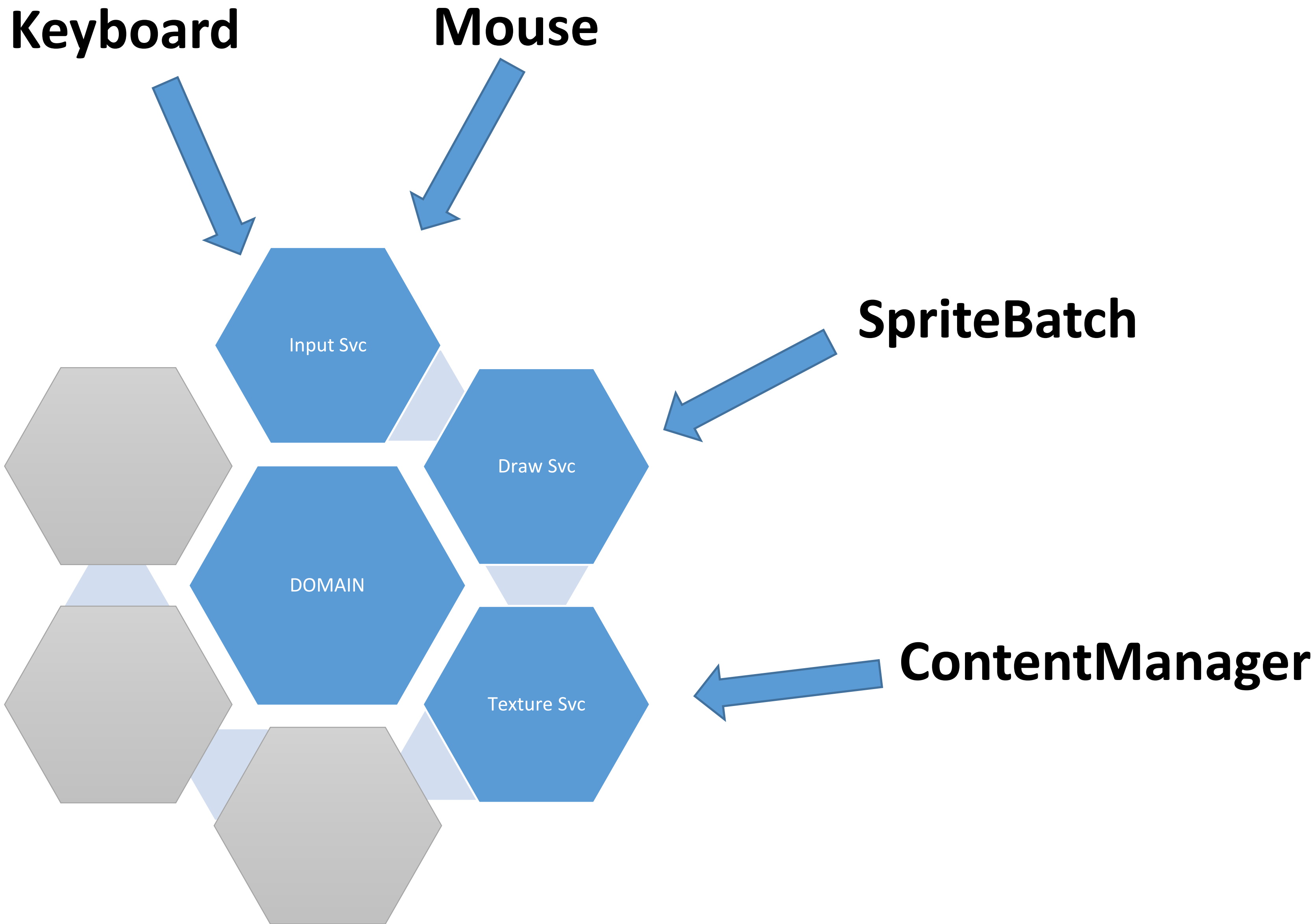
ACT IV: DDD



ACT IV: DDD



ACT IV: DDD


Keyboard
Mouse
SpriteBatch
ContentManager
Input Svc
Draw Svc
DOMAIN
Texture Svc

ACT IV: DDD

```
public class Game {

  //called from Update()
  void Update(InputStruct input)

  //called from Draw()
  Point GetCameraPosition()
  Point GetPlayerPosition()
  Vector2 GetPlayerFacingDirection()
  IEnumerable<Bullet> GetBullets()
  IEnumerable<Enemy> GetEnemies()
  IEnumerable<Shrubbery> GetShrubbery()
  IEnumerable<CollisionSplash> GetCollisionSplashes()
  IEnumerable<ExplosionFragment> GetFragments()
  bool ShouldTriggerPowerUpText()

  //object design is hard - not sure where to put this
  bool OutOfBounds(float position)
}
```

ACT IV: DDD

```
public class MonogameDemoGame : Game {

  protected override void Update(GameTime gameTime) {
    var input = _inputService.ProcessInput();
    _lob.Update(input);
  }

  protected override void Draw(GameTime gameTime) {
    var vm = ViewModelMapper.CreateViewModel(_lob);
    _drawService.InitializeFrame(vm.CameraPosition);
    foreach (var entity in vm.Entities)
      if (entity.HasRotation)
        _drawService.DrawEntityWithRotation(…);
      else
        _drawService.DrawEntity(…);
  }
}
```

ACT IV: DDD

SCORECARD:

Green: ok
Yellow: caution
Red: abort

In review:

✓DDD provides better guiding principles than “naked” OO, which means your abstractions are better, which means you can think in the abstract “ubiquitous language”

ϴ Large learning curve, which means that in a large endeavor, your team will create many bad domain models

ϴ Bad domain models are a tragedy – you get none of the benefits, but mental overhead and N+1s

Aside: Is F# the “pit of success” we need?

(No.)

(Maybe a little, but basically no.)

Questions?

Questions about how the application works?

Act 5: **DSLs**

Domain-specific language

A **specialized language** designed to match your solution space.

ACT V: Domain-specific languages

```
Entity Player
  color < Cyan
  > Mouse1
    fire
  > Keyboard.W
    move -1 0
  > Keyboard.A
    move 0 -1
  > Keyboard.S
    move 1 0
  > Keyboard.D
    move 0 1

  > level_up
    #todo implement
```

ACT V: Domain-specific languages

```
DefineEntity("Player")
                 .Color(@cyan)
                 .On(@mouse1, () => Fire())
                 .On(@w, () => Move(-1, 0))
                 .On(@a, () => Move(0, -1))
                 .On(@s, () => Move(1, 0))
                 .On(@d, () => Move(0, 1))
                 .On(@level_up, () =>
                 {
                     /* todo implement */
                 });
```

ACT V: Domain-specific languages

Implementing an external DSL requires one of the following:

1. Irony
2. M Lang (Oslo)
3. JetBrains MPS
4. ANTLR
5. Sending your ASTs to Roslyn

Aside: if DSLs are so scary to implement, are they **ever** needed?

```
@model WebApplication1.Models.ChangePasswordViewModel
@{
    ViewBag.Title = "Change Password";
}

<h2>@ViewBag.Title.</h2>

@using (Html.BeginForm("ChangePassword", "Manage", FormMethod.Post, new 
{
    @Html.AntiForgeryToken()
    <h4>Change Password Form</h4>
    <hr />
    @Html.ValidationSummary("", new { @class = "text-danger" })
    <div class="form-group">
        @Html.LabelFor(m => m.OldPassword, new { @class = "col-md-2 contr
        <div class="col-md-10">
            @Html.PasswordFor(m => m.OldPassword, new { @class = "form-c
        </div>
    </div>
}
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

```xml
<UserControl x:Class="MonogameDemoGame.DslInAction"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  mc:Ignorable="d"
  d:DesignHeight="300" d:DesignWidth="300">
    <Grid>

    </Grid>
</UserControl>
```

Aside: Workflow Foundation is a DSL (or at its core, a platform to help you build a DSL). **Does it suffer the same problems** as other DSLs?

ACT V: Domain-specific languages

In review:

- ✓ With the correct abstractions, DSLs change the way you think about a problem

- ϴ With incorrect abstractions, DSLs are crippling
- ϴ You are bad at making DSLs

Questions?

Epilogue:
Takeaways

Improve your build & deployment process. Your current setup is **terrible**.

(I'll explain why)

Epilogue: **Takeaways**

Agile plateaus without better engineering practices (i.e. architecture)

Epilogue: Takeaways

Make the smallest change possible
(c.f. “Clean Code”)

Epilogue: Takeaways

Choose the simplest abstraction that works

Epilogue:
Takeaways

When in doubt, make a <Thing>Helper.

Move it later.

Maybe never move it.

Epilogue:

Takeaways

Unspoken rule that must now be spoken:

YAGNI – You Ain’t Gonna Need It

Epilogue:
Takeaways

Don’t feel rushed

Epilogue:
Takeaways

Use your “unit” test projects as **practice** working with abstractions

Epilogue:
Takeaways

Every time you are modifying code,

Find Usages

~ ~ Thank you ~ ~

github.com/pseale/presentation-architecture-madness

Full refunds available at the box office