REPORTE TÉCNICO SOBRE VISUAL STUDIO 2010

Este reporte tiene como finalidad resumir el conocimiento adquirido sobre Microsoft Visual Studio 2010, como herramienta de apoyo a la *gestión de pruebas* de software (no desde el punto de vista de *desarrollo* de software). Es necesario mencionar que la versión de Microsoft Visual Studio a la que se refiere este reporte es la versión Ultimate 2010. Esta versión incluye el Microsoft Test Manager, que es una aplicación "separada" pero complementaria de Visual Studio, y que se incluye en la misma distribución.

Este documento contiene aportes de varios investigadores involucrados en el proyecto de investigación No. 824-B1-133 "Adecuación de prácticas básicas de aseguramiento de la calidad y mejora del proceso de software a una unidad de desarrollo de software de la UCR".

Este reporte está organizado en tres grandes partes: la primera parte ofrece una visión general de la herramienta, la segunda parte se enfoca hacia la gestión de historias de usuario, casos de prueba, e incidentes (énfasis en pruebas manuales), y la tercera parte se enfoca hacia el desarrollo de pruebas automatizadas.

Autores:

Dra. Alexandra Martínez Porras

Bach. Marco González

Bach. Gustavo López

Bach. Francisco Cocozza

1. Visión General de la herramienta

Visual Studio Ultimate 2010 (VS2010) de *Microsoft* es un conjunto completo de herramientas para la gestión del ciclo de vida de una aplicación de software. VS2010 se acopla con el servidor *Team Foundation Server* y con el *Test Manager*, ambos de la familia *Microsoft*.

2.1. Visual Studio

El *Visual Studio* (VS) es un entorno de desarrollo de software integrado, que permite programar en los lenguajes C++, C#, J#, y VB.NET. Entre las funcionalidades que se incluyen en la versión 2010 de Visual Studio y que están relacionadas con la gestión de las pruebas dentro del ciclo de vida de la aplicación, están:

- Mejoramiento de los esfuerzos de pruebas con herramientas para una mejor documentación de los escenarios de prueba y de las colecciones de datos de prueba.
- Identificación y ejecutación de sólo aquellos casos de prueba que fueron impactados por un cambio en el código.
- Capacidades aumentadas de control de versiones, incluyendo *check-in* cerrado, visualización de los *branchs*, y flujo de compilación.

Visual Studio 2010 evoluciona la gestión del ciclo de vida de una aplicación mediante:

- La construcción de calidad en el ciclo de vida.
    - Eliminando los defectos "no-reproducibles".
    - Asegurando compilaciones (*builds*) de alta calidad.
    - Incorporando rendimiento en el ciclo de vida.
- El impulso a la eficiencia en el esfuerzo de pruebas.
    - Eliminando tareas tediosas.
    - Mejorando la instalación, configuración y despliegue de las pruebas.
    - Escogiendo las pruebas adecuadas.
- La garantía de mayor completitud de las pruebas.
    - Planeando las pruebas y monitoreando su progreso.
    - Encontrando brechas en las pruebas y solucionándolas.
    - Asegurando que los cambios son probados apropiadamente.

2.2. Team Foundation Server

El *Team Foundation Server* (TFS) es el sucesor de *Visual SourceSafe*, y consiste en un repositorio centralizado con un conjunto de herramientas que permiten la colaboración

entre diferentes roles dentro del ciclo de vida de la aplicación. La configuración básica del TFS ofrece un flujo de trabajo (*work flow*) integrado que incluye control de código fuente, elementos de trabajo (*work items*) y compilaciones de versiones (*builds*). Esto permite escenarios de trabajo comunes, por ejemplo: edición del código fuente, compilación del producto, prueba del producto, reporte de defectos, corrección de defectos, repita. La versión completa del TFS agrega características nuevas tales como pruebas automatizadas, laboratorios virtuales, validación de la arquitectura, entre otras.

El TFS trabaja con "colecciones de proyectos de equipo" (*team project collections*), que son agrupaciones de "proyectos de equipo" (*team projects*) que permiten manejar al grupo como un recurso autónomo con sus propios grupos de usuarios, recursos de servidor, etc. Esencialmente estas colecciones son unidades organizativas que ofrecen la posibildad de agrupar y controlar proyectos similares.

Un "proyecto de equipo" es una infraestructura única que comprende los elementos de trabajo (e.g., historias de usuario, tareas, casos de prueba, y errores), código fuente, pruebas, métricas, documentos, y todas las demás herramientas y artefactos asociados a una aplicación en desarrollo.

2.3. Test Manager

El *Microsoft Test Manager* (MTM) le permite gestionar y ejecutar casos de prueba fuera de *Visual Studio*, además de crear y administrar ambientes físicos o virtuales. Esta aplicación cliente se instala con *Visual Studio Ultimate* 2010, y ayuda a:

- Crear y gestionar casos de prueba
- Administrar las propiedades de las pruebas
- Generar los pasos para las prueba
- Ejecutar las pruebas
- Crear Planes de pruebas
- Gestionar las configuraciones para un Plan de pruebas
- Monitorear el avance de la ejecución de un Plan de pruebas
- Crear defectos (*bugs*) durante la ejecución de los casos de pruebas, y ligarlos para efectos de trazabilidad.
- Llevar una bitácora del resultado de la ejecución de las pruebas.

2. Gestión de historias de usuario, casos de prueba, e incidentes

2.1. Historias de Usuario

¿Que es una historia de usuario?

Una historia de usuario (*user story*) comunica la funcionalidad que es de valor para el usuario final del sistema. El ideal es que cada historia de usuario indique lo que el usuario desea realizar con determinada característica del sistema. Una historia de usuario debe centrarse en la función, es decir, lo que se desea lograr. Después con esa información se podrá definir cómo se va a lograr, por lo que se debe evitar una especificación en la que se detalla cómo se va a desarrollar.

Beneficios de las historias de usuario

Facilitan la rápida administración de requerimientos de los usuarios sin usar una gran cantidad de documentos formales.

Permiten responder rápidamente a los requerimientos cambiantes.

Facilitan la validación de requerimientos ya que tienen pruebas de validación asociadas.

Características de las historias de usuario

Independientes unas de otras. Pueden ligarse mediante "links", pero deben ser independientes

Negociables: Al no ser un documento formal no puede considerarse un "contrato".

Valoradas por los clientes y los usuarios.

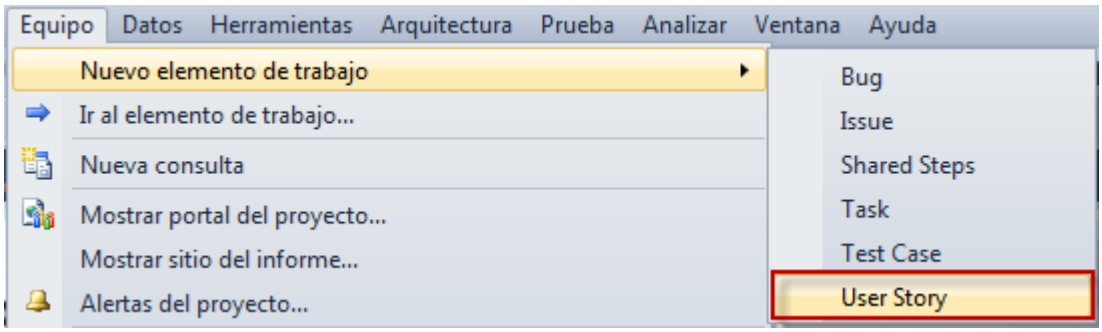
Estimables: Debe permitir la estimación de tiempo necesario para concluirlo.

Pequeñas: En tamaño y estimación de tiempos.

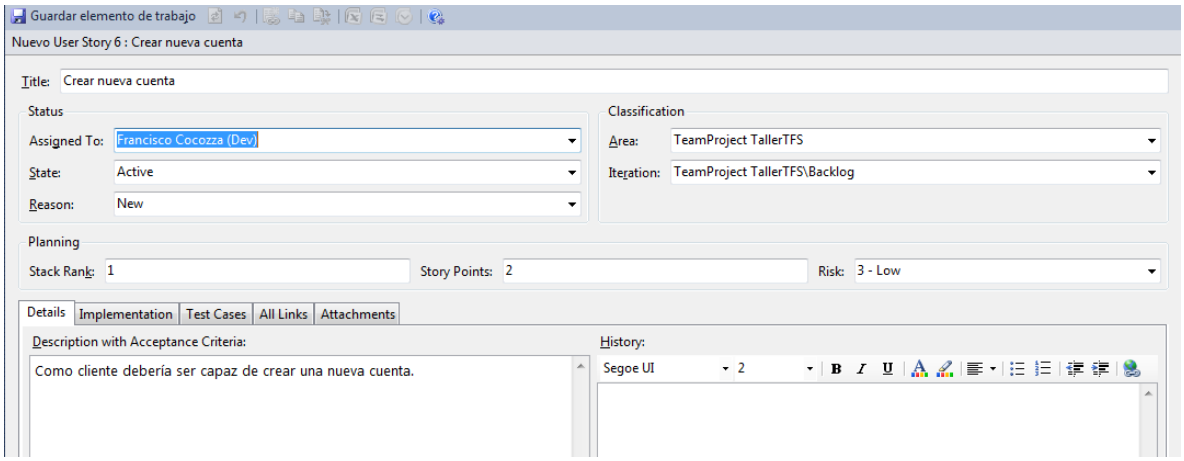
Verificables: Fácil validación de funcionalidad

Creación de historias de usuario

Para crear una historia de usuario, accedemos a Equipo → Nuevo elemento de trabajo → *User Story*.



Se abre el formulario para ingresar los datos del *user story*.



Los datos más relevantes en la información son:

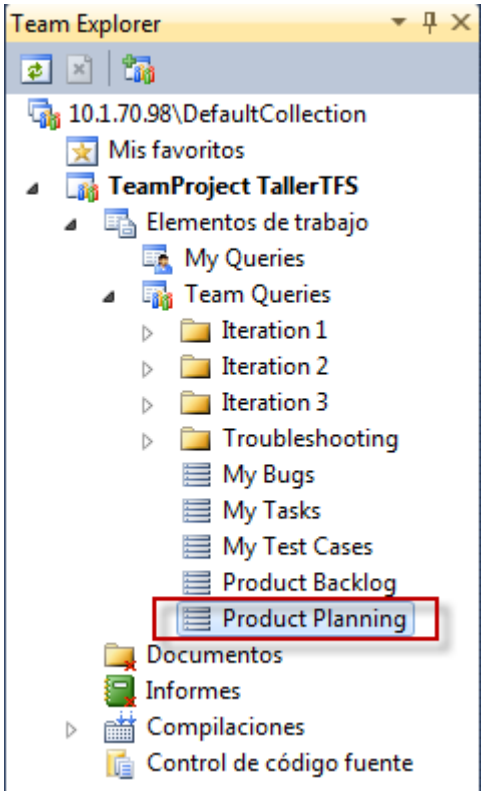
- **Titulo:** Escoger un título significativo del *user story.*
- **Asignada a:** Seleccionar de la lista de participantes del proyecto a la persona encargada de darle seguimiento al *user story.*
- **Área:** Defina el área de la empresa o el equipo al que pertenece el *user story.*
- **Iteración:** Define la iteración en la que se encuentra el *user story.*
- **Estado:** Dado que el *user story* es nuevo, el único posible estado es active. Los posibles estados de un *user story* se muestran en el siguiente diagrama.

- **Razón:** Determina los motivos de los cambios entre estados, este se modifica automáticamente de acuerdo a las siguientes reglas:
  - ✓ Si se pasa de estado Activo a Resuelto, la razón es que se completaron la codificación y los test de unidad
  - ✓ Si se pasa de Resuelto a Cerrado, la razón es que se paso el test de aceptación
  - ✓ Si pasa de Activo a Cerrado, se debe seleccionar manualmente la razón de las posibles existentes
- ***Stack Rank*:** Indica la importancia relativa del caso comparada con la de los demás casos existentes en el trabajo pendiente (*Area: Backlog*).
- **Puntos de historia:** Especifica una valoración subjetiva de la cantidad de trabajo que se necesitará para completar la historia de usuario (estas métricas son definidas por la organización).
- **Riesgo:** Una calificación subjetiva de la incertidumbre relativa en torno a la finalización con éxito del *user story*, puede ser: *High, médium, low.*
- **Detalles:** Descripción detallada tanto del caso de usuario como del criterio de aceptación que se utilizará para el mismo. Además provee un historial de cambios.
- **Implementación:** Permite asociar las tareas (*tasks*) o los casos de usuario (*user stories*) que tengan alguna relación de padre o hijo con el presente *user story*.

- **Casos de prueba:** Permite asociar los casos de prueba que tengan alguna relación de "*tested by*" con el presente *user story.*
- **Todos los enlaces (*links*):** Muestra la trazabilidad de los *work items* relacionados con el presente *user story*.
- **Adjuntos:** Permite adjuntar archivos o documentos relacionados con el *user story*.

Visualización de historias de usuario

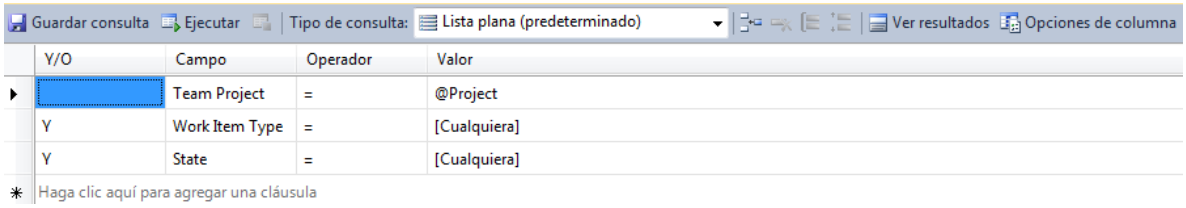
Visual Studio provee la funcionalidad para acceder a los *work items* mediante consultas, similares a las de una base de datos, mediante una herramienta gráfica. Para acceder a la herramienta y observar las historias de usuario ingrese a: ***Team Explorer*** → **Elementos de trabajo** → ***Team Queries***, podemos encontrar consultas predefinidas asociadas al *team Project* por ejemplo para poder visualizar todos los *user stories* utilizamos la consulta conocida como ***Product Planning***.



Ahora bien, si lo que se desea es visualizar únicamente los *user stories* asociados a un usuario específico podemos crear nuevas consultas. Para crear nuevas consultas accedemos al *Team Explorer,* **clic derecho en *My Queries* → *New Querie*.**

1. Modifique los campos de la consulta como considere adecuado, en este caso vamos a limitar la consulta a los *user stories* asociados a mi usuario. Agregue las siguientes condiciones:

   - Team Project = @Project
   - Work Item Type = User Story
   - State <> Closed
   - Assigned to = @Me

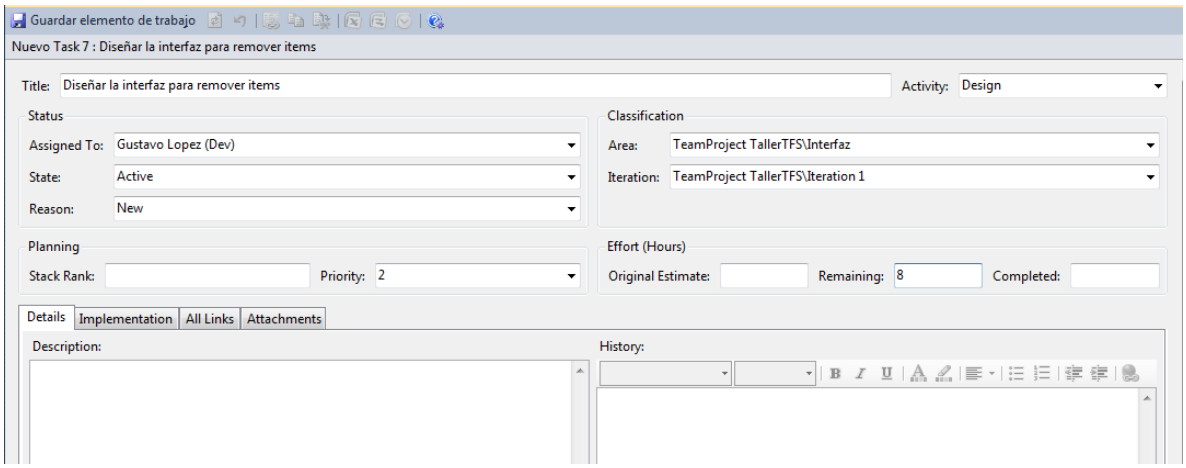


2. Ejecute y guarde la consulta. Todas las consultas que guarde serán accesibles desde **Team Project → Work Items → My Queries → Nombre de consulta**.

Definición de tareas

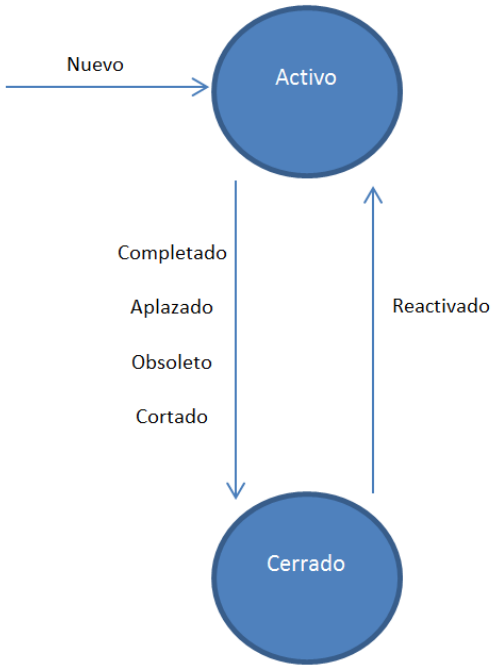
Para crear una tarea accedemos a: **Equipo → Nuevo elemento de trabajo → *Task***.

Se abre el formulario para ingresar los datos de la tarea.



Los datos más relevantes en la información son:

- **Título:** Escoger un título significativo de la tarea, donde se describa de manera general lo que se va a ejecutar.
- **Actividad**: Define el área del ciclo de vida de la aplicación a la que pertenece la tarea.
- **Asignado a:** Define el usuario que tiene asignada la tarea.
- **Estado:** Define los posibles estados de la tarea.



- **Razón:** Causa de la modificación de la tarea.
- **Rango:** Ingrese un número que indica la importancia relativa de la tarea comparada con la de las demás tareas existentes.
- **Prioridad:** Define la importancia de la tarea, 1 siendo el más importante.
- **Área:** Defina el área de la empresa o el equipo al que pertenece el task.
- **Iteración:** Define la iteración en la que se encuentra el *task.*

- **Estimación original:** Número de horas de trabajo que tardará la tarea en completarse.
- **Completado:** Inicialmente cero, conforme se avanza hay que irlo modificando.
- **Restante:** Estimación original menos completado.
- **Descripción:** Opcional, detalla la tarea.

2.2. Casos de Prueba

¿Que es un caso de prueba?

Un caso de prueba (*test case*) es un conjunto de entradas, condiciones de ejecución, y resultados esperados, desarrollado con un objetivo específico (por ejemplo, ejercitar un determinado camino de un programa, o bien, verificar la conformidad de un programa contra un requerimiento específico).

Creación de casos de prueba

Para generar un nuevo caso de prueba ingrese al menú **Equipo → Nuevo Elemento de Trabajo → Test Case.**

Los datos más importantes para crear un *Test Case* son los siguientes:

- **Título:** Escoger un título significativo para el caso de prueba.
- **Asignado a:** Define el usuario que tiene asignado el caso de prueba.
- **Iteración:** Define la iteración en la que se encuentra el caso de prueba.
- **Estado de automatización:** Define si el caso de prueba está automatizado o no.

Asociación de un caso de prueba a una historia de usuario

- Para asociar un *test case* a un *user story*, seleccione la pestaña **Tested User Stories**.
- Vaya a botón Vincular → Examinar → Tailspin Toys →Team Queries → ProductPlanning → Buscar.
- Seleccione el *user story* del que se deriva el *test case*. Aceptar y Aceptar.
- Guarde el elemento de trabajo.

Definir los pasos de un caso de prueba

Para definir los pasos que comprenden un caso de prueba manual, vamos a la pestaña **Steps** → clic al botón "**Abrir para editar**". Esto nos abre la herramienta **Microsoft Test Manager**.

- En el Test Manager, vaya a: pestaña Organizar → pestaña Administrador de casos de prueba. Doble clic la prueba que acaba de crear.
- Inserte el primer paso del *test case*, para esto de clic en el botón "Insertar paso".
- Haga lo mismo para el resto de los pasos. Guardar y cerrar.

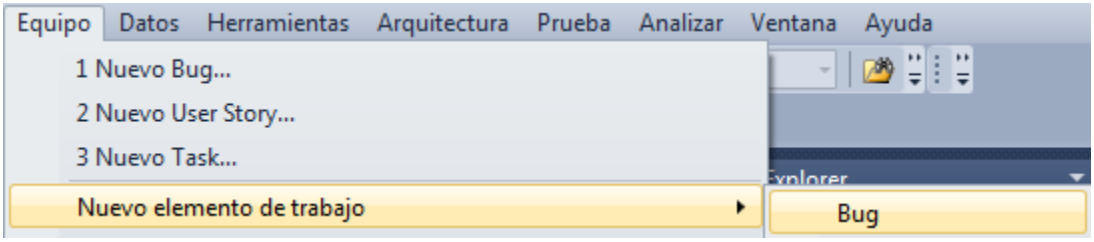
2.3. Incidentes

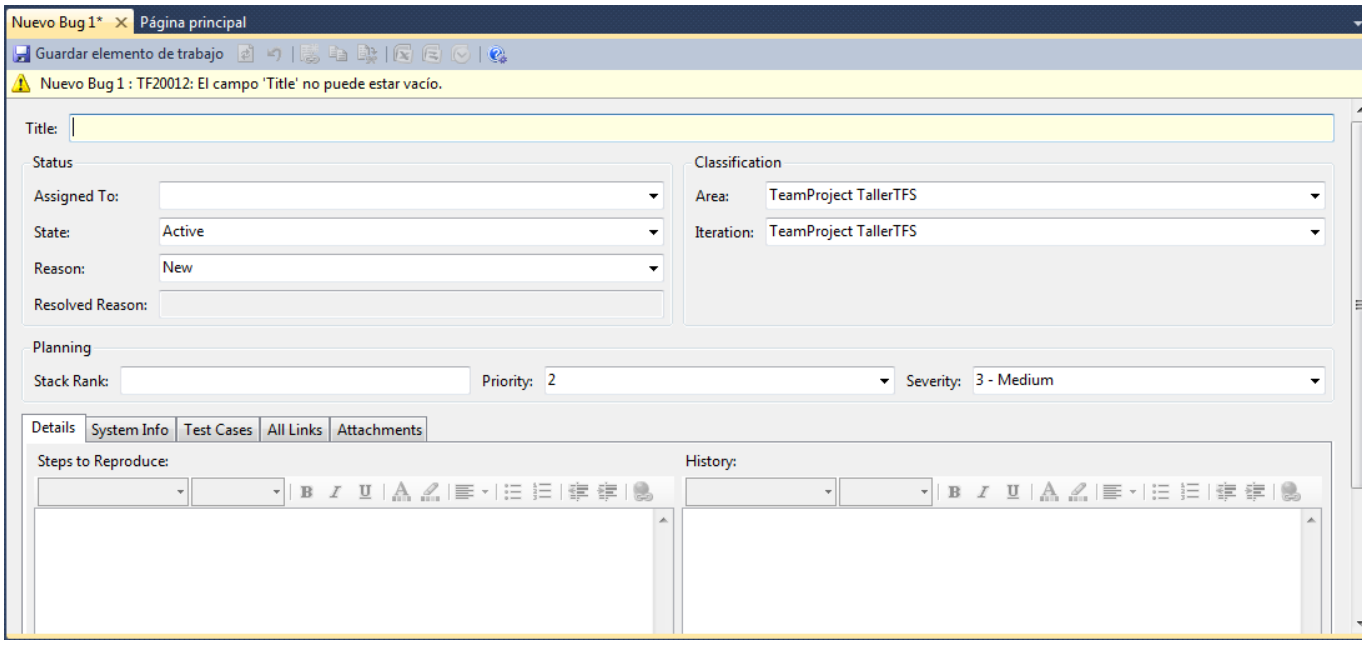
¿Que es un incidente?

Un incidente es una no-conformidad del sistema con respecto a su especificación. Más ampliamente se puede decir que un incidente es todo aquello que hace que el sistema deje de ser útil al usuario. También se le llama como error, falla, o defecto, y popularmente se le llama "pulga" (*bug*).

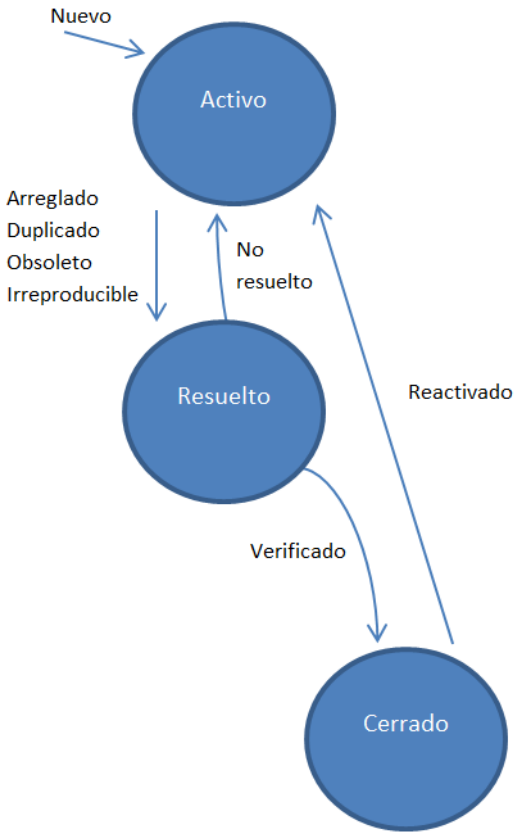
Creación de incidentes

Para generar un reporte de un incidente ingrese al menú **Equipo → Nuevo Elemento de Trabajo → Bug.**





- **Título**: Escriba un título adecuado en Título (representativo del bug).
- En la **clasificación** elija el área y la iteración a la que pertenezca el código o módulo en el que se encontró el bug.
- En el **estatus** seleccione inicialmente la persona a la que se le asignará el bug. Regularmente es la persona que programó el modulo en el que se encontró el problema. No obstante puede ser cualquier miembro del equipo de trabajo.
- El **estado** inicialmente es activo y la razón puede ser porque es nuevo o porque se presentó un error en el build de la aplicación. Durante el ciclo de vida del bug su estado puede cambiar por diversas razones, la siguiente imagen muestra los posibles estados y transiciones de un bug.



- La **prioridad** determina la velocidad con la que se considera necesario corregir el bug, puede variar de 1 a 4.
- Para asignar una **severidad** al bug, puede seleccionar entre crítico, alto, medio o bajo dependiendo de la severidad que considere adecuada.
- **Detalles:** En Detalles, especifique qué se debe hacer para reproducir el bug.
- **Información del sistema:** En Información del sistema debe especificar las características del sistema sobre el que se corrió la prueba cuando se generó el bug.
- **Casos de prueba:** Para vincular este bug a casos de prueba concretos.
- **Todos los vínculos:** Puede agregar archivos adjuntos como imágenes que prueben el bug o cualquier archivo digital.
- Para guardar el bug, haga clic en Guardar en la barra de herramientas.

Visualización de incidentes reportados

Los reportes de bugs deben monitorearse a lo largo del ciclo de vida del software. Por ejemplo, cada desarrollador debe revisar frecuentemente si existen bugs asociados a su usuario, cada líder de equipo debe revisar si hay bugs asignados a su equipo de trabajo, y cada *project manager* debe revisar si hay bugs asociados a su proyecto.

Para visualizar los bugs acceda a **Team Explorer → Team Project Taller TFS → Elementos de trabajo → Team Queries → My Bugs.**

En esta consulta se presentan los bugs asignados a su usuario únicamente y que no estén cerrados. Si lo que se desea es generar nuevas consultas relacionadas con los bugs, siga el mismo procedimiento que para las historias de usuario, generando una nueva consulta que muestre **TODOS** los bugs asociados a su usuario sin importar su estado, por ejemplo.

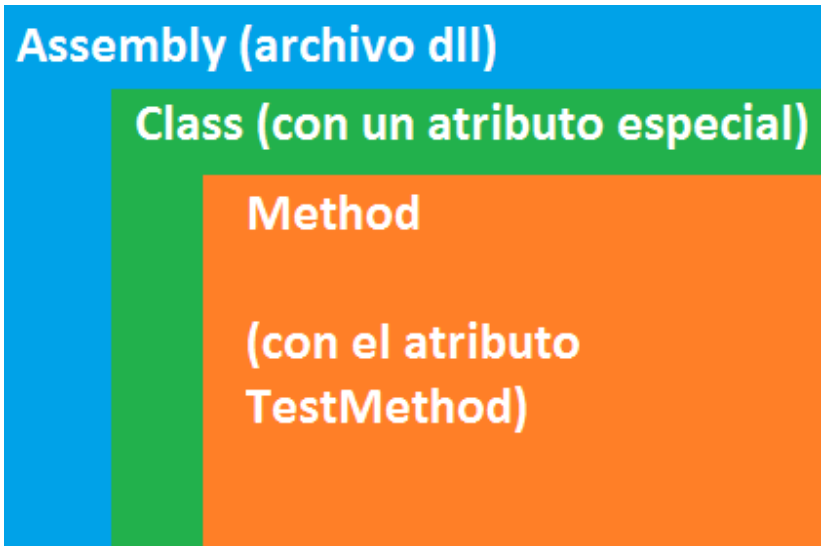
3. Pruebas Automatizadas

Antes de adentrarse en cómo automatizar pruebas con la herramienta, es recomendable conocer algunos conceptos básicos de Visual Studio en el área de pruebas automatizadas, que se presentan a continuación.

3.1. Conceptos Preliminares

Caso de prueba automatizado

Cada caso de prueba automatizado en Microsoft Visual Studio sin importar su tipo (prueba unitaria, prueba de interfaz de, etc) tiene una estructura como la siguiente:



Cada prueba automatizada corresponde a un método adornado con el atributo "TestMethod", el cual está contenido dentro de una clase con un atributo especial que indica el tipo de pruebas que contiene (unitarias, de interfaz gráfica, etc), que a su vez se encuentra dentro de un proyecto cuya salida será un assembly (para mayor información sobre assemblies consulte http://msdn.microsoft.com/en-us/library/k3677y81.aspx, para efectos de pruebas automatizadas un assembly será un archivo .dll con una o más clases compiladas).

El código contenido dentro de cada prueba puede ser generado automáticamente mediante Visual Studio o escrito manualmente.

Ejecución de pruebas automatizadas

Ejecutor de pruebas

Una vez que la o las pruebas fueron creadas y compiladas en uno o más assemblies, un programa denominado "ejecutor de pruebas" (MSTest.exe es el nombre de la implementación para Windows), se encarga de procesar cada uno de los assemblies y ejecutar las pruebas automatizadas contenidas en ellas.

Este programa de línea de comandos normalmente es desconocido por el "tester", pues es ejecutado automáticamente por Microsoft Visual Studio (de manera similar como el IDE ejecuta el compilador cuando lo necesita).

Orden de ejecución de las pruebas automatizadas

Como las pruebas en la teoría son independientes, es decir, el resultado de una prueba no afecta a la ejecución o resultado de otra, el ejecutor de pruebas ejecuta las pruebas automatizadas sin ningún orden en particular con las siguientes excepciones:

- Una vez seleccionado un assembly, no se selecciona otro hasta que se hayan ejecutado todas las pruebas contenidas en él.
- Una vez seleccionada una clase, no se selecciona otra hasta que se hayan ejecutado todas las pruebas contenidas en ella.

Preparación y limpieza de ambiente para las pruebas

Como cada prueba automatizada requiere que la aplicación esté en un estado o ambiente en particular, a veces es necesario preparar el ambiente antes de la ejecución y limpiarlo luego de la ejecución.

Estas preparaciones se realizan con métodos en las clases de los assemblies adornados con atributos especiales con los sufijos "Initialize" y "Cleanup". Las preparaciones y limpiezas de ambiente se pueden realizar en tres niveles distintos:

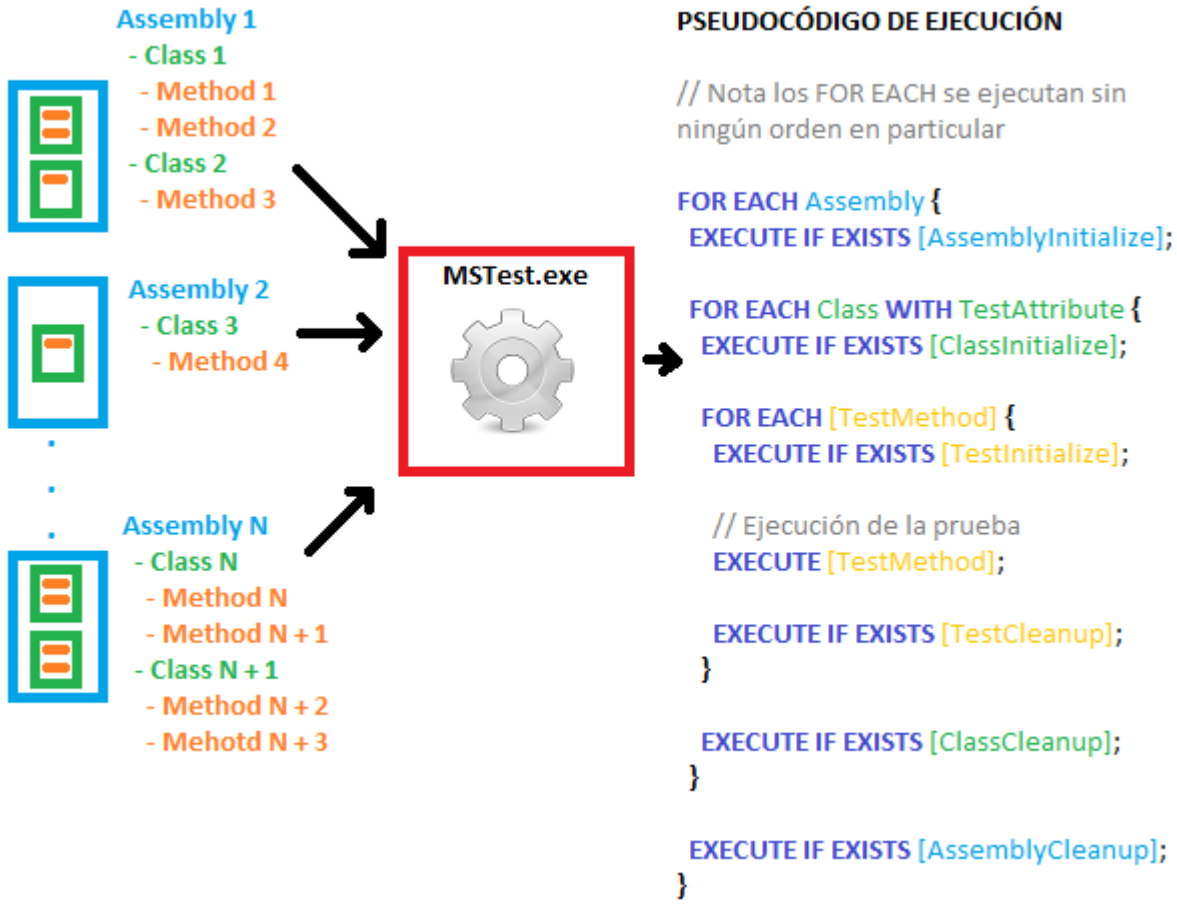
- Assembly:
    - **[AssemblyInitialize]:** Método que se ejecuta antes de la ejecución de cualquier prueba dentro del assembly seleccionado. Se ejecuta una vez por assembly. Sólo puede haber un método con este atributo por assembly.
    - **[AssemblyCleanup]:** Método que se ejecuta después de la ejecución de todas las pruebas dentro del assembly seleccionado. Se ejecuta una vez por assembly. Sólo puede haber un método con este atributo por assembly.

- Clase:
    - **[ClassInitialize]:** Método que se ejecuta antes de la ejecución de cualquier prueba dentro de la clase seleccionada. Se ejecuta una vez por clase. Sólo puede haber un método con este atributo por clase.
    - **[ClassCleanup]:** Método que se ejecuta después de la ejecución de todas las pruebas dentro de la clase seleccionada. Se ejecuta una vez por clase. Sólo puede haber un método con este atributo por clase.
- Método:
    - **[TestInitialize]:** Método que se ejecuta antes de la ejecución de cada prueba dentro de la clase seleccionada. Se ejecuta una vez por cada método de prueba dentro de la clase. Sólo puede haber un método con este atributo por clase.
    - **[TestCleanup]:** Método que se ejecuta después de la ejecución de cada prueba dentro de la clase seleccionada. Se ejecuta una vez por cada método de prueba dentro de la clase. Sólo puede haber un método con este atributo por clase.

Tip:

Los métodos de preparación y limpieza pueden ser muy útiles. Por ejemplo en un conjunto de pruebas automatizadas web, la preparación a nivel de assembly puede montar el servidor web y el servidor de base de datos de prueba; la preparación a nivel de clase puede crear las tablas en la base de datos que utilizarán las pruebas dentro de la clase; y la preparación a nivel de método puede llenar la base de datos con los datos de prueba necesarios para la ejecución de las mismas y abrir el navegador. La limpieza, por su parte, cierra el navegador, elimina las tablas y baja los servidores.

Para comprender de una manera más clara la ejecución de las pruebas automatizadas en Visual Studio se presenta el siguiente diagrama:



Criterio de éxito de una prueba automatizada

Todo caso de prueba tiene un criterio de éxito, es decir una o varias condiciones que determinan si un caso de prueba pasó o falló. En Visual Studio un caso de prueba automatizado se considera exitoso si logró ejecutarse sin arrojar ninguna excepción.

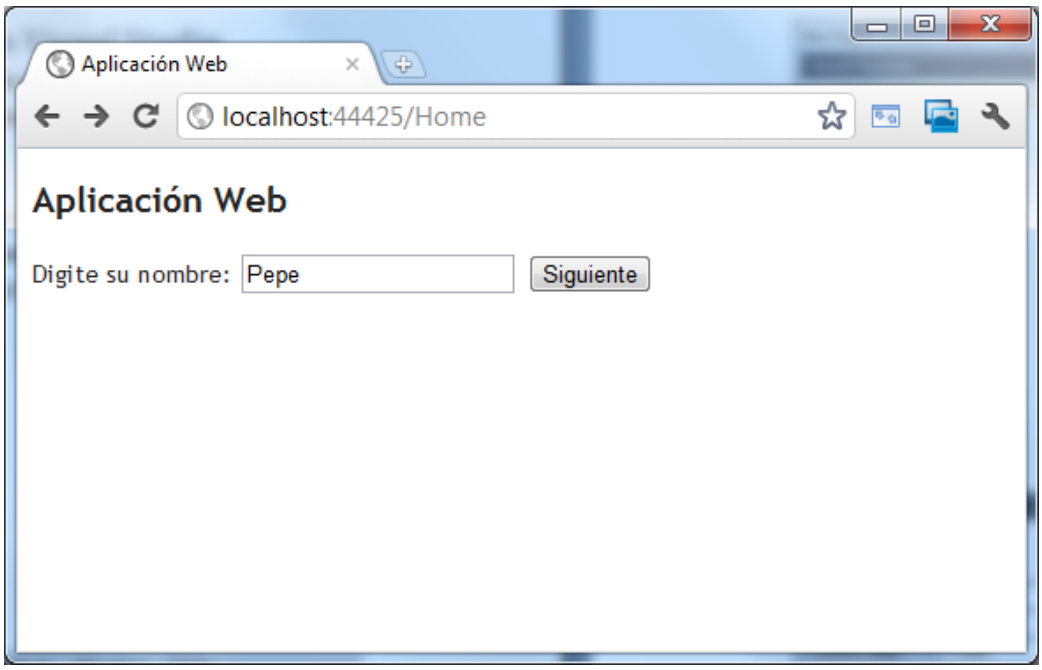
Para simular los "resultados esperados" de los casos de prueba, se utilizan métodos que clases del framework que verifican condiciones especificadas por el "tester" y arrojan una excepción si no se cumple la condición. De esta manera el caso de prueba fallará si la condición no se cumple.

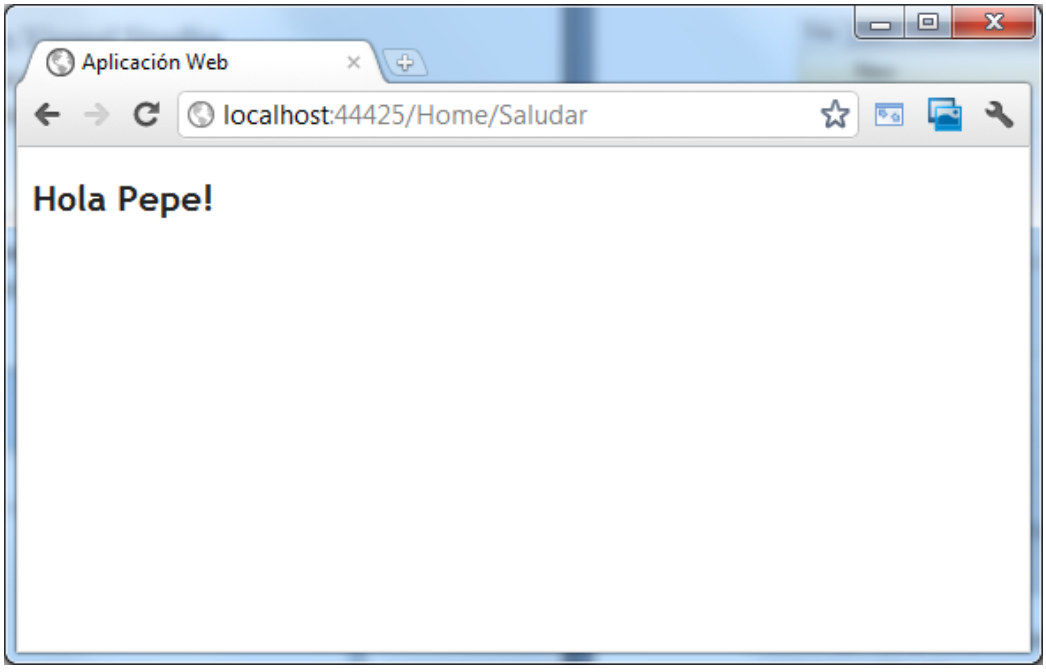
3.2. Generación de Pruebas Web Automatizadas

Una vez explicados los conceptos preliminares, procederemos a describir cómo generar pruebas web automatizadas en Visual Studio.

Aplicación de ejemplo

Para poder explicar más concretamente cómo realizar pruebas web automatizadas en Visual Studio se harán pruebas a una aplicación de prueba extremadamente sencilla. La aplicación únicamente consta de dos páginas: en la primera página solicita el nombre del usuario y en la siguiente página saluda al usuario por su nombre, como se muestra a continuación:





Pruebas automatizadas de interfaz de usuario

Las pruebas de interfaz de usuario en Visual Studio se llaman "*CodedUI Tests*". Visual Studio no hace diferenciación si la aplicación a probar es de escritorio o web.

Búsqueda de controles independiente de resolución de pantalla y posición

A diferencia de las herramientas antiguas para hacer pruebas de interfaz automatizadas, las pruebas automatizadas de interfaz de usuario en Visual Studio no dependen de la resolución de la pantalla ni de la posición de la ventana de la aplicación a probar. Es decir, las acciones del mouse no están ligadas a coordenadas (x, y) de pixeles sino que más bien, utilizan unas tecnologías más sofisticadas para poder "encontrar" el control con el que se debe interactuar, independientemente de donde se encuentre en la pantalla.

Las tecnologías utilizadas para hacer esto posible son:

- **Microsoft Active Accessiblity (MSAA):** Utilizada para encontrar ventanas y controles de aplicaciones de escritorio clásicas.
- **Internet Explorer Document Object Model (IE DOM):** Utilizada para encontrar controles de aplicaciones web dentro de una ventana de Internet Explorer.
- **User Interface Automation (UIA):** Utilizada para encontrar controles de aplicaciones de escritorio y web de generación XAML: Microsoft Presentation Foundation (WPA) y Microsoft Silverlight.

Tip:

*Al ver la tecnología IE DOM, se deduce que únicamente se pueden hacer pruebas automatizadas de aplicaciones web con Internet Explorer, otro inconveniente es que únicamente la versión de 32 bit es soportada. No obstante, Microsoft liberó una actualización de Visual Studio llamada "**Microsoft Visual Studio 2010 Feature Pack 2**" la cual permite hacer pruebas automatizadas de aplicaciones web con Mozilla Firefox (para más información consulte http://msdn.microsoft.com/en-us/library/gg269474.aspx).*

Los archivos .uitest

Las pruebas automatizadas de interfaz de usuario se crean de una manera un poco diferente de las demás prueba automatizadas en Visual Studio. Estas pruebas se crean en su mayor parte con código autogenerado, mediante una herramienta llamada "Coded UI Test Builder". Esta herramienta permite realizar dos cosas:

- Grabar una o más acciones que el "tester" realiza sobre la aplicación a probar y almacenarlas para su posterior ejecución.
- Almacenar un conjunto de verificaciones sobre las propiedades visibles e invisibles de uno o más controles de la aplicación a probar y de igual manera poder ejecutar las verificaciones posteriormente.

Los conjuntos de acciones o verificaciones son codificados automáticamente en métodos y clases por la herramienta, los cuales luego son compilados en uno o más assemblies para su ejecución. Para realizar esta labor, la herramienta crea un archivo especial de extensión .uitest.

Los archivos .uitest son archivos XML independiente del lenguaje de programación en que se codifica la prueba automatizada, los cuales almacenan la siguiente información:

- Los conjuntos de acciones que se convertirán en métodos de acción.
- Los conjuntos de verificaciones que se convertirán en métodos de verificación.
- El "mapa" de la interfaz de usuario de la aplicación sobre la que se realizan las acciones y verificaciones para que el ejecutor de pruebas pueda encontrar los controles a la hora de ejecutar las pruebas.

> ***Tip:***
>
> *El archivo más importante generado por la herramienta "Coded UI Test Builder" es el .uitest; sin embargo, la herramienta genera dos archivos más:*
>
> *1- Un archivo .designer.cs que contiene el código autogenerado derivado del archivo .uitest, el cual será compilado y luego ejecutado por el ejecutor de pruebas.*
> *2- Un archivo .cs que permite realizar modificaciones al archivo .designer.cs sin que se pierdan los cambios en caso de que se regenere el código autogenerado.*

Fallo de pruebas automatizadas por cambios en la interfaz de usuario

Anteriormente se mencionó que el criterio de éxito de una prueba automatizada en Visual Studio es si ésta se logra ejecutar sin que se arroje ninguna excepción. Naturalmente, si un conjunto de verificaciones falla, se arrojará una excepción que ocasionará que la prueba falle, no obstante existe otra situación que puede causar que una prueba de este tipo falle.

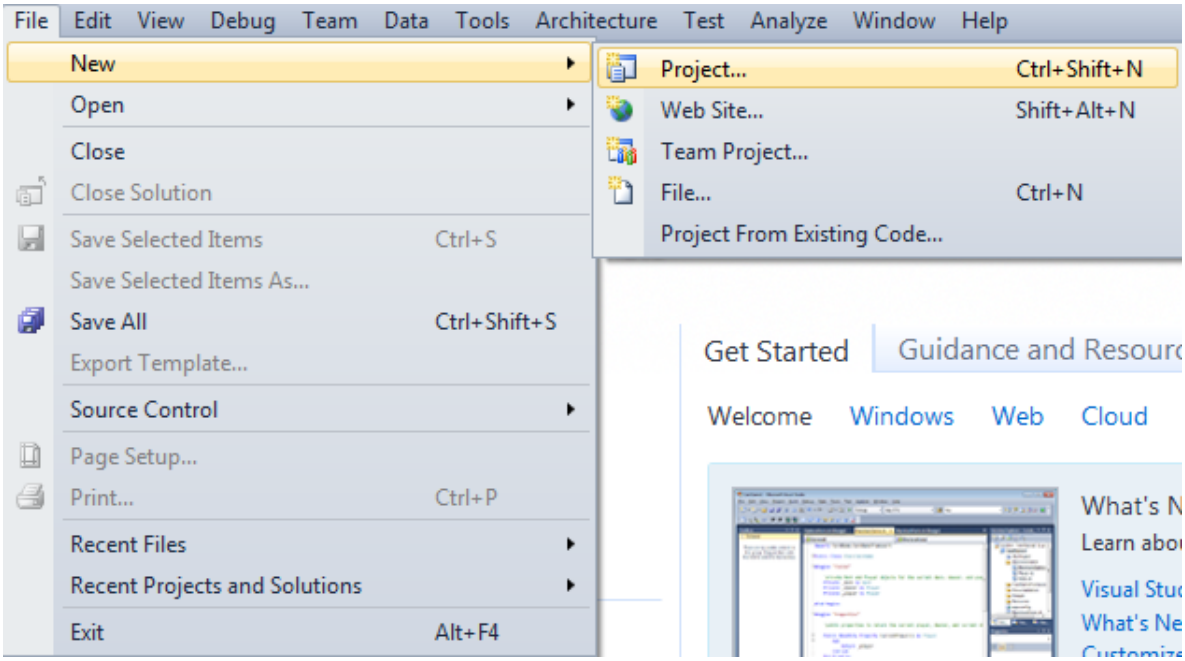
El ejecutor de pruebas utiliza las propiedades de los controles, tales como el id, el tipo de control, el control contenedor, entre otros, para "encontrar" un control en la aplicación bajo

prueba y realizar una acción o verificación sobre éste. Esta información es conocida como el "mapa" de la interfaz de usuario que se almacena en el archivo .uitest. Si estas propiedades sufren cambios bruscos o el control es removido de la aplicación, el ejecutor de pruebas no podrá encontrar el control y automáticamente arrojará una excepción, lo cual causará que la prueba falle.

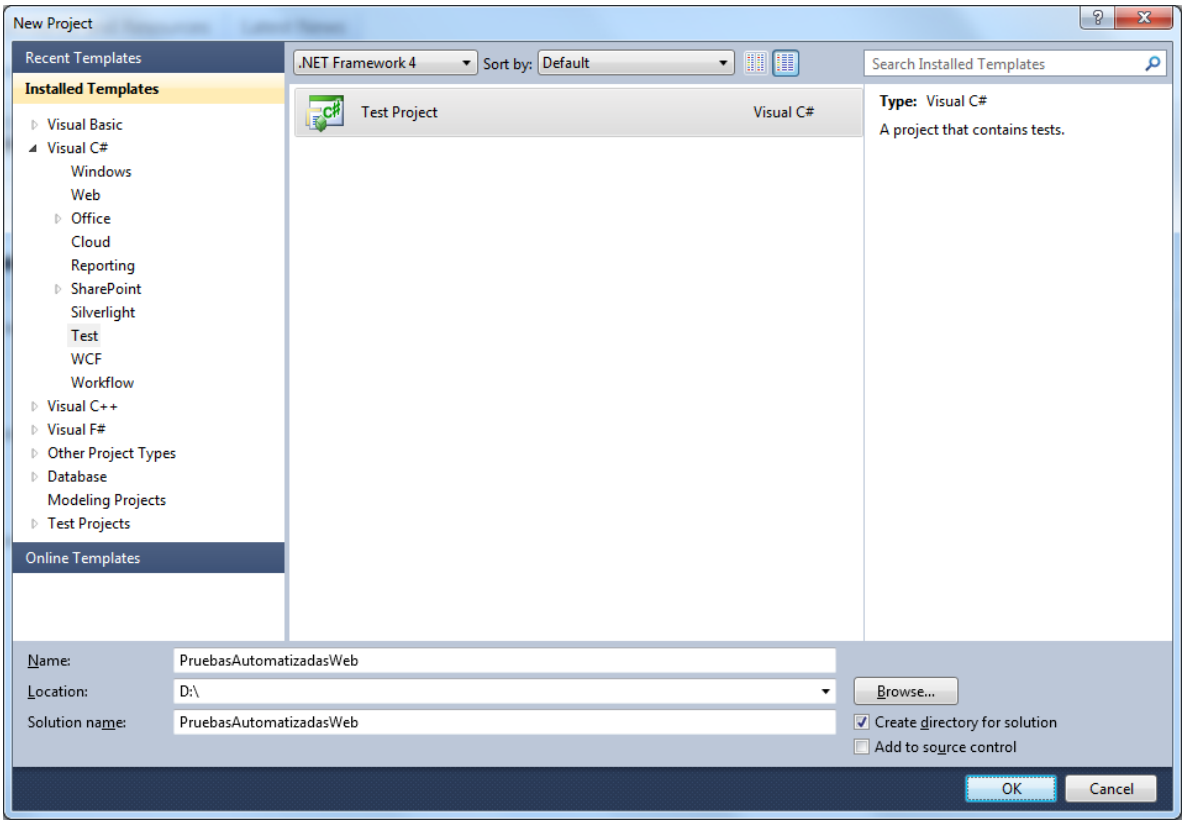
> ***Tip:***
>
> *No se recomienda realizar pruebas automatizadas de interfaz de usuario si la interfaz es muy propensa a sufrir cambios constantemente, pues el costo de mantenimiento de las pruebas automatizadas será muy alto.*

Crear un proyecto de pruebas

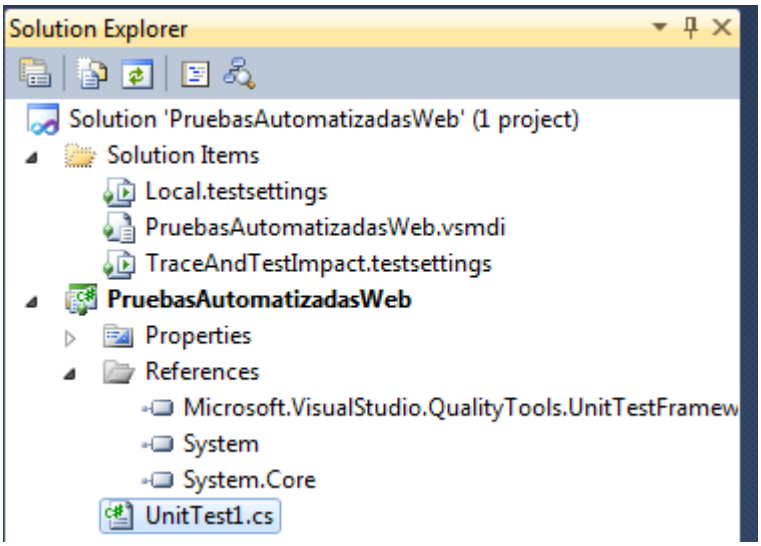
1. Abrir Visual Studio.
2. Seleccionar "File" -> "New" -> "Project".



3. En la columna izquierda, seleccionar el idioma "Visual Basic" o "C#" y luego seleccionar "Test".
4. Digitar nombres para la solución y el proyecto.
5. Presionar el botón "Ok".



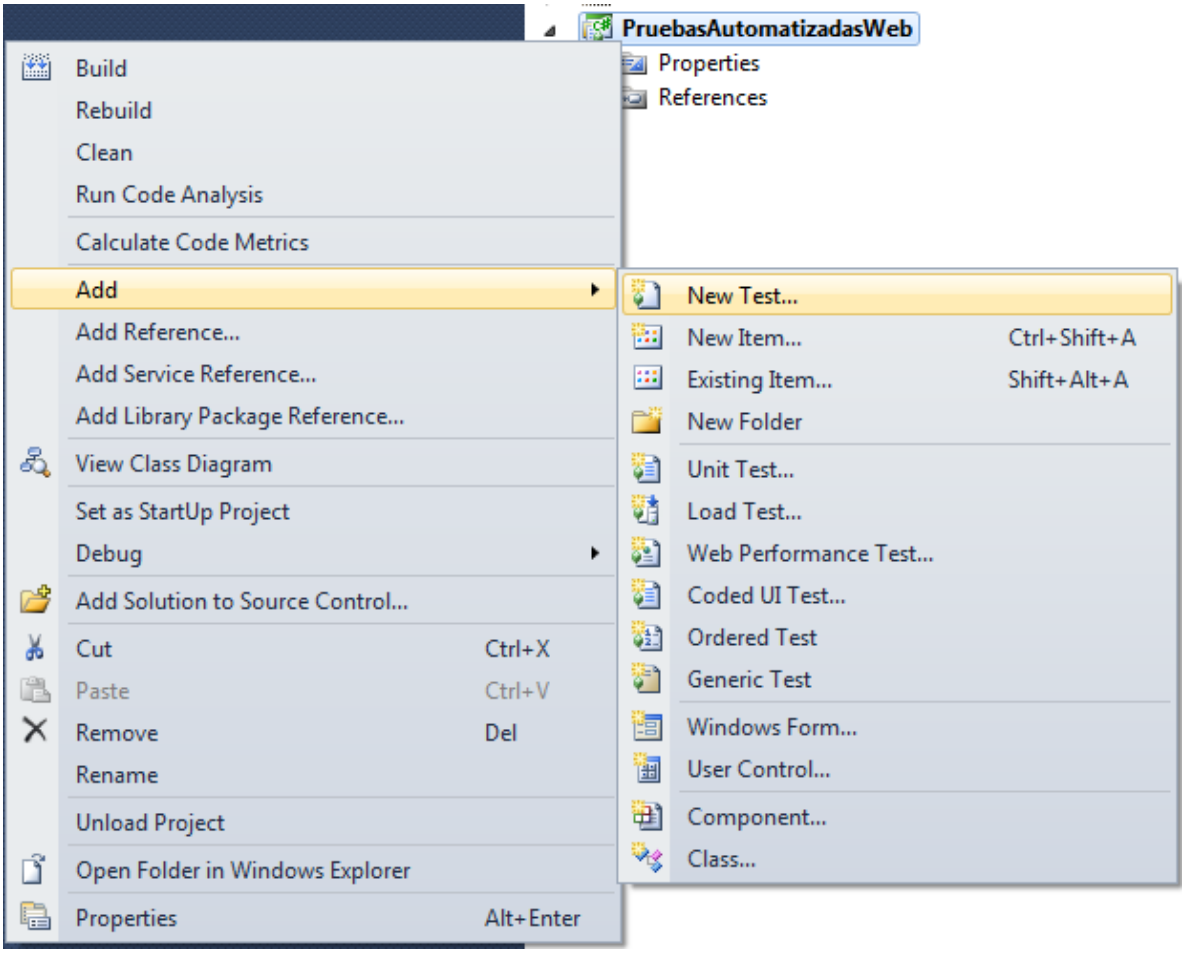
6. Eliminar el archivo autogenerado "UnitTest1.cs".



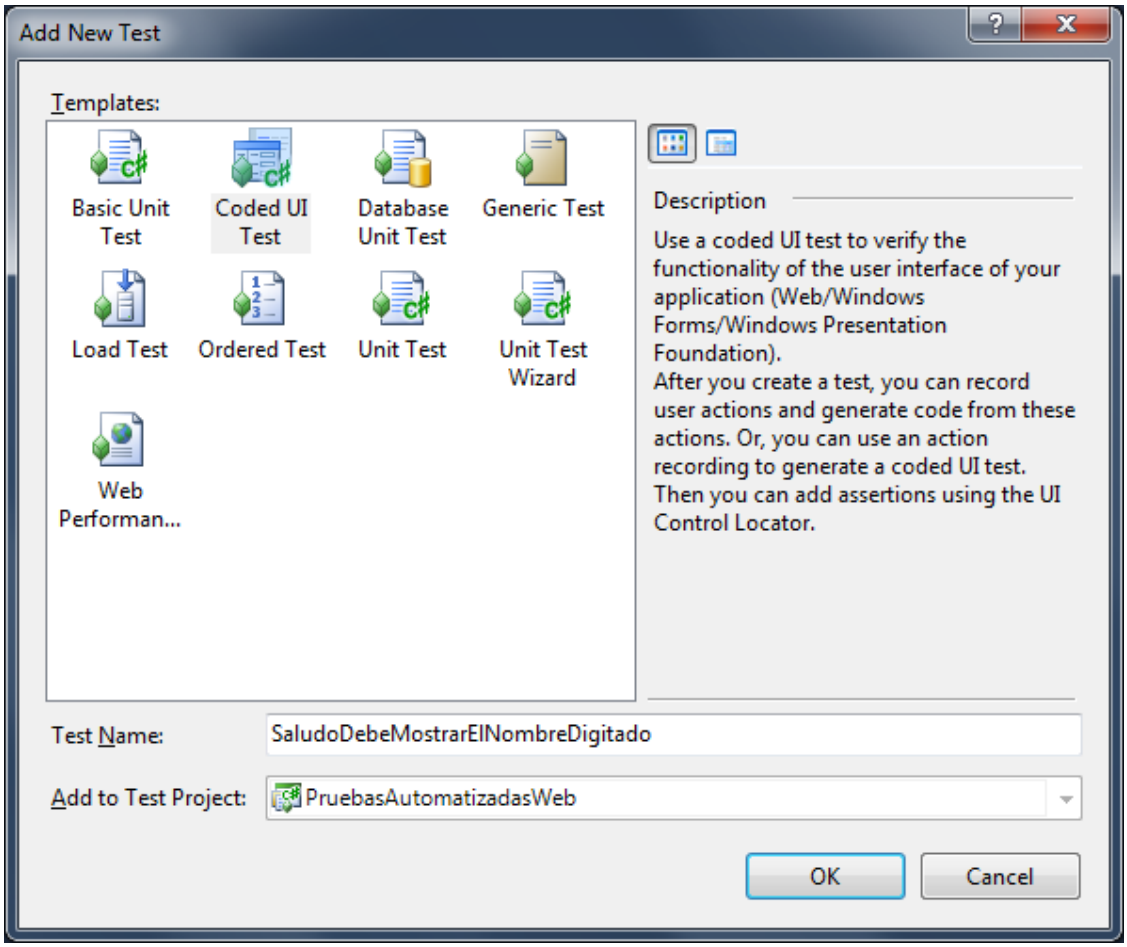
Crear una prueba automatizada web

Una vez creado el proyecto de pruebas, se pueden empezar a crear pruebas automatizadas. Para crear una prueba automatizada web:

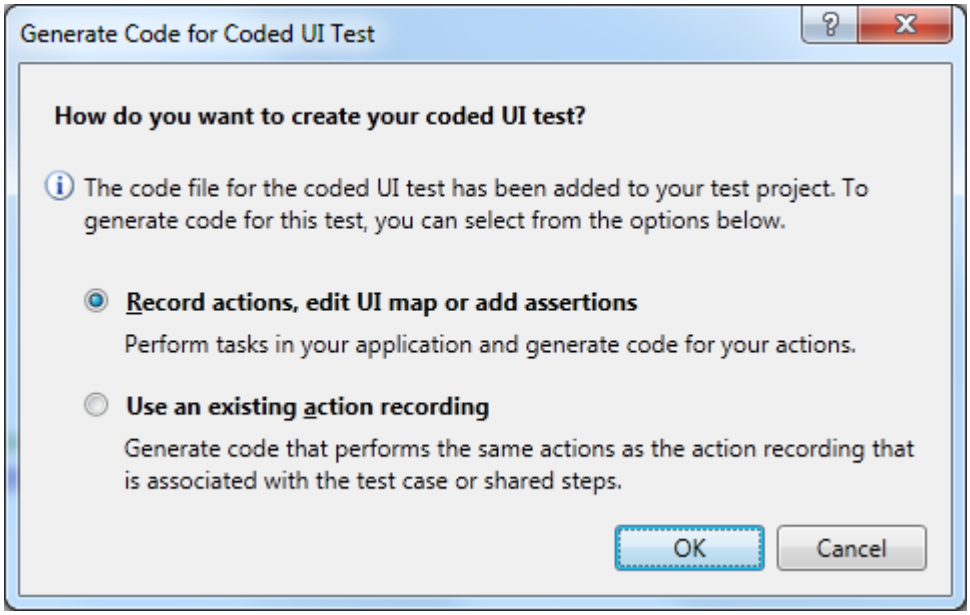
1. Hacer clic derecho en el proyecto.
2. Seleccionar "Add" -> "New Test".



3. Seleccionar "Coded UI Test".
4. Digitar un nombre para la prueba.
5. Presionar el botón "Ok".



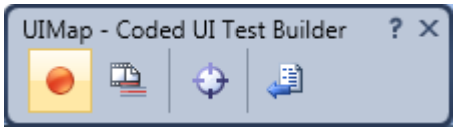
6. Seleccionar la opción "Record actions, edit UI map or add assertions".
7. Presionar el botón "Ok".



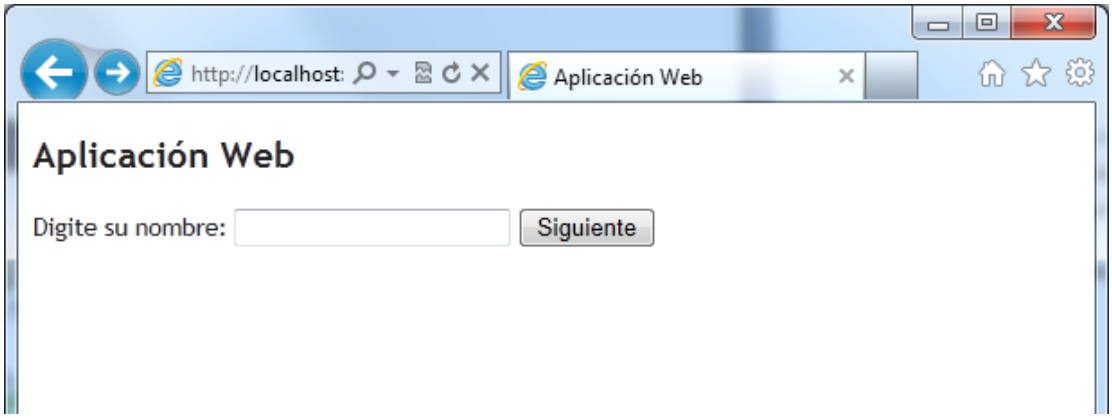
Crear un conjunto de acciones

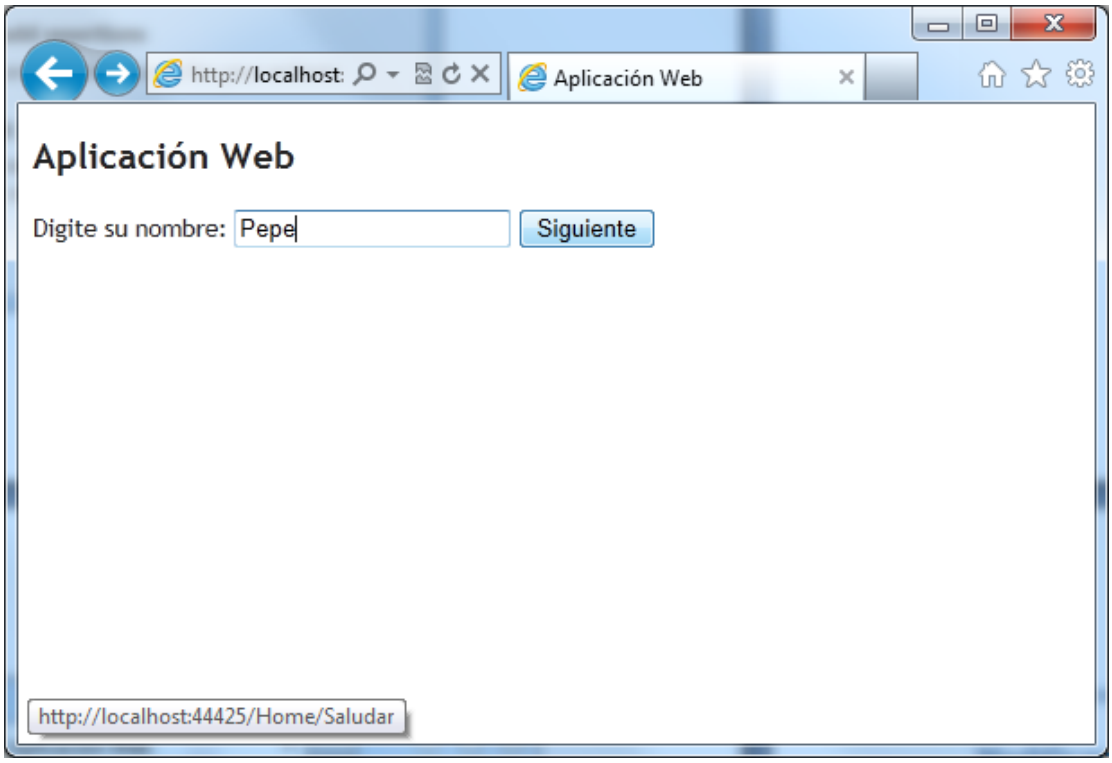
Una vez creada la prueba automatizada, se iniciará la herramienta "Coded UI Test Builder" automáticamente. Para crear un conjunto de acciones se deben seguir los siguientes pasos:

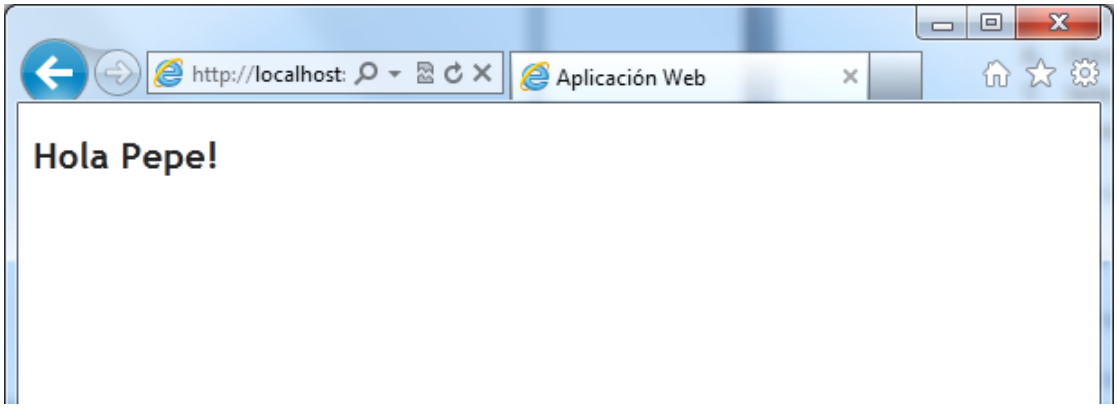
1. Seleccionar el ícono con el círculo rojo (Start Recording).



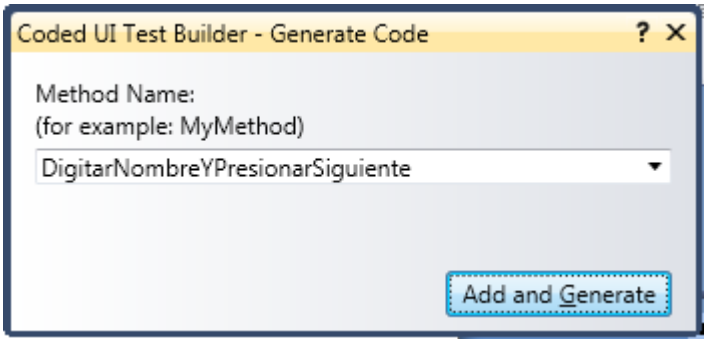
2. Realizar un conjunto de acciones.







3. Seleccionar el ícono de la página de código con la flecha azul (Generate Code).
4. Poner un nombre al método en el cual se guardarán las acciones.
5. Presionar el botón "Add and Generate".

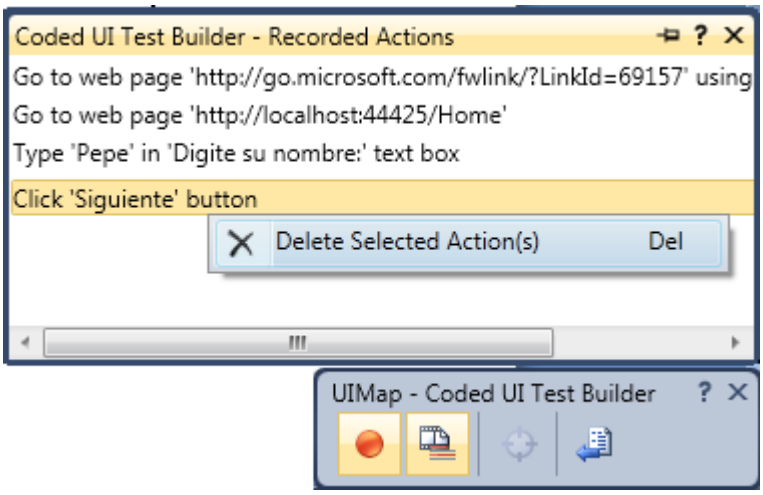


Tip:

El botón con el círculo rojo se transforma en un botón de pausa apenas comienza la grabación. Esto permite pausar la grabación de acciones en cualquier momento para luego reanudar la grabación.

Tip:

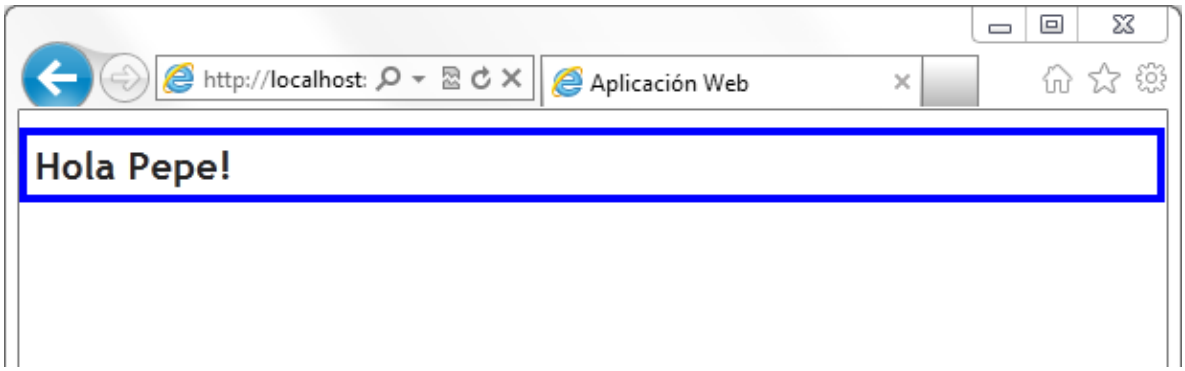
En cualquier momento durante una grabación de acciones, se pueden consultar las acciones grabadas mediante el botón con una película (Show Recorded Steps). Esto permite eliminar acciones que se hicieron accidentalmente y que no se desea que formen parte de la prueba haciendo clic derecho sobre la acción deseada y seleccionando "Delete selected Action(s)".



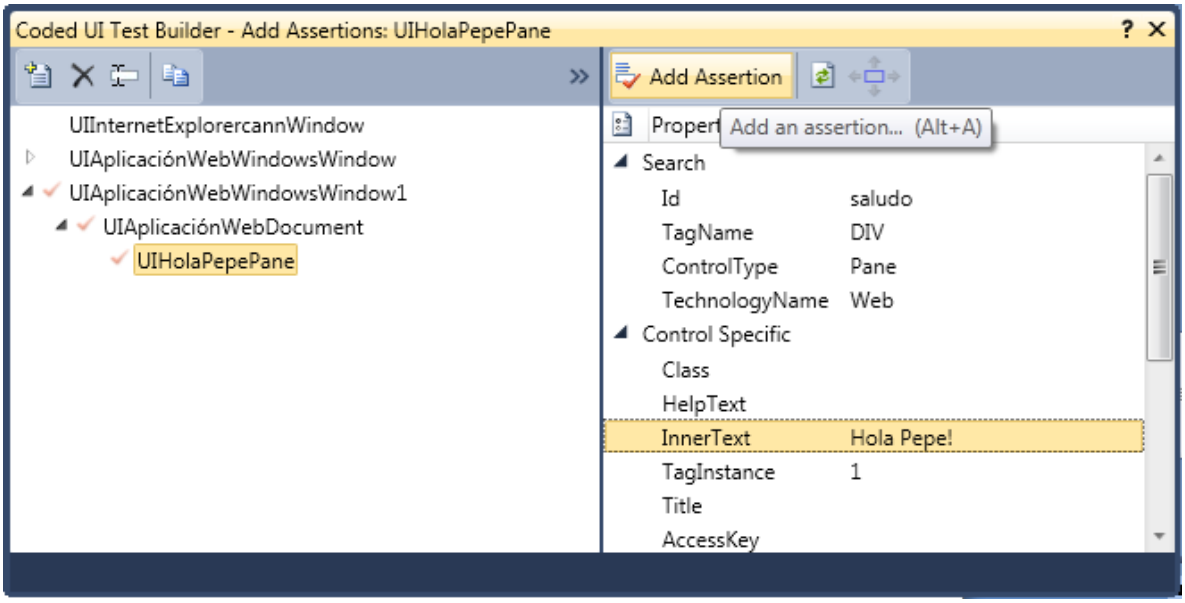
Crear un conjunto de verificaciones

Una prueba con sólo acciones y sin verificaciones carece un poco de sentido. Para agregar un conjunto de verificaciones, se deben seguir los siguientes pasos:

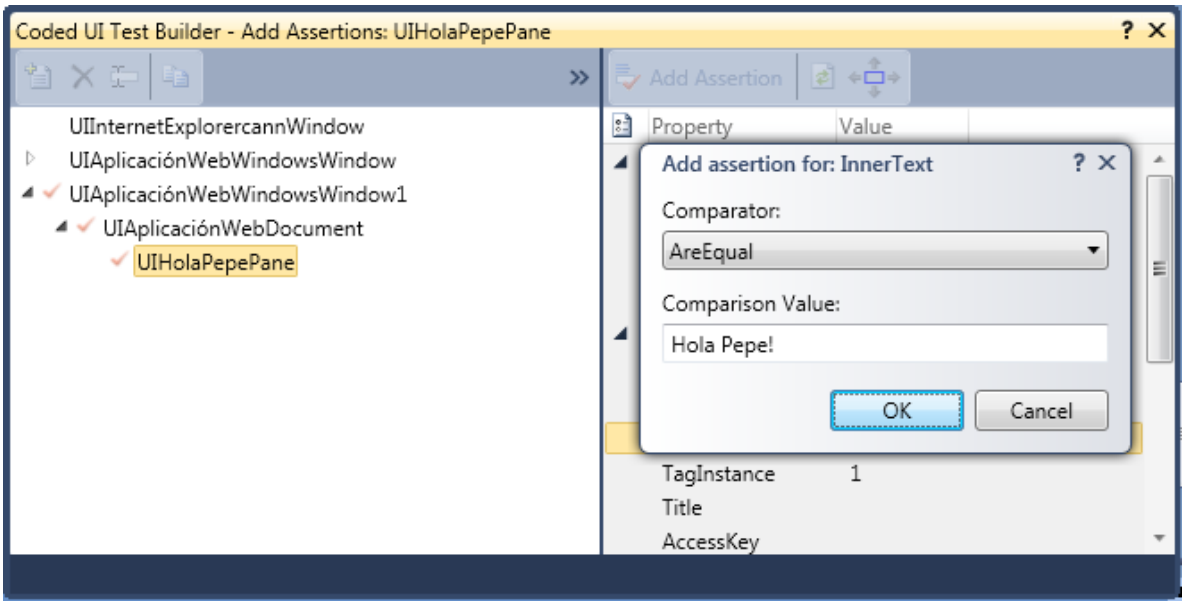
1. Presionar el botón con el ícono con la mira (Add assertion) y sin soltar el clic, arrastrar la mira hasta el control que se le desean hacer las verificaciones.



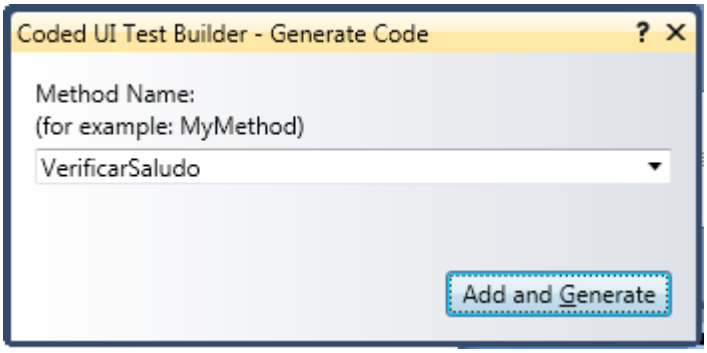
2. Seleccionar la propiedad que se deseé verificar y presionar el botón "Add Assertion".



3. Seleccionar el tipo de comparador que se desee utilizar, digitar el valor esperado y presionar el botón "Ok".



6. Seleccionar el ícono de la página de código con la flecha azul (Generate Code).
7. Poner un nombre al método en el cual se guardarán las verificaciones.
8. Presionar el botón "Add and Generate".



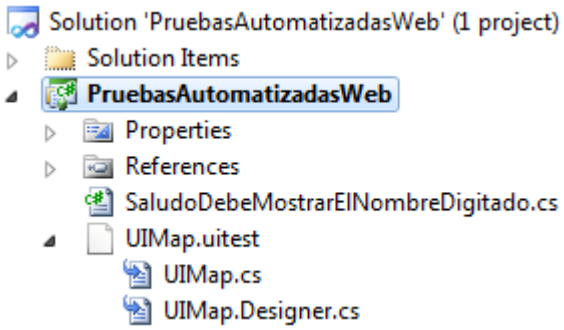
Tip:

Se puede aplicar más de una validación al mismo control por método, simplemente se deben repetir los pasos 1-3 cuantas veces como se desee. También se puede agregar más de una validación a la misma propiedad de un control.

Tip:

El árbol jerárquico que se muestra en la ventana de agregación de validación representa al mapa de la interfaz de usuario. Éste mapa representa la jerarquía de controles de la aplicación a probar. Se puede navegar a través de este árbol en esta ventana.

Una vez se hayan agregado todas los conjuntos de acciones y verificaciones deseados, se debe cerrar la herramienta "Coded UI Test Builder". Se generarán los siguientes archivos:



- El archivo UIMap.uitest, contiene los conjuntos de acciones y verificaciones recién grabadas y el mapa de la aplicación a probar. Además se crean los dos archivos derivados del archivo .uitest.
- El archivo "SaludoDebeMostrarElNombreDigitado.cs" contiene la clase de pruebas que será procesada por el ejecutor de pruebas. A continuación se muestra el código de este archivo (el código ha sido simplificado para efectos didácticos):

```
namespace PruebasAutomatizadasWeb
{
    [CodedUITest]
    public class SaludoDebeMostrarElNombreDigitado
    {
        [TestMethod]
        public void CodedUITestMethod1()
        {
            this.UIMap.DigitarNombreYPresionarSiguiente();
            this.UIMap.VerificarSaludo();
        }

        public UIMap UIMap
        {
            get
            {
                if ((this.map == null))
                {
                    this.map = new UIMap();
                }

                return this.map;
            }
        }

        private UIMap map;
    }
}
```

Esta clase no es dependiente del archivo .uitest, por lo que los cambios que se realicen aquí, no se perderán si se regenera el código.

El método "CodedUITestMethod1" representa a la propia prueba que será ejecutada por el ejecutor de pruebas. La propiedad UIMap de tipo UIMap representa a la clase autogenerada por la herramienta "Coded UI Test Builder" la cual contiene el código que realiza las acciones y las verificaciones. En el método de prueba se pueden editar el orden en que se ejecutan los métodos, eliminar métodos o agregar métodos que se generen posteriormente.

Modificar un archivo .uitest

¿Qué sucede si se debe modificar un archivo .uitest ya creado? ¿Cómo se procede si se debe agregar nuevos conjuntos de acciones o verificaciones o modificar conjuntos de acciones o verificaciones existentes? Para hacer esto es necesario modificar el archivo .uitest.

Se puede modificar un archivo .uitest de tres maneras:

- **Modificar el archivo directamente con un editor de texto:** Poco recomendado, muy riesgoso pues cualquier pequeño error puede dañar el archivo dejándolo inservible.
- **Modificar el archivo con la herramienta "Coded UI Test Builder":** Recomendado, no obstante no se puede modificar conjuntos de acciones existentes, sólo se pueden remplazar por completo.
- **Modificar el archivo con un editor especial:** Recomendado, no obstante requiere que el Microsoft Visual Studio 2010 Feature Pack 2 esté instalado.

AGREGAR UN NUEVO CONJUNTO DE ACCIONES O VERIFICACIONES

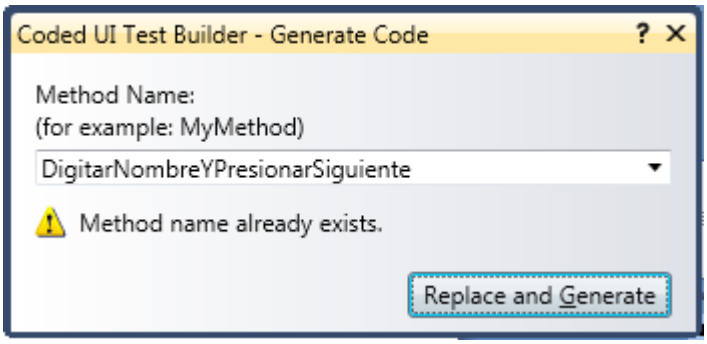
Para agregar un nuevo conjunto de acciones o verificaciones se deben seguir los siguientes pasos:

1. Hacer clic derecho en el archivo .uitest.
2. Seleccionar la opción "Edit With Coded UI Test Builder".
3. Proceder de la misma manera que se enseñó anteriormente para agregar conjuntos de acciones o verificaciones.

MODIFICAR UN CONJUNTO DE ACCIONES O VERIFICACIONES EXISTENTE

Para modificar un conjunto de acciones o verificaciones existente se deben seguir los pasos de la creación de conjuntos de acciones o verificaciones con la pequeña diferencia que a la

hora de ponerle nombre al método a generar, se debe poner el nombre del método existente que se desee remplazar. La ventana mostrará una advertencia de que el método ya existe y será remplazado.

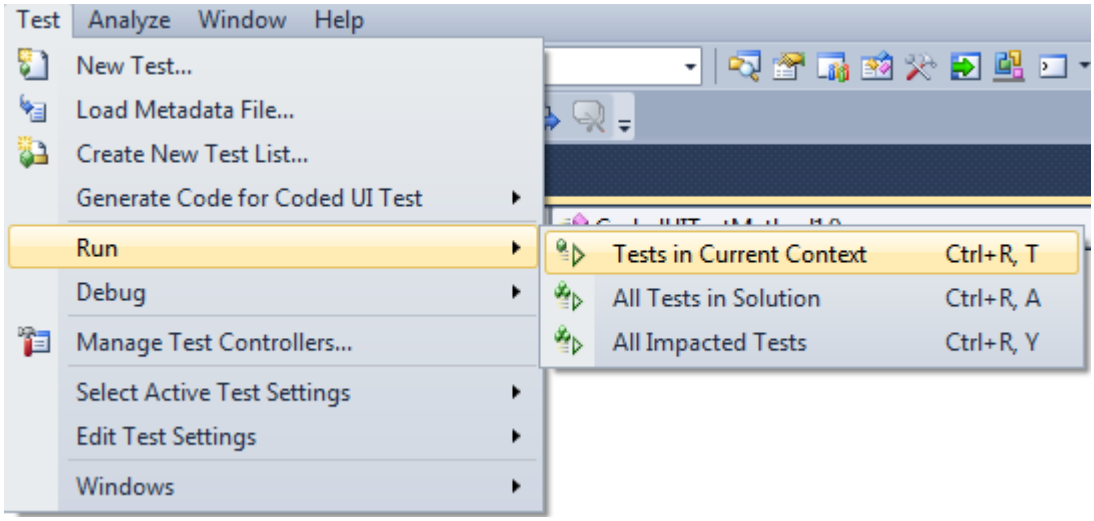


Ejecutar una prueba web automatizada

La ejecución de pruebas automatizadas se puede hacer de muchas maneras, en este reporte se enseñará cómo ejecutarlas directamente desde Visual Studio, pues será la manera más común de realizarlas.

Para ejecutar una prueba automatizada individual:

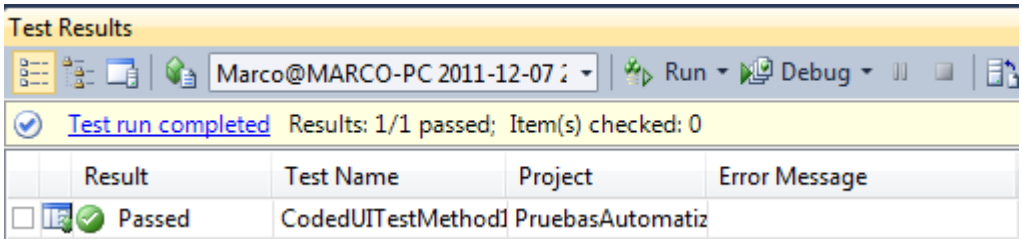
1. Posicione el cursor en el cuerpo de la prueba que se desee ejecutar.
2. Seleccione la opción de menú "Test" -> "Run" -> "Tests in Current Context".



Para ejecutar todas las pruebas automatizadas:

1. Seleccionar la opción de menú "Test" -> "Run" -> "All Tests in Solution".

Una vez se corra la prueba, se mostrará una ventana con el(los) resultado(s) de la(s) prueba(s):



Si una prueba pasa, tendrá un ícono verde con la frase "Passed", si una prueba falla tendrá un ícono rojo con la frase "Failed". Al hacer doble clic en la prueba fallida se podrá ver información más detallada de porqué la prueba falló.

Tip:

Si las pruebas automatizadas se guardan en control de código fuente de Visual Studio Team System y se programa un Team Build, las pruebas pueden ejecutarse automáticamente luego de ejecutarse el Team Build. Para más información consulte http://msdn.microsoft.com/en-us/library/dd793166.aspx.

Tip:

Cuando se ejecuta una prueba automatizada web, se abre una instancia de Internet Explorer que luego debe ser cerrada, de lo contrario la memoria del computador se acabará por el exceso de ventanas de Internet Explorer abiertas. Para asegurarse que siempre se cierren las ventanas de Internet Explorer y que no se quede sin memoria el computador se puede aprovechar el método con el atributo [TestCleanup]. Simplemente créese un conjunto de acciones cuya única acción sea cerrar el navegador y póngase el llamado de este método dentro del método con el atributo [TestCleanup]. De esta manera, aunque una prueba falle, siempre se asegurará de que la ventana de Internet Explorer se cierre.

Tip:

Las pruebas automatizadas en páginas web que abren muchas conexiones asíncronas pueden ejecutarse extremadamente lentas (un ejemplo de este tipo de páginas es la página principal de Google). La razón de esto es porque el ejecutor de pruebas por defecto no realiza la siguiente acción hasta que todas las conexiones que se hayan realizado hayan finalizado, incluyendo las asíncronas. La lógica detrás de esta medida es impedir que erróneamente se declare un control como que no se puede encontrar porque todavía no ha terminado de responder el servidor, no obstante conexiones asíncronas que retornan interfaz de usuario puede ralentizar enormemente su ejecución.

Se recomienda entonces analizar la ejecución de cada prueba automatizada y utilizar dos propiedades de una clase especial llamada "PlayBack" para aminorar este problema:

La instrucción:
Playback.PlaybackSettings.WaitForReadyLevel = WaitForReadyLevel.Disabled;
Evita que el ejecutor de pruebas espere a que todas las conexiones hayan finalizado para empezar a buscar los controles para ejecutar las acciones o verificaciones.

La propiedad:

Playback.PlaybackSettings.SearchTimeout

Permite cambiar el tiempo máximo (en milisegundos) que dura el ejecutor de pruebas buscando un control antes de declararse como no encontrado.

Se debe ajustar este tiempo según cada ambiente para que no se arrojen excepciones. Aunque en la mayoría de los casos se comporta bastante bien con un límite de 5 segundos.

Trabajar con más de un archivo .uitest

Trabajar con más de un archivo .uitest es algo completamente opcional, sin embargo se recomienda enormemente por las siguientes dos razones:

- Cualquier aplicación web de moderado tamaño puede hacer crecer enormemente el archivo .uitest haciendo que su mantenibilidad sea muy complicada. Por ejemplo, si se tiene archivo .uitest con 400 conjuntos de acciones y verificaciones que trabajan sobre una ventana principal (control raíz de la aplicación) y las propiedades de búsqueda de esta ventana cambian, se tendría que actualizar los 400 conjuntos de acciones y verificaciones para que puedan encontrar al control raíz, de lo contrario, todas las pruebas fallarán por fracaso a la hora de buscar el control.
- El archivo .uitest es de texto xml que puede ser extremadamente verboso, si se trabaja en un proyecto en equipo con control de código fuente, el hecho de que todos los "testers" trabajen sobre el mismo archivo .uitest puede causar enormes complicaciones a la hora de hacer combinar de los cambios.

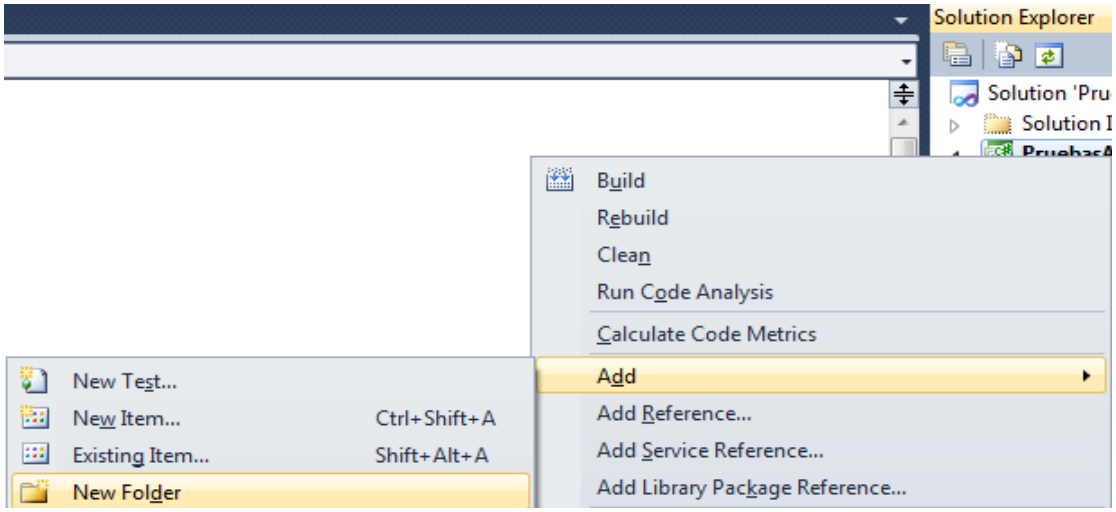
En el caso de las pruebas de aplicaciones web se recomienda seguir el siguiente patrón:

- Crear una carpeta llamada "UIMaps" la cual contendrá los archivos .uitest.
- Crear un archivo .uitest por cada página distinta de la aplicación.
- Crear una clase Helper para acceder los UIMaps los cuales se crearán bajo demanda.

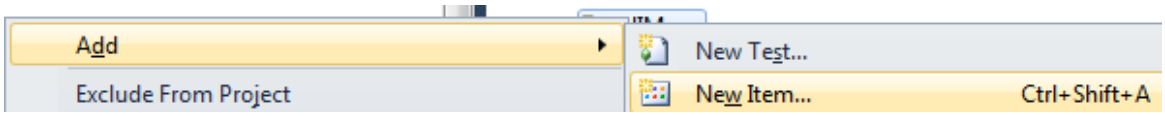
El siguiente ejemplo se basará en el estado del ejemplo anterior, y modificará el caso de prueba automatizado anteriormente para que utilice dos UIMaps, uno para la página donde se digita el nombre y otro para la página que muestra el saludo.

CREACIÓN DE LOS ARCHIVOS .UITEST

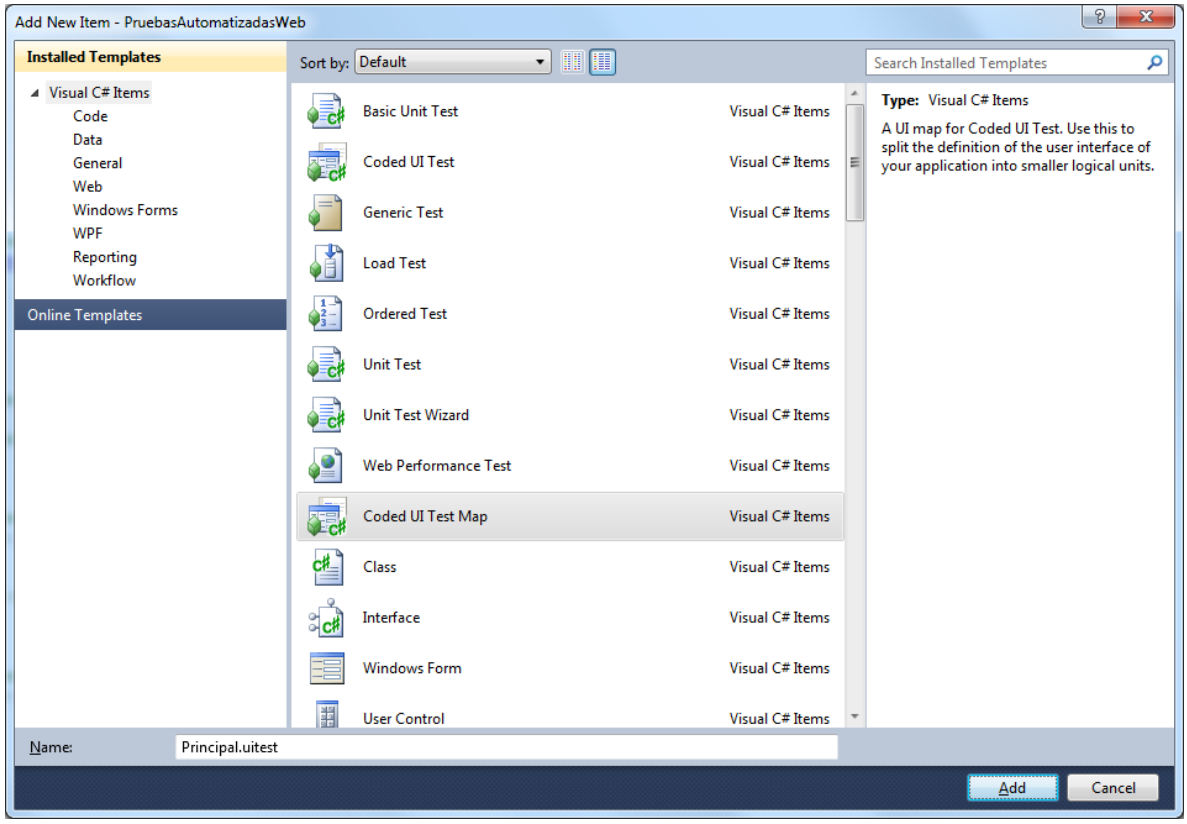
1. Hacer clic derecho en el proyecto y seleccionar la opción "Add" -> "New Folder". Llamar a la carpeta "UIMaps".



2. Hacer clic derecho en la carpeta y seleccionar la opción "Add" -> "New Item".



3. Seleccionar "Coded UI Test Map", llamar al archivo "Principal", y presionar el botón "Add".



4. Cerrar la herramienta "Coded UI Test Builder".
5. Repetir los pasos 1-4 y llamar al archivo .uitest "Saludo".

CREACIÓN DE LAS ACCIONES

1. Hacer clic derecho sobre el archivo "Principal.uitest" y seleccionar la opción "Edit With Coded UI Test Builder".
2. Seguir las instrucciones de cómo crear un conjunto de acciones.
3. Cerrar la herramienta "Coded UI Test Builder".
4. Hacer clic derecho sobre el archivo "Saludo.uitest" y seleccionar la opción "Edit With Coded UI Test Builder".
5. Seguir las instrucciones de cómo crear un conjunto de verificaciones.
6. Cerrar la herramienta "Coded UI Test Builder".

CREACIÓN DE LA CLASE HELPER

1. Hacer clic derecho sobre la carpeta "UIMaps" y seleccionar la opción "Add" -> "Class".
2. Llamar a la clase UIMapHelper.

A continuación se presenta el código de la clase helper:

```
using PruebasAutomatizadasWeb.UIMaps.PrincipalClasses;
using PruebasAutomatizadasWeb.UIMaps.SaludoClasses;

namespace PruebasAutomatizadasWeb.UIMaps
{
    public class UIMapHelper
    {
        private Principal principal;
        public Principal Principal
        {
            get
            {
                if (principal == null)
                    principal = new Principal();
                return principal;
            }
        }
        private Saludo saludo;
        public Saludo Saludo
        {
            get
            {
                if (saludo == null)
                {
                    saludo = new Saludo();
                    saludo.UIAplicaciónWebWindowsWindow.CopyFrom
                    (Principal.UIInternetExplorercannWindow);
                }
                return saludo;
            }
        }
    }
}
```

Hay dos puntos de especial atención en esta clase:

- Los mapas de interfaz de usuario se crean bajo demanda, es decir, son instanciados únicamente cuando son solicitados por el método de prueba, no antes. La razón de esto es porque el mapa "Saludo" depende del mapa "Principal" y la ventana de la cual copia las propiedades del otro mapa debe existir en el momento de realizar la copia (para efectos prácticos esto quiere decir que la ventana de Internet Explorer debe estar abierta cuando se crea un mapa de interfaz dependiente, de lo contrario se arrojará una excepción de que no se puede encontrar el control).

- Los mapas de interfaz de usuario dependen del mapa principal y copian las propiedades de búsqueda del mapa principal. De esta manera si las propiedades del control raíz cambian, únicamente se deberán actualizar el archivo Principal.uitest.

RE-ESCRITURA DEL MÉTODO DE PRUEBA

Una vez escrita la clase helper, el método de prueba quedará así:

```
[TestMethod]
public void CodedUITestMethod1()
{
    UIMapHelper ui = new UIMapHelper();
    ui.Principal.DigitarNombreYPresionarSiguiente();
    ui.Saludo.VerificarSaludo();
}
```

Parametrizar una prueba web automatizada

Se ha visto cómo se puede crear, modificar, optimizar y ejecutar una prueba automatizada web. No obstante, las pruebas automatizadas no serían tan convenientes si no se pudieran automatizar. ¿Cómo se hace para que la prueba automatizada anterior escriba el nombre "Juan" en vez de "Pepe" y que la verificación refleje este cambio?

ENTENDER LAS CLASES AUTOGENERADAS DE PARÁMETROS

Cada vez que se crea un método de acciones que requiere una entrada de datos (en el ejemplo el método que digita un nombre), en el archivo autogenerado derivado del archivo .uitest se crea una clase con el nombre "[NombreMetodo]Params" donde se puede parametrizar los datos a ingresar. De la misma manera, en los métodos de verificación se crea una clase con el nombre "[NombreMetodo]ExpectedValues" para poder modificar esta información.

A continuación se muestra un ejemplo de cómo se podría parametrizar la prueba del ejemplo anterior para que escriba "Juan" en vez de "Pepe":

```
[TestMethod]
public void CodedUITestMethod1()
{
    UIMapHelper ui = new UIMapHelper();

    ui.Principal.DigitarNombreYPresionarSiguienteParams.
    UIDigitesunombreEditText ="Juan";
    ui.Principal.DigitarNombreYPresionarSiguiente();
    ui.Saludo.VerificarSaludoExpectedValues.UIHolaPepePaneInnerText = "Hola
    Juan!";
    ui.Saludo.VerificarSaludo();
}
```

Nótese que los valores parametrizados deben ser asignados antes del llamado del método, de lo contrario no se hará efectiva la parametrización.

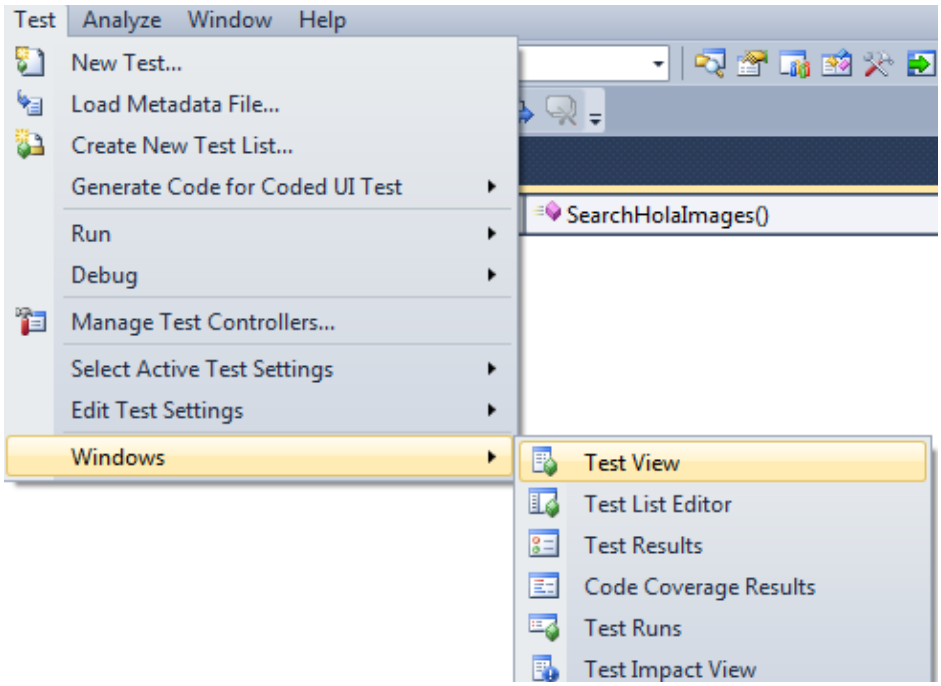
LEER PARÁMETROS DE ORÍGENES DE DATOS EXTERNOS

Para parametrizar los casos de prueba automatizados para que lean los parámetros de orígenes de datos externos y que se ejecuten automáticamente tantas veces como registros externos existan, se procede de una forma similar al ejemplo anterior, con dos excepciones:

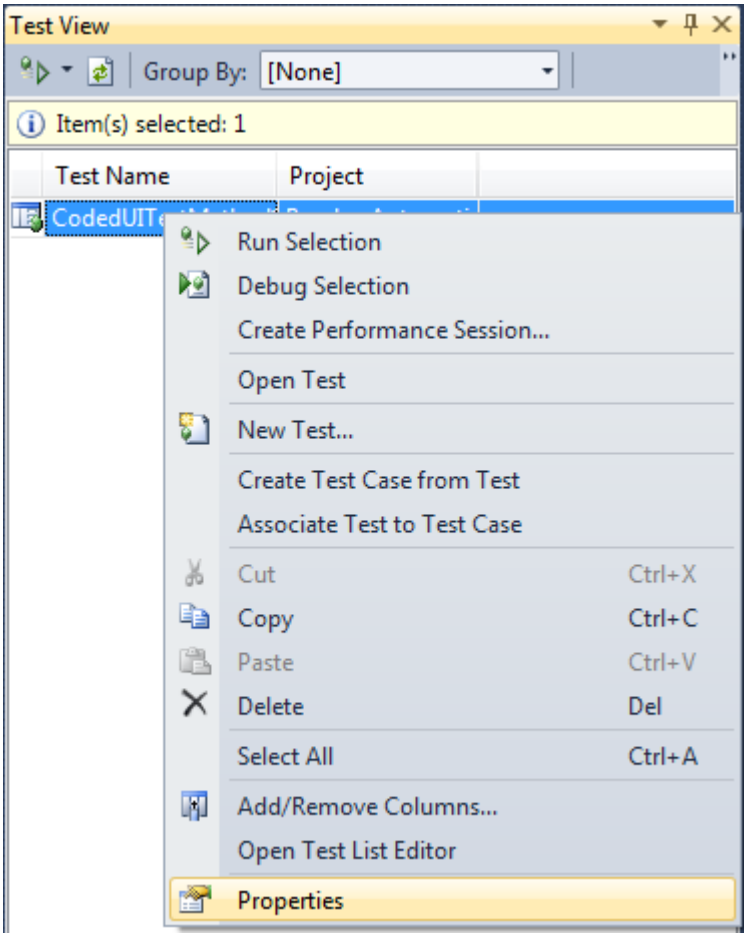
- En vez de asignar el parámetro con un valor directo (entre comillas en el caso del ejemplo), se debe utilizar un objeto que contienen las clases de prueba llamado "TestContext". Este objeto contiene información acerca del contexto en que se están ejecutando las pruebas y los valores de los parámetros del origen de datos externo, entre otras cosas. El objeto TestContext contiene la propiedad DataRow desde donde se leen los parámetros.
- Se debe adornar el método a ejecutar con un atributo que indique el origen de datos externo desde el cual se leerán los parámetros. Escribir este atributo manualmente puede ser un poco complicado por lo que se recomienda generarlo mediante la interfaz gráfica.

Para generar el atributo de origen de datos externo se deben seguir las siguientes instrucciones:

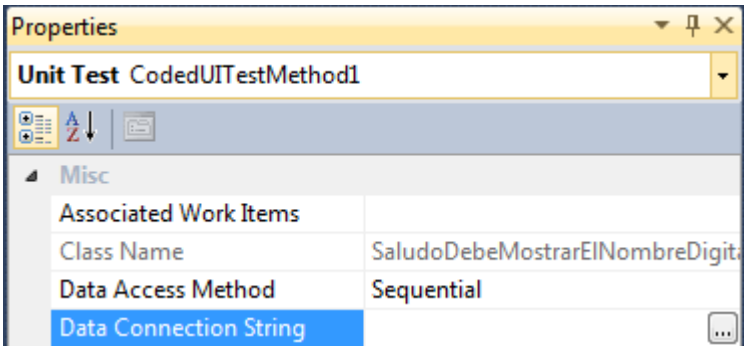
1. Seleccionar la opción de menú "Test" -> "Windows" -> "Test View".



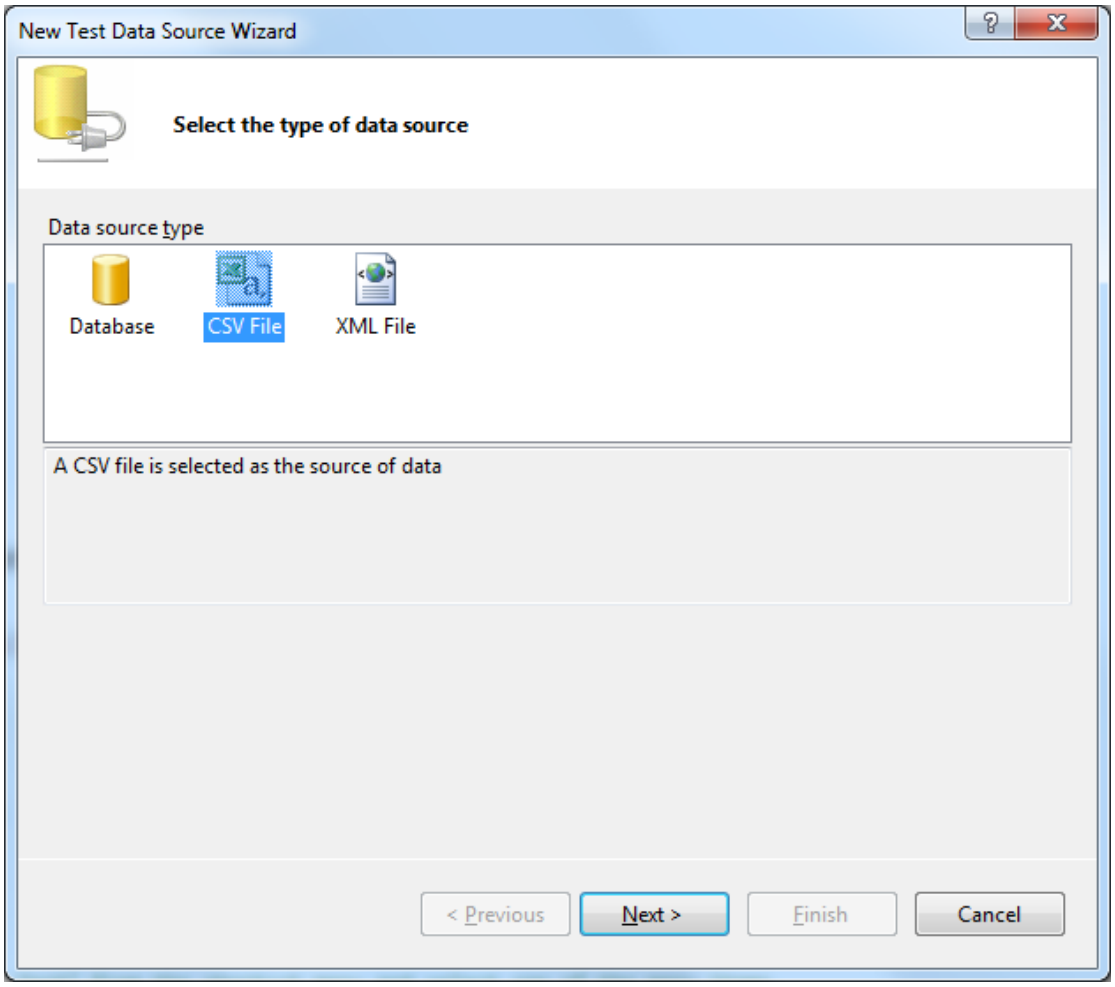
2. Presionar el botón de refrescar en la ventana mostrada.
3. Hacer clic derecho en la prueba deseada y seleccionar la opción "Properties".



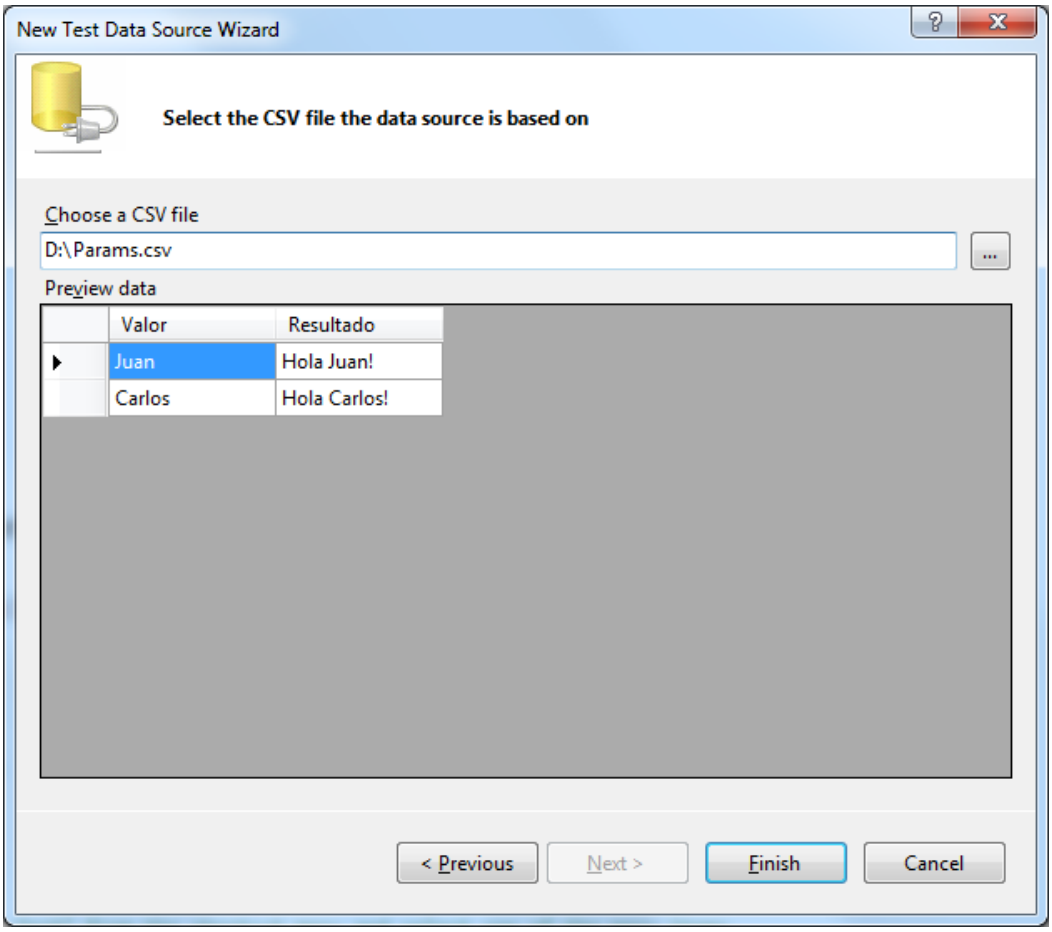
4. Seleccionar la propiedad "Data Connection String" y seleccionar el botón de los puntos suspensivos.



5. Seleccionar el tipo de origen de datos que se desea utilizar.



6. Localizar el archivo o completar las instrucciones en caso de seleccionar la opción "Database".



7. Seleccionar el botón "Finish".
8. Seleccionar la opción "Yes" en caso de haber seleccionado un archivo.

Finalmente el método deberá quedar de la siguiente manera:

```
        [TestMethod]
        [DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",
"|DataDirectory|\\Params.csv",      "Params#csv",      DataAccessMethod.Sequential),
DeploymentItem("PruebasAutomatizadasWeb\\Params.csv")]
        public void CodedUITestMethod1()
        {
            UIMapHelper ui = new UIMapHelper();


ui.Principal.DigitarNombreYPresionarSiguienteParams.UIDigitesunombreEditText       =
(string)TestContext.DataRow["Valor"];
            ui.Principal.DigitarNombreYPresionarSiguiente();

            ui.Saludo.VerificarSaludoExpectedValues.UIHolaPepePaneInnerText       =
(string)TestContext.DataRow["Resultado"];
            ui.Saludo.VerificarSaludo();
        }
```

3.3. Integración con Microsoft Test Manager

Finalmente, se mostrarán tres aplicaciones útiles de la integración de Microsoft Visual Studio con Microsoft Test Manager.

Creación de una prueba automatizada a partir de un caso de prueba

Para crear una prueba automatizada a partir de un caso de prueba son necesarias tres cosas:

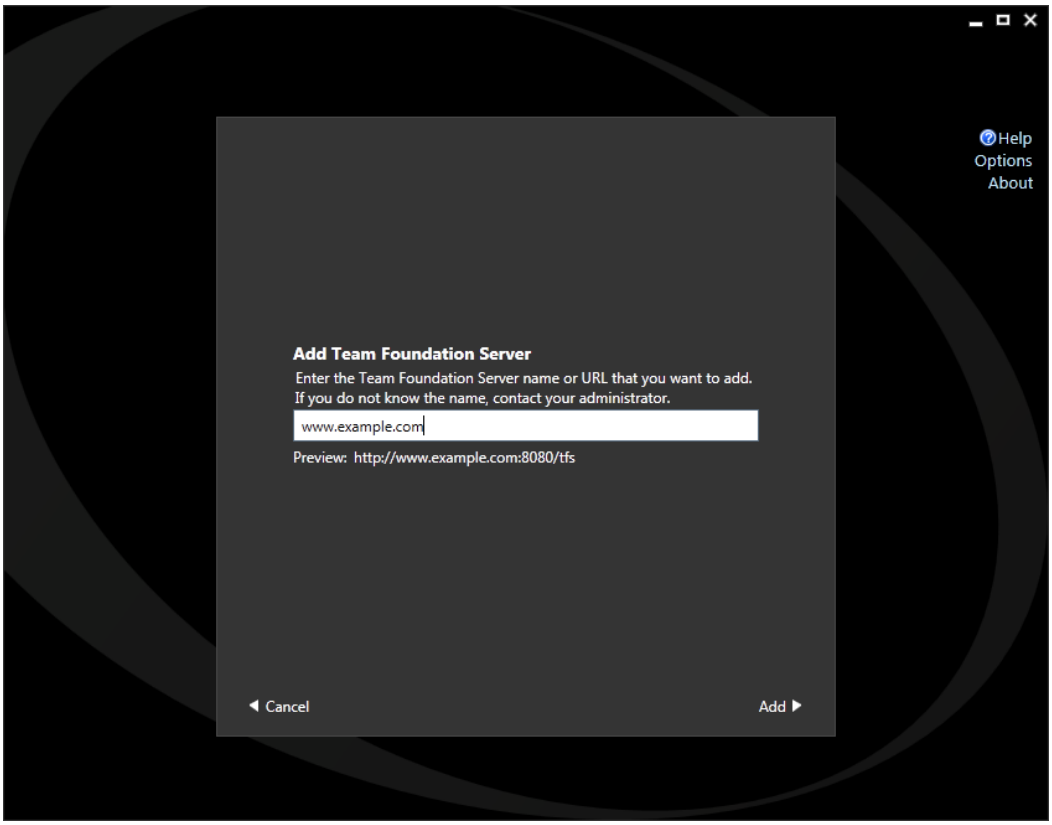
- Contar con un servidor Microsoft Team Foundation Server (TFS) en el cual se puedan crear casos de prueba.
- Tener al menos un caso de prueba ingresado en el servidor.
- Haber corrido manualmente el caso de prueba al menos una vez seleccionando la opción "Create Action Recording".

La instalación y configuración del servidor Team Foundation Server están fuera del ámbito de este reporte por lo que únicamente se mostrarán los pasos para crear el caso de prueba, ejecutarlo con "Action Recording" y crear la prueba automatizada a partir del caso.

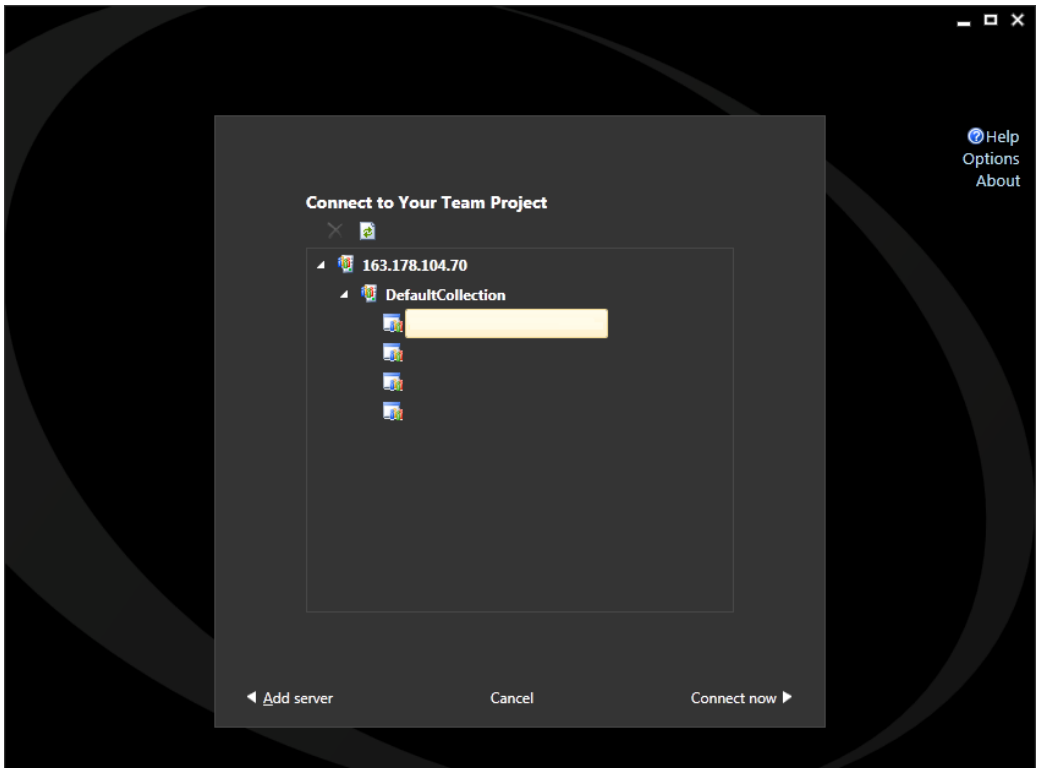
Crear un caso de prueba

Para crear un caso de prueba, se deben seguir las siguientes instrucciones:

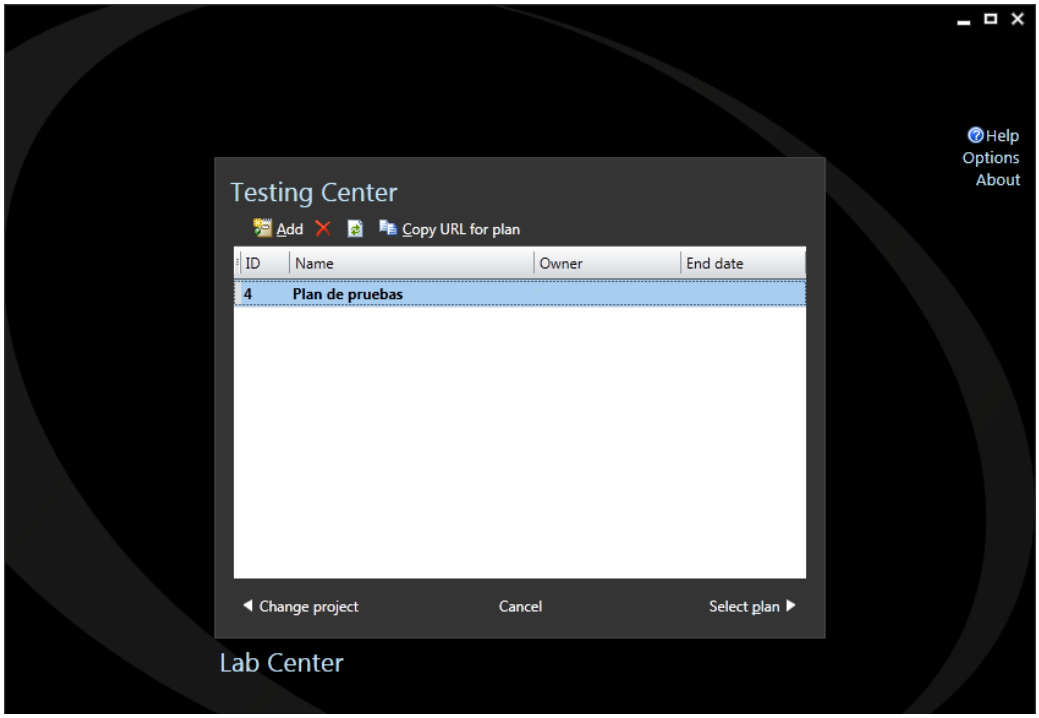
1. Abrir la herramienta Microsoft Test Manager.
2. Digitar el nombre del servidor TFS al que se desee conectar y seleccionar la opción "Add".



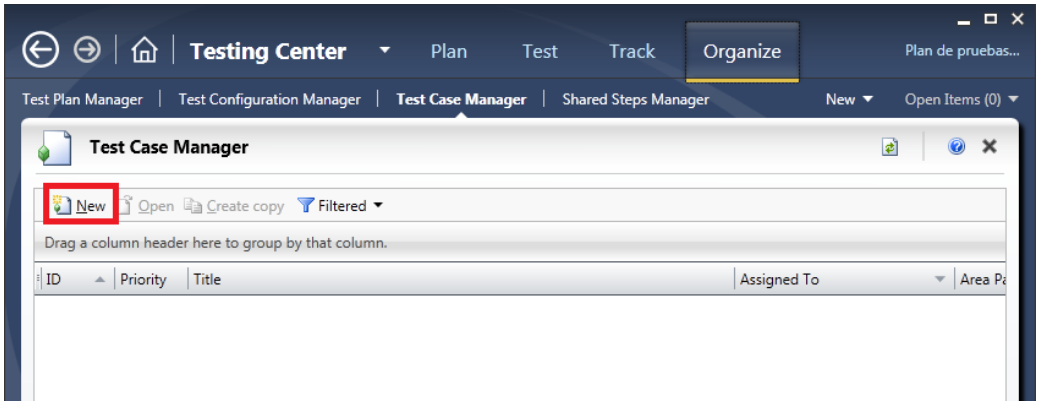
3. Seleccionar el proyecto en el cual se desea ingresar el caso de prueba y seleccionar la opción "Connect now".



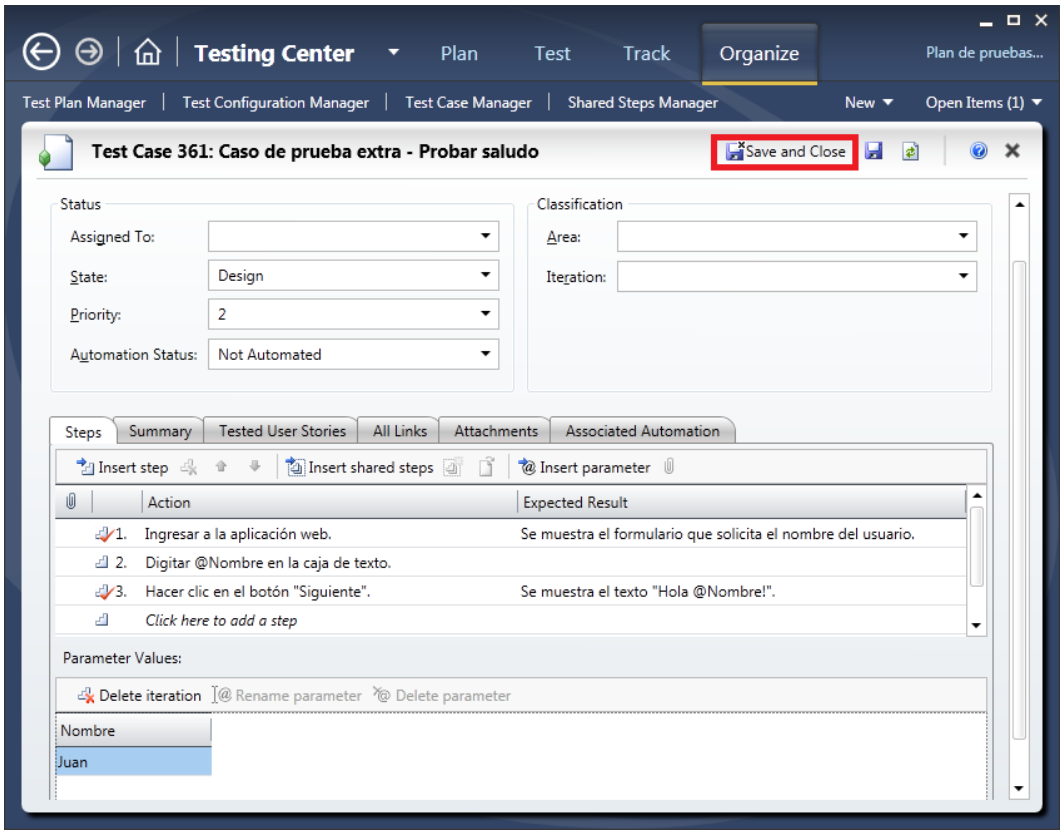
4. Seleccionar el plan de pruebas en el que se desea ingresar el caso de prueba y seleccionar la opción "Select plan".



5. Seleccionar la pestaña "Organize", y la subpestaña "Test Case Manager", seguidamente hacer clic en el botón "New".



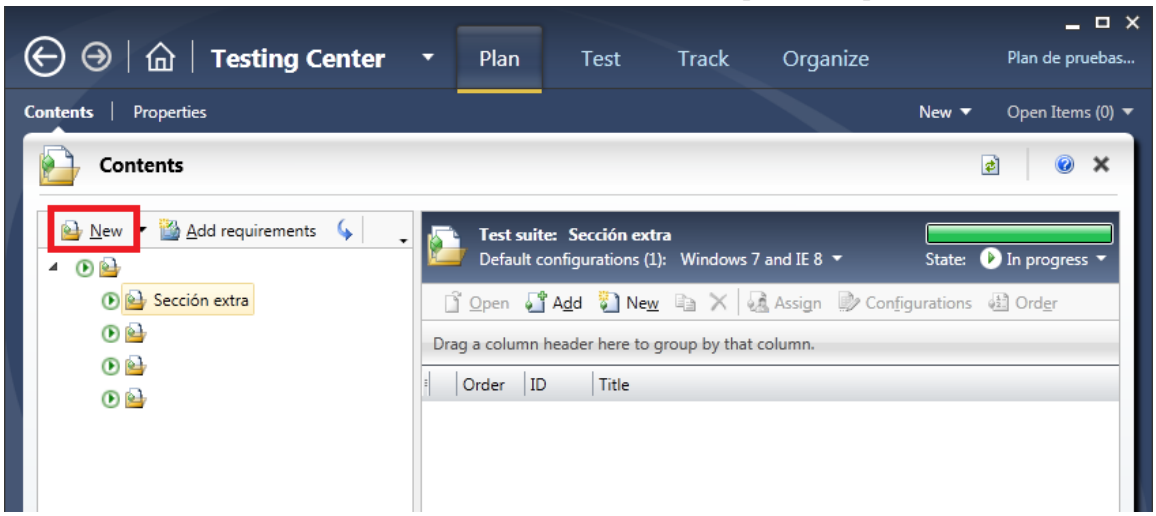
6. Escribir los pasos del caso de prueba, los parámetros y resultados esperados y presionar el botón "Save and close".



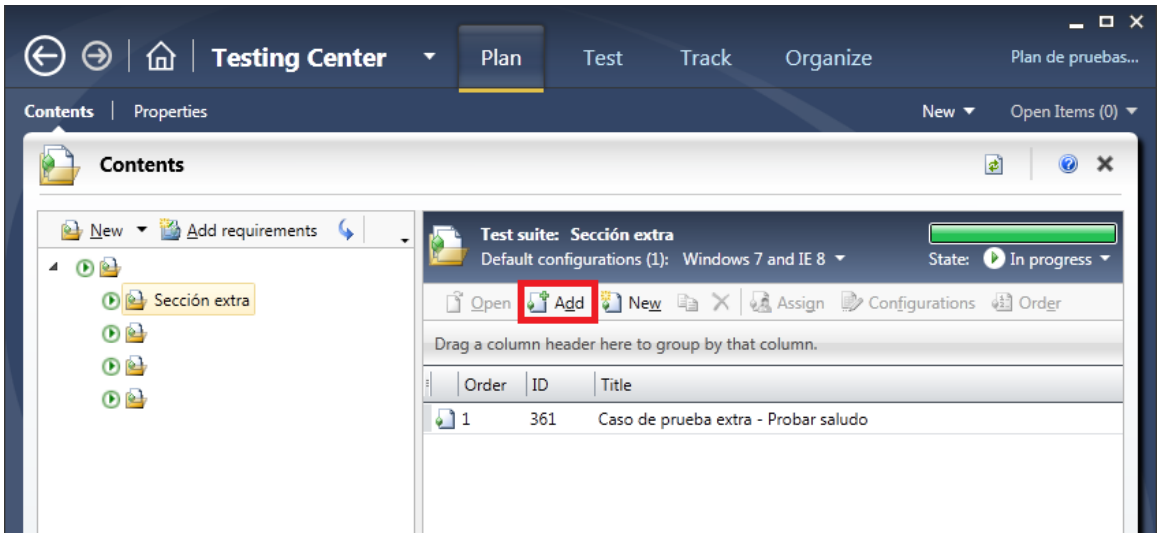
Ejecutar el caso de prueba con "Action Recording"

Una vez creado el caso de prueba, se deben seguir los siguientes pasos para ejecutar el caso de prueba con "Action Recording":

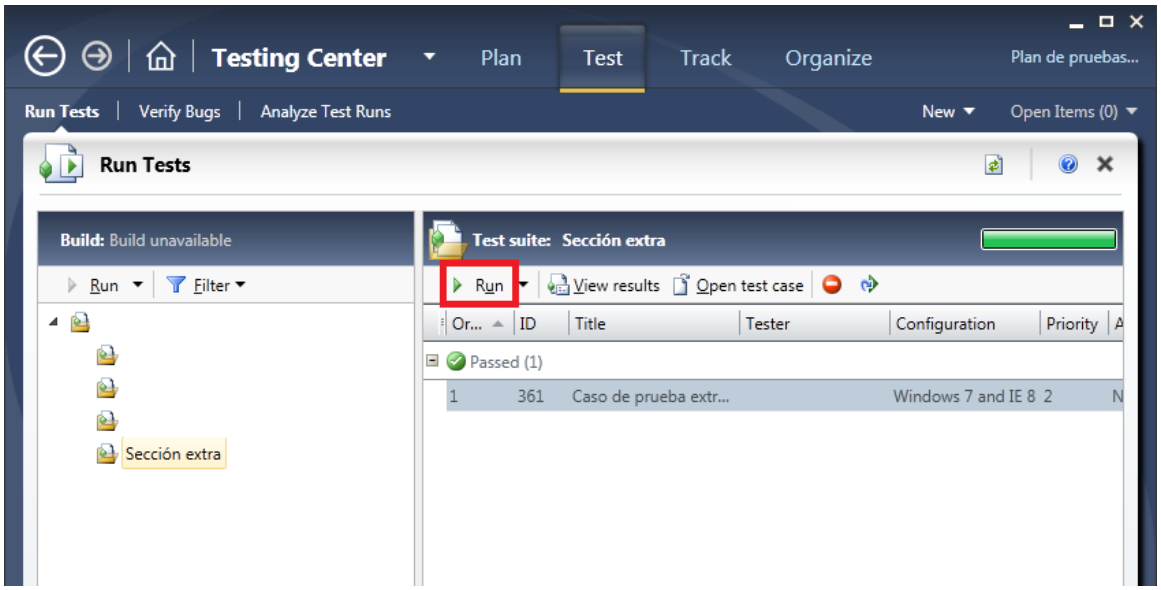
1. Seleccionar el tab "Plan" y seguidamente hacer clic en el botón "New" y ponerle un nombre al ítem recién creado. Esto creará un "suite" de pruebas que serán corridas.



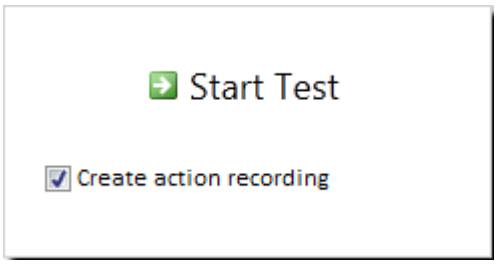
2. Seguidamente seleccionar el botón "Add" en la sección derecha de la pantalla y buscar el caso de prueba creado anteriormente.



3. Seleccionar el tab "Test", seguidamente seleccionar el "suite" recién creado y seleccionar el botón "Run".

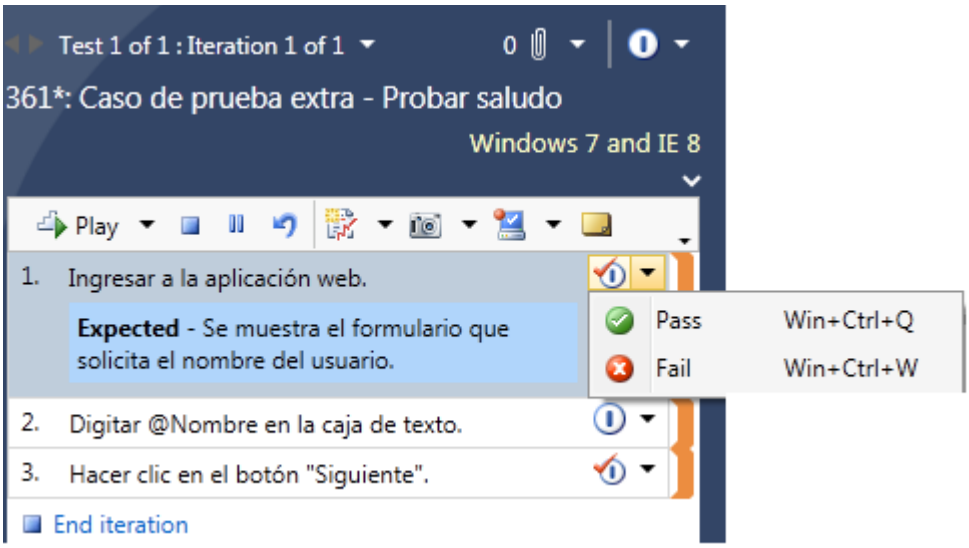


4. Esto iniciará la herramienta de ejecución manual de pruebas. Antes de iniciar con la prueba, se debe chequear la opción "Create action recording" para que sea posible crear la prueba automatizada a partir del caso de prueba.



5. Hacer clic en el botón "Start Test".

6. Ejecutar manualmente cada paso del caso de prueba y seleccionar los resultados de los mismos mediante el menú contextual con las opciones "Pass" o "Fail".



> ***Tip:***
>
> *Se debe seleccionar el resultado de cada paso inmediatamente después de ejecutar las acciones del mismo. De lo contrario, todas las acciones grabadas se guardarán en un único método de acción cuando se convierta el caso de prueba en una prueba automatizada.*

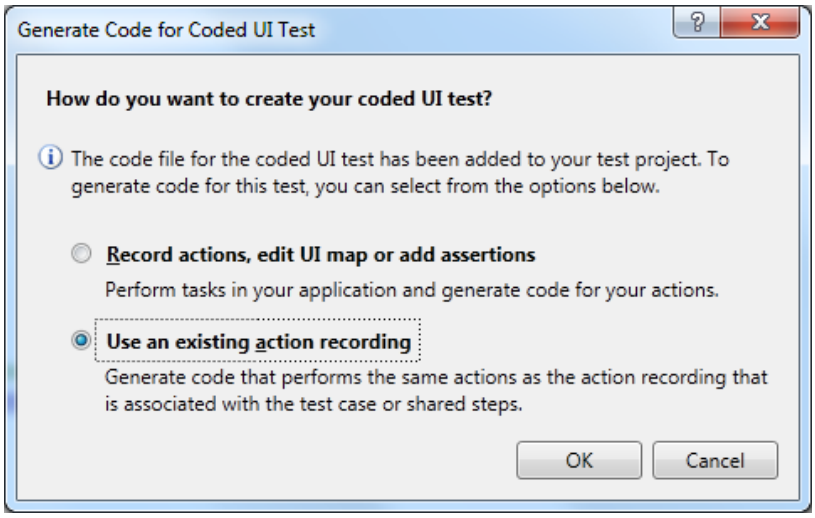
7. Cuando se terminó de ejecutar todos los pasos, únicamente hay que cerrar la herramienta de ejecución manual de pruebas.

Crear la prueba automatizada a partir del caso de prueba

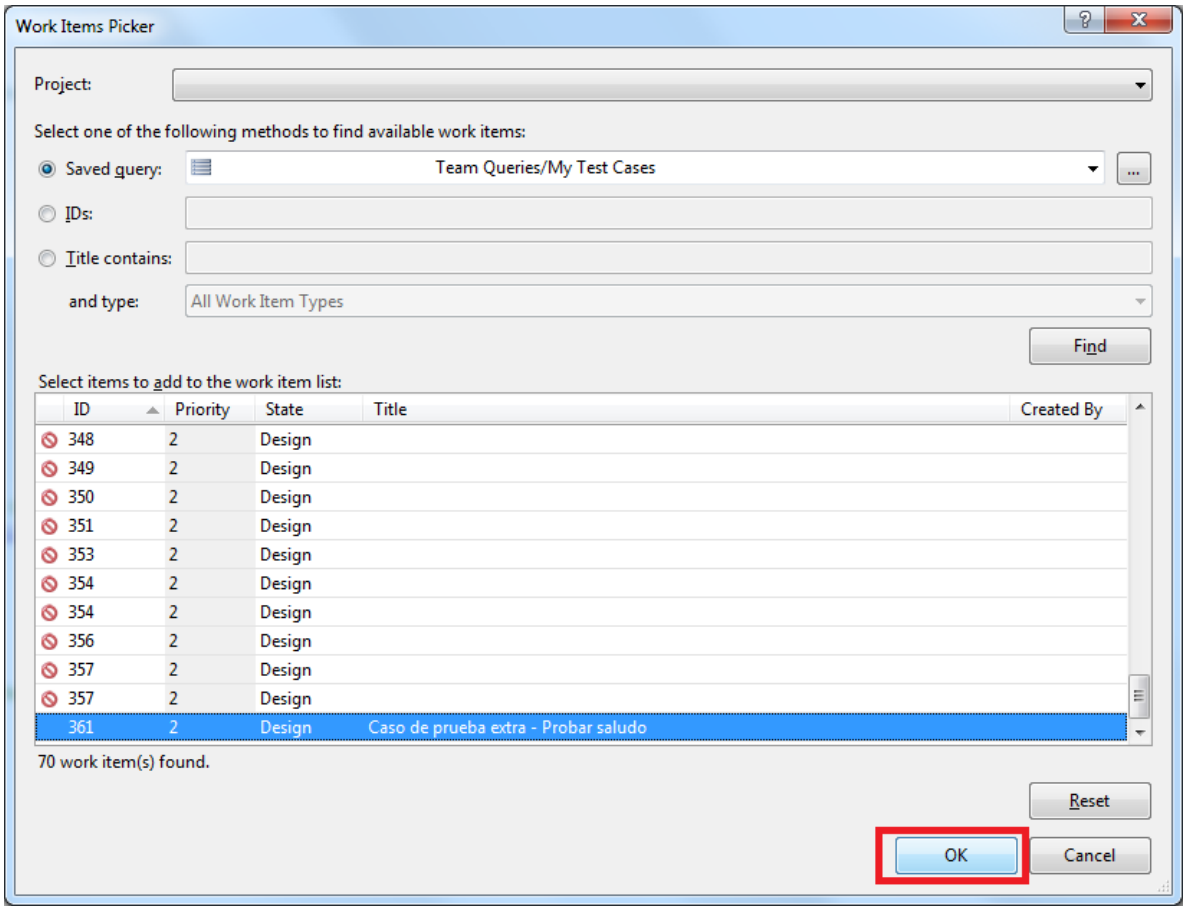
Una vez ejecutado el caso de prueba con un "Action Recording" es posible crear la prueba automatizada utilizando como base el caso de prueba anteriormente grabado. Para crear la prueba automatizada, se deben seguir las mismas instrucciones en la sección "Crear una

prueba automatizada web" hasta el paso 5 inclusive, seguidamente se deben seguir las siguientes instrucciones:

1. Seleccionar la opción "Use an existing action recording" y seleccionar el botón "Ok".



2. Seleccionar la consulta de los casos de prueba y seleccionar en la lista el caso de prueba que se creó y ejecutó anteriormente. Finalmente hacer clic en el botón "Ok".



Tip:

Los casos de prueba que no hayan sido ejecutados con la opción "Create action recording" no podrán seleccionarse para la creación de la prueba automatizada. Para distinguir los casos utilizables de los no utilizables, Visual Studio muestra el ícono rojo  *al lado de cada caso de prueba listado, si éste no contiene una grabación de acciones.*

La prueba automatizada se creará automáticamente, generando un método por cada paso del caso de prueba manual, cada método ejecutará las acciones manuales que se hayan ejecutado entre cada selección de "Pass" o "Fail" mientras se ejecutaba el caso de prueba. Además, la prueba automatizada quedará parametrizada, utilizando como origen de datos el servidor TFS.

Tip:

El servidor TFS como origen de datos requiere autenticación de usuario y contraseña. Hasta el momento no es posible asignar estos credenciales manualmente, por lo que se utilizarán los credenciales del usuario actual que ejecuta las pruebas. Si las pruebas automatizadas se ejecutan con un usuario que no tiene permisos para ingresar al servidor TFS, la prueba fallará por problemas de autenticación.

Si no se puede cambiar el usuario que ejecuta las pruebas automatizadas o asignar permisos al mismo por motivos de seguridad, debe considerarse utilizar otro origen de datos.

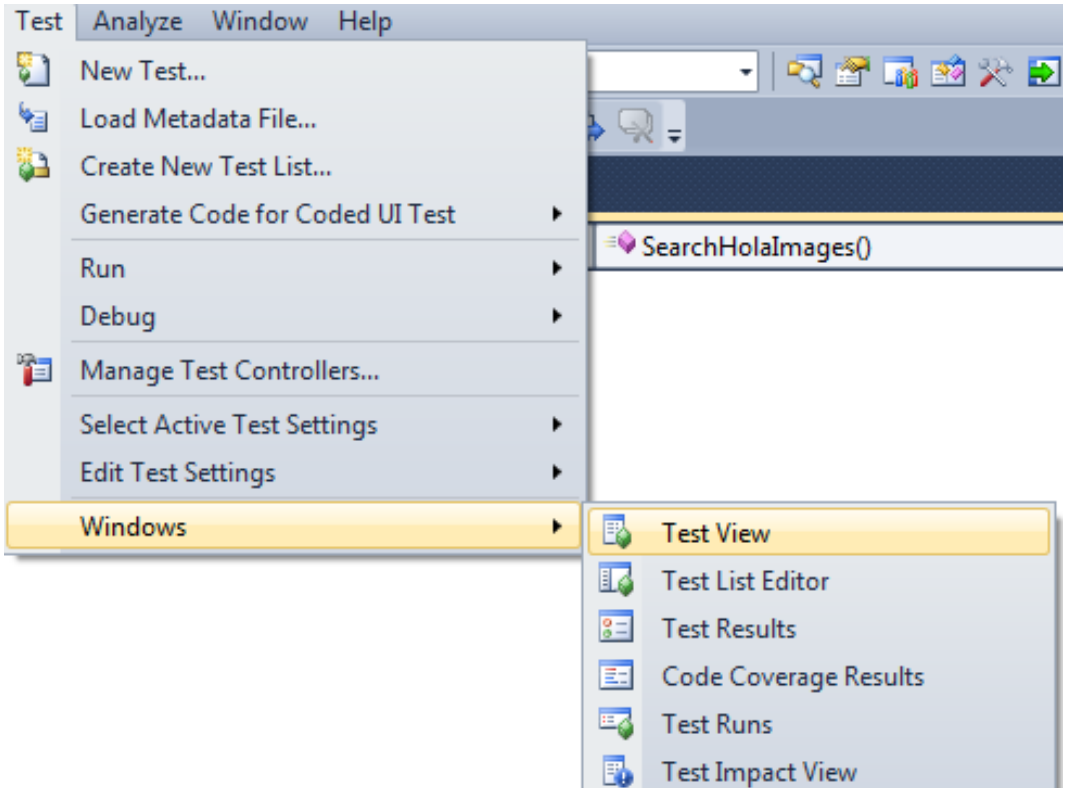
Tip:

Únicamente los métodos de acción pueden ser grabados en una ejecución de un caso de prueba. Los métodos de verificación deberán crearse e ingresarse manualmente a la prueba automatizada luego de su creación.

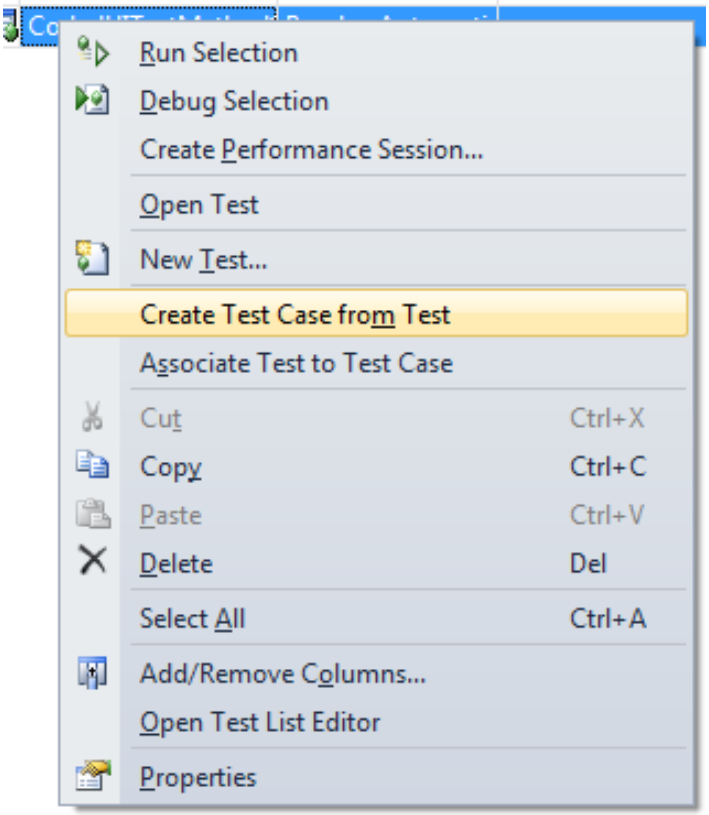
Creación de un caso de prueba a partir de una prueba automatizada

Crear un caso de prueba a partir de una prueba automatizada es una tarea muy simple de dos pasos. Para crear el caso de prueba, se deben seguir las siguientes instrucciones.

1. Seleccionar la opción de menú "Test" -> "Windows" -> "Test View".



2. Hacer clic derecho en la prueba automatizada a partir de la cual se desea crear un caso de prueba y seleccionar la opción "Create Test Case from Test".

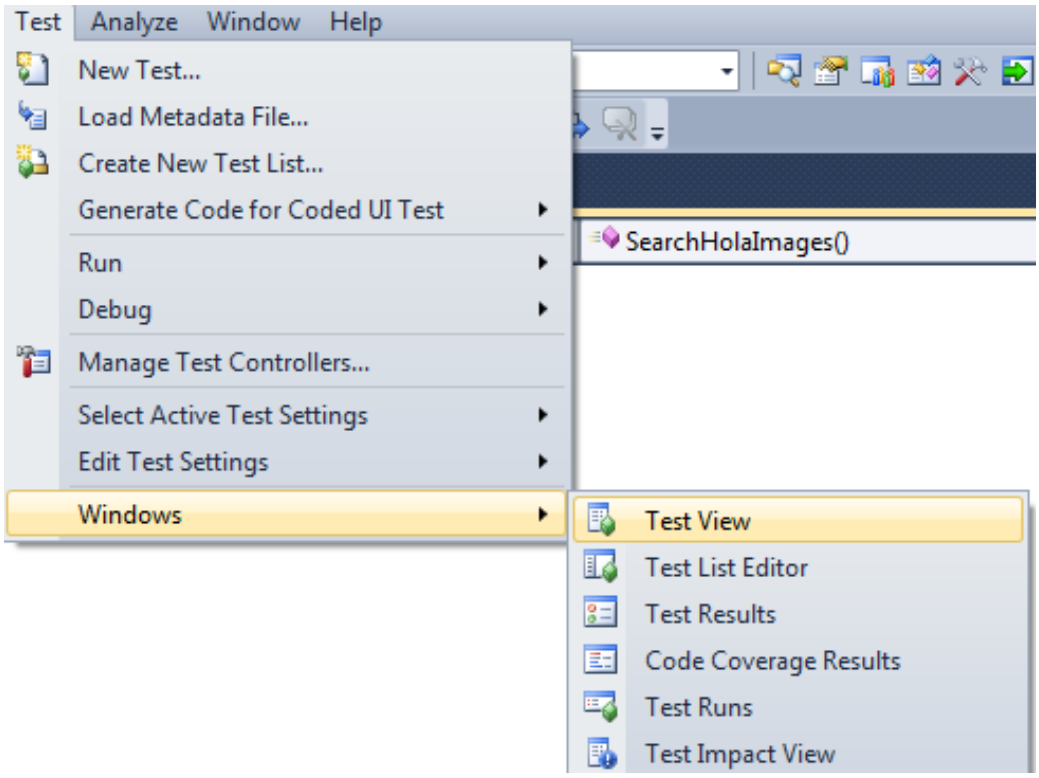


El nuevo caso de prueba automáticamente será asociado a la prueba automatizada.

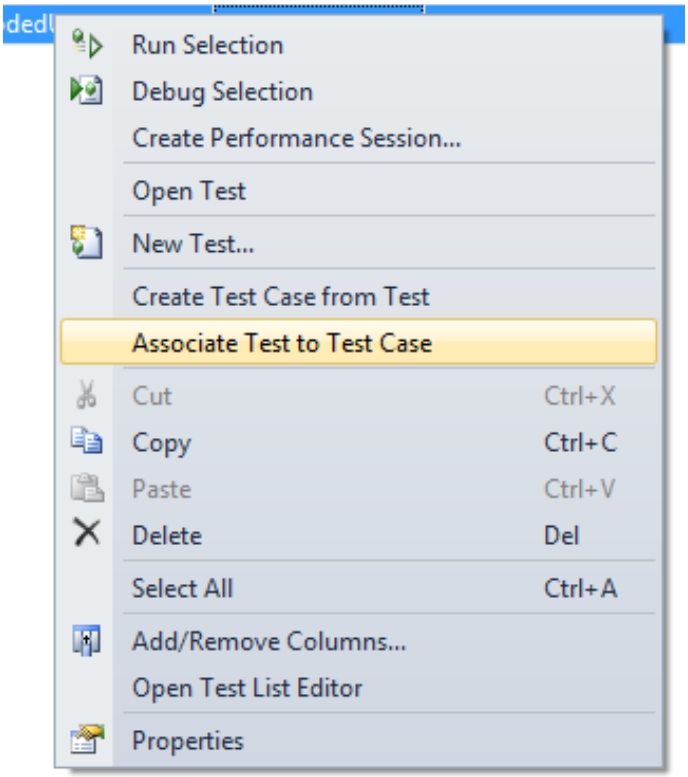
Asociación de una prueba automatizada a un caso de prueba

Cuando el código de las pruebas automatizadas se encuentra en control de código fuente en el servidor TFS, es muy útil asociar pruebas automatizadas con casos de prueba, ya que esto facilita la trazabilidad de requerimientos, un aspecto muy importante en el desarrollo de software. Asociar pruebas automatizadas con casos de prueba existentes es una tarea muy sencilla. Para asociar una prueba automatizada con un caso de prueba existente, se deben seguir las siguientes instrucciones:

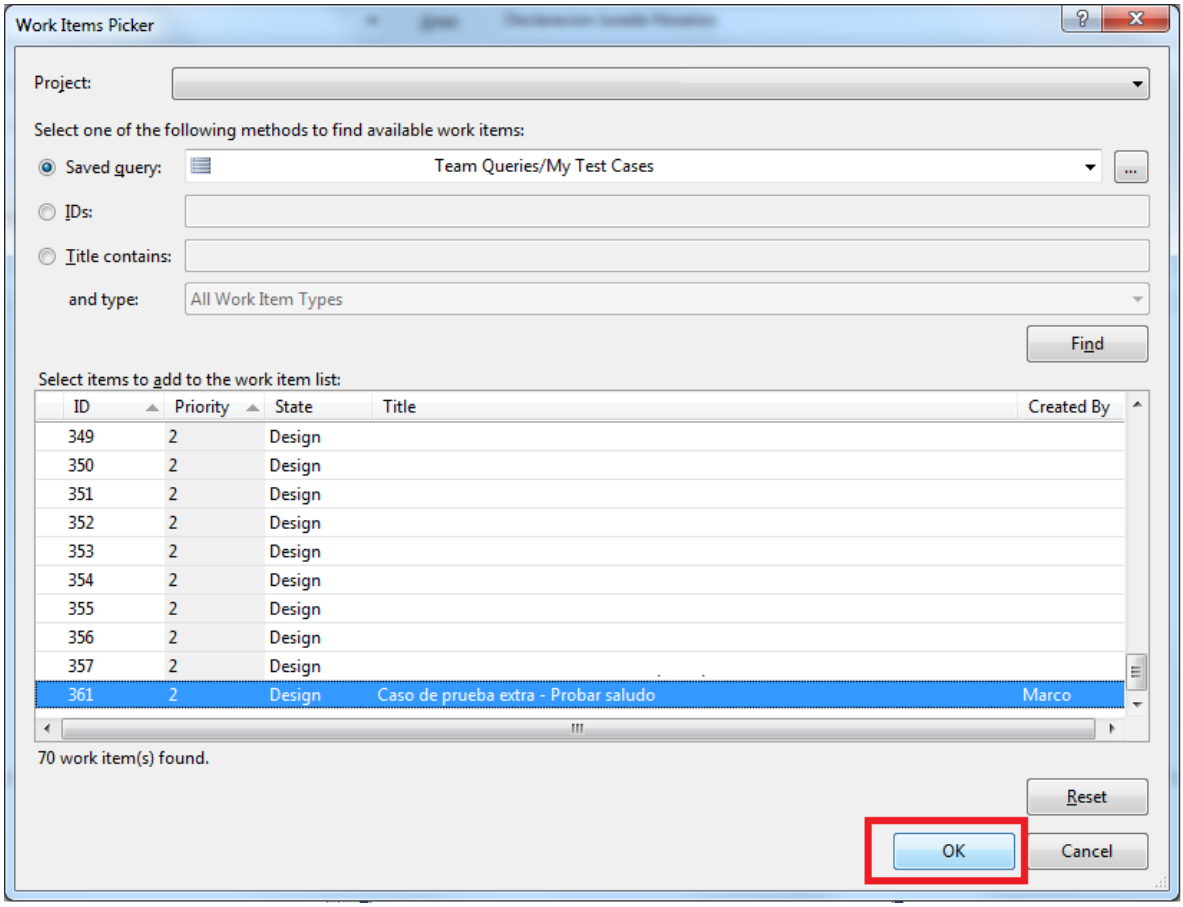
1. Seleccionar la opción de menú "Test" -> "Windows" -> "Test View".



2. Hacer clic derecho en la prueba automatizada que se desea asociar y seleccionar la opción "Associate Test to Test Case".



3. Seleccionar la consulta de los casos de prueba y seleccionar en la lista el caso de prueba con el cual se desea crear la asociación y finalmente hacer clic en el botón "Ok".



Referencias

[1] Jeff Levinson. Software Testing with Visual Studio® 2010. Addison-Wesley Professional, 2011.

[2] Microsoft. Visual Studio 2010. URL: http://msdn.microsoft.com/en-us/library/dd831853(v=vs.100).aspx (último acceso: 23 de octubre 2012).

[3] Microsoft. Microsoft Developer Network. URL: http://msdn.microsoft.com/en-US/