



# COURS .NET

## ANALYSE APPLICATIVE

Lemettre Arnaud

[Arnaud.lemettre@gmail.com](mailto:Arnaud.lemettre@gmail.com)

# SOMMAIRE



- Introduction
- Quelques raccourcis ...
- Les tests unitaires
- .Net Reflector
- Analyse de performance
- FXCop

# INTRODUCTION



- Nous allons voir un aperçu de l'écosystème du Framework .Net, en abordant les tests unitaires, ainsi que les différents outils qui sont nécessaires au bon développement d'une application.



# QUELQUES RACCOURCIS

# QUELQUES RACCOURCIS

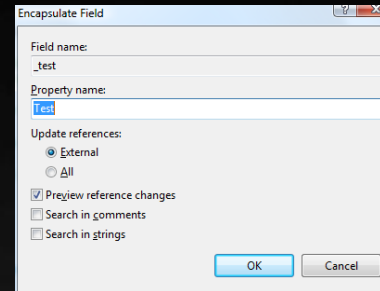


Ctrl+r, ctrl+e génèrent les propriétés.

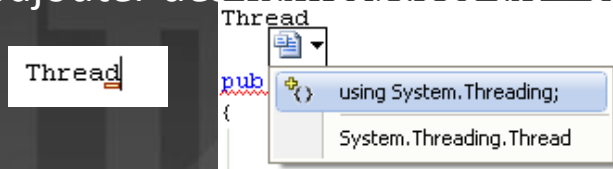
Exemple :

```
private string _test;

public string Test
{
    get { return _test; }
    set { _test = value; }
}
```



- /// permet de générer automatiquement les commentaires, au dessus de fonctions, variables.
- Pour commenter toute une portion de code ctrl+k, ctrl +c
- Pour décommenter toute une portion de code ctrl+k, ctrl +u
- Après avoir sélectionné une partie du code, faire ctrl+k, ctrl+s permet d'englober la sélection avec des instructions (if, boucle, region ...)
- Le double tab permet de compléter certaines instructions (for, if, ...)
- Let ctrl+espace permet l'auto complétion
- Pour rajouter des namespaces alt + shift + f10 ou ctrl + ;





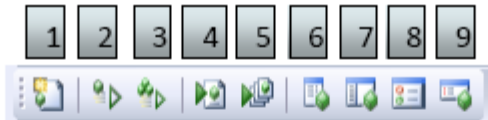
# LES TESTS UNITAIRES

# LES TESTS UNITAIRES



- Les tests unitaires sont un procédé permettant de s'assurer du fonctionnement correct d'une partie déterminée d'un logiciel ou d'une portion d'un programme (appelée « unité ») (Wikipédia).
- Le principe de fonctionnement d'un test unitaire est d'isoler la portion de code à tester du reste du programme, et de tester la partie de code dans un environnement prédéterminé et de voir si cela réagit comme on le souhaite.

# LES INTERFACES



Numéros	Description
1	Permet de créer un projet de test
2	Démarre les tests dans le projet courant
3	Démarre les tests pour toute la solution
4	Démarre les tests en mode « debug » dans le projet courant
5	Démarre les tests en mode « debug » pour toute la solution
6	Affiche la fenêtre de sélection des tests
7	Affiche l'éditeur de liste de tests
8	Affiche le résultat des tests
9	Affiche la fenêtre pour surveiller le système de test (tests en cours, tests en attente ...)

Astuce :

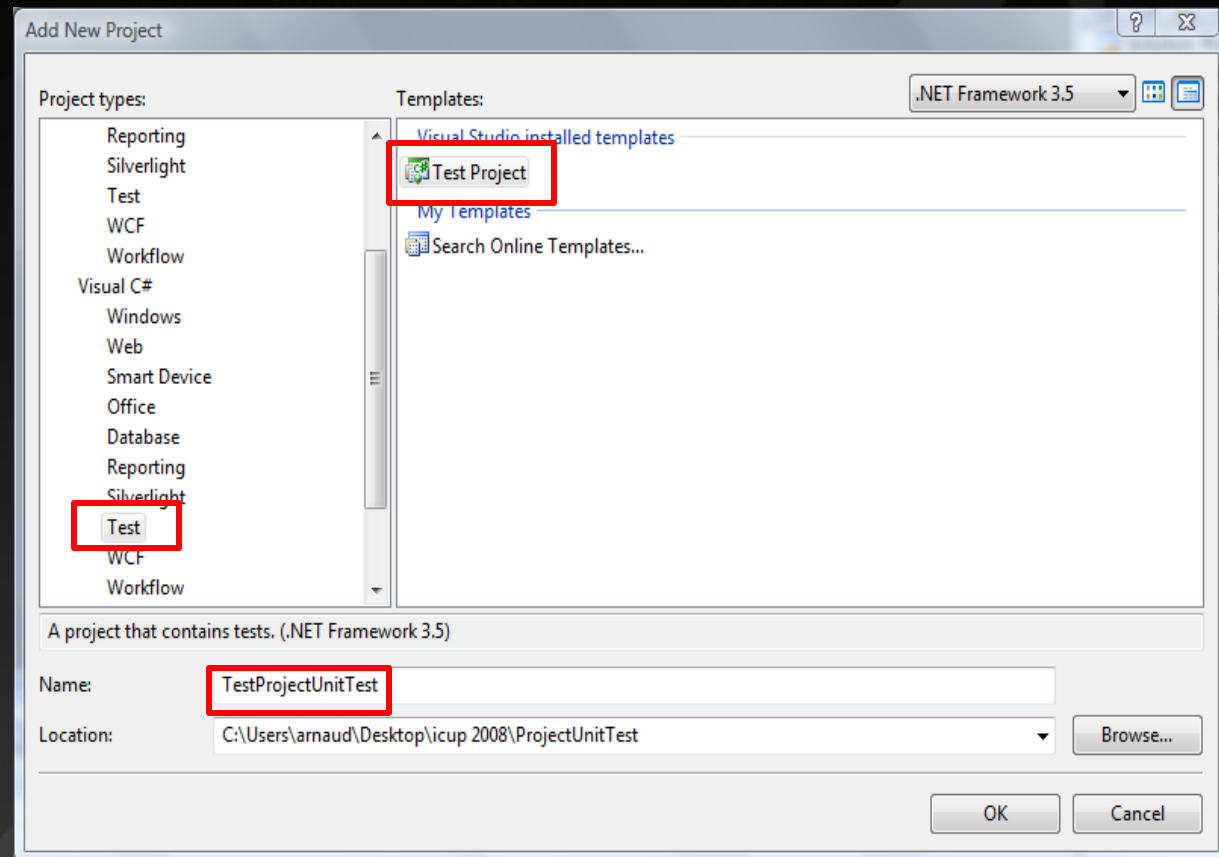
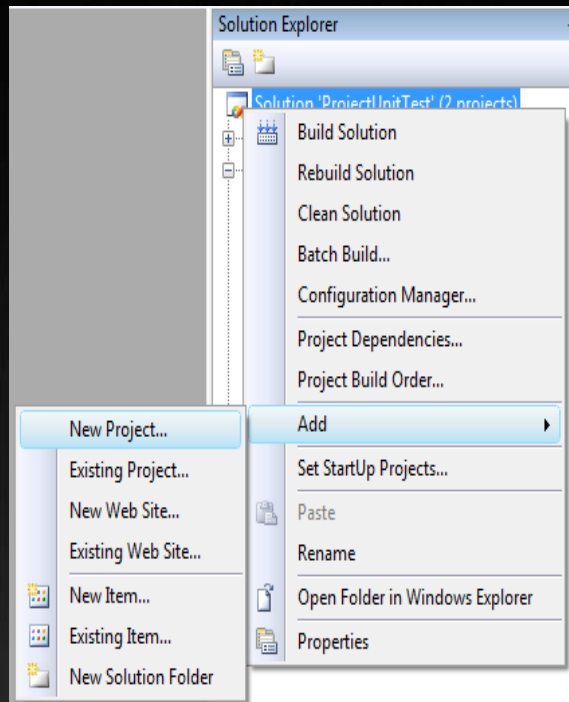
Si l'interface de test n'apparaît pas il faut faire un clic droit sur la barre de menu et cocher le menu Test Tools.



# LES INTERFACES



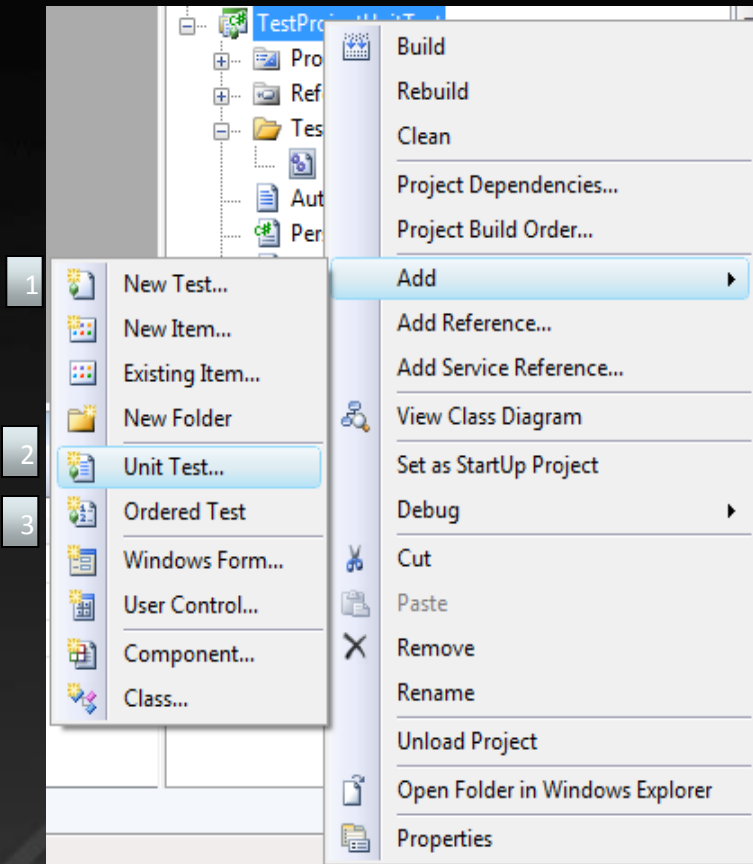
Création du projet :



# LES INTERFACES



Création des tests :

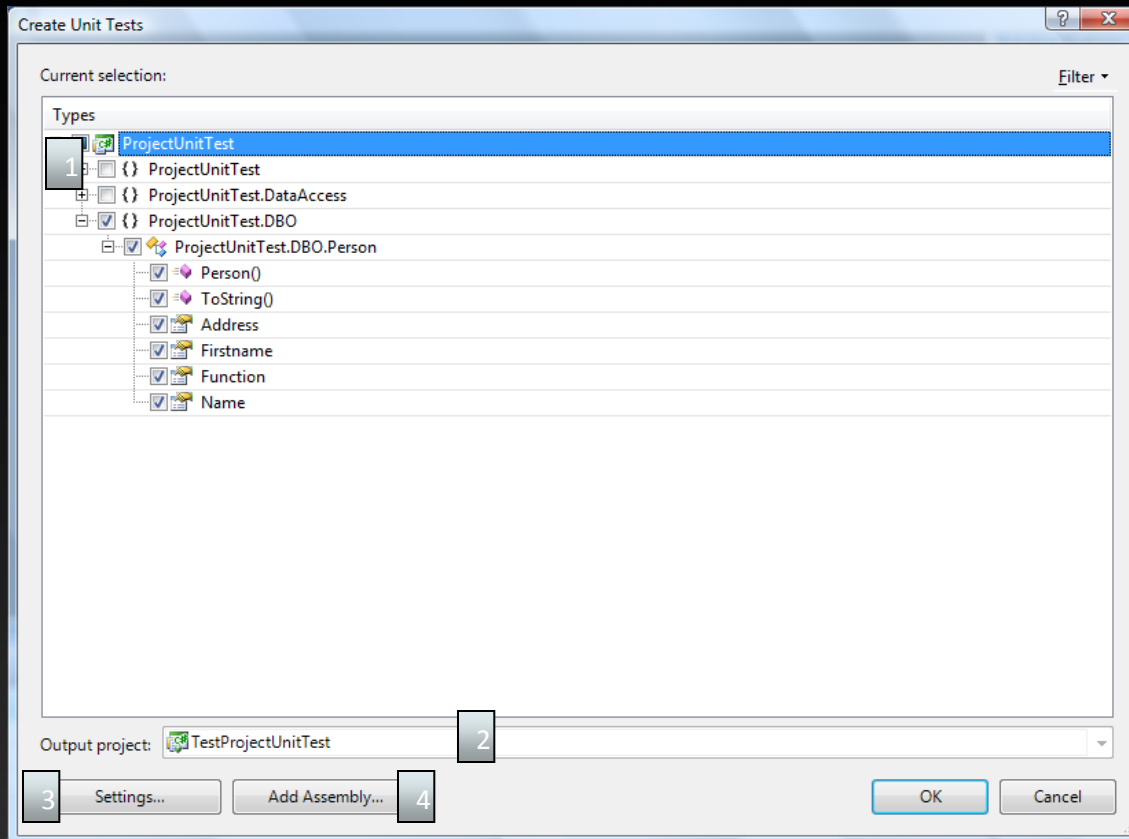


Numéros	Description
1	Crée un fichier de test simple
2	Génère des tests automatiquement à partir de classe
3	Permet de générer un ordonnancement des tests

# LES INTERFACES



Création des tests :

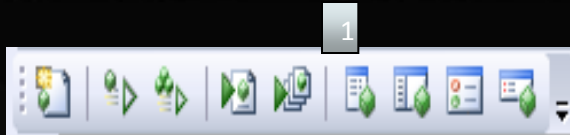


Numéros	Description
1	Case pour sélectionner les classes à tester
2	Sélection du projet de test lorsque l'on a plusieurs projets de test
3	Permet de configurer la façon dont seront nommées les méthodes, ...
4	Permet d'ajouter des assembly si celles-ci ne sont pas présentes par défaut dans la zone « Types »

# LES INTERFACES



Pour voir les tests :

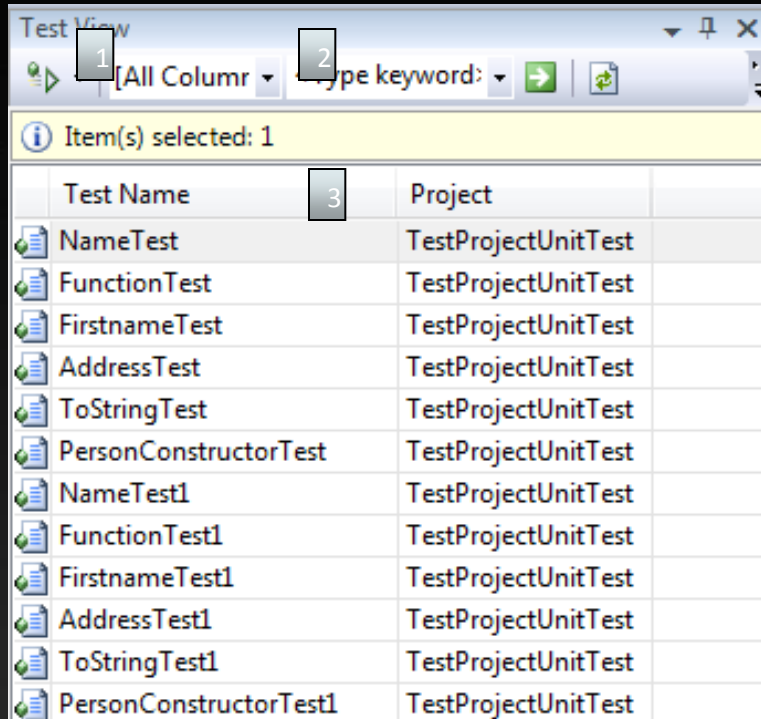


Pour voir les tests disponibles dans le projet on peut utiliser le visualisateur de tests. 1

# LES INTERFACES



Visualisateur des tests :



Numéros	Description
1	Permet de lancer les tests sélectionnés dans la liste
2	Permet de filtrer les tests affichés dans la liste
3	Affichage des tests

# LES INTERFACES



Le code d'un test:

```
/// <summary>
///A test for Name
///</summary>
[TestMethod()]
[Owner("arnaud lemettre")]
public void NameTest()
{
    Person target = new Person();
    string expected = "lemettre";
    string actual;
    target.Name = expected;
    actual = target.Name;
    Assert.AreEqual(expected, actual);
}
```

Attribut de méthode permet de  
Spécifier l'auteur du test entre autre

# LES INTERFACES

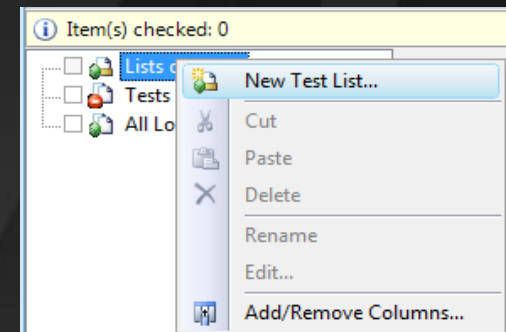
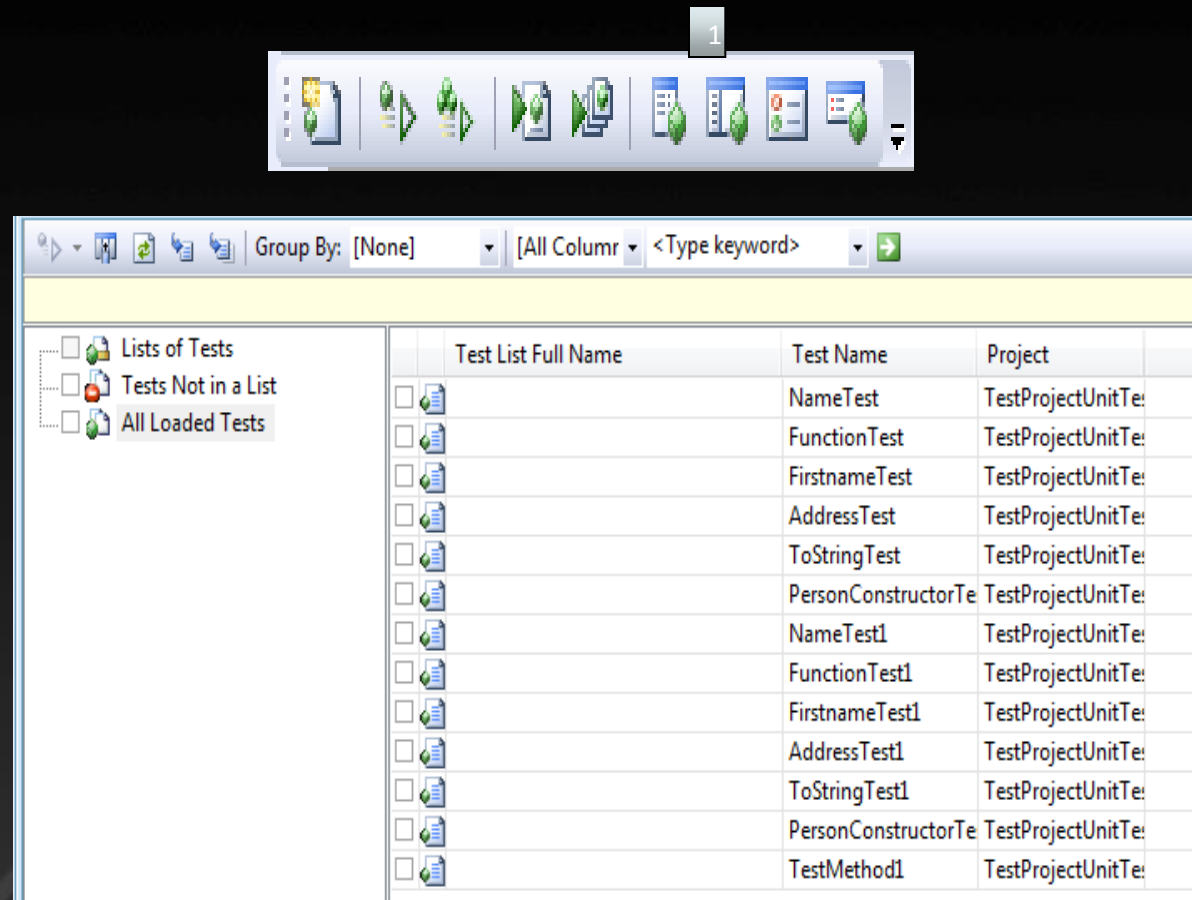


Le code qui permet de faire les tests est la classe Assert. On peut lui fournir différents tests :

Nom	Description
AreEqual	Permet de tester l'égalité des deux paramètres
AreNotEqual	Permet de tester la différence entre les deux paramètres
AreSame	Permet de tester l'égalité sur les objets
AreNotSame	Permet de tester la différence sur les objets
Fail	Fait échouer automatiquement les tests
Inconclusive	Permet d'indiquer qu'un test n'est pas encore implémenté
IsFalse	Vérifie que la condition passée en paramètre est fausse
IsTrue	Vérifie que la condition passée en paramètre est vrai
IsInstanceOfType	Vérifie que l'objet passé en paramètre est bien du bon type
IsNotInstanceOfType	Vérifie que l'objet passé en paramètre n'est pas du type passé en paramètre
IsNull	Vérifie que l'objet passé en paramètre est null
IsNotNull	Vérifie que l'objet passé en paramètre n'est pas null

# LES INTERFACES

Création d'une liste de tests :





# LES INTERFACES



Interface de Création d'une liste de tests :

A screenshot of a Windows-style dialog box titled 'Create New Test List'. It contains three main sections: a 'Name:' field with the text 'TestPerson', a 'Description:' text area with the text 'Permet de tester la classer Person', and a section titled 'Select where in test list hierarchy to place this list:' which contains a tree view with a single item 'Lists of Tests'. At the bottom are 'OK' and 'Cancel' buttons.

Create New Test List

Name:  
TestPerson

Description:  
Permet de tester la classer Person

Select where in test list hierarchy to place this list:

..... Lists of Tests

OK Cancel

Astuce : Pour créer une liste de tests on peut directement, sélectionner les tests dans l'interface, réaliser un clic droit dessus et faire « New Test List ... », les tests seront alors automatiquement ajoutés à la liste.

# LES INTERFACES



Exécution d'un test :

Item(s) checked: 1

	Test List Full Name	Test Name	Project
<input checked="" type="checkbox"/>	/Lists of Tests/TestPerson	NameTest	TestProjectUnitTe
<input type="checkbox"/>	/Lists of Tests/TestPerson	FunctionTest	TestProjectUnitTe
<input type="checkbox"/>	/Lists of Tests/TestPerson	FirstnameTest	TestProjectUnitTe
<input type="checkbox"/>	/Lists of Tests/TestPerson	AddressTest	TestProjectUnitTe

1

Group By: [None] [All Columr] <Type keyword>

Item(s) checked: 1

	Test List Full Name	Test Name	Proje
<input checked="" type="checkbox"/>	/Lists of Tests/TestPerson	NameTest	TestP

Résultat :

Test Results

arnaud@PC-DE-ARNAUD 2008-05 Run Debug

Group By: [None] [All Columr]

Test run completed Results: 1/1 passed; Item(s) checked: 0

Result	Test Name	Project	Error Message
<input checked="" type="checkbox"/> Passed	NameTest	TestProjectUnitTest	

# FAIRE PLUS DE TESTS



■ ■ ■  
Exécuter des tests c'est bien, mais ça prend du temps si on doit écrire chaque contexte (différentes valeurs, différents résultats ...):

Pour ça on peut utiliser les fichiers XML ou une base de données :  
Dans notre exemple on traitera avec un fichier XML :

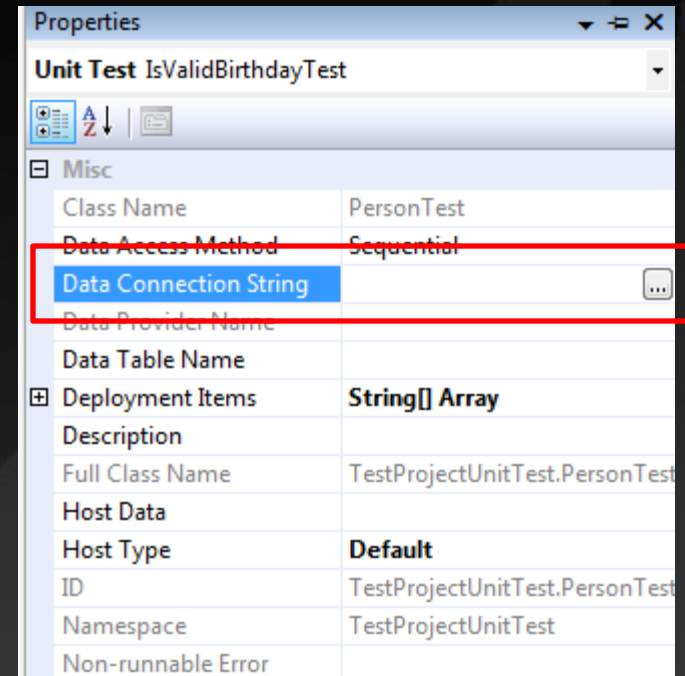
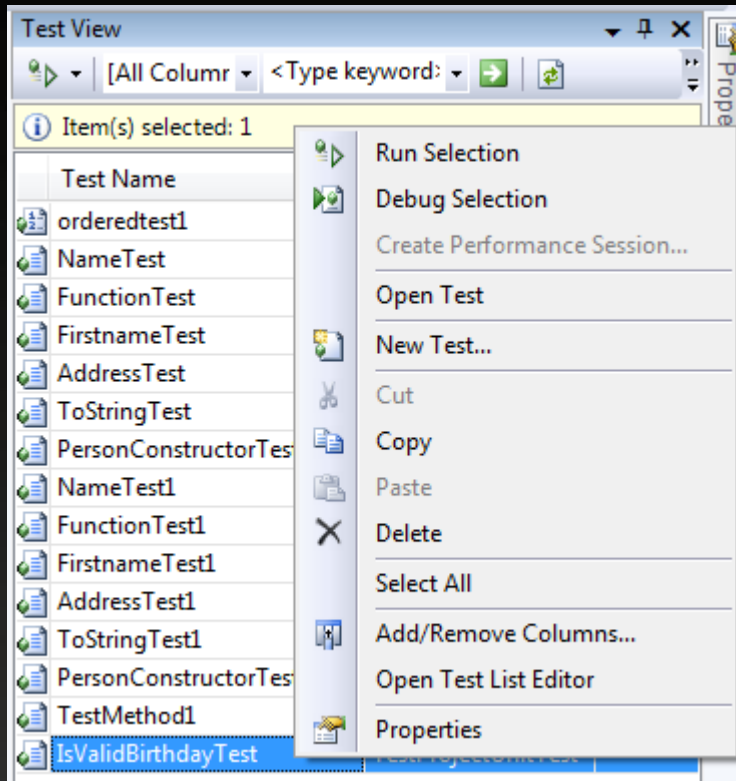
```
<?xml version="1.0" encoding="utf-8" ?>
<values>
  <person>
    <name>titi</name>
    <birthday>20/12/2005</birthday>
  </person>
  <person>
    <name>toto</name>
    <birthday>20/12/2009</birthday>
  </person>
</values>
```

# FAIRE PLUS DE TESTS



■ ■ ■

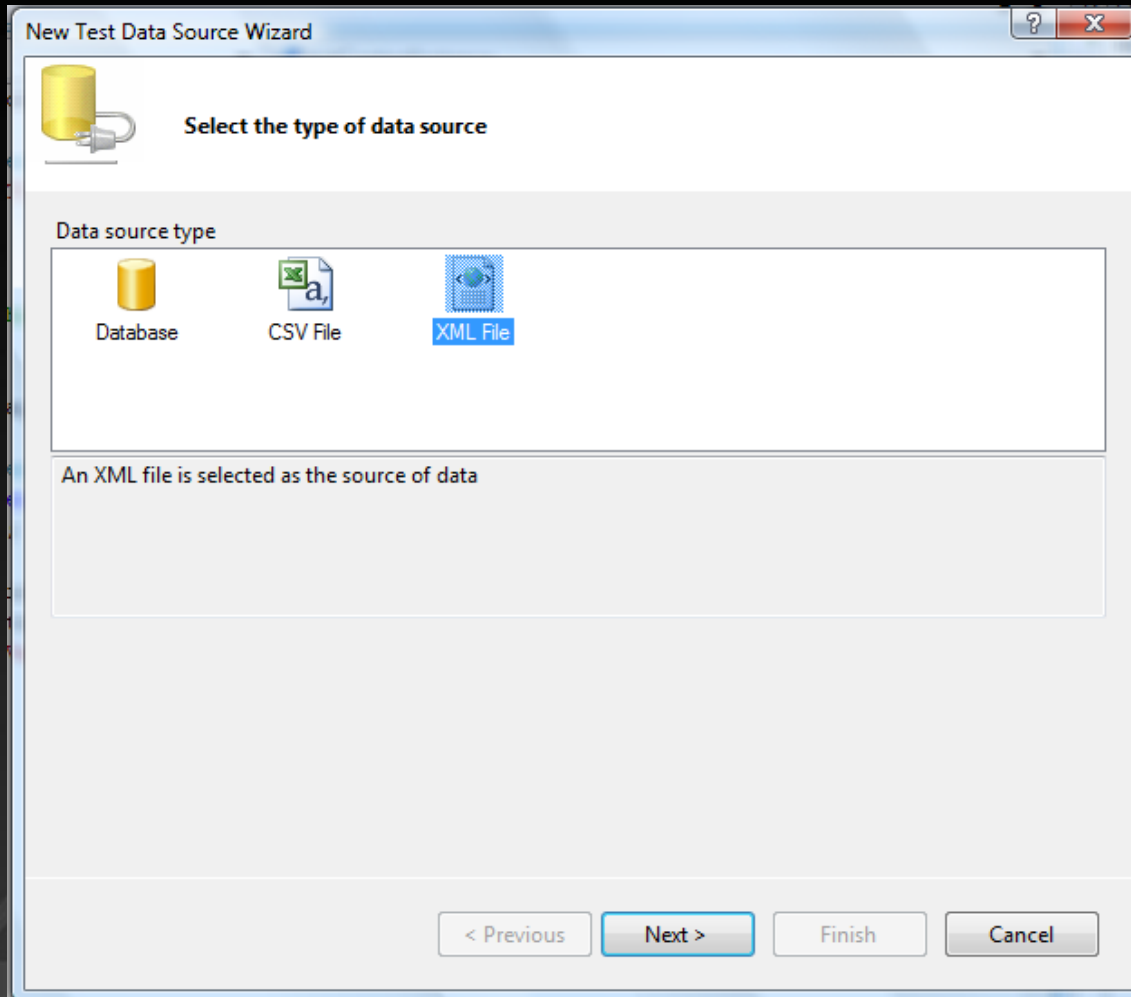
Il faut maintenant lier le fichier XML au test, en spécifiant :



# FAIRE PLUS DE TESTS



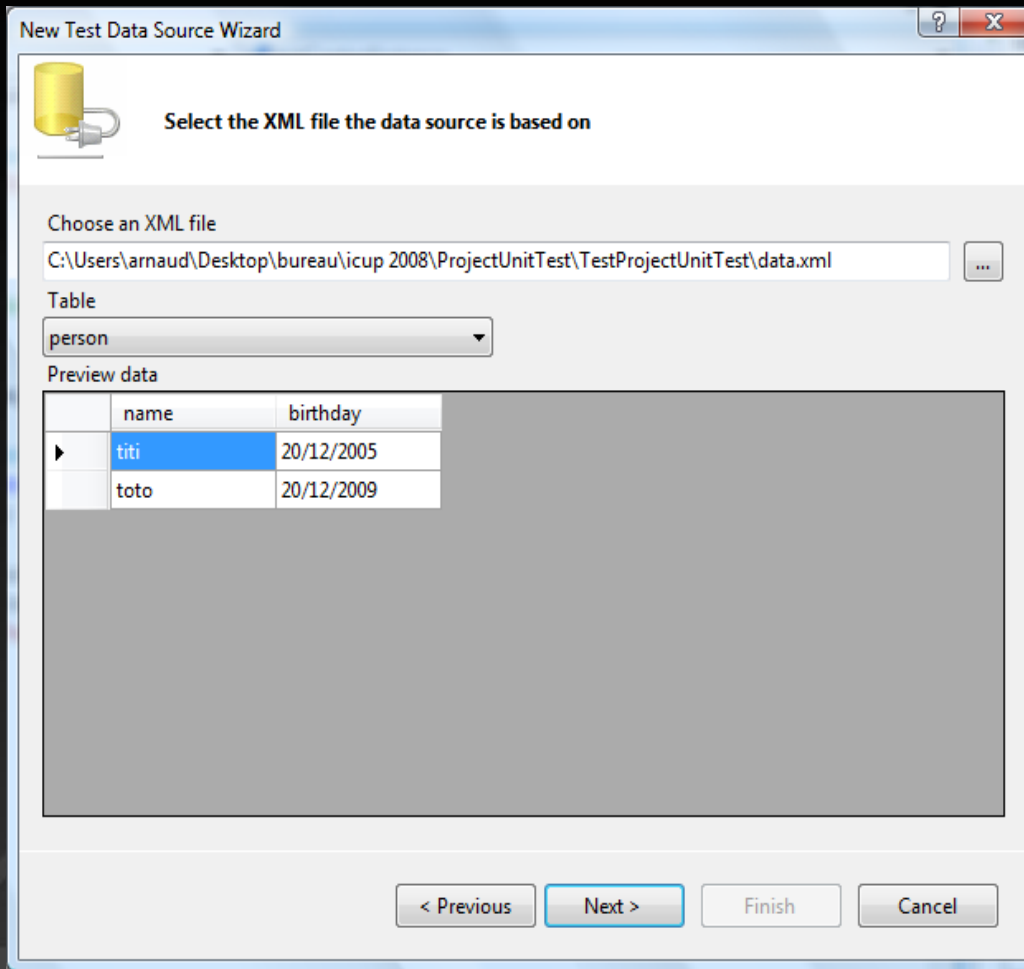
Il faut maintenant lier le fichier XML au test, en spécifiant :



# FAIRE PLUS DE TESTS



Il faut maintenant lier le fichier XML au test, en spécifiant :



The dialog box is titled "New Test Data Source Wizard". It contains a yellow folder icon and the text "Select the XML file the data source is based on". Below this, there is a section "Choose an XML file" with a text box containing the path "C:\Users\arnaud\Desktop\bureau\icup 2008\ProjectUnitTest\TestProjectUnitTest\data.xml" and a browse button "...". Below the text box is a "Table" dropdown menu showing "person". Below the dropdown is a "Preview data" section with a table:

	name	birthday
▶	titi	20/12/2005
	toto	20/12/2009

At the bottom of the dialog box are four buttons: "< Previous", "Next >", "Finish", and "Cancel".

L'assistant vient alors de rajouter des informations au dessus de la méthode de test. Une des options intéressantes est de mettre l'exécution des données de manière aléatoire ou séquentiel, en le spécifiant dans la propriété « Data Access Method ».

# FAIRE PLUS DE TESTS



...

Code à modifier dans la zone du test sur la valeur, result par exemple.

```
DateTime birthday =  
    Convert.ToDateTime (TestContext.DataRow["birthday"]);
```

# FAIRE PLUS DE TESTS



...

- Et dans vs 2010 ... :
  - Le système de test est beaucoup plus efficace, on peut directement coder le test qui générera le code métier associé (classes et méthodes)
    - Ctrl + alt + espace

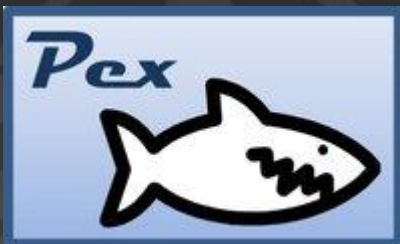


# FAIRE PLUS DE TESTS



...

- Faire des tests unitaires c'est bien, mais cela peut devenir vite compliqué.
- Pour nous aider nous avons 2 nouveaux outils : [lien pour télécharger](#)
  - Pex : Permet de d'explorer le code pour déterminer les tests utiles
  - Moles : Permet d'isoler les méthodes de leur environnement (fichier, base de données, ...) en les substituant par des délégués



# FAIRE PLUS DE TESTS

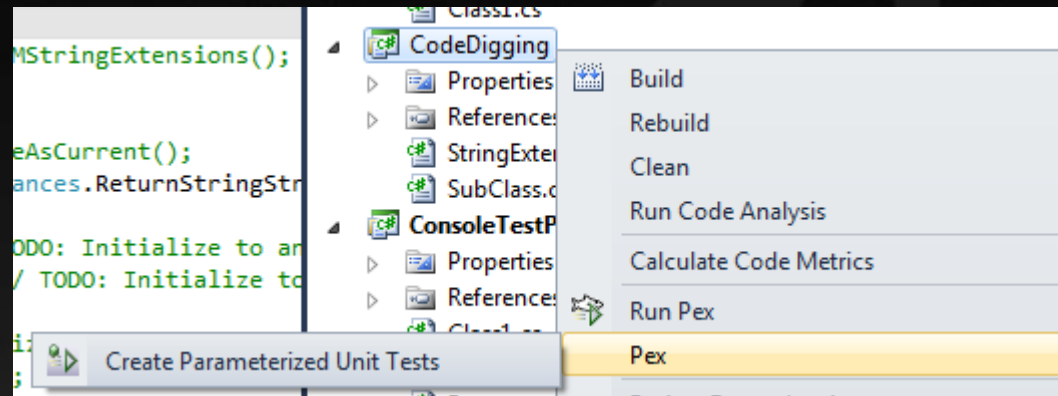


...

- Utilisation de Pex :

Soit Pex vous indique directement les résultats

soit vous pouvez  
générer directement  
des tests unitaires



Résultat :

Pex Exploration Results - stopped

StringExtensions.Capitalize(String)

11 1 17/17 Block, 0/0 asserts, 202 runs

Review bold issues: All Events 5 Uninstrumented Methods 1 Boundary 4 Imprecisions

	value	result	Summary/Exception	Error Message
1	null		<b>NullReferenceException</b>	Object reference not set to an instance of type.
2	""	""		
3	"\0"	""		
4	","	","		
5	"a"	"A"		
6	"a\0"	"A"		
7	"aa"	"Aa"		
8	"\0"	""		
9	","	","		
10	"a.\u8061\0"	"A_\u8061"		
11	"aaa"	"Aaa"		
12	"aaBaaaA:aa"	"AaBaaaA_..."		

Details:

```
[TestMethod]
[PexGeneratedBy(typeof(StringExtensionsTest))]
public void Capitalize870()
{
    string s;
    s = this.Capitalize("\0");
    Assert.AreEqual<string>("", s);
}
```

# FAIRE PLUS DE TESTS

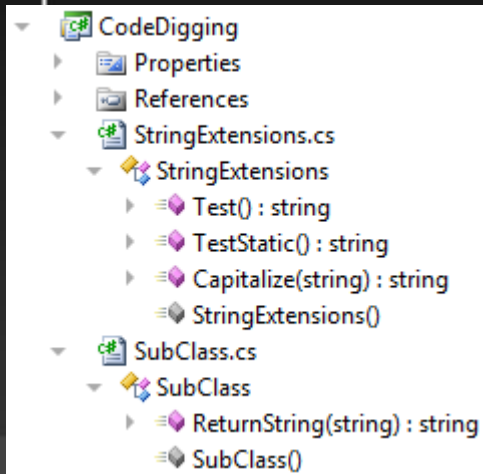


...

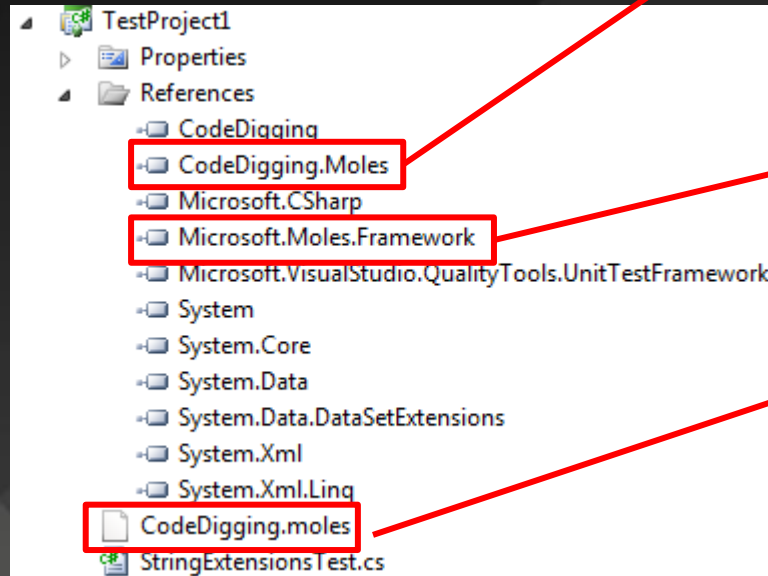
- Utilisation de Moles:

Moles s'utilise directement dans les tests unitaires, mais pour cela il faut d'abord déclarer des fichiers \*.moles

Soit le code suivant :



Dans le projet de test:



Référence rajoutée automatiquement par Moles

A rajouter pour utiliser le framework Moles

Fichier à rajouter à la main indiquant quel projet « moler »



# FAIRE PLUS DE TESTS



...

Les fichiers \*.moles dans notre exemple ressemblent à cela :

Nomenclature : NomDuProjet.moles

```
<Moles xmlns="http://schemas.microsoft.com/moles/2010/">  
  <Assembly Name="NomDuProjet" />  
</Moles>
```

Exemple :

```
<Moles xmlns="http://schemas.microsoft.com/moles/2010/">  
  <Assembly Name="CodeDigging" />  
</Moles>
```

# FAIRE PLUS DE TESTS



Extrait du code C# pour le test : . . .

```
public class StringExtensions
{
    public string Test()
    {
        SubClass sub = new SubClass();

        return sub.ReturnString("plop");
    }
}
```

```
public class SubClass
{
    public string ReturnString(string str)
    {
        return "return String";
    }
}
```

Dans notre cas pour SubClass on peut substituer la méthode par un delegate pour que celle ci renvoie toujours Good

```
[TestMethod()]
[HostType("Moles")]
public void TestSubClass()
{
    var stubSubClass = new CodeDigging.Moles.MSubClass();
    stubSubClass.ReturnString = (a) => "Good";

    SubClass subClass = stubSubClass.Instance;
    // Sans la class stub la méthode aurait renvoyée "return String"
    Assert.AreEqual(subClass.ReturnString("test1"), "Good");
}
```

Obligatoire pour moler une méthode

Lambda Expression  
cette syntaxe sera  
abordée dans les  
prochains cours



# FAIRE PLUS DE TESTS



...

Maintenant un peu plus complexe : Pour tester la méthode Test de la classe StringExtension, nous pourrions appliquer la même méthode que précédemment mais Imaginons que le résultat dépend beaucoup plus de la classe SubClass alors il faudrait l'exclure du test pour maîtriser l'environnement de test :

```
[TestMethod()]
[HostType("Moles")]
public void TestSubClass()
{
    CodeDigging.Moles.MSubClass.AllInstances.ReturnStringString = (b, a) => "Good";
    StringExtensions strClass = new StringExtensions();

    // Sans la class stub la méthode aurait renvoyée "return String"
    Assert.AreEqual(strClass.Test(), "Good");
}
```



# .NET REFLECTOR

# .NET REFLECTOR



- .Net Reflector est un logiciel permettant de desassembler les assembly .Net
- Une assembly est un .exe ou une dll compilée en MSIL

[Téléchargement](#)



# .NET REFLECTOR



- Cela permet donc d'avoir accès au code source.
- Bien sûr cela est d'autant plus facile que le MSIL est un langage intermédiaire.  
Cependant vous pouvez également avoir accès au Framework de cette façon.

# .NET REFLECTOR



- On peut également se protéger de ce genre d'outil. Pour cela on peut utiliser un logiciel d'obfuscation.
- Dans visual studio il existe une version gratuite : Dotfuscator Community Edition  
Cependant elle n'est pas très efficace.

Une version commerciale existe regroupant plus d'options.

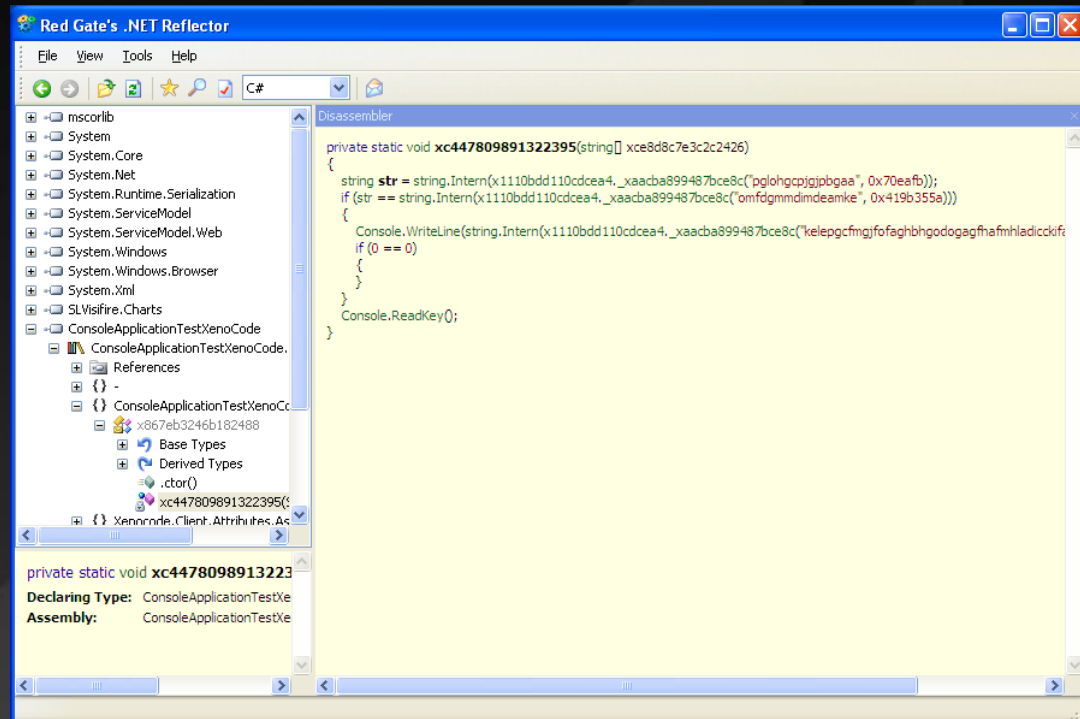
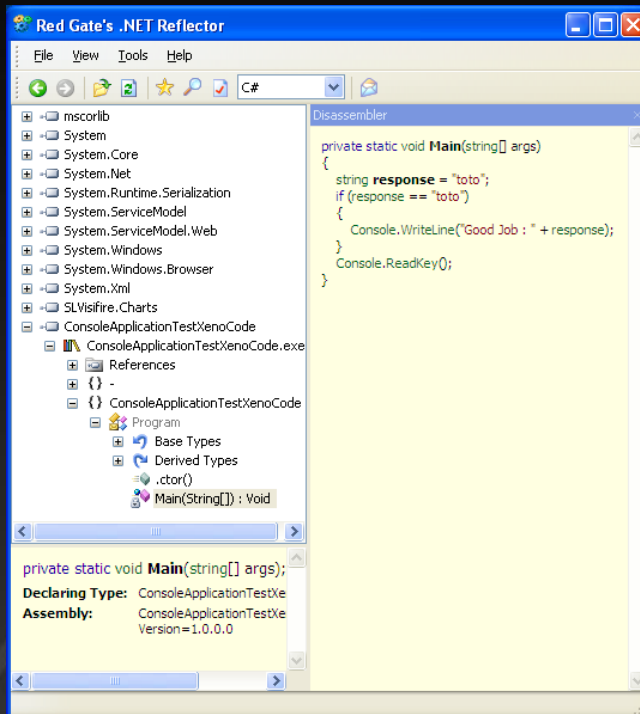
- Un autre outil appelé Xenocode est très efficace

# .NET REFLECTOR



Sans protection :

Avec protection :

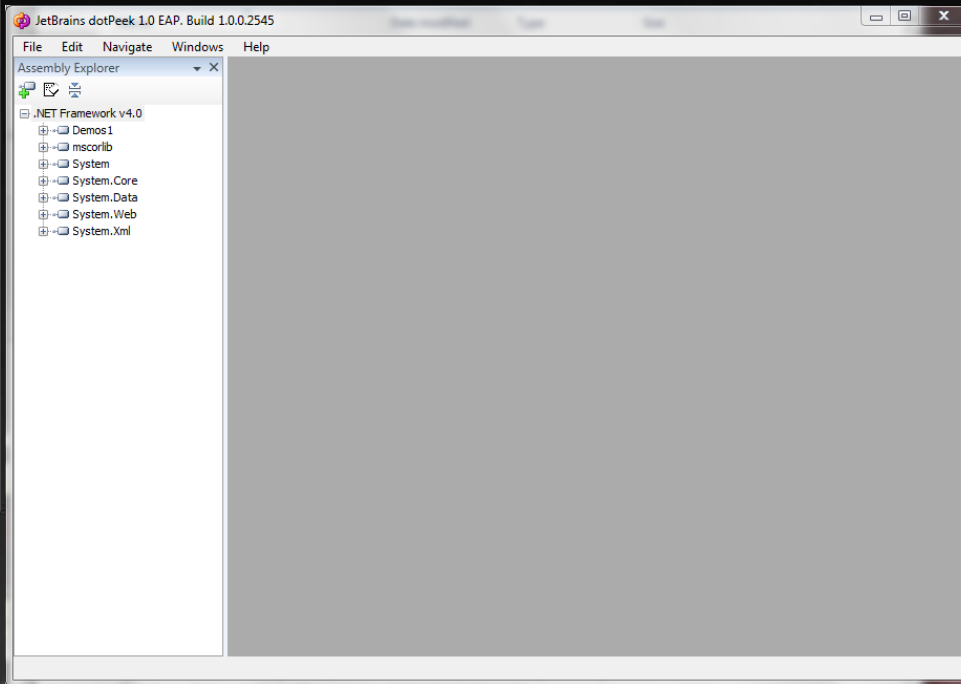


# .NET REFLECTOR



.Net reflector est devenu payant depuis quelques mois. Cependant un outil peut le remplacer gratuitement :

<http://www.jetbrains.com/decompiler/>





# PROFILING D'UNE APPLICATION

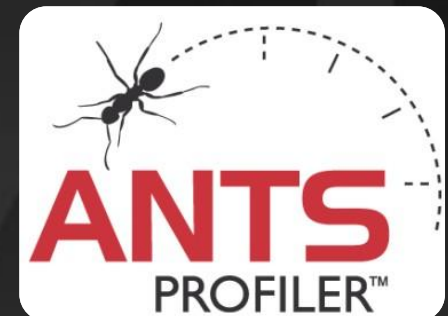
# PROFILING



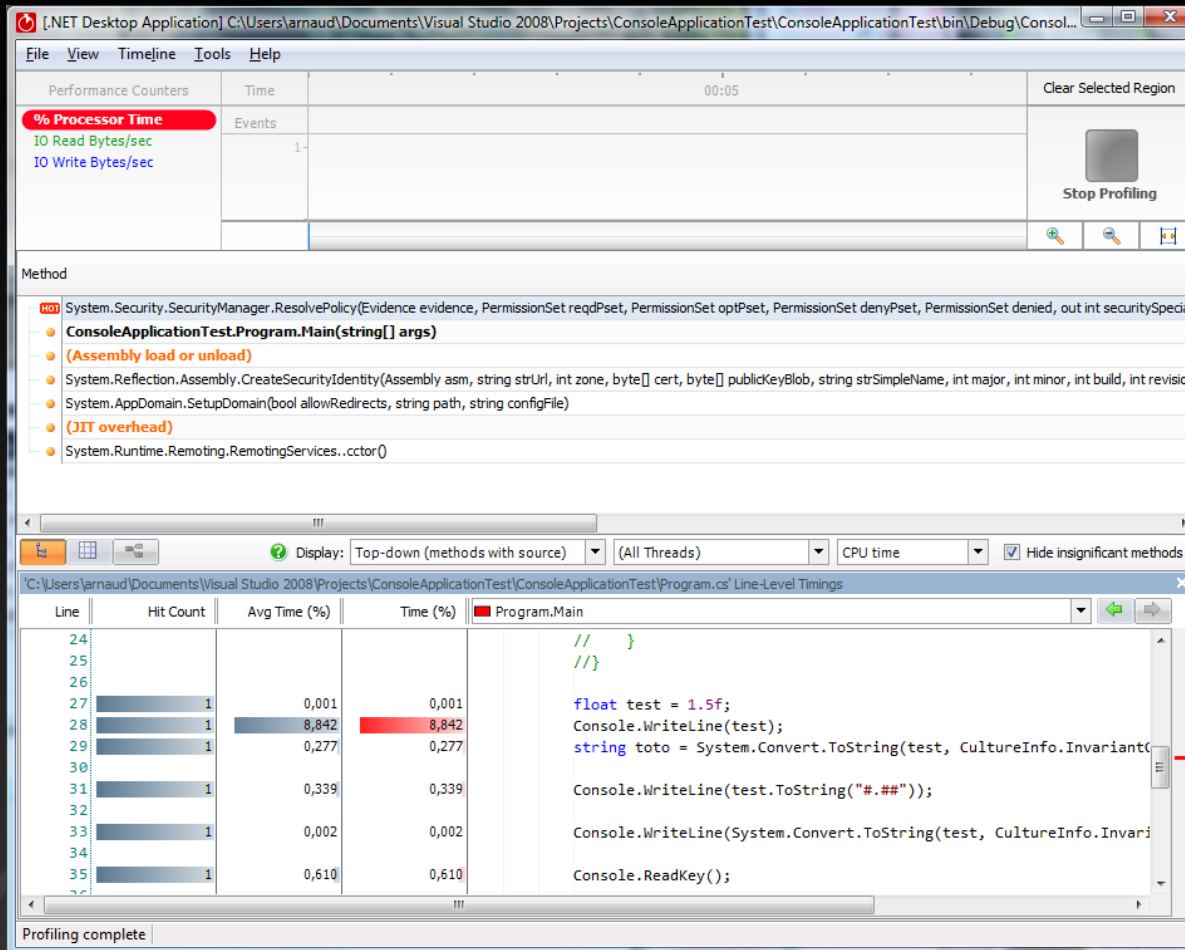
- Dans certains contextes les applications doivent répondre à des critères de performances. Pour nous aider dans cette tâche à tracker l'utilisation mémoire intensive, l'appel à des méthodes coûteuses en temps processeurs, il existe des outils soit incorporés dans visual studio ou des outils externes.

Permet de monitorer les performances d'une application. Et aussi les accès mémoire :

[Téléchargement](#)



# PROFILING



Time line

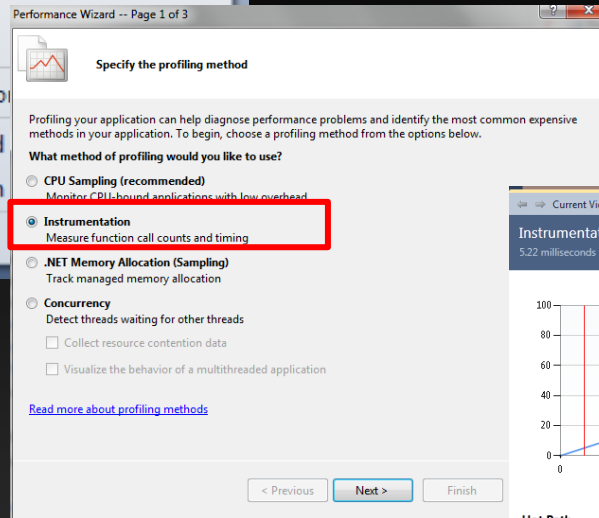
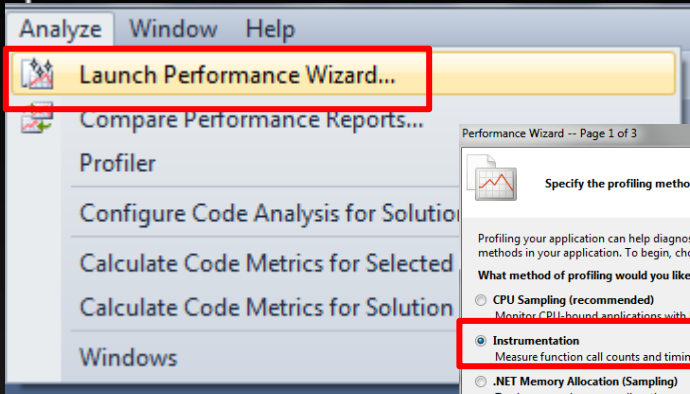
Mise en évidence  
Des lignes qui  
prennent du temps

CPU Temps

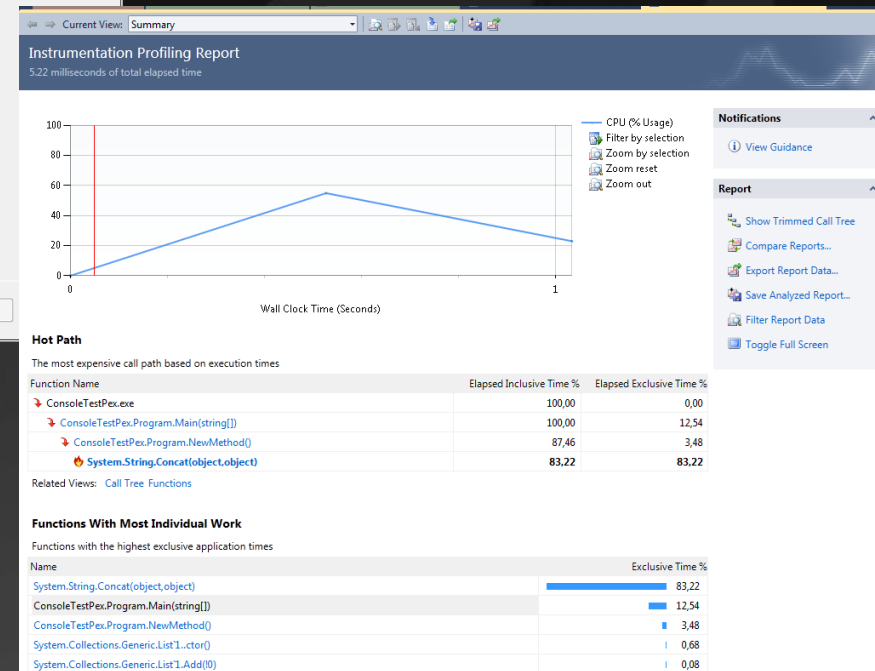
# PROFILING



- Avec visual studio (au dessus des versions pro) des outils permettent d'instrumenter les mesures de performance.



## Rapport de l'instrumentation :







# Fx COP

# Fx COP



- Permet de checker des règles de programmation. De base il y a celles de microsoft, mais on peut rajouter ses propres normes de programmation.
- [Téléchargement](#)

# Fx COP



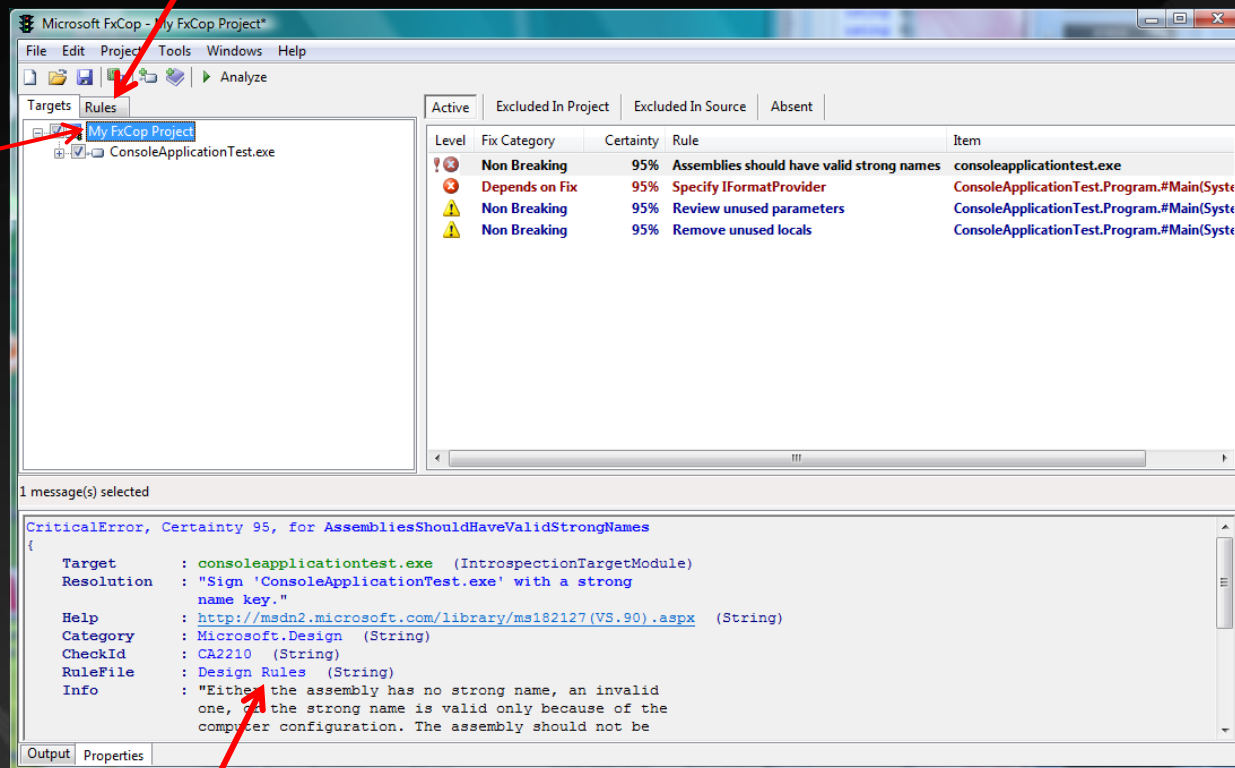
- Permet d'analyser des assemblies .Net aussi bien Dll qu'un projet.
- Pour l'utiliser il faut lancer le logiciel

# Fx COP



Permet de sélectionner les règles à appliquer

Add->target  
Permet d'ajouter  
l'assembly



Liste des  
« infractions »

Détails de l'infraction et la façon de résoudre le problème



# POUR ALLER PLUS LOIN

# POUR ALLER PLUS LOIN



- Des outils tel que Reflector utilisent la technologie de la réflexion. Ce chapitre sera l'occasion de voir ce type de programmation ainsi que les nouveaux concepts introduits dans .NET 4. et mot clé dynamique
- La réflexion est l'art de découvrir des types et d'invoquer leurs membres à l'exécution. La réflexion permet d'inspecter dynamiquement le contenu d'assemblages, d'en lire ses types, de créer des instances de ces types durant l'exécution du programme et d'appeler leurs méthodes ou champs dynamiquement.

# POUR ALLER PLUS LOIN



- Quelques méthodes d'exploitation de la réflexion :

```
//permet de charger l'assembly en mémoire
Assembly program = Assembly.LoadFile("test.dll");
//permet de récupérer un type de classe en particulier
Type typeConsole = program.GetType('ConsoleTest.Program');
//permet de récupérer une méthode en particulier
MethodInfo methodFacto = typeConsole.GetMethod("Facto", BindingFlags.NonPublic | BindingFlags.Static);
//passage de paramètre et appel à la fonction
object result = methodFacto.Invoke(typeConsole, new object[] { 3 });
```

Il existe bien entendu  
encore de nombreuses  
méthodes pour exploiter la  
réflexion

[http://msdn.microsoft.com/fr-fr/library/ms173183\(v=VS.100\).aspx](http://msdn.microsoft.com/fr-fr/library/ms173183(v=VS.100).aspx)

Paramètres d'appel

```
namespace ConsoleTest
{
    class Program
    {
        private static int Facto(int nb)
        {
            if (nb >= 13 || nb < 0)
            {
                return -1;
            }
            int res = 1;
            while (nb > 0)
            {
                res *= nb--;
            }
            return res;
        }

        static void Main(string[] args)
        {
        }
    }
}
```

# POUR ALLER PLUS LOIN



- La reflexion est très efficace, cependant depuis la version 4.0, Microsoft a introduit le mot clé `dynamic`. Ce mot clé permet de simplifier l'utilisation



Attention l'utilisation du mot clé `dynamic` est beaucoup plus couteux en ressource que l'utilisation de la réflexion. Il faut donc faire attention entre simplicité d'écriture et performance



# POUR ALLER PLUS



## LOIN

- Exemple d'utilisation :

```
static void Main(string[] args)
{
    //au travers de la reflexion
    TmpObject obj = new TmpObject();
    obj.GetType().InvokeMember("PropertyToSet",
        BindingFlags.Instance |
        BindingFlags.Public |
        BindingFlags.SetProperty,
        Type.DefaultBinder,
        obj,
        new object[] { "MyName" });

    //Avec le mot clé dynamic
    dynamic dyn = new TmpObject();
    dyn.PropertyToSet = "value";
}
```

```
public class TmpObject
{
    private string _propertyToSet;

    public string PropertyToSet
    {
        get { return _propertyToSet; }
        set { _propertyToSet = value; }
    }
}
```



# QUESTIONS ?