Agile tour Québec 2018

Tester c'est douter…

De qui tu te « Mock »?

Vincent VIAL – Développeur – Mock<IConférencier>

Plan

- Contenu pour les non développeurs
- Coding Dojo (code Kata)
  - Qu'est ce qu'un « coding Dojo » ?
  - Présentation du défi
  - Revue de code
  - Refactoring (4 techniques)
  - Conclusion
- Repas ! 11h50

Introduction: Tester c'est ~~douter~~ *difficile*

Pas le temps !

On cherche de la performance avant tout !

Ca n'a pas besoin d'être testé !

C'est risqué !

Ca nécessite l'accord du PO

Développer c'est ..

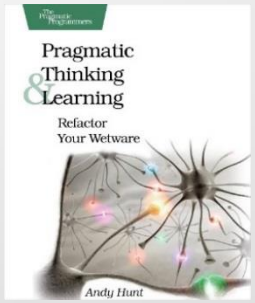
- une activité créative
- dont on attend des résultats concrets
- réalisée en équipe

Ce sont ces paradoxes qui rendent le métier de développeur autant passionnant que difficile !

Comment survivre dans le « Legacy Code » ?

- Agir en professionnel
  - ~~Demander~~ **Prendre** le temps qu'il faut pour faire de la qualité
  - Demander les outils nécessaires
  - Demander du coaching (technique)

- Adopter des principes d'équipe et non des règles d'équipe



- S'entrainer mais pas sur du code de production
  - Formation + coding dojo !

…

</Fin du contenu pour les chargés de projet>

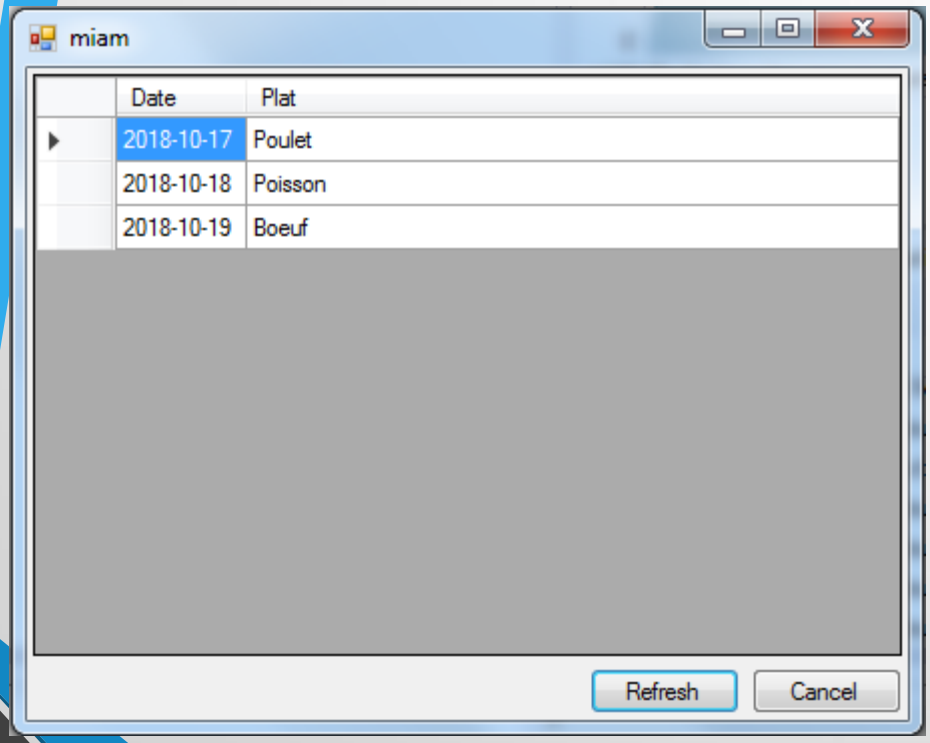
Qu'est ce qu'un « coding dojo » ?

- Aborder un défi de programmation de façon Collective (Code Kata ou Randori)
- Permet d'expérimenter en dehors du code de production
- Apprend aux développeurs à collaborer ensemble en partageant un clavier
- Pas nécessairement organisé par un expert

Objectif du défi proposé

- Découvrir des techniques pour ajouter des tests dans du « Legacy code »
- Découvrir des moyens d'exécuter son code en isolation de dépendances (fake, stub, mock, shim, …)
- Améliorer son efficacité avec son IDE (ici Visual Studio + Resharper)
- Tout ca … en moins d'une heure

L'application « miam » !



- Affiche les repas de la semaine en cours dans une grille
- L'action « Refresh » déclenche le chargement de la grille
- 2 exigences fonctionnelles :
  - Affiche les repas d'aujourd'hui jusqu'au prochain vendredi
  - Si aucun repas, affiche un message à l'utilisateur

No developers were harmed in the making of this code.

Le code derrière l'application

```
public partial class MainForm : Form
{
  public MainForm()
  {
      InitializeComponent();
      _dataGridView1.AutoGenerateColumns = false;
  }

  private void btnRefresh_Click(object sender, EventArgs e)
  {
      using (MiamDbContext context = new MiamDbContext())
      {
          int nb = ((int)DayOfWeek.Friday - (int)DateTime.Today.DayOfWeek + 7) % 7;
          DateTime dateTime2 = DateTime.Today.AddDays(nb);
          List<Repas> repas = context.Repas.Where(x => DateTime.Today <= x.Date
                                                  && x.Date <= dateTime2).ToList();
          if (!repas.Any())
          {
              _dataGridView1.DataSource = new List<Repas>();
              MessageBox.Show("Pas de repas!");
          }
          else
              _dataGridView1.DataSource = repas;
      }
  }
}
```

La revue de code de capitaine « insight »

« Captain Insight », personnage de la série south park

- Il y a des « var »!
- La méthode fait 12 lignes au lieu de 10 maximum
- Il manque des accolades « { »
- 7 devrait être une constante
- Il aurait fallu faire ce code en TDD en appliquant tous les principes SOLID

Code review de l'application

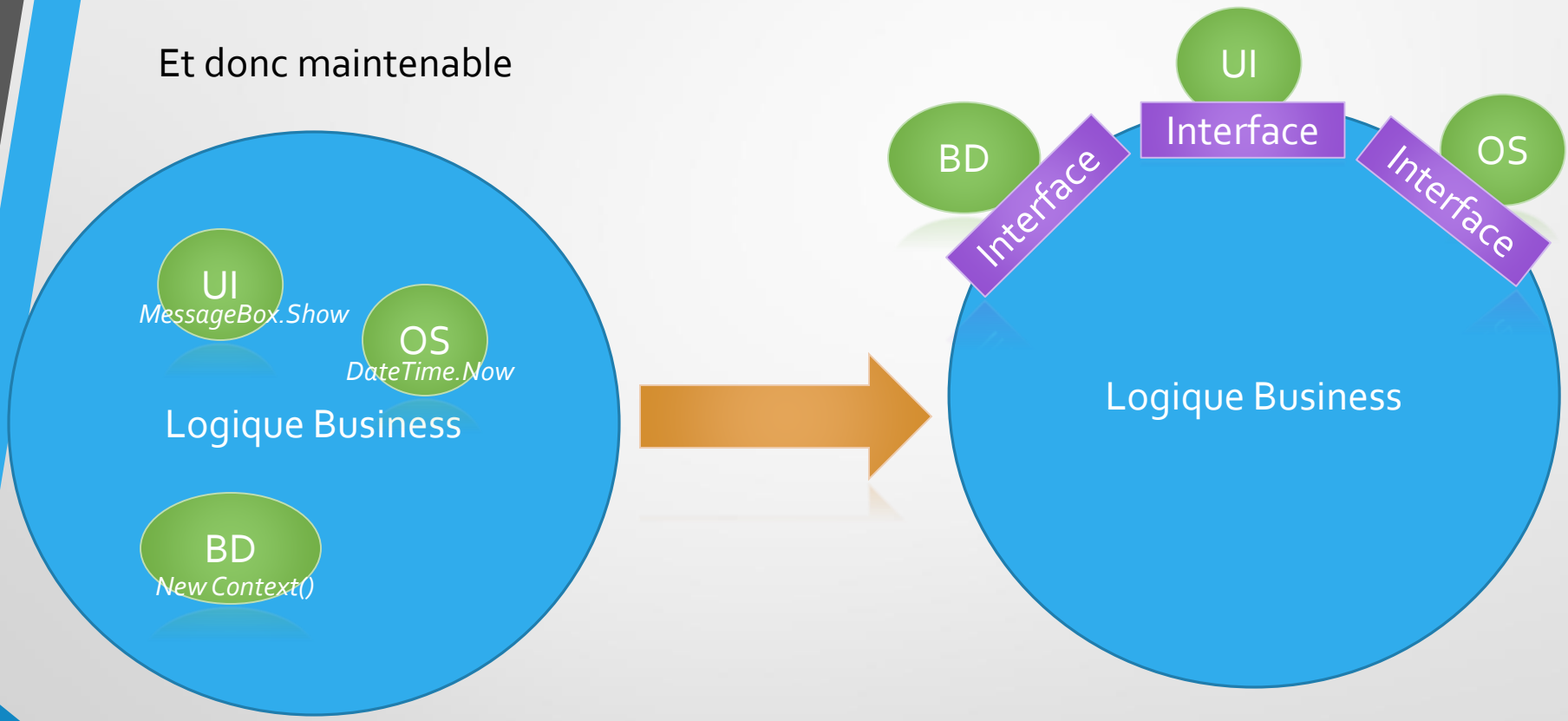
```
public partial class MainForm : Form
{
  public MainForm()
  {
      InitializeComponent();
      _dataGridView1.AutoGenerateColumns = false;
  }

  private void btnRefresh_Click(object sender, EventA
  {
      using (MiamDbContext context = new MiamDbContex
      {
          int nb = ((int)DayOfWeek.Friday - (int)Date
          DateTime dateTime2 = DateTime.Today.AddDays
          List<Repas> repas = context.Repas.Where(x =

          if (!repas.Any())
          {
              _dataGridView1.DataSource = new List<Re
              MessageBox.Show("Pas de repas!");
          }
          else
              _dataGridView1.DataSource = repas;
      }
  }
}
```

Non testable car :

- 1 Appel à des méthodes statiques (DateTime.Today)
- 2 Appel direct à des éléments UI (MessageBox.Show)
- 3 Dépendance à la BD via le context entity framework
- 4 Logique affaire et présentation mélangées

Objectif : Rendre l'application testable

Et donc maintenable



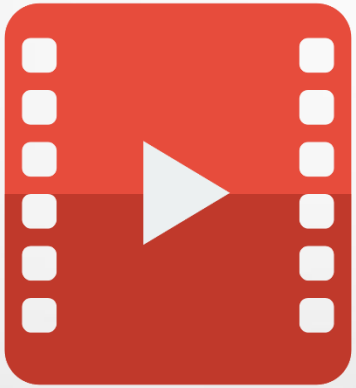
Passer de ça ...

... à ça

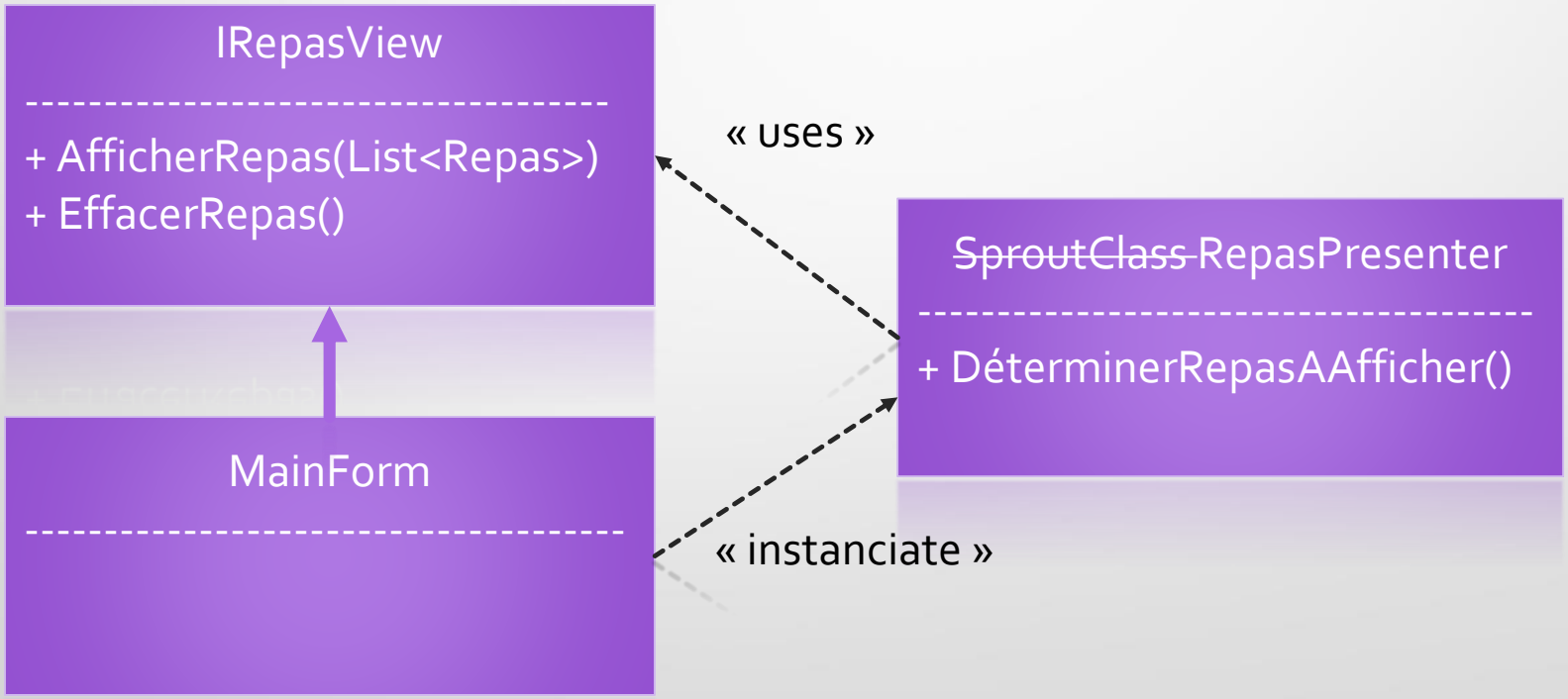
Essayons d'écrire un test…

- Exigences fonctionnelles
  - Affiche les repas d'aujourd'hui jusqu'au prochain vendredi
  - Si aucun repas, affiche un message à l'utilisateur
- Deux exemples
  - Supposons qu'il y a un repas par jour et que l'on est jeudi, alors on devrait voir les repas de jeudi et vendredi
  - Supposons qu'on a pas de repas, alors un message doit s'afficher à l'utilisateur

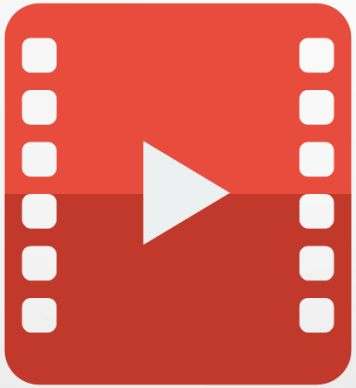
Démo technique 1 – Sprout Class



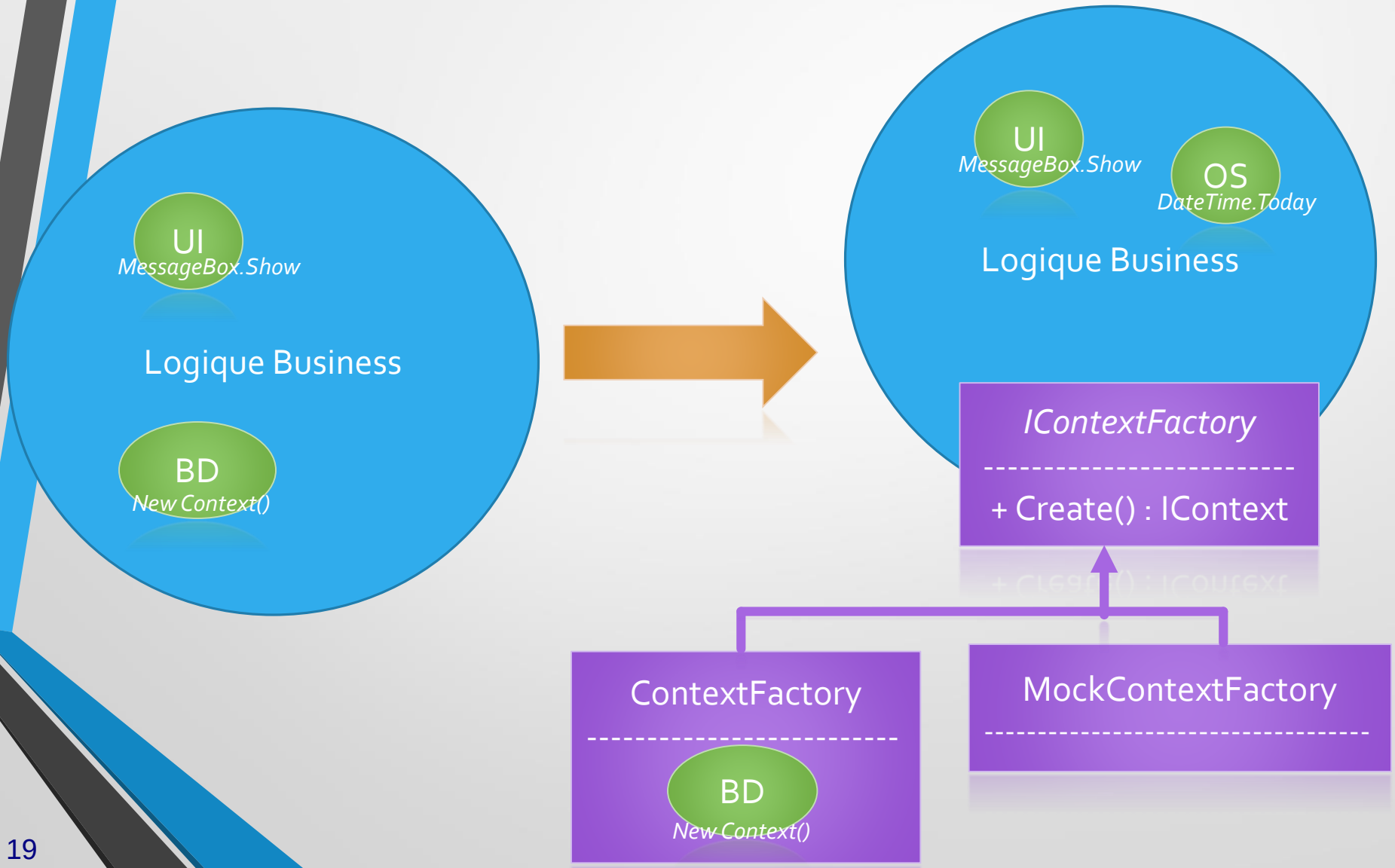
La sprout class permet de séparer la logique de présentation de la logique d'affaire



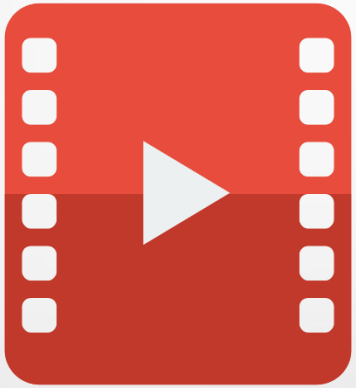
Démo technique 2 – Design Pattern Factory



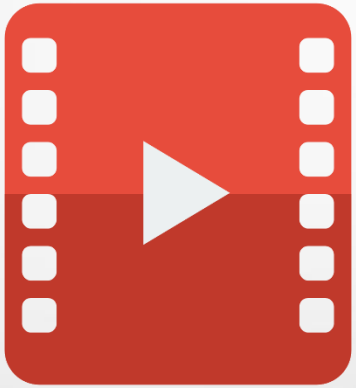
Mocker la BD


UI
MessageBox.Show
Logique Business
BD
New Context()
UI
MessageBox.Show
OS
DateTime.Today
Logique Business
IContextFactory
+ Create() : IContext
ContextFactory
BD
New Context()
MockContextFactory

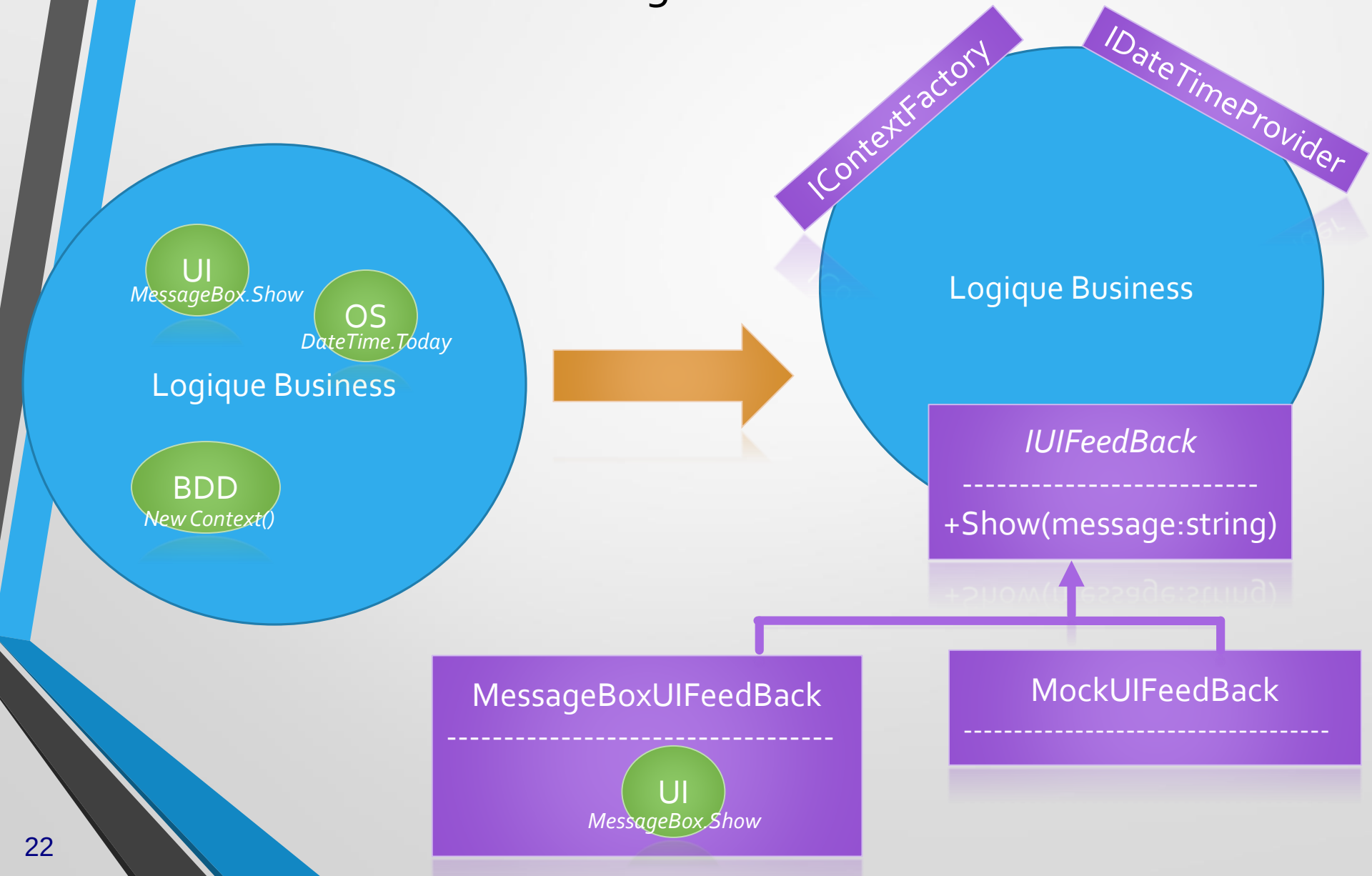
Démo technique 3 – Introduire un paramètre



Démo technique 4 – L'injection de dépendance du pauvre

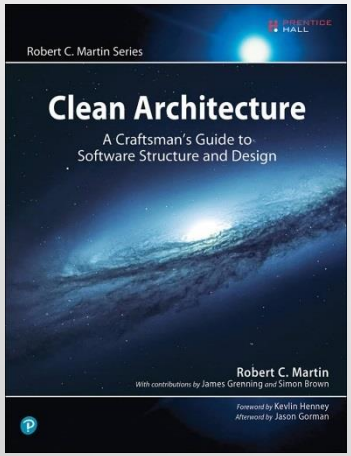


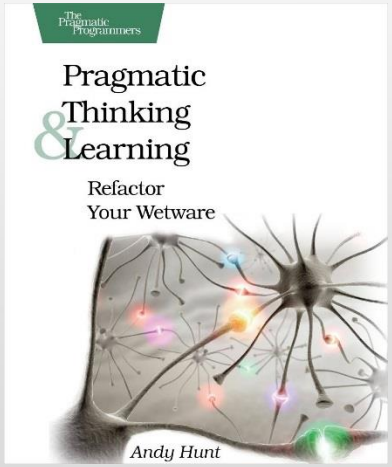
Etape 4 : Enlever la dépendance à MessageBox.Show

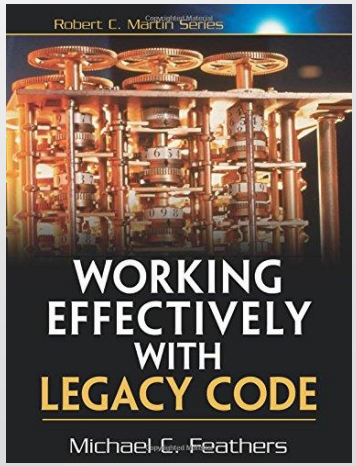


Références

- https://martinfowler.com/articles/mocksArentStubs.html
- « Working Effectively with Legacy Code » de Michael C. Feathers
- « Clean Architecture » de Robert C. Martin
- « Pragmatic Thinking & Learning » de Andrew Hunt







Merci et bon appétit !

vincent.vial@yahoo.fr

http://linkedin.com/in/vincentvial

https://twitter.com/vincevial

https://github.com/vinsoyo/CodingDojoMock