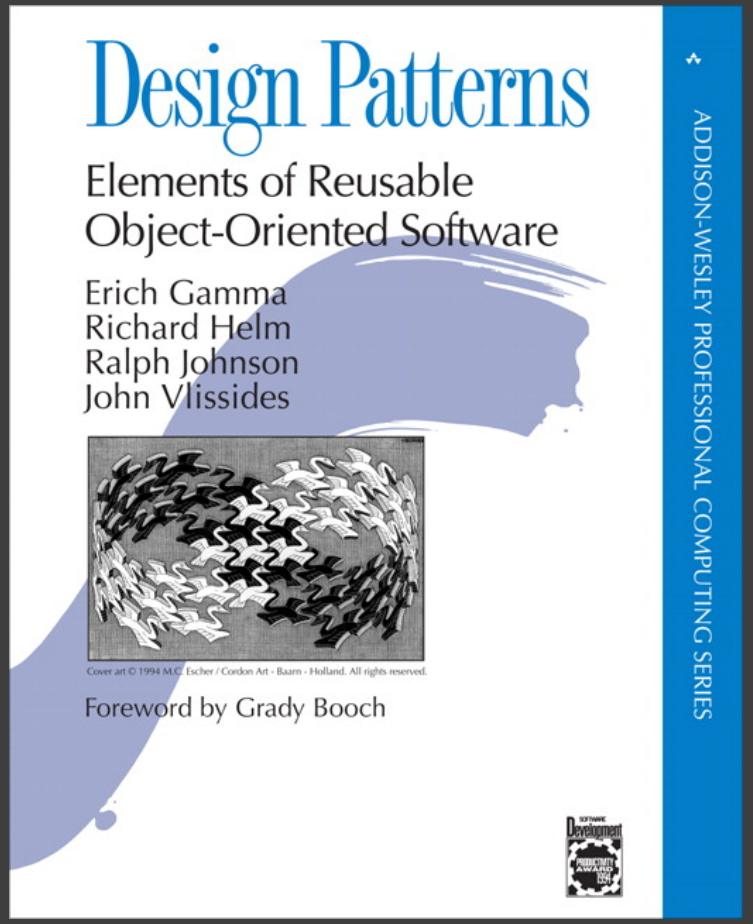
Observations on the Implementation of Design Patterns in C# and .NET

Dmitri Nesteruk

@dnesteruk

Why design patterns?

- Design patterns are no longer as fashionable
- Still relevant, translated to most OOP languages
- We assume everyone knows them
- Made a video course on them (book might happen)
- Design pattern and architecture-related observations



Local Inversion of Control

Normal control: x.foo(y)

Inverted control: y.foo(x)

Implemented using e.g. extension methods

```
Names.Add(name);
```
“Names should add to itself the name”

```
name.AddTo(Names);
```
“Take name and add to Names”

```
public static T AddTo<T>(
    this T self,
    ICollection<T> c)
{
    c.Add(self);
    return self;
}
```

```
name.AddTo(Names);

name.AddTo(Names)
    .AddTo(OtherNames)
    .SomeOtherStuff();
```

```
op == "AND" || op == "OR" || op == "XOR"
```
Operation is AND or OR or XOR

```
new[]{"AND","OR","XOR"}.Contains(op)
```
This list of operations contains our operation (ugly)

```
op.IsOneOf("AND","OR","XOR")
```
Operation is one of the following

```
public static bool IsOneOf<T>(
   this T self,
   params T[] items)
{
   return items.Contains(self);
}
```

Composite

Exposing collections and scalar objects in a uniform way

Different scales:

- Properties
- Entire objects

Property Composite

I have names {FirstName, MiddleName, LastName}

Sometimes, I want an individual name

```
person.FirstName
```

Sometimes, I want to print the full name

```
string.Join(" ", person.Names)
```

How do I get person.Names?

```
public class Person
{
  string FirstName, … { get; }
  public IEnumerable<string> Names
  {
    yield return FirstName;
    …
  }
}
```

Array-Backed Properties

Suppose I want to add Title before the names…

How should I expose it?

Why not store names contiguously?

Never accidentally fail to yield a name from Names

Easier serialization (never miss a property)

Many aggregate get-only properties (e.g., full name without title)

```
public class Person
{
  private string names[4];

  public string Title
  {
    get { return names[0]; }
    set { names[0] = value; }
  }

  public string FullNameNoTitle
    => names.Skip(1).Join(' ');
}
```

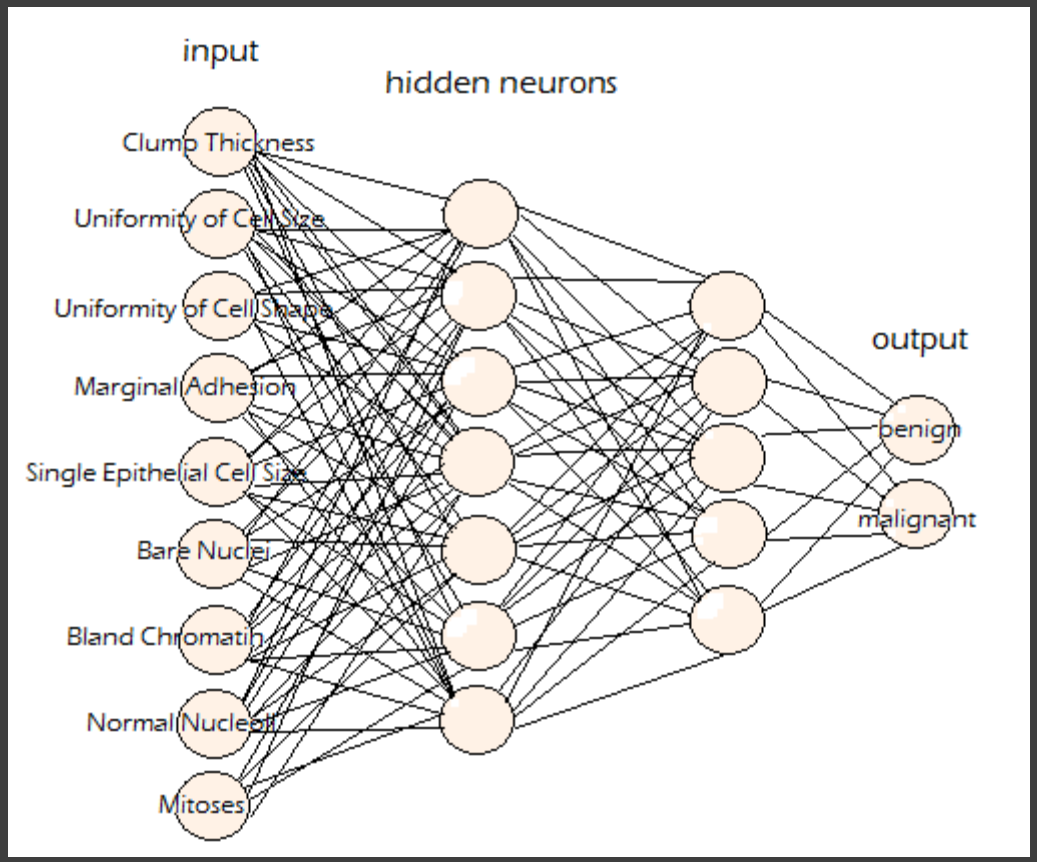
Composite at Class Scale

Neural network model

A single neuron can be connected to another neuron

A bunch of neurons form a layer

We want all forms of entities to be connectable



```
class Neuron
{
  public List<Neuron> In, Out;
}
```

```
class Neuron
{
  public List<Neuron> In, Out;

  public void ConnectTo(Neuron other)
  {
    Out.Add(other);
    other.In.Add(this);
  }
}
```

```
public class NeuronLayer :
Collection<Neuron> {}
```

```
var neuron1 = new Neuron();
var neuron2 = new Neuron();
var layer1 = new NeuronLayer();
var layer2 = new NeuronLayer();

neuron1.ConnectTo(neuron2);
neuron1.ConnectTo(layer1);
layer2.ConnectTo(neuron1);
layer1.ConnectTo(layer2);
```

Hot to make a single ConnectTo()?

Cannot make a base class: NeuronLayer already has one

Use a common interface

NeuronLayer is an IEnumerable<Neuron>

So why not make Neuron IEnumerable<Neuron> too?

```
class Neuron : IEnumerable<Neuron>
{
  public List<Neuron> In, Out;

  public void ConnectTo(Neuron other)
  {
    Out.Add(other);
    other.In.Add(this);
  }
}
```

```
public class Neuron : IEnumerable<Neuron>
{
  public List<Neuron> In, Out;

  public IEnumerator<Neuron> GetEnumerator()
  {
    yield return this;
  }

  IEnumerator IEnumerable.GetEnumerator()
  {
    return GetEnumerator();
  }
}
```

```
public static void ConnectTo(
  this IEnumerable<Neuron> self,
  IEnumerable<Neuron> other)
{
  if (ReferenceEquals(self, other)) return;

  foreach (var from in self)
    foreach (var to in other)
    {
      from.Out.Add(to);
      to.In.Add(from);
    }
}
```

```
var neuron1 = new Neuron();
var neuron2 = new Neuron();
var layer1 = new NeuronLayer();
var layer2 = new NeuronLayer();

neuron1.ConnectTo(neuron2);
neuron1.ConnectTo(layer1);
layer2.ConnectTo(neuron1);
layer1.ConnectTo(layer2);
```

Dynamic in Design Patterns

Runtime-constructed Null Object

Dynamic Proxy

Dynamic Visitor

Null Object

A no-op object that can be injected when required

Typically conforms to an interface

We might not want to explicitly construct such an object (e.g., for testing)

Dynamic to the rescue!

```
public interface ILog
{
  void Info(string msg);
  void Warn(string msg);
}

class ConsoleLog : ILog
{
  public void Info(string msg)
  {
    WriteLine(msg);
  }
  ⋮
}
```

```
public class BankAccount
{
  private ILog log;
  private int balance;

  public BankAccount(ILog log) { this.log = log; }
  public void Deposit(int amount)
  {
    balance += amount;
    log.Info(
      $"Deposited ${amount}, balance is now {balance}");
  }
}
```

Problem

We have a hard dependency on ILog
We cannot supply null – too many NREs
We cannot rewrite existing code to use ?. everywhere
Log may be passed on to other components
How to make a log that does nothing?

```
sealed class NullLog : ILog
{
    public void Info(string msg) {}
    public void Warn(string msg) {}
}
```

```
public class Null<T> :
   DynamicObject where T:class
{
   ⋮
}
```

```
override bool TryInvokeMember(
  InvokeMemberBinder binder,
  object[] args, out object result)
{
  result = Activator.CreateInstance(
    binder.ReturnType);
  return true;
}
```

```
public static T Instance
{
  get
  {
    // ImpromptuInterface
    return new Null<T>().ActLike<T>();
  }
}
```

```
var log = Null<ILog>.Instance;
var ba = new BankAccount(log);
```

Dynamic Visitor

Dispatch = how many pieces of info do I need to know what to call?

Static dispatch = all information must be known at compile time ☹

Then what's the point of polymorphism?

```
interface IStuff { }
class Foo : IStuff { }
class Bar : IStuff { }

static void f(Foo foo) { }
static void f(Bar bar) { }

IStuff i = new Foo();
f(i); // cannot resolve
      // call needs to be on i
```

```
public abstract class Expression
{
  public abstract void Accept(IExpressionVisitor visitor);
}

public interface IExpressionVisitor
{
  void Visit(DoubleExpression de);
  void Visit(AdditionExpression ae);
}

public class DoubleExpression : Expression
{
  override void Accept(IExpressionVisitor visitor)
  {
    visitor.Visit(this);
  }
} // f★★★ this, too much work!
```

```
interface IStuff { }
class Foo : IStuff { }
class Bar : IStuff { }

static void f(Foo foo) { }
static void f(Bar bar) { }

IStuff i = new Foo();
f((dynamic)i); // woo-hoo!
               // dynamic dispatch!
```

Dynamic Proxy

Null Object: remove logging from entity

Dynamic Proxy: add logging to entity (at runtime!)

Implementations of Null<T> and Log<T> are almost identical…

```
public class Log<T> : DynamicObject
  where T : class, new()
{
  private readonly T subject;
  private Dictionary<string, int> methodCallCount =
    new Dictionary<string, int>();

  protected Log(T subject)
  {
    this.subject = subject;
  }

  ⋮
```

```
public override bool TryInvokeMember(InvokeMemberBinder binder, object[] args, out object result)
{
  try
  {
    // logging
    WriteLine($"Invoking {subject.GetType().Name}.{binder.Name} ([{string.Join(",", args)}])");

    // more logging
    if (methodCallCount.ContainsKey(binder.Name)) methodCallCount[binder.Name]++;
    else methodCallCount.Add(binder.Name, 1);

    result = subject.GetType().GetMethod(binder.Name).Invoke(subject, args);
    return true;
  }
  catch
  {
    result = null;
    return false;
  }
}
```

```
public static I As<I>() where I : class
{
  // ensure I is an interface

  return new Log<T>(new T())
    .ActLike<I>();
}
```

```
public string Info
{
  get
  {
    var sb = new StringBuilder();
    foreach (var kv in methodCallCount)
      sb.AppendLine($"{kv.Key} called {kv.Value} time(s)");
    return sb.ToString();
  }
}

// will not be proxied automatically
public override string ToString()
{
  return $"{Info}{subject}";
}
```

```
var ba = Log<BankAccount>.As<IBankAccount>();

ba.Deposit(100);
ba.Withdraw(50);

WriteLine(ba); // ToString()
               // DynamicObject overrides :(
```

That's it!

Questions? Answers? Hate mail? @dnesteruk

Design Patterns in .NET online course at http://bit.ly/2p3aZww

ImpromptuInterface: https://github.com/ekonbenefits/impromptu-interface