Atelier 3 : Interagir avec l'équipe

Introduction

Vous avez à présent réalisé les ateliers 1 et 2 du coach MSDN dédié aux tests fonctionnels. Dans le premier atelier, vous avez endossé le rôle du chef de projet ou du responsable fonctionnel afin de créer et organiser un plan de test (création de suite, récupération de requirements, création de cas de tests…). Dans le second, vous avez pris la casquette de testeur fonctionnel afin d'exécuter une campagne de test.

Vous avez appris comment utiliser **Microsoft Test and Lab Manager** (« MTLM » par la suite), l'outil qui vous est dédié dans la gamme de produit **Visual Studio 2010**, afin de réaliser des tests fonctionnels sur vos applications, de suivre l'évolution de ces tests et de remonter des bugs à l'équipe de développement via **Team Foundation Server 2010**.

Dans ce dernier atelier, vous allez endosser plusieurs rôles, alternativement. Le premier sera bien entendu celui du testeur fonctionnel utilisant **MTLM**, le second sera celui du développeur utilisant **Visual Studio 2010**. L'enjeu de cette partie et de découvrir le rôle central de **Team Foundation Server** et de ces deux outils dans l'interaction entre les équipes de testeurs et les équipes de développeurs.

Il s'agira ici de répondre à plusieurs questions :

- Comment le développeur va pouvoir utiliser le travail du testeur afin d'améliorer la qualité de son code ?
- Comment développeurs et testeurs vont collaborer afin d'éliminer les bugs dans une application ?
- Comment le testeur va pouvoir valider les corrections apportées par un développeur ?

Ce chapitre est sans doute le plus important du coach puisqu'il met en évidence un problème qui a longtemps été rencontré dans les projets de développement : le manque de communication entres les équipes de test et les équipes de développement.

Ce problème est dû à plusieurs facteurs, le premier étant le manque d'outils mis à disposition des équipes pour communiquer entre elles. Le second facteur est un problème évoqué dans l'atelier 2 : souvent le développeur est incapable de reproduire un bug, parce qu'il n'a pas connaissance de toutes les actions effectuées par le testeur avant de produire ce bug. Les outils qui sont aujourd'hui proposés par Microsoft dans la gamme Visual Studio 2010 apportent une solution concrète à ces problématiques.

Afin de préciser à quel rôle s'applique chaque partie de l'atelier, vous retrouverez les pictogrammes suivants :

 L'explication se base sur le rôle du testeur fonctionnel et de son outil : **MTLM**

 L'explication se base sur le rôle du développeur et de son outil : **Visual Studio 2010**

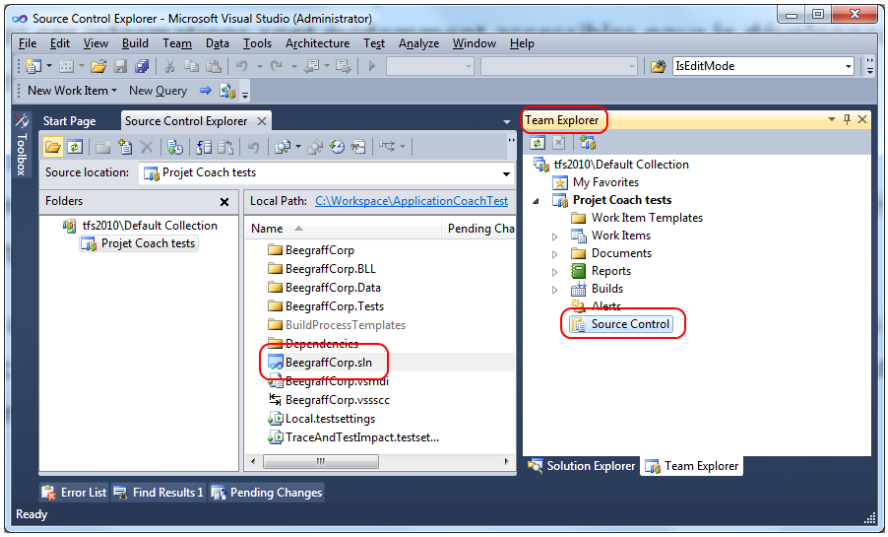
Le travail du testeur au service du développeur

Dans l'atelier précédent, vous avez exécuté plusieurs tests fonctionnels, enregistré des actions, remontés des bugs…Toutes ces informations sont évidemment accessibles pour le développeur et ce dernier va même les utiliser pour améliorer la qualité de son code au quotidien.

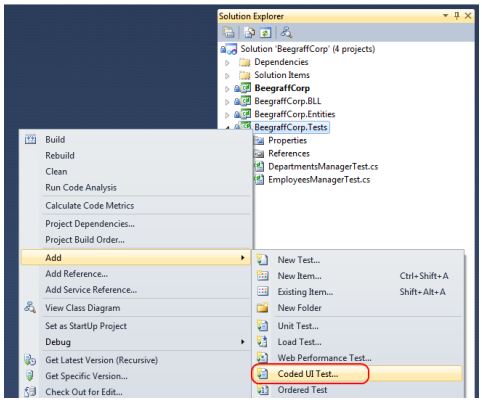
Tests d'interfaces graphiques automatisés

En effet, un développeur va pouvoir créer un test d'interface graphique automatisé (Coded UI Test), à partir d'un enregistrement d'actions réalisées lors d'un test fonctionnel : il n'a pas besoin de réenregistrer le scénario lui-même.

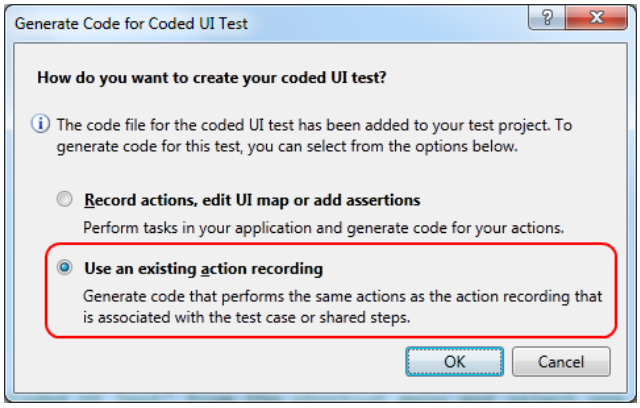
Pour faire cela, ouvrez **Visual Studio 2010** et connectez-vous au projet d'équipe via le **Team Explorer**. Une fois connecté, rendez-vous dans le **Source Control** et double cliquez sur la solution (fichier BeegraffCorp.sln) pour l'ouvrir :



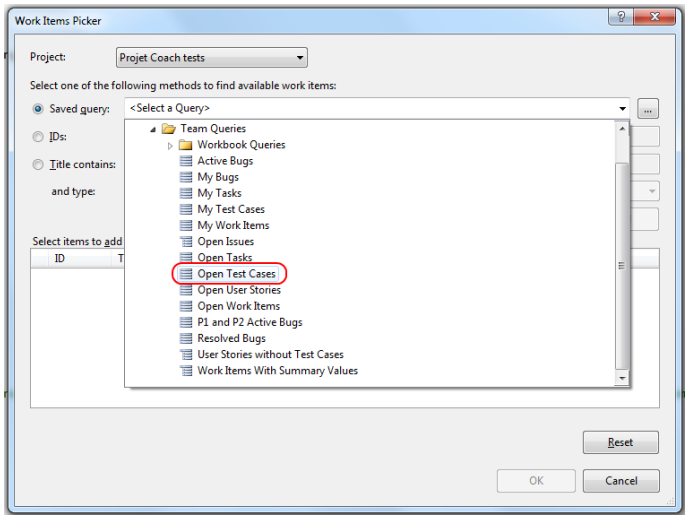
Cette solution contient un projet de tests (BeegraffCorp.Tests) contenant des tests unitaires sur lesquels nous ne nous attarderons pas ici. Faîtes un clic droit sur ce projet et cliquez sur l'entrée de menu **Coded UI Test…** :



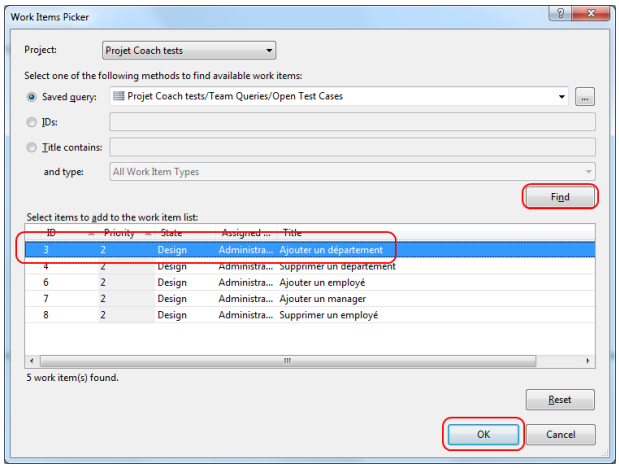
Une boîte de dialogue apparaît et vous propose deux solutions pour créer un test d'interface graphique automatisé. La première consiste à lancer un outil qui permettra d'enregistrer vos actions sur une application afin de les rejouer par la suite (comme le **Test Runner** de **MTLM** en somme), alors que la seconde vous propose d'utiliser un enregistrement d'actions déjà existant. Choisissez la deuxième proposition. En effet, le testeur a déjà réalisé un ensemble de tests fonctionnels sur l'application et a enregistré ses actions, inutile de refaire son travail :



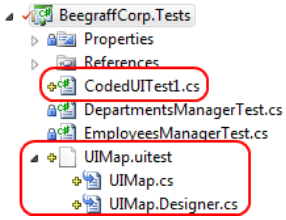
Cliquez sur le bouton **OK** pour valider. Dans la fenêtre qui s'ouvre, vous allez devoir sélectionner un cas de test sur lequel Visual Studio devra se baser pour générer le test d'interface graphique. Pour cela, déroulez la liste **Saved Queries** et choisissez l'élément **Open Test Cases** :



Une fois la requête sélectionnée, cliquez sur le bouton **Find**. Vous voyez alors apparaître dans la grille la liste de tous les tests fonctionnels qui ont été créés dans l'atelier 1 de ce coach. Sélectionnez le test « Ajouter un département » puis cliquez sur **OK** :



Visual Studio 2010 va alors générer un ensemble de fichiers de code qui vont permettre d'exécuter automatiquement le test d'interface graphique :



Le fichier **CodedUITest1.cs** est le point d'entrée du test d'interface graphique. C'est ce dernier qui contient l'appel vers les différentes étapes du test :

```
[DataSource(
    "Microsoft.VisualStudio.TestTools.DataSource.TestCase",
    "http://tfs2010:8080/tfs/default collection;Projet Coach tests",
    "3",
    DataAccessMethod.Sequential),
TestMethod]
public void CodedUITestMethod1()
{
    this.UIMap.Démarrerlapplication();
    this.UIMap.DanslongletDepartmentscliquezsurleboutonNew();
    this.UIMap.EntrerunnomdedépartementdanslazonedetexteetappuyersurlatoucheTABdepartmentParams.ItemEditText =
        TestContext.DataRow["department"].ToString();
    this.UIMap.EntrerunnomdedépartementdanslazonedetexteetappuyersurlatoucheTABdepartment();
    this.UIMap.CliquersurleboutonSave();
    this.UIMap.Fermerlapplication();
}
```

La première chose qu'il est intéressant de remarquer est l'attribut **DataSource** placé au-dessus de la méthode **CodedUITestMethod1**. En effet, cette nouveauté de **Visual Studio 2010** permet au test d'aller rechercher les paramètres directement dans la définition du cas de test dans **Team Foundation Server**. Ainsi, dans ce cas, il exécutera 3 fois le test de manière séquentielle avec les paramètres défini dans

l'atelier 1 pour le nom du département (Development, IT, Sales). L'avantage de cette nouvelle source de données est que si le testeur fonctionnel rajoute des valeurs dans la définition du cas de test (pour tester des caractères spéciaux, valeurs trop longues etc…) celles-ci seront automatiquement prises en compte par le test d'interface graphique automatisé.

La seconde chose à remarquer dans l'extrait de code ci-dessus est que Visual Studio a bien généré une méthode par étape présente dans la définition du cas de test. Celles-ci sont définies dans la classe **UIMap** et plus précisément dans le fichier **UIMap.designer.cs**. Elles sont tout simplement en charge de retrouver les contrôles au sein d'une application (un bouton, par exemple) et d'y appliquer des actions (« Cliquer », par exemple).

```
/// <summary>
/// DanslongletDepartmentscliquezsurleboutonNew - Test Case 3
/// </summary>
public void DanslongletDepartmentscliquezsurleboutonNew()
{
    #region Variable Declarations
    WpfButton newButton = this.BeegraffCorporationMWindow.TabControlTabList.DepartmentsTabPage.NewButton;
    #endregion

    // Click 'New' button
    Mouse.Click(newButton, new Point(27, 17));
}
```

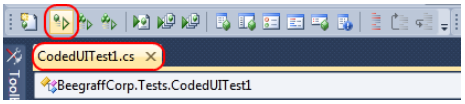
Dans l'exemple ci-dessus, on récupère le bouton « New » de l'onglet « Departments » et on simule un clic dessus. L'arbre des contrôles est maintenu par le fichier **UIMap.uitest** (XML).

*NB : Le **Point** passé en paramètre de la méthode **Click** dans l'exemple ci-dessus ne représente pas les coordonnées du bouton sur la fenêtre de l'application, mais bien les coordonnées du point où le clic doit être simulé sur le bouton. Sans quoi le déplacement d'un contrôle dans l'interface invaliderait le test, ce qui n'est pas le cas !*

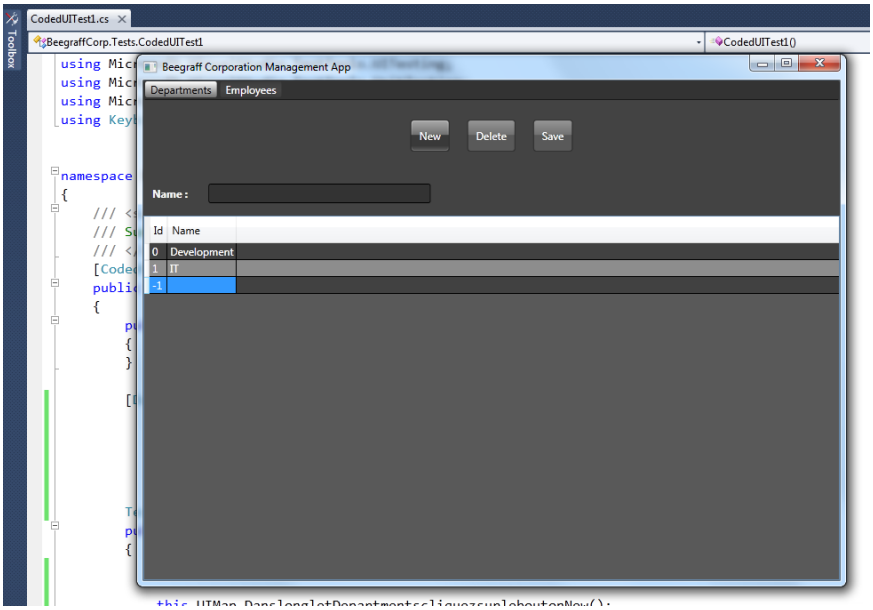
*NB 2 : ouvrez le fichier **UIMap.uitest** dans **Visual Studio** est assurez-vous que le chemin de l'application à tester est bon (notamment si vous êtes sur une autre machine que celle utilisée pour l'exécution des tests fonctionnels dans l'atelier précédent) :*

```
<LaunchApplicationAction>
  <ParameterName />
  <FileName>C:\Users\Julien CORIOLAND\Desktop\Beegraff Corp\BeegraffCorp.exe</FileName>
  <AlternateFileName>%USERPROFILE%\Desktop\Beegraff Corp\BeegraffCorp.exe</AlternateFileName>
</LaunchApplicationAction>
```

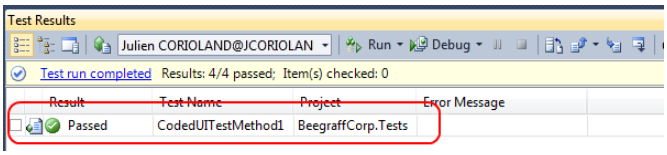
Vous pouvez à présent rejouer le test en vous plaçant dans le fichier CodedUITest1.cs et en cliquant sur le bouton  dans la barre d'outils de tests de Visual Studio 2010 :



Dès lors, vous verrez que l'application se lance 3 fois (correspondant aux trois itérations du cas de test) et ajoute sans votre intervention les départements suscités :



La fenêtre de résultats de tests de Visual Studio 2010 vous permet de vous assurer que le test est bien passé :



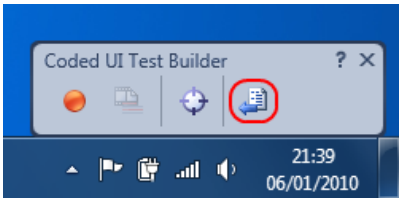
Faites un check-in pour envoyer le code du **Coded UI Test** généré ci-dessus dans le contrôleur de code source. Au préalable, assurez-vous que le chemin vers l'exécutable à lancer référencé dans le fichier UIMap.uitest est bien un répertoire partagé sur le serveur de compilation. Si ce n'est pas le cas (comme ci-dessus par exemple) modifiez le fichier XML :

```
<LaunchApplicationAction>
  <ParameterName />
  <FileName>\\tfs2010\Tests\BeegraffCorp\BeegraffCorp.exe</FileName>
  <AlternateFileName>\\tfs2010\Tests\BeegraffCorp\BeegraffCorp.exe</AlternateFileName>
</LaunchApplicationAction>
```

Note : bien entendu, rendez l'application disponible sur ce chemin réseau, vous en aurez besoin dans la dernière partie de cet atelier.

Vous allez devoir demander à Visual Studio de re-générer le code du **Coded UI Test à partir du fichier XML d'extension .uitest** afin que ce nouveau chemin soit pris en compte. Pour cela, rendez-vous dans le fichier **CodedUITest1.cs**. Placez-vous sur la méthode de test CodedUITestMethod1 et faites un clic droit, puis choisissez le menu **Generate Code for Coded UI Test** puis **Use Coded UI Test Builder…**

Visual Studio se réduit alors pour laisser place au **Coded UI Test Builder**. Cliquez sur le bouton  pour demander la génération du code :

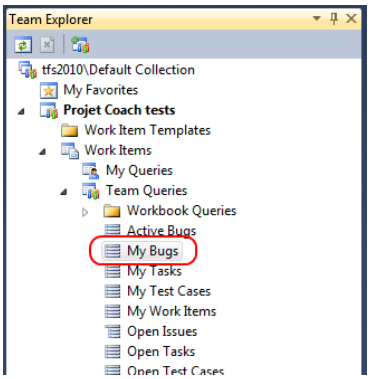


Validez en cliquant sur le bouton **Generate** dans la fenêtre qui s'affiche. Une fois l'opération terminée, fermez le **Coded UI Test Builder** et relancez l'exécution du **Coded UI Test** afin d'être sûr que celui-ci fonctionne.

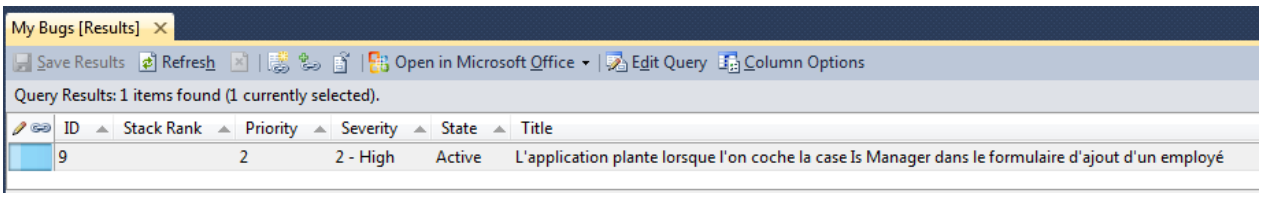
Résolution d'un bug remonté par un testeur fonctionnel

Dans l'atelier précédent vous avez remonté un bug rencontré lors d'un test fonctionnel. Dans cet atelier, en tant que développeur, vous allez accéder aux informations de ce bug via Team Foundation Server et de ce fait, vous allez pouvoir le corriger.

Pour commencer, rendez-vous dans le **Team Explorer**, dépliez le nœud **Work Items** et double cliquez sur la requêtes **My Bugs** :

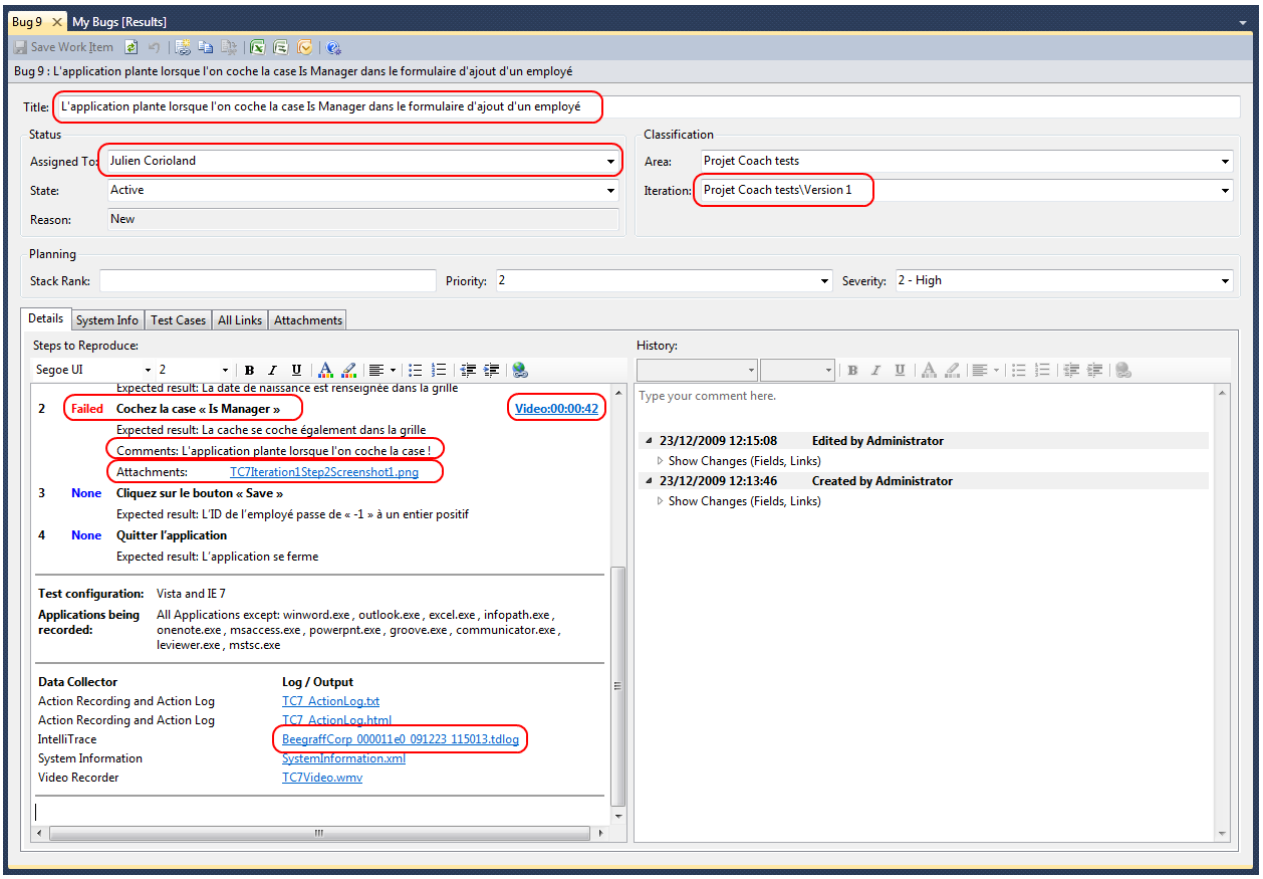


Dans la fenêtre de résultat de la requête, vous devriez voir apparaître le bug « L'application plante lorsque l'on coche la case Is Manager dans le formulaire d'ajout d'un employé » :



*NB : si aucun bug ne s'affiche, vérifiez que vous êtes bien connecté sur **Team Foundation Server** avec le compte de la personne à qui vous avez assigné le bug dans l'atelier 2. Le cas échant reconnectez-vous avec la bonne personne ou ouvrez la requête **Active Bugs** via le **Team Explorer**.*

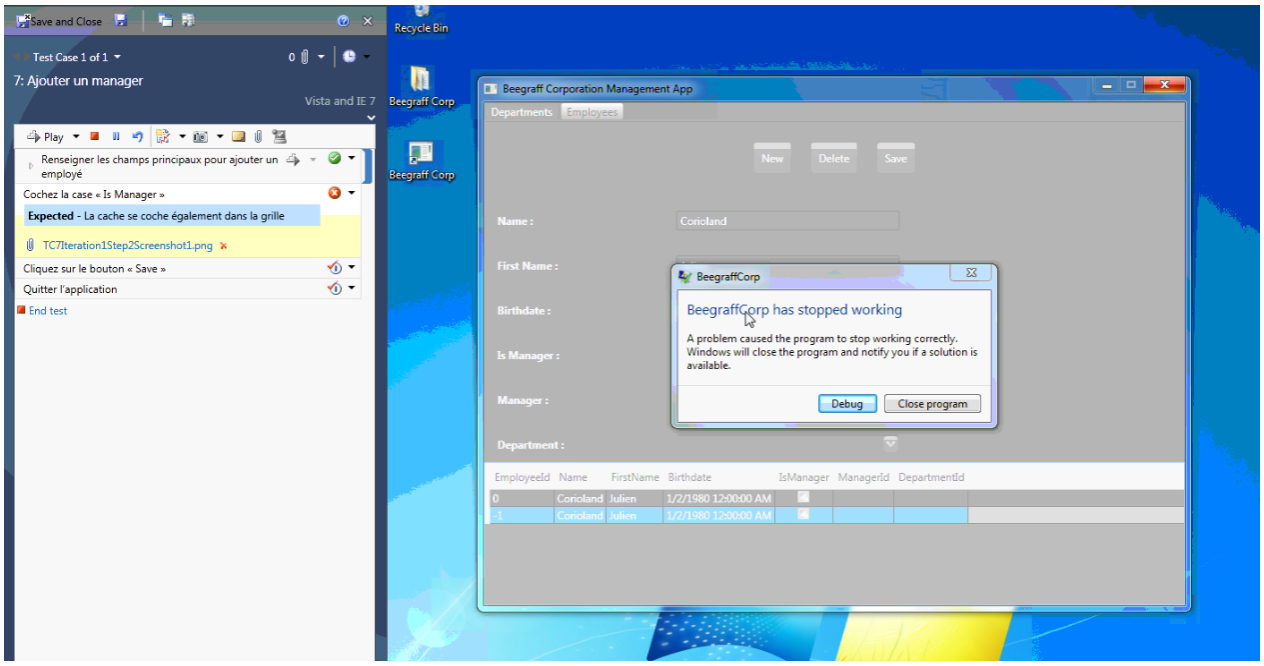
Double cliquez sur le bug afin d'avoir accès à toutes les informations :



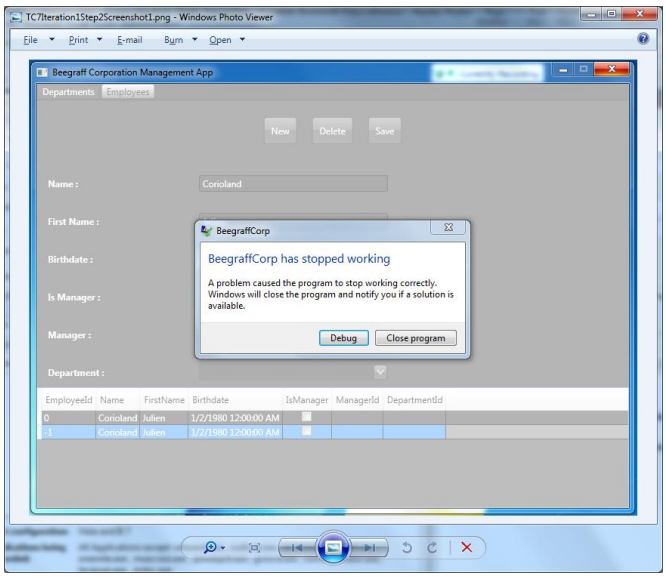
Comme vous pouvez le constater, toutes les informations entrées par le testeur fonctionnel sont disponibles :

- Vidéos
- Etapes réussies, étapes échouées
- Commentaire et capture d'écran sur l'étape échouée
- IntelliTrace™
- Etc…

Vous pouvez visualiser la vidéo à l'endroit où l'application a planté en cliquant sur le lien :



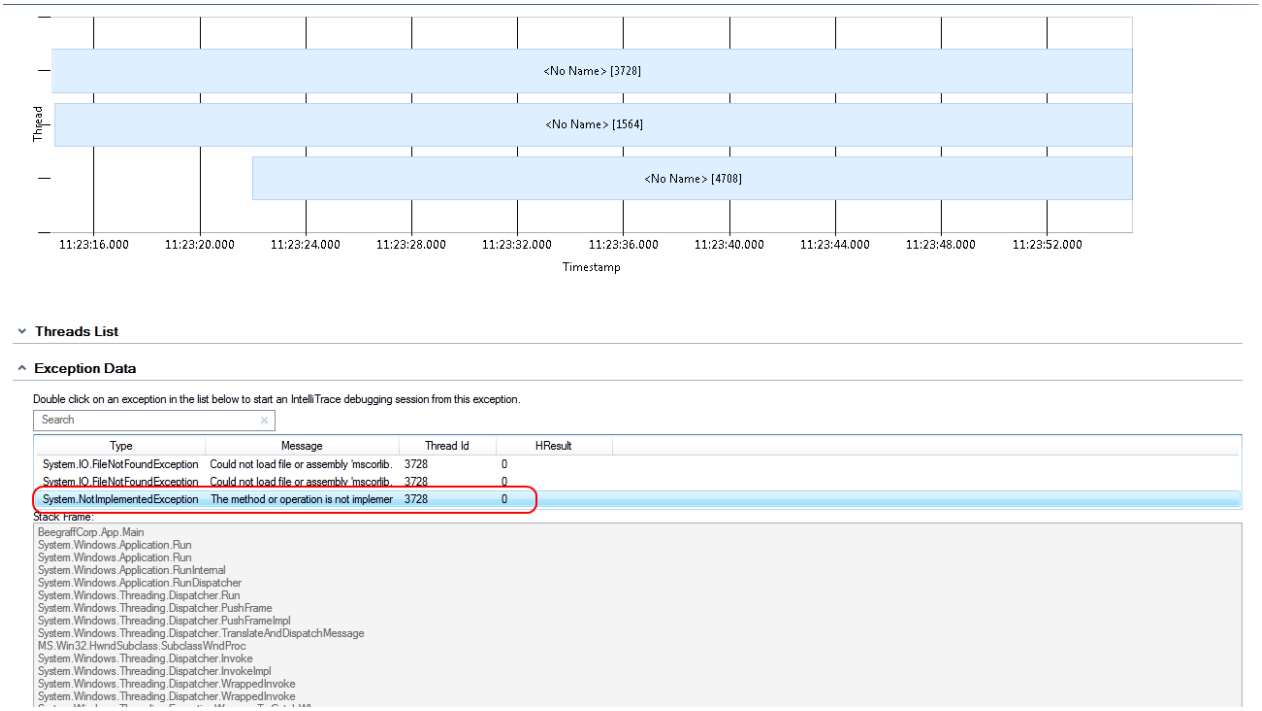
Vous pouvez également récupérer la capture prise par le testeur :



Enfin, l'IntelliTrace™ vous permet de vous mettre en condition de debug dans Visual Studio, mais au moment où l'application a planté. Pour cela, commencez par cliquer sur le lien vers le fichier .tdlog dans le rapport de bug :

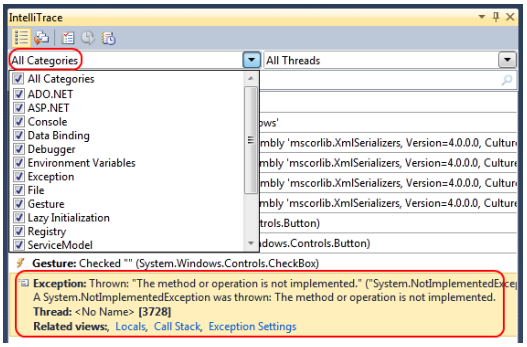
Data Collector	Log / Output
Action Recording and Action Log	TC7_ActionLog.txt
Action Recording and Action Log	TC7_ActionLog.html
IntelliTrace	BeegraffCorp_000011e0_091223_115013.tdlog
System Information	SystemInformation.xml
Video Recorder	TC7Video.wmv

Vous avez alors accès à un rapport vous indiquant les différents événements qui se sont produit dans votre application, les différents threads qui étaient en cours d'exécution etc. :

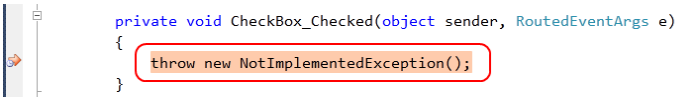


La ligne entourée ci-dessus correspond à une exception qui a été levée et qui n'a pas été gérée dans l'application. Double-cliquez sur la ligne pour lancer le debug sur cette exception.

Une nouvelle fenêtre s'ouvre dans **Visual Studio** : **IntelliTrace** et vous permet de visualiser tous les événements qui sont survenue depuis cette exception, ainsi que la pile d'appel telle qu'elle était. Vous avez la possibilité de filtrer les événements à l'aider de la liste déroulante en haut à gauche :



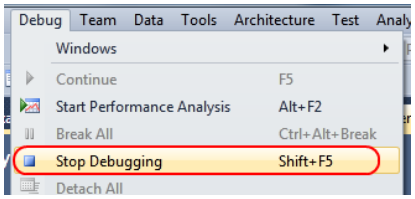
Visual Studio vous place également dans le fichier de code, à la ligne où l'exception c'est produite :



```
private void CheckBox_Checked(object sender, RoutedEventArgs e)
{
    throw new NotImplementedException();
}
```

Ici, on remarque de suite qu'il s'agit d'une faute d'inattention et que le développeur a oublié d'implémenter la méthode appelé lorsque que la case à cocher est cochée.

Stoppez le mode Debug de **Visual Studio** via le menu **Debug** puis **Stop Debugging** :

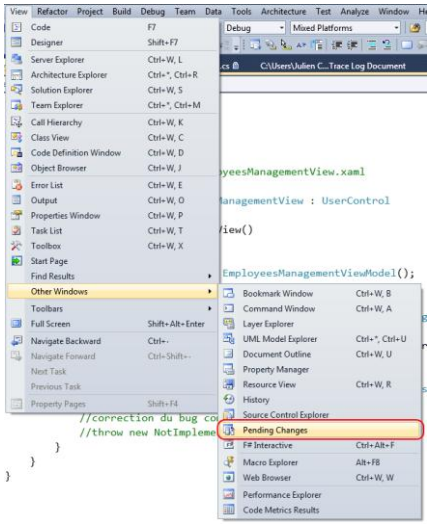


Corrigez l'erreur dans le fichier de code (commentez la levée de l'exception) :

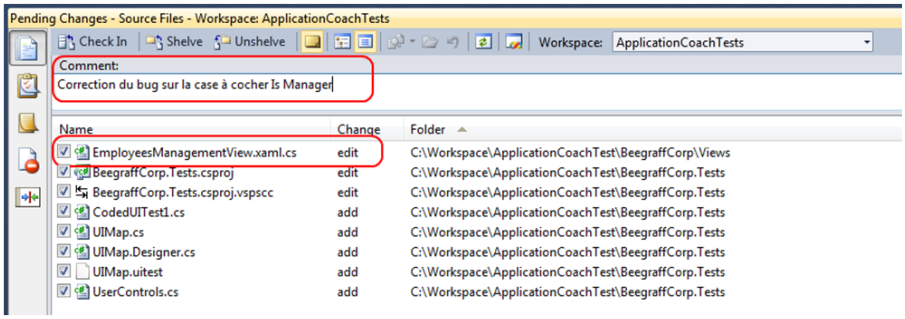
```
private void CheckBox_Checked(object sender, RoutedEventArgs e)
{
    //correction du bug complexe
    //throw new NotImplementedException();
}
```

Recompilez l'application, lancez-la et tester de cocher la case Is Manager afin de vous assurer que votre correction fonctionne.

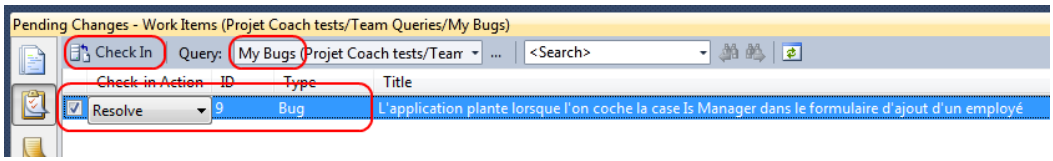
Une fois chose faite, ouvrez la fenêtre **Pending Changes** à partir du menu **View** puis **Other Windows** et enfin **Pending Changes** :



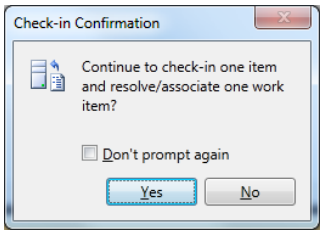
Dans cette fenêtre, ajoutez un commentaire qui sera associé à l'envoi des modifications sur le contrôle de code source, assurez-vous que tous les fichiers de **Coded UI Test** et modification de bug sont cochés :



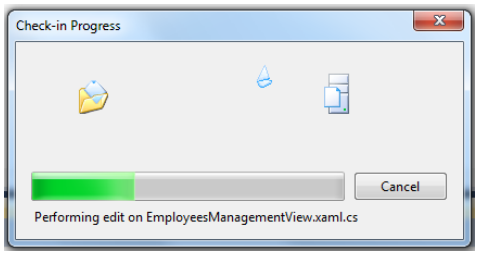
Ensuite, cliquez sur l'icône pour associer ce check-in à un work item. Dans le cas présent, le bug que vous venez de corriger :



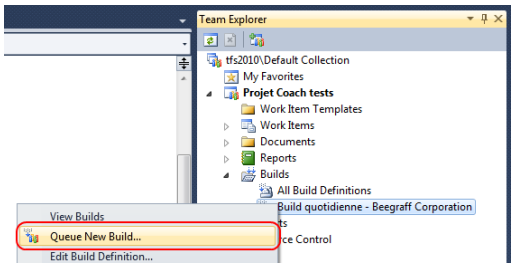
NB : faites attention a bien sélectionner la requête **My Bugs** (ou **Active Bugs**, le cas échéant) dans la liste déroulante. Cochez le bug qui vous intéresse ici et assurez-vous que la valeur de **Check in Action** soit bien **Resolved** afin d'indiquer que c'est le code qui va être envoyé qui résout ce bug. Il ne vous reste plus qu'à cliquer sur le bouton **Check In** en haut à gauche. Dans la fenêtre de confirmation qui s'affiche, cliquez sur **Yes** :



Patientez jusqu'à la fin de l'opération :



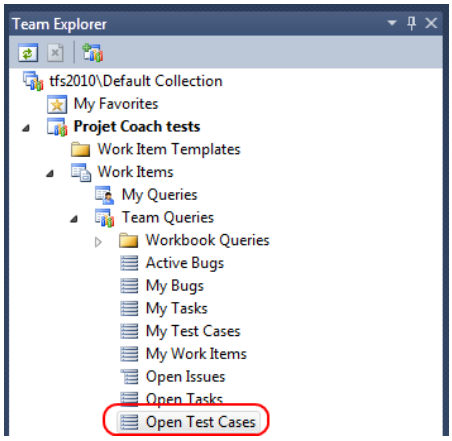
Enfin, et pour ne pas attendre jusqu'au lendemain que la build configurée lors de l'atelier 0 s'exécute, ouvrez le Team Explorer, dépliez le nœud Builds, faîtes un clic droit sur la définition « Build quotidienne – Beegraff Corporation » et cliquez sur Queue New Build… :



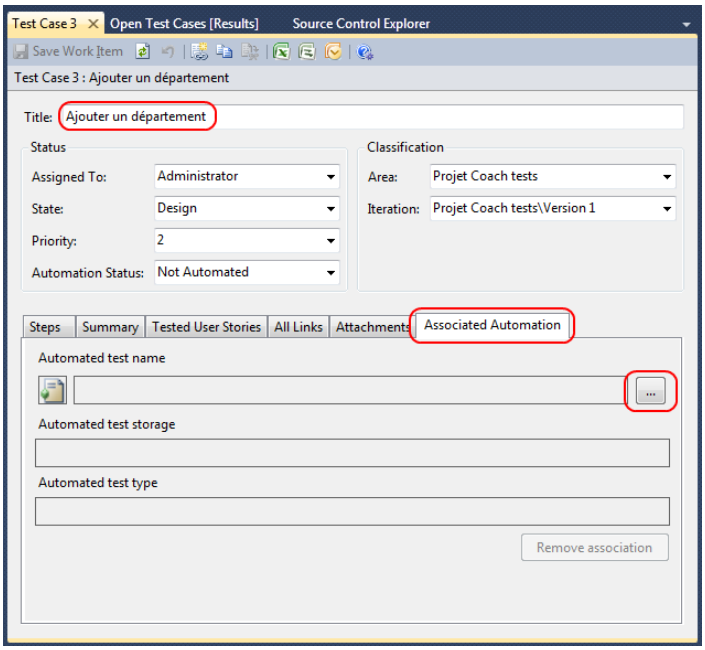
Dans la fenêtre qui s'affiche, confirmez en cliquant sur **Build**.

Pour terminer, vous allez devoir associer votre Coded UI Test au cas de test dans Team Foundation Server afin de permettre son automatisation via Test and Lab Manager par un testeur fonctionnel.

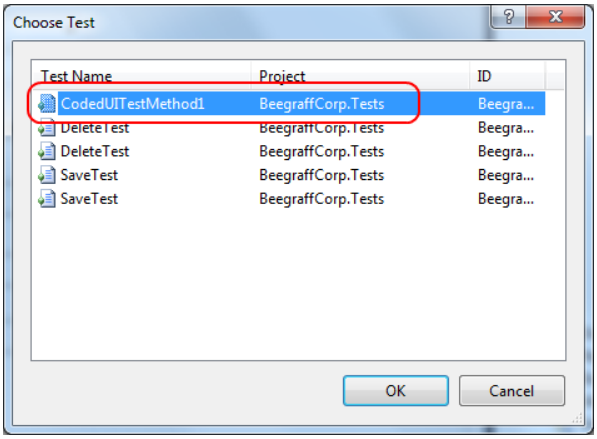
Pour cela, ouvrez le **Team Explorer**, dépliez le nœud **Work Items**, puis **Team Queries**. Double-cliquez alors sur la requête **Open Test Cases** :



Dans la fenêtre qui s'affiche, double-cliquez sur le cas de test « Ajouter un département » pour ouvrir la fiche qui lui est associé. Dans cette fiche, rendez-vous dans l'onglet **Associated Automation** :



Cliquez sur le bouton **…** pour accéder aux différents tests disponibles. Dans l'écran qui s'affiche, choisissez **CodedUITestMethod1** :



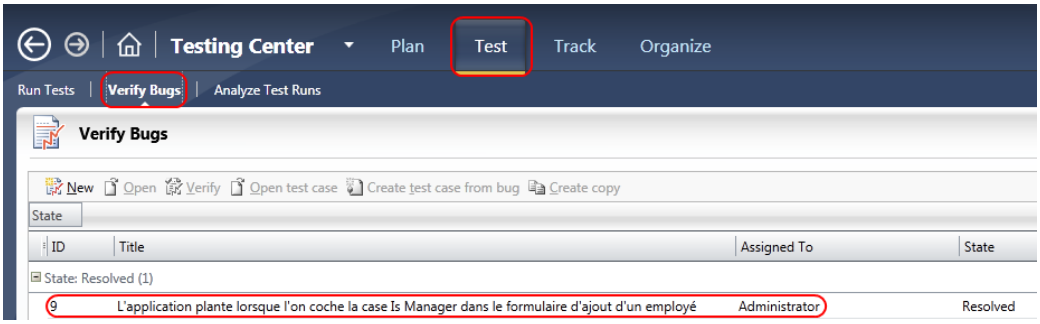
Validez en cliquant sur **OK** puis enregistrer la fiche du cas de test. Vous pouvez fermer Visual Studio 2010.

Test d'une correction de bug

En règle général, lorsqu'un développeur de l'équipe valide une correction, c'est-à-dire marque un bug comme étant résolu dans **Team Foundation Server**, il est prudent de re-tester la fonctionnalité qui était incriminée afin de valider son bon fonctionnement.

Dans **Microsoft Test and Lab Manager** (MTLM), vous avez accès à une section qui vous permet de suivre tous les bugs que vous avez créez ou ceux qui vont ont été assignés par le développeur. Pour cela, ouvrez

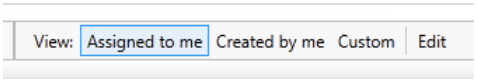
MTLM et rendez-vous dans le **Testing Center** après avoir sélectionné votre projet et le plan de test utilisé depuis le début du coach. Rendez-vous dans l'onglet **Tests** et choisissez le sous-menu **Verify Bugs** :



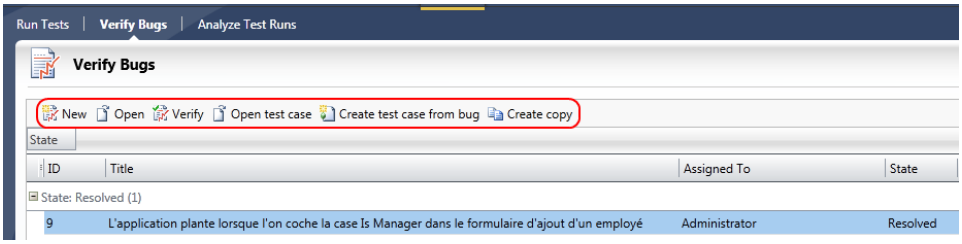
Comme vous pouvez le constater, le bug qui a été remonté à l'équipe de développement lors de l'atelier précédent et qui a été corrigé dans la partie ci-dessus est bien présent, vous est assigné et est marqué comme résolu. Ainsi, vous avez directement accès à la liste de tous les bugs corrigés que vous aviez remonté et qui vont ont été assigné lors de leur correction.

Dans la même fenêtre, il vous est possible de modifier les filtres appliqués pour afficher :

- Les bugs qui vous sont assignés (par défaut)
- Les bugs que vous avez créés
- Une requête personnalisée



Lorsque vous sélectionnez le bug dans la liste, la barre d'outils s'active :



Via celle-ci, plusieurs fonctionnalités s'offrent à vous :

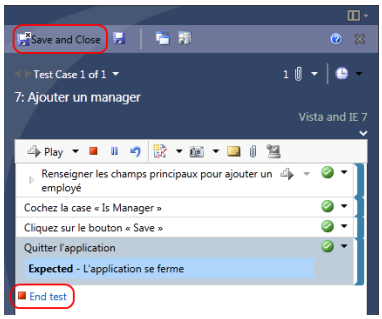
- Créer un nouveau bug
- Ouvrir la fiche de bug sélectionnée
- Vérifier la correction : rejouer le test afin de s'assurer que la correction soit effective.
- Ouvrir le cas de test associé
- Créer un cas de test associé au bug
- Créer une copie du bug

Cliquez sur le bouton « Verify ». Dès lors, **MTLM** se réduit pour laisser place au **Test Runner** et ce afin d'exécuter le cas de test associé au bug.

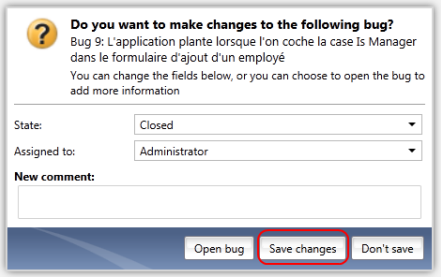
***Note** : au préalable, pensez à récupérer les binaires issus de la dernière build dans le répertoire partagé prévu à cet effet sur votre serveur **Team Foundation Server**.*

Exécutez alors le test fonctionnel. Vous pouvez utiliser la fonction « fast-forward » qui vous a été présentée dans l'atelier précédent afin de réaliser automatiquement les premières étapes de la shared steps.

Validez le fait qu'à présent toutes les étapes du test fonctionnent. Une fois ceci fait, cliquez sur le bouton **End test** pour arrêter le test puis sur **Save and Close** pour revenir dans MTLM :



Lorsque vous revenez dans MTLM, ce dernier vous affiche un popup permettant de changer directement l'état du bug et de le passer en **Closed** puisque l'exécution du test a réussi :



Cliquez sur le bouton **Save changes** pour valider la fermeture de la fiche de bug. Si le test fonctionnel avait à nouveau échoué, vous auriez pu passer la fiche en **Open** et la réassigner au développeur afin de lui indiquer que le bug n'était pas résolu. Dès lors le développeur aurait eu connaissance du bug et aurait pu recommencer à travailler dessus comme vous l'avez appris dans la partie précédente.

Automatisation d'un test manuel réalisé avec Microsoft Test and Lab Manager

Dans la première partie de cet atelier vous avez appris, en tant que développeur, à créer des tests d'interface graphique codés (**Coded UI Tests**) à partir de l'enregistrement d'un test fonctionnel réalisé

par un testeur fonctionnel via **Microsoft Test and Lab Manager**. Cela vous donnait alors la possibilité de le rejouer via **Visual Studio 2010**.

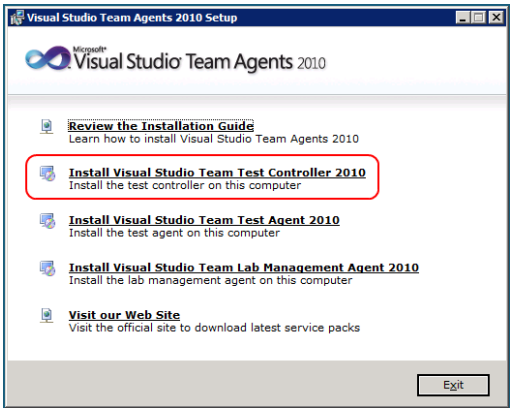
Dans certains cas, il peut être utile d'automatiser l'exécution de ces tests, par exemple pour les intégrer dans un processus de build. En effet, dans des problématiques d'intégration continue il est fréquent d'exécuter une batterie de tests (test unitaires par exemple) afin de valider certaines fonctionnalités. Avec **Visual Studio 2010**, **Microsoft Test and Lab Manager** et **Team Foundation Server 2010**, il est possible d'automatiser les tests fonctionnels afin de les exécuter sur une machine précise. Pour cela, il va falloir configurer une machine (le serveur qui est en charge de la build dans le cas présent) afin qu'un agent soit capable d'ouvrir une session graphique, lancer le test de l'application et enregistrer les résultats dans **Team Foundation Server**.

Installation de Microsoft Visual Studio Team Test Controller 2010

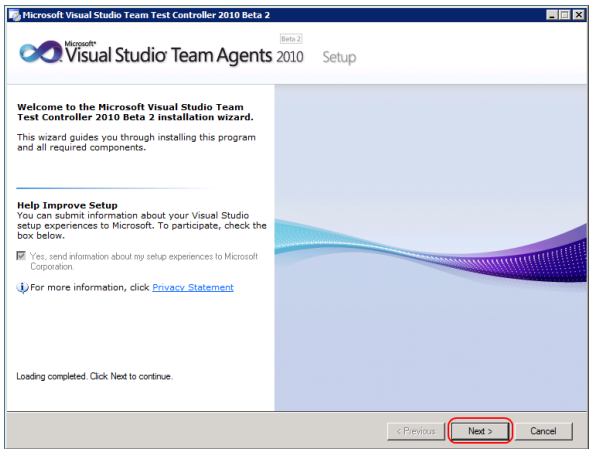
L'exécution automatique de tests sur un environnement nécessite d'installer un agent (i.e. un service Windows) sur la machine cible de manière à pouvoir contrôler cette machine. Potentiellement, dans une infrastructure de tests réaliste, on retrouvera plusieurs machines contenant chacune un agent. On aura par exemple un environnement sous Windows Vista, un autre sous Windows 7, etc. Ces différents agents sont pilotés par un contrôleur identifié auprès d'une collection de projet. C'est ce contrôleur qui contacte les agents pour leur faire exécuter les différents tests.

Pour commencer, vous devez télécharger l'image ISO **Visual Studio Team Agents** à l'adresse suivante : http://go.microsoft.com/fwlink/?LinkID=165574

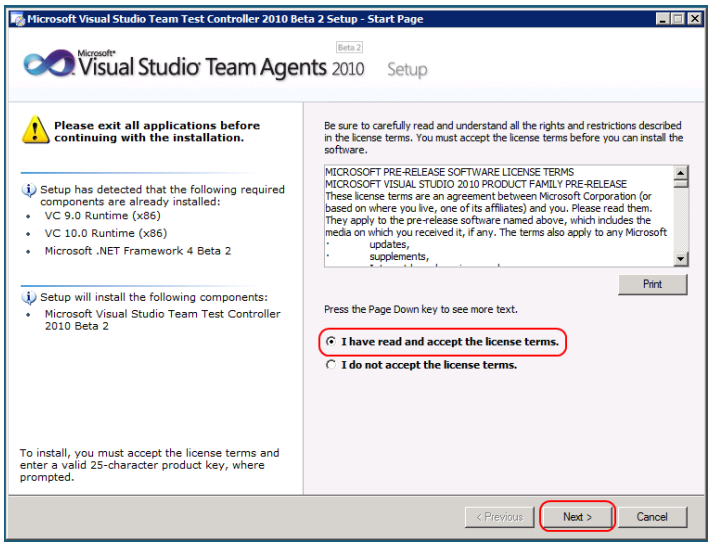
Lancer l'assistant d'installation disponible sur l'image CD et cliquez sur le lien **Install Visual Studio Team Test Controller 2010** :



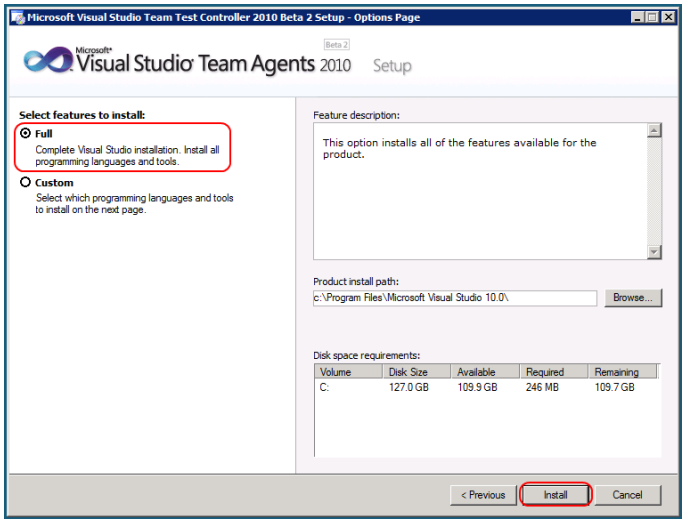
Patientez pendant le chargement des composants d'installation puis cliquez sur **Next** :



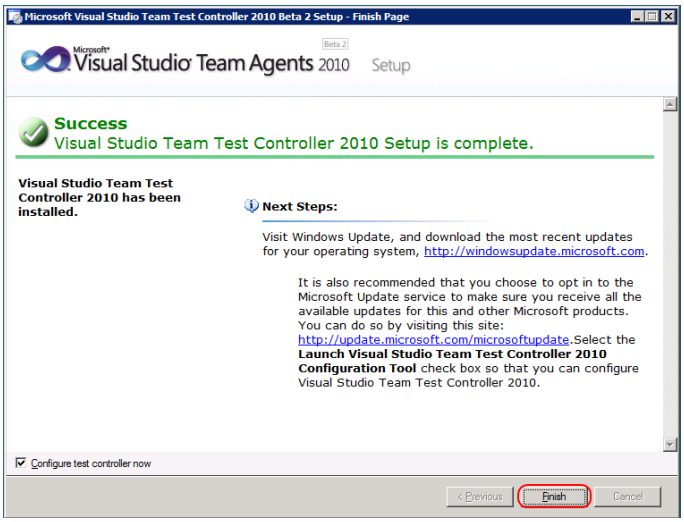
Dans l'écran suivant, acceptez les termes du contrat de licence et cliquez sur **Next** :



Choisissez enfin le mode d'installation **Full** et cliquez sur **Install** :

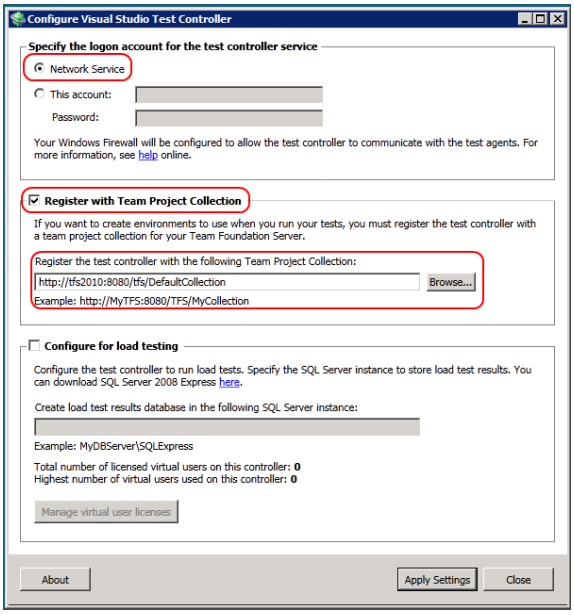


Patientez pendant l'installation du contrôleur de tests et cliquez sur **Finish** pour terminer :

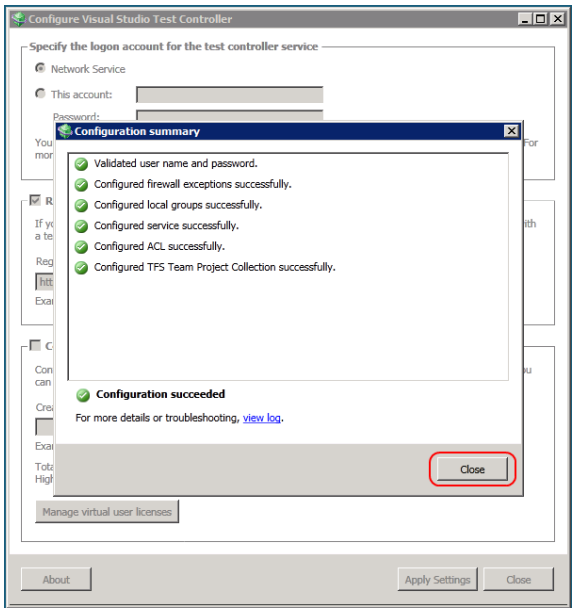


Note : Faites attention au fait que la case ***Configure test controller now*** *soit cochée.*

Dans l'assistant qui s'affiche, laissez Network Service comme compte de service pour le contrôleur de test et associé le à la collection de projet qui contient le projet que vous utilisez depuis le début de ce coach :



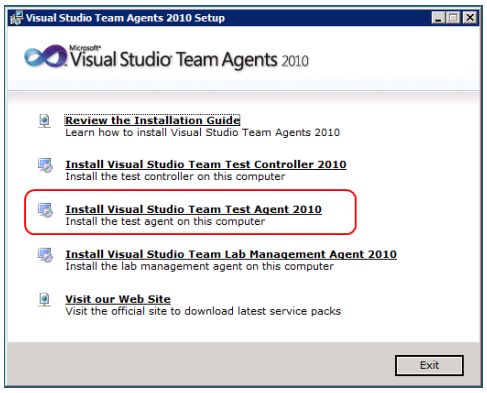
Cliquez sur **Apply Settings** pour valider la configuration et cliquez sur Close pour terminer :



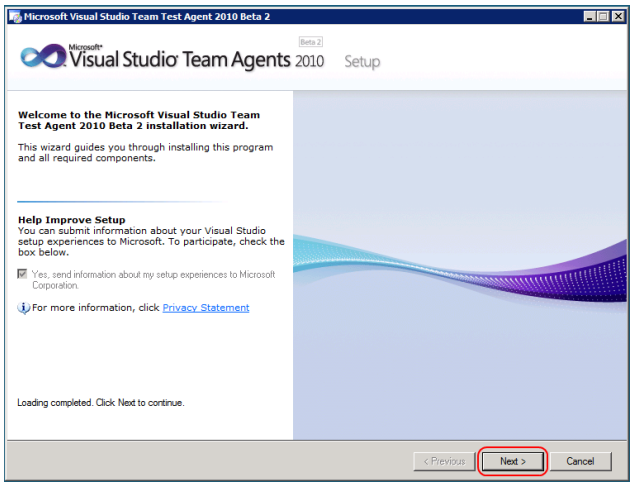
Installation de Microsoft Visual Studio Team Test Agent 2010

Pour commencer, vous allez devoir installer un agent de test sur la machine cible (celle qui sera en charge de l'exécution du test). L'ISO que vous avez téléchargez pour installer le contrôleur de tests sur le serveur vous permet également d'y installer un agent.

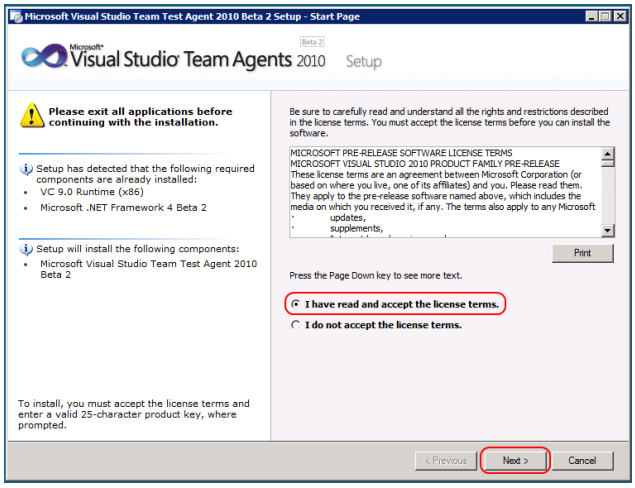
Une fois l'assistant d'installation lancé, cliquez sur le lien **Install Visual Studio Team Test Agent 2010** pour lancer l'installation de l'agent de test :



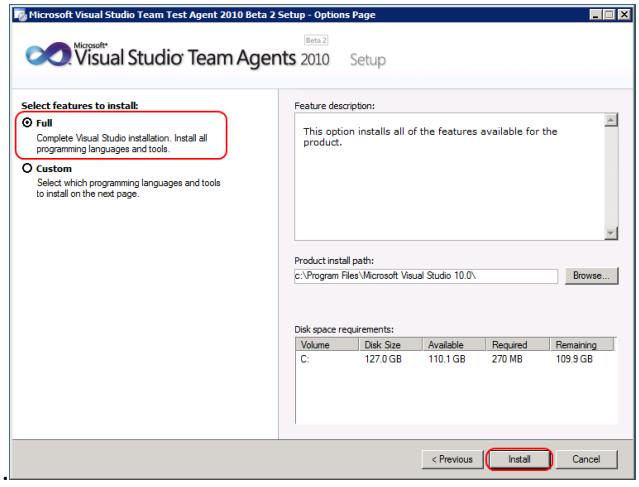
Patientez pendant le chargement des composants d'installation puis cliquez sur **Next** :



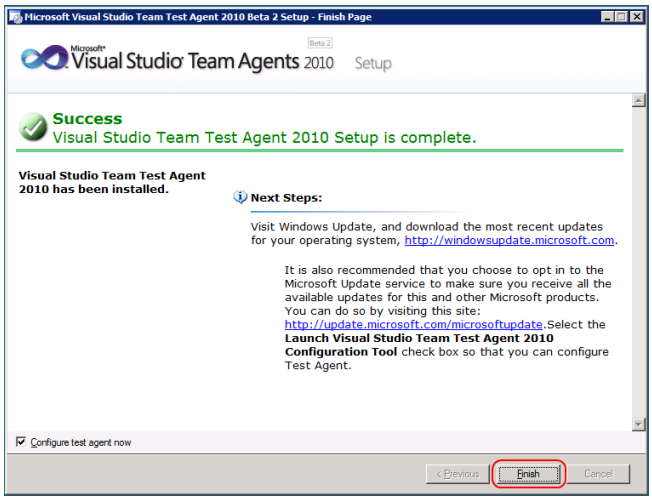
Acceptez les conditions de licence et cliquez sur **Next** :



Choisissez l'installation complète, puis cliquez sur **Install** :



Patientez jusqu'à la fin de l'installation puis cliquez sur Finish :

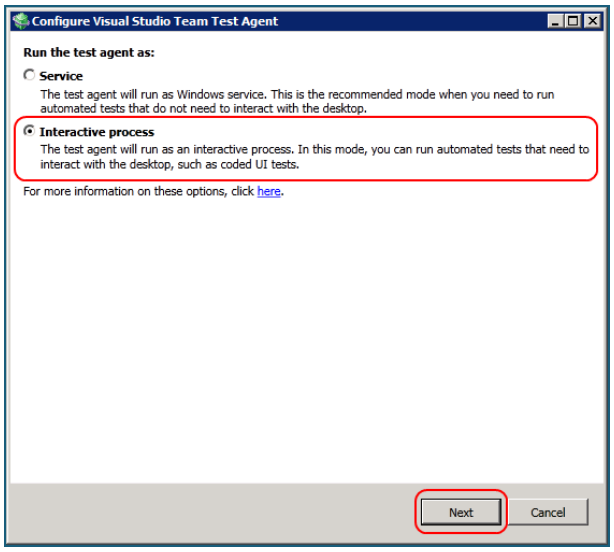


*Note : veillez à laisser cochée la case **Configure test agent now**.*

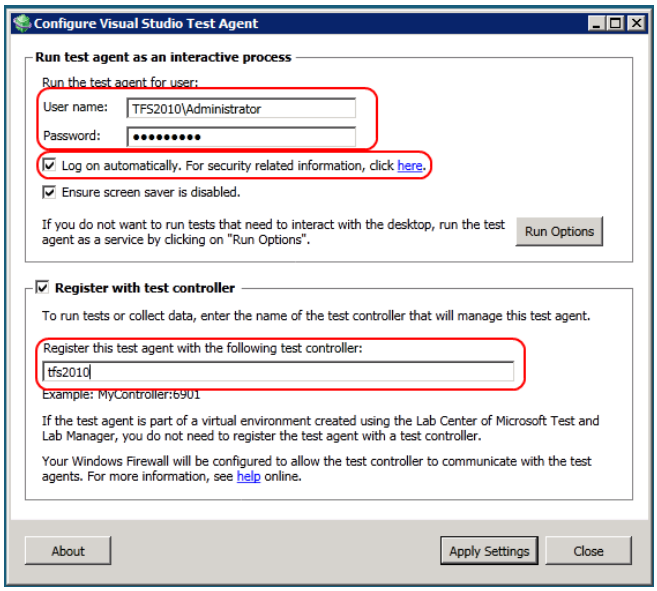
Dans l'écran de configuration qui s'affiche, la première information à donner est le type de compte qui doit être utilisé. Il existe deux possibilités :

- **Service** : service windows (sans session graphique)
- **Interactive Process** : utilisateur possédant le droit de se loguer sur la machine est donc ayant la possibilité de lancer une session graphique et donc d'exécuter un **Coded UI Test**.

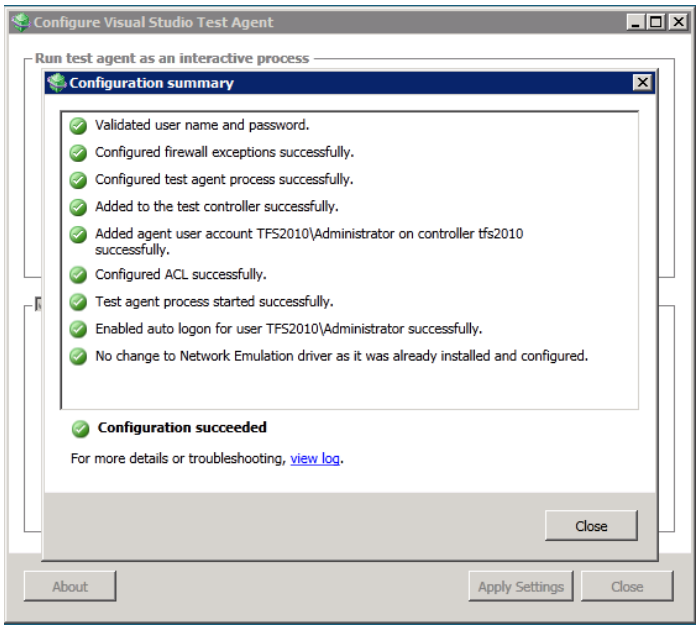
Choisissez la deuxième option :



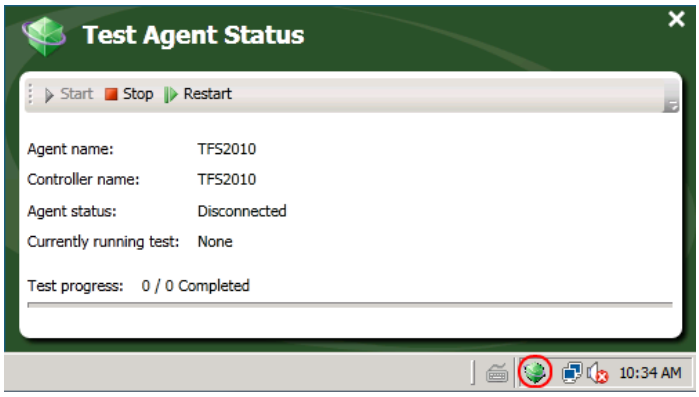
Dans l'écran suivant, renseignez un compte utilisateur ayant le droit de se loguer sur la machine. Cochez également la case **Log on automatically** afin d'ouvrir automatiquement la session de l'utilisateur en cas de redémarrage de la machine, par exemple :



Enfin cliquez sur **Apply Settings** et attendez la validation de la configuration. Enfin, cliquez sur **Close** pour fermer les assistants :

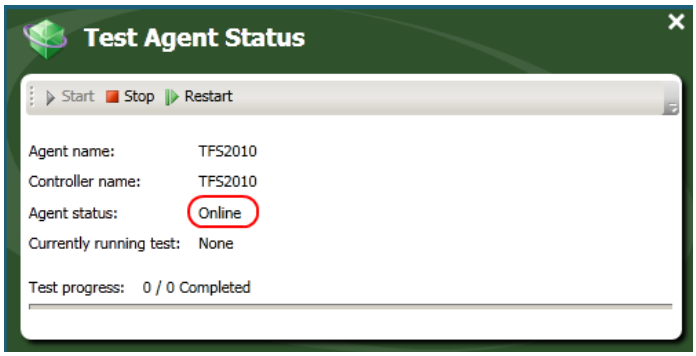


Dès lors, une nouvelle icône est disponible dans la barre d'état du serveur. Celle-ci vous permet de voir l'état de l'agent de test sur le serveur :



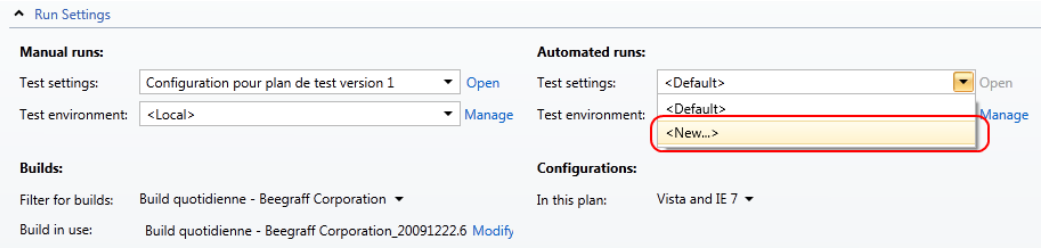
Actuellement vous constatez que l'agent est déconnecté. Vous devez redémarrer la machine sur laquelle vous l'avez installée pour qu'il soit correctement initialisé.

Une fois le redémarrage effectué, l'agent passe **Online** :

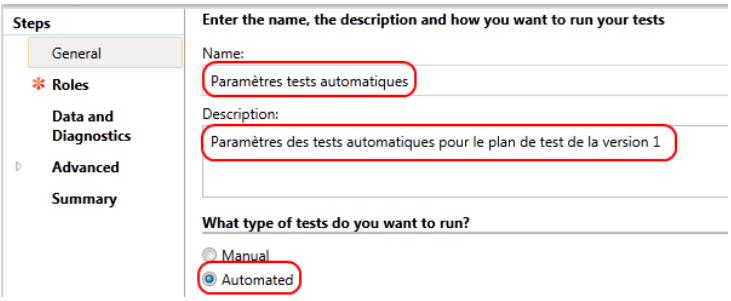


Configuration du plan de test pour permettre l'automatisation

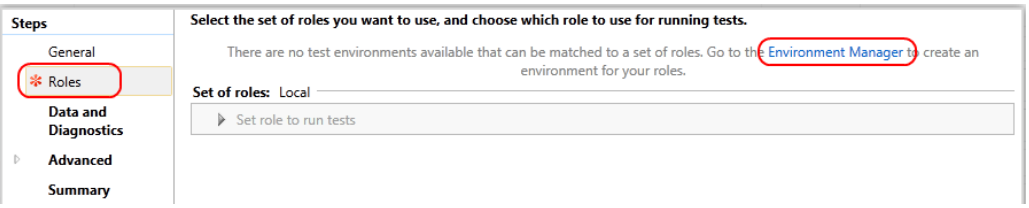
La première chose que vous allez devoir faire est de configurer des paramètres de tests automatisés pour le plan de test actuel. Pour cela, ouvrez **Microsoft Test and Lab Manager** et rendez-vous dans l'onglet **Test** du **Testing Center**. Cliquez sur le sous-menu **Properties** pour accéder aux propriétés du plan de test. Dans la partie **Automated Runs** déroulez la liste **Test Settings** et cliquez sur **<New…>** :



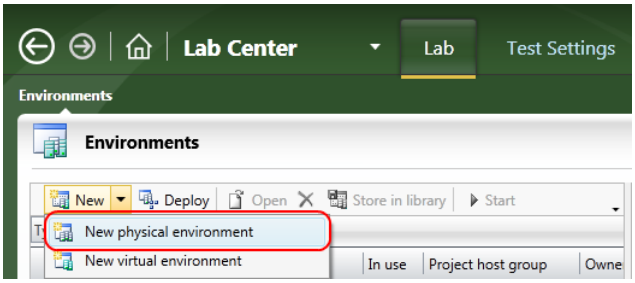
Dans la fenêtre qui s'affiche, dans l'onglet **General**, donnez un nom et une description à la configuration qui va être créée :



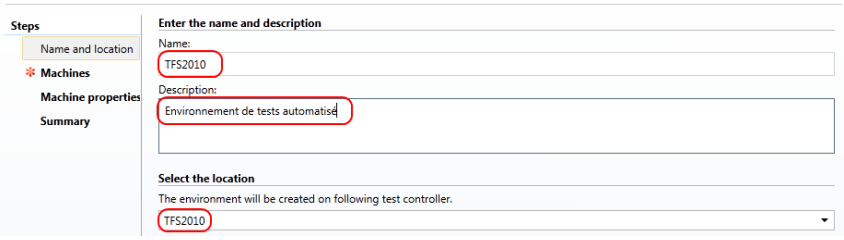
Cliquez sur **Next** pour passer à l'onglet **Roles**. Dans celui-ci, cliquez sur le lien **Environment Manager** pour accéder à la gestion des environnements d'exécution de test :



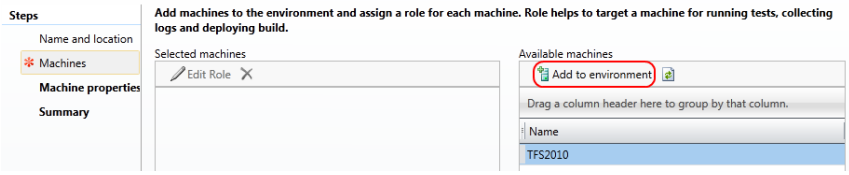
Ceci aura pour effet de vous faire basculer dans la seconde « grande » zone de MTLM : le Lab Center. Dans la partie **Environments** cliquez sur le bouton **New** et choisissez **New physical environment** :



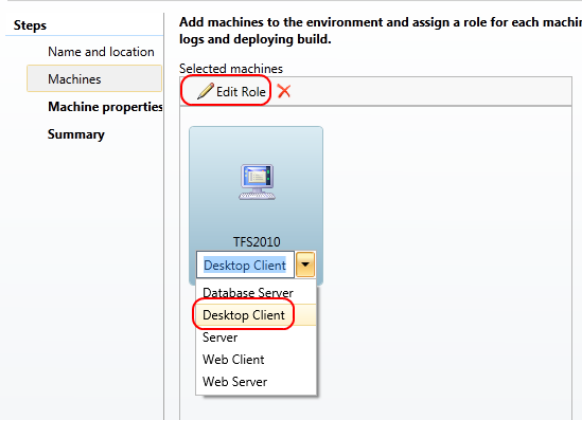
Dans la fenêtre qui s'affiche, saisissez un nom pour l'environnement, une description et choisissez le contrôleur de test sur lequel sera créé l'environnement :



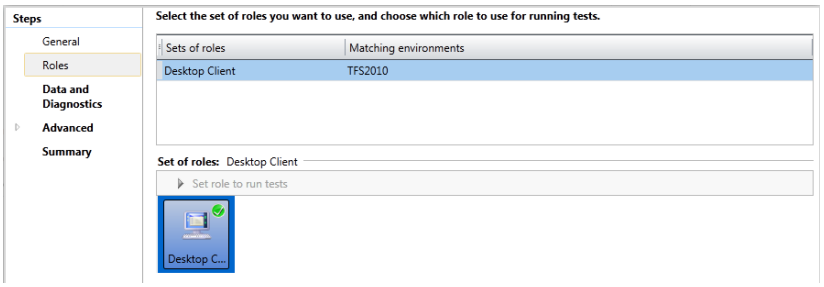
Cliquez sur **Next** pour passer à l'onglet **Machines**. Dans celui-ci, vous allez devoir ajouter des machines (une ici) à l'environnement de test. Pour cela, sélectionnez l'agent de test dans la liste de droite et cliquez sur le bouton **Add to environment** :



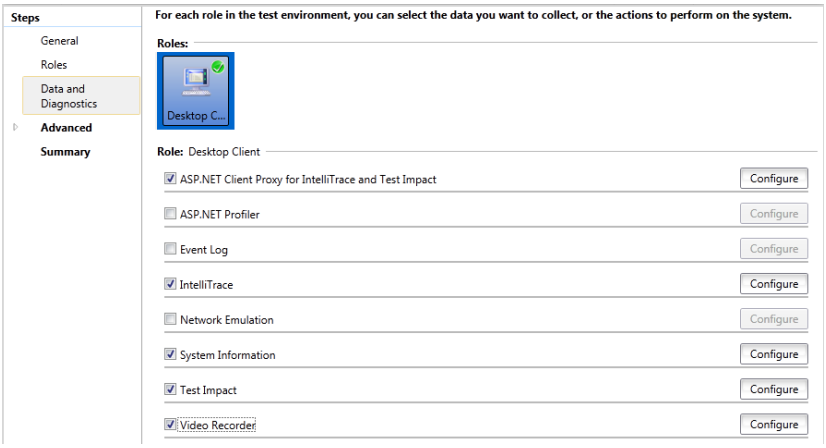
Ensuite, vous devez affecter un rôle à la machine (ici Desktop Client, bien que ce soit un Server…). Pour cela, cliquez sur **Edit Role** et choisissez **Desktop Client** dans la liste déroulante :

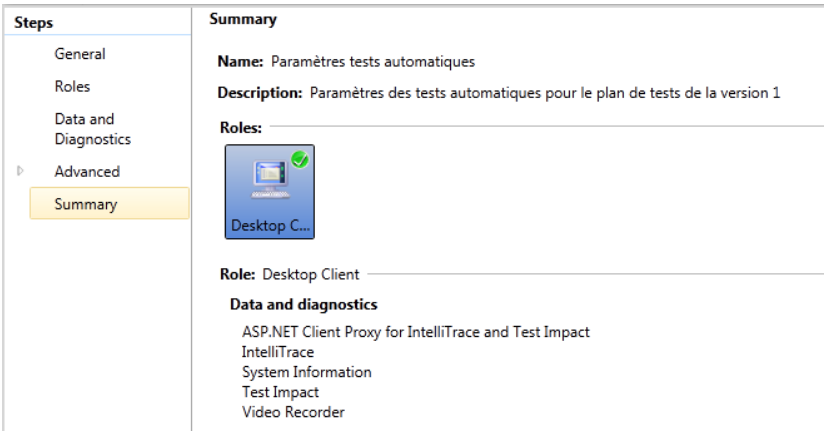


Cliquez sur **Finish** pour terminer la configuration. Vous pouvez alors fermer la fenêtre **Environments** du **Lab Center** pour revenir à la configuration des paramètres de tests automatisés du plan de tests. Comme vous pouvez le constater, l'onglet **Roles** vous propose maintenant de choisir la machine que vous venez d'ajouter dans l'environnement fraîchement créé :



Cliquez alors sur **Next** pour passer à l'onglet **Data and Diagnostics** qui vous permet de définir quelles sont les données à collecter lors de l'exécution du test automatisé :



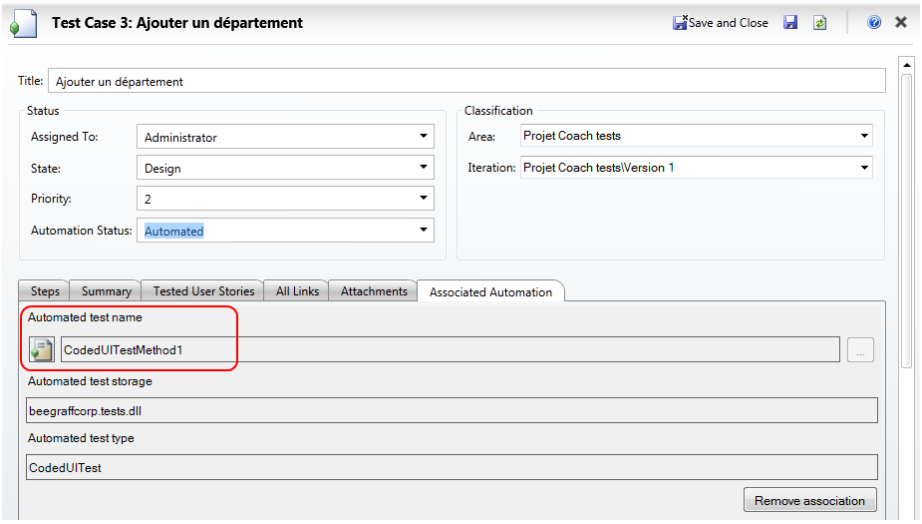


Enfin terminez l'assistant en cliquant sur **Finish**.

Validez les paramètres du plan de test en cliquant sur le bouton **Save and Close**.

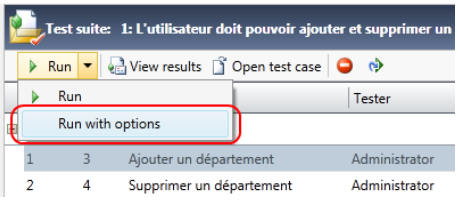
Exécution du test automatisé via Microsoft Test and Lab Manager

Ouvrez **Microsoft Test and Lab Manager** et rendez-vous dans l'onglet **Tests** du **Testing Center**. Dans la suite de test « L'utilisateur doit pouvoir ajouter et supprimer un département », sélectionnez le cas de test « Ajouter un département ». Ouvrez le cas de test et rendez-vous dans l'onglet **Associated Automation** :

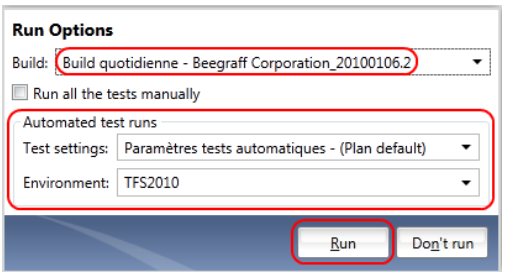


Comme vous pouvez le constater, le cas de test est associé à un **Coded UI Test**, suite au travail que le développeur a effectué dans la première partie de cet atelier. Cette constatation étant faite, vous pouvez fermer la fiche du test.

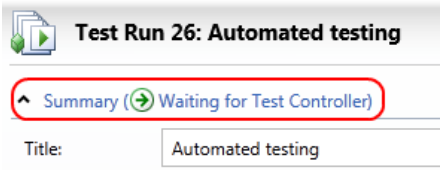
De retour dans l'onglet Test, vérifiez que le test « Ajouter un département » est bien sélectionné dans la liste et déroulez le menu **Run**. Choisissez l'entrée de menu **Run with options** :



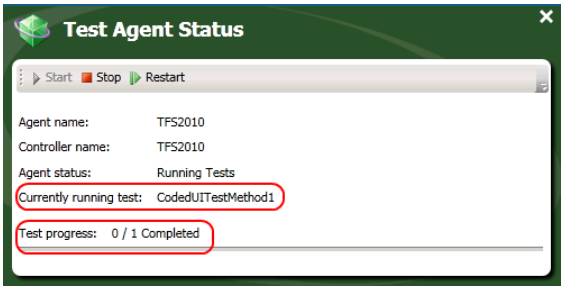
Dans la fenêtre qui s'affiche, choisissez la dernière Build en date. Vous pouvez également voir que le test est bien marqué comme automatisé sur l'environnement TFS 2010 que vous avez configuré ci-dessus :



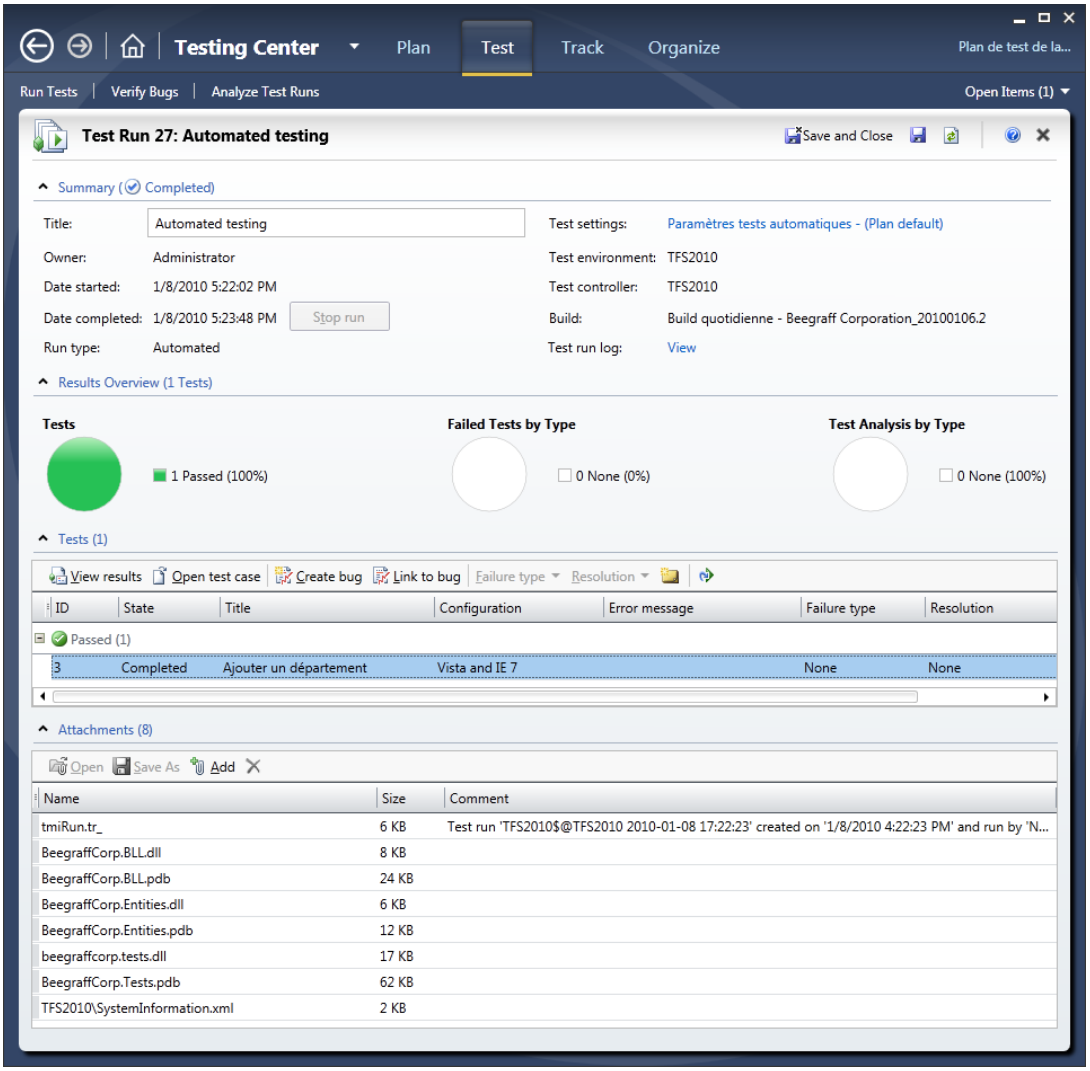
Cliquez sur **Run** pour démarrer le test. Patientez pendant l'exécution du test :



Sur le contrôleur de test (ici, le serveur que vous avez configuré plus tôt dans cet atelier), vous pouvez voir qu'un test est en cours :



Une fois le test terminé, retournez dans Microsoft Test and Lab Manager et cliquez sur le bouton  pour rafraîchir la page concernant le test automatisé. Vous pouvez voir que celui-ci est complété et vous obtenez alors le résultat de celui-ci :



Comme vous pouvez le constater, le test s'est bien exécuté sur l'agent de test.

Note : *si vous aviez une session ouverte sous le compte d'utilisateur utilisé par l'agent de test sur la machine de test, vous avez pu constater que l'application se lançait et que le test se déroulait de la même façon que le test automatisé lancé depuis Visual Studio 2010 dans la première partie de cet atelier.*