Un exemple de configuration d'une Solution Visual Studio d'un projet C#.

Contenu de ce document

Introduction 1
Outils nécessaires 1
Architecture de la Solution 1
Création de la Solution 3

Introduction

La qualité de la conception d'une application repose en grande partie sur un choix d'architecture adapté à la complexité final.

La structure de projet qui est présentée dans ce document est simple. Elle offre une séparation claire entre le code d'interface, le code métier (potentiellement utilisable dans d'autres environnement technique qu'une application cliente) et un support de tests unitaires. Cette structure peut évoluer si besoin est.

Le nom de cette application type est CryptIT. Le choix d'un nom pertinent et parlant qui remplacera ce nom type est la première chose à faire.

Outils nécessaires

- Visual Studio 2012 Professional, Premium ou Ultimate.
- NuGet (http://nuget.codeplex.org)
- GitExtension (http://code.google.com/p/gitextensions/)
- NUnit Test Adapter (http://nunit.org/index.php?p=vsTestAdapter&r=2.6)

Ces outils sont les outils standards des développements .Net à Intech'INFO mais également dans de nombreuses entreprises.

Architecture de la Solution

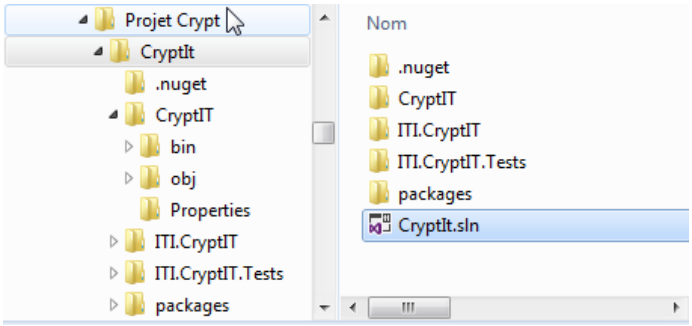
Une solution Visual Studio (**CryptIT.sln**) qui contient 3 projets :

- **ITI.CryptIT** : bibliothèque de classes (couche « métier »).
- **ITI.CryptIT.Tests** : bibliothèque de classes contenant les tests unitaires. Ce projet doit référencer nunit.framework.dll via NuGet (et, bien sûr, le projet **ITI.CryptIT**).
- **CryptIT** : Application Windows Forms. Référence le projet **ITI.CryptIT**.

Cette structure représente l'architecture minimaliste que nous vous conseillons d'appliquer pour tous vos développements.

La partie suivante détaille la création complète de la Solution.

Note : En termes de répertoires physiques, la structure reproduit celle de la Solution : le fichier *.sln* est à la racine, chaque projet contient son propre *.csproj*). Cette structure facilite la maintenance et l'organisation de projets (qui peuvent devenir complexes)[1].

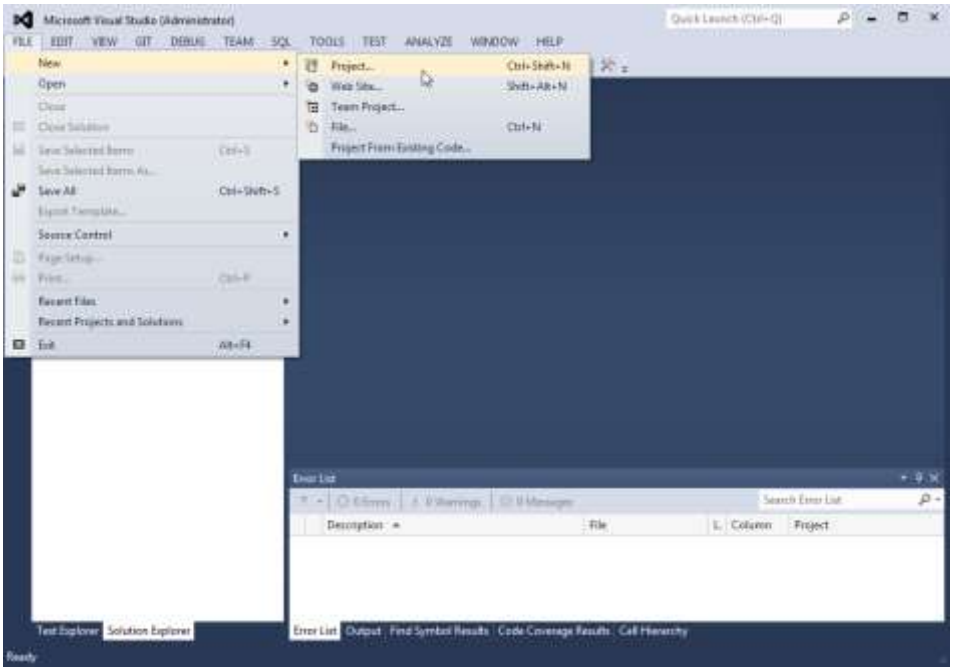


[1] Les Solutions (*.sln*) dans Visual Studio peuvent référencer des Projets (*.csproj*) divers n'importe où sur le disque dur. De même, on peut référencer depuis un Projet des fichiers en créant des « alias ».
Si dans certains cas (rares) cela peut aider, il vaut toujours mieux viser la simplicité d'une structure physique en cohérence avec la structure logique : en l'occurrence, une Solution contient des Projets et un Projet ne peut appartenir qu'à une et une seule Solution.

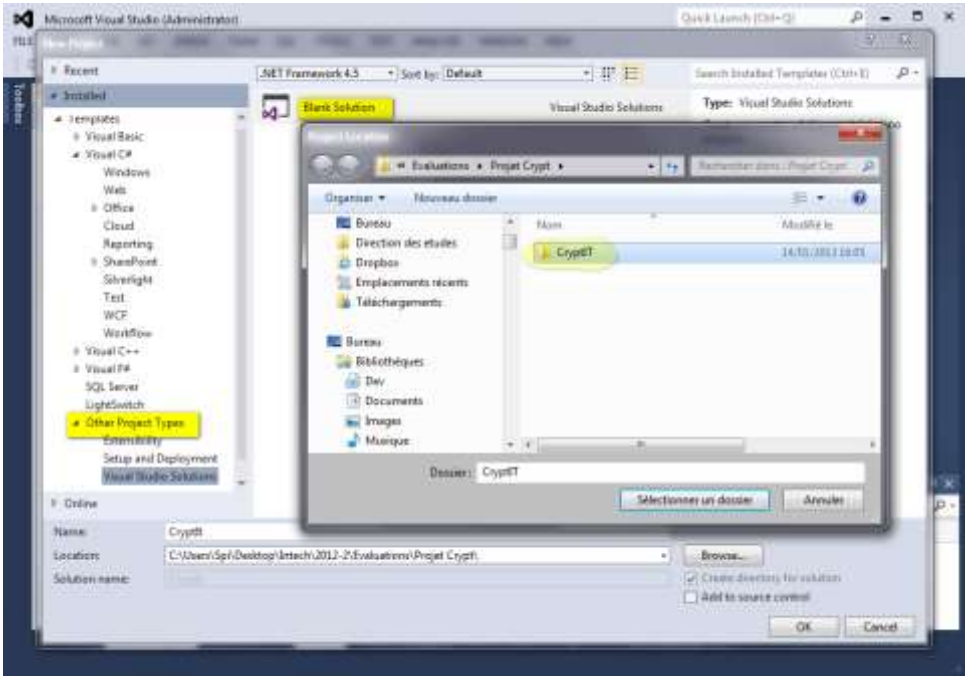
Création de la Solution

Note : Les outils listés en en page 1 doivent avoir été installés.

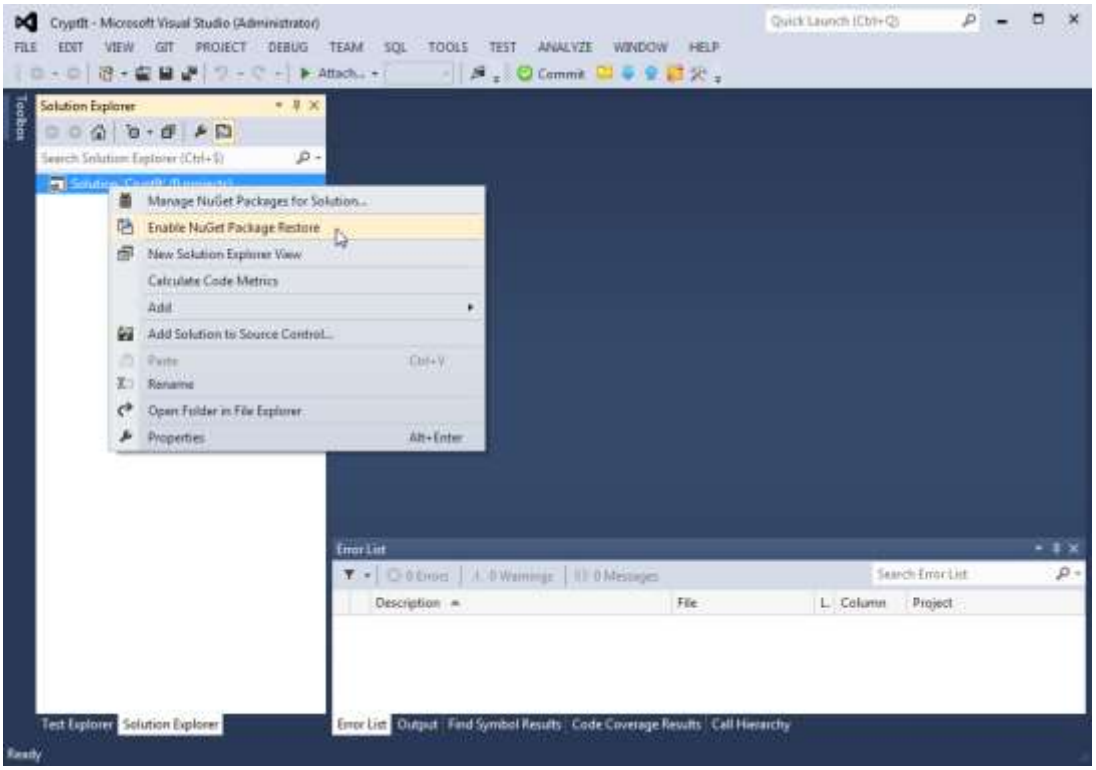
1 – Au lancement de Visual Studio, il faut créer une nouvelle Solution (*.sln*).



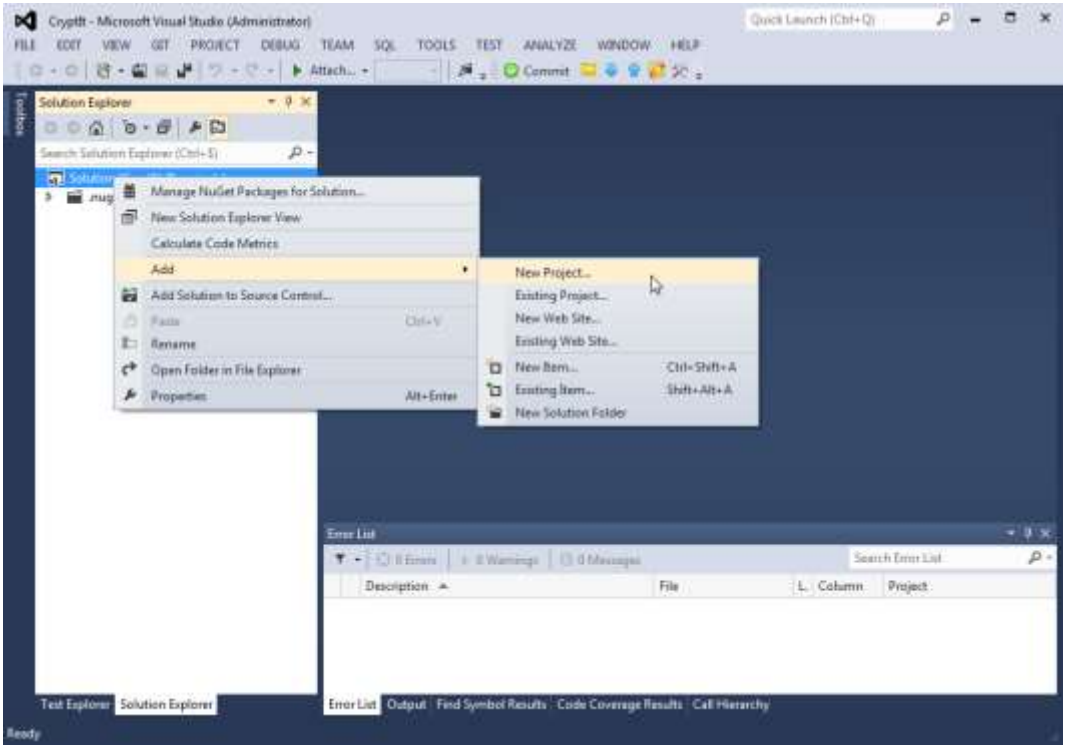
2 – On choisit de partir d'une Solution vide, puis de créer les différents Projets un à un à l'intérieur. Lors de cette étape, on choisit intelligemment l'emplacement et le nom de la Solution. Ici, on choisit de l'appeler CryptIT dans le répertoire du même nom.



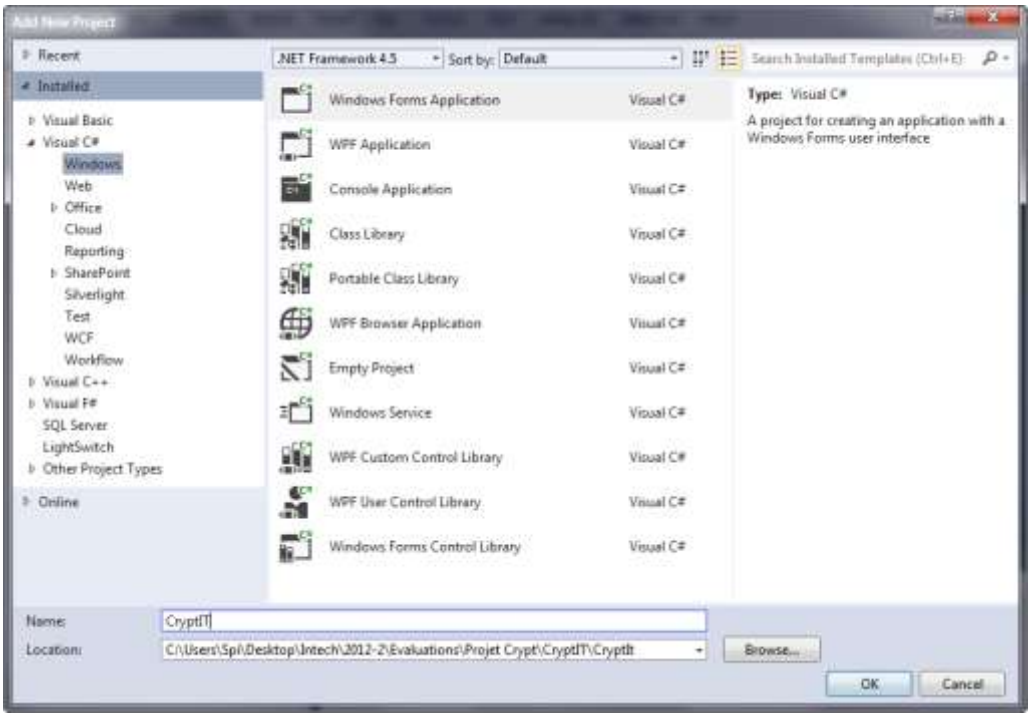
3 – NuGet ayant été préalablement installé, on configure la Solution pour se mettre automatiquement à jour si, d'aventure, de nouvelles versions de Packages que vous êtes amené à utiliser sont disponibles :



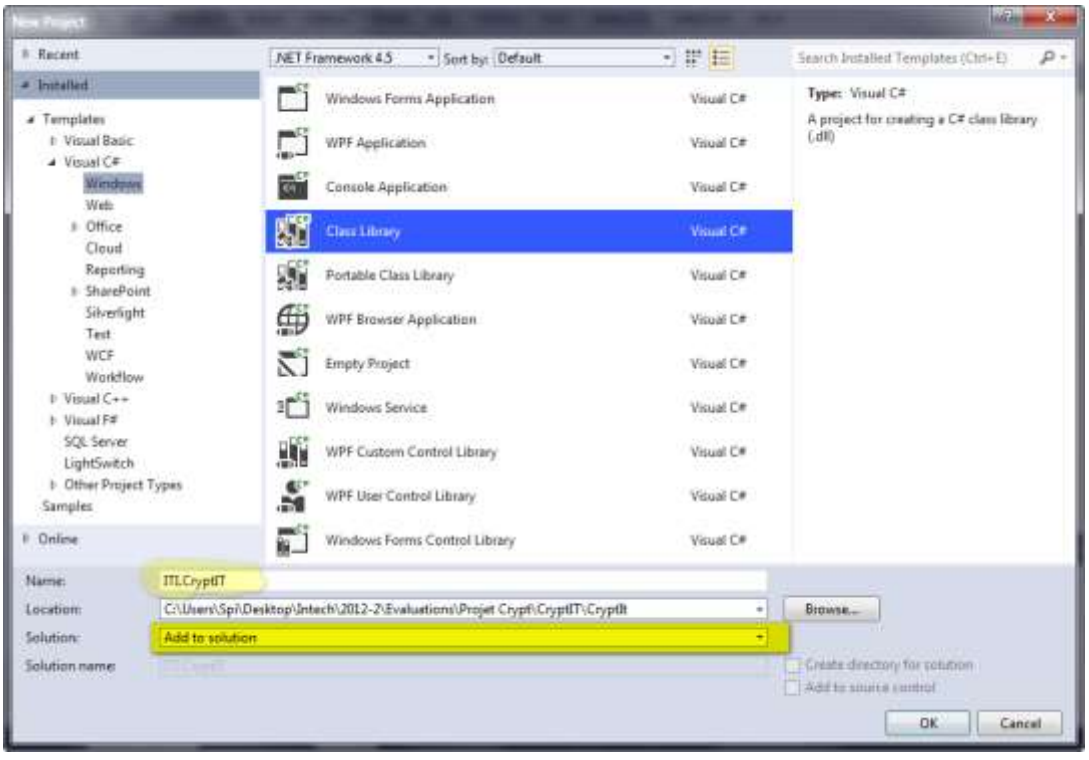
4 – On ajoute un premier Projet à la Solution : l'application.



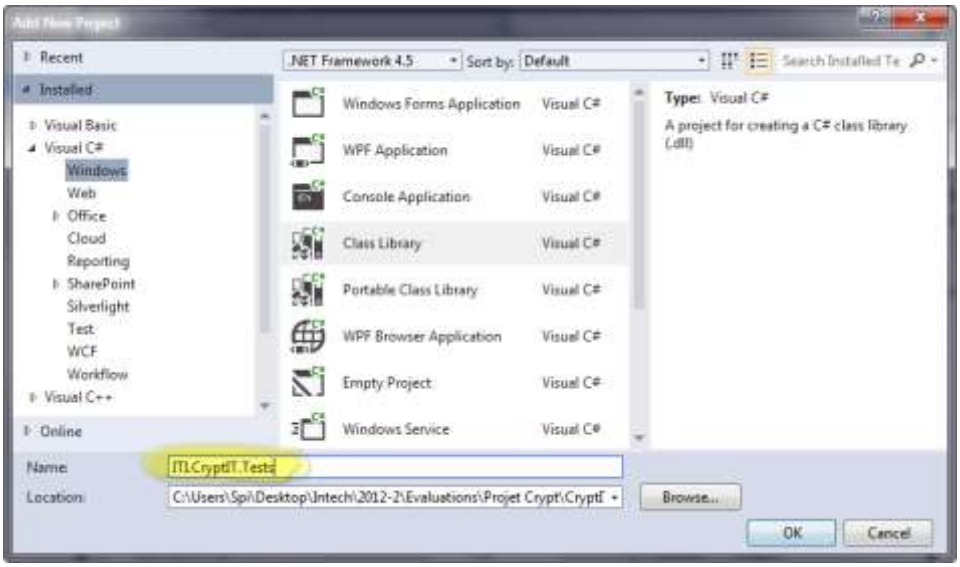
5 – On choisit une « Windows Forms Application » et on l'appelle, elle aussi, CryptIT (l'application compilée sera donc *CryptIT.exe*).



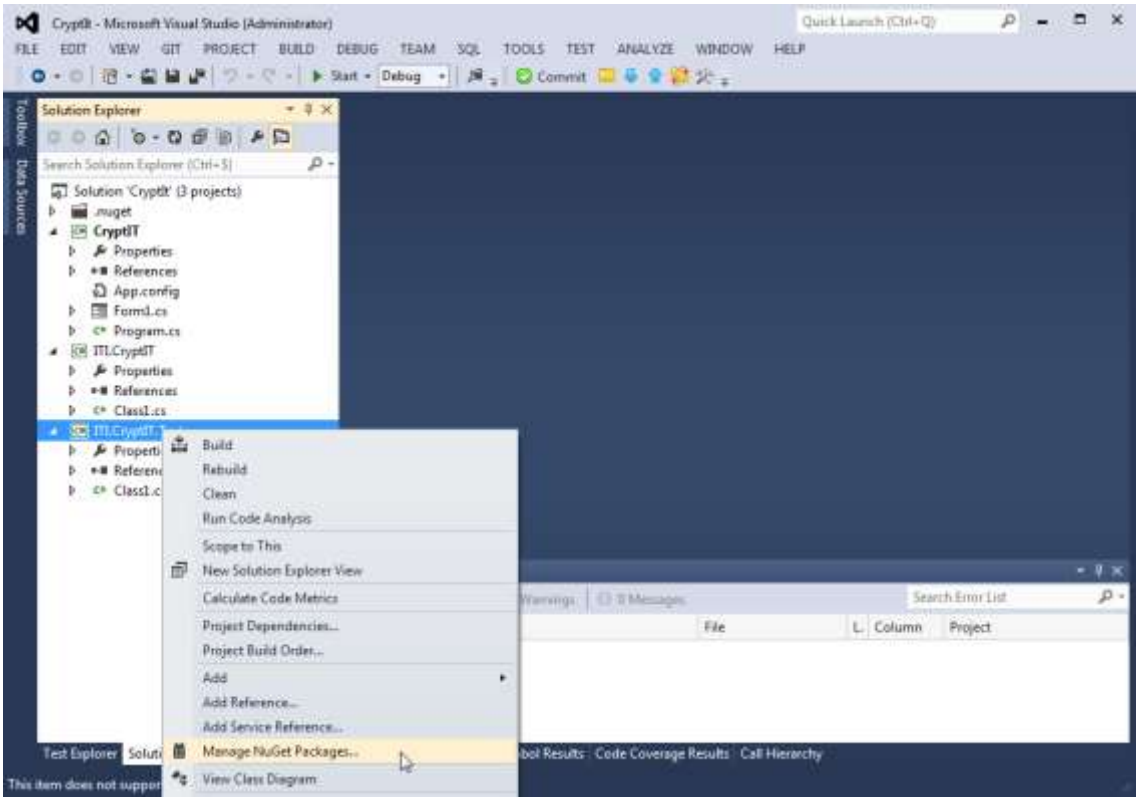
6 – On ajoute la dll qui contiendra le « vrai » code des fonctionnalités de l'application (cette dernière ne devant contenir que des aspects liés au GUI : fenêtres, menus, etc.). Il s'agit d'une « Class Library », une « Bibliothèque de Classe » en français, c'est l'unité de base d'architecture qui produira une dll : *ITI.CryptIT.dll*.



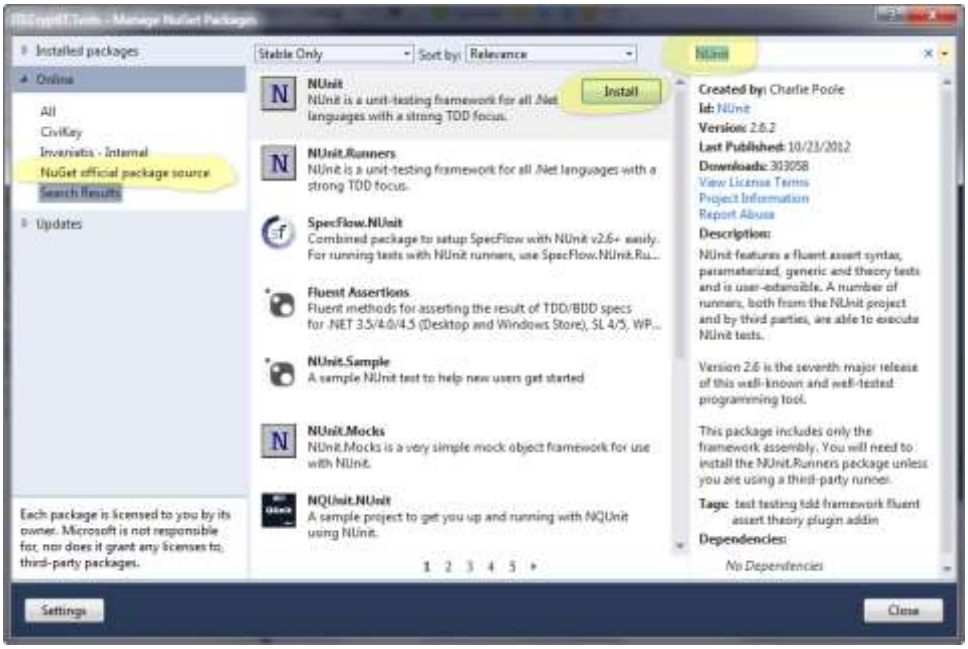
7 – On ajoute une dernière pièce : la dll qui contiendra les tests unitaires.



8 – Nous utilisons NUnit pour faire les tests unitaires. Notre projet *ITI.Crypt.Tests* a besoin de *nunit.framework.dll* qui contient un ensemble de classes, attributs, helpers qui permettent de créer les *fixtures*.
Nous utilisons l'outil NuGet (qui est un plugin de Visual Studio) pour aller chercher sur le site *nuget.org* le package qui contient cette dll.



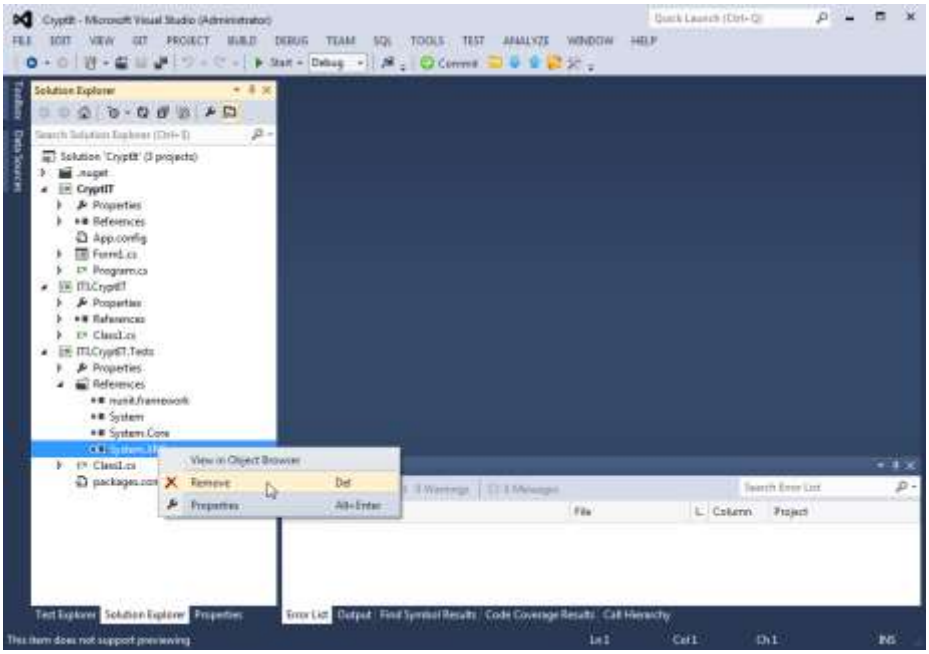
9 – Il existe des centaines de packages disponibles sur nuget. Utilisez le filtre pour chercher le package NUnit et installez-le.



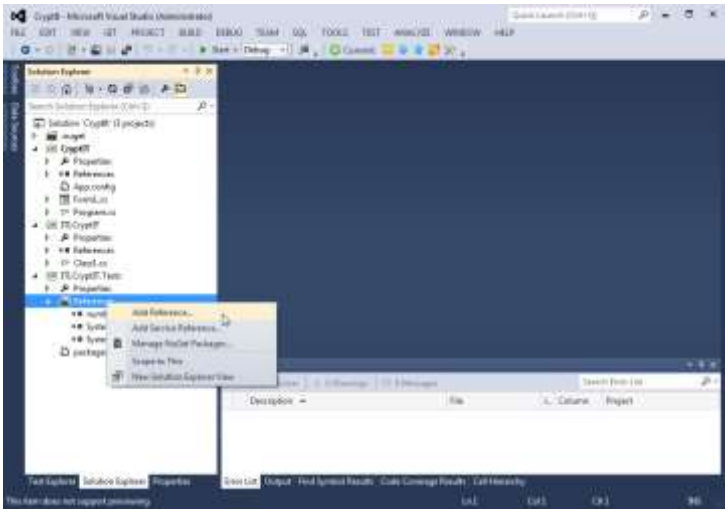
Note : NuGet peut être configuré pour aller chercher des Packages depuis de multiples sources. Sur l'écran ci-dessus, on peut voir qu'il existe d'autres « feeds » que le « NuGet official package source ».

10 – Comme vous l'apprendrez (certainement à vos dépends) dans les prochaines années, un des problèmes fondamentaux de l'ingénierie logicielle est de gérer les dépendances. Un bon réflexe à acquérir est de nettoyer ces projets de toutes les références à des composants logiciels inutiles.

Ici, notre dll de tests n'a besoin que de 3 références : *System*, *System.Core* (noyau d'exécution du framework .Net), et de *nunit.framework.dll*. On supprime donc tout ce qui est superflu : rappelez-vous qu'en développement logiciel, **ce qui est inutile est nuisible** !

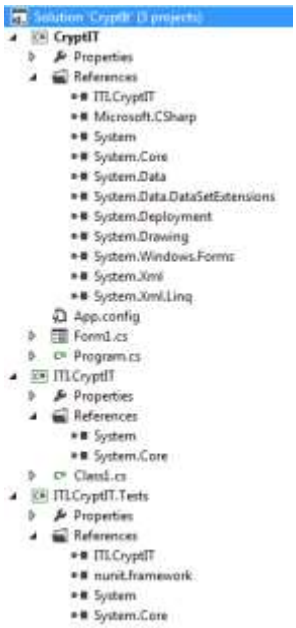


11 – Il s'agit maintenant de « tisser nos propres références » au sein de notre architecture, entre nos différents projets.



12 – On ajoute les références nécessaires (et uniquement nécessaires) entre nos projets…

13 – …pour aboutir à la configuration finale ci-dessous. On peut commencer à travailler.



Note : Je n'ai pas « nettoyé » l'application elle-même, celle-ci conserve toutes les références créées par défaut par le template de projet « Windows Application Forms ».