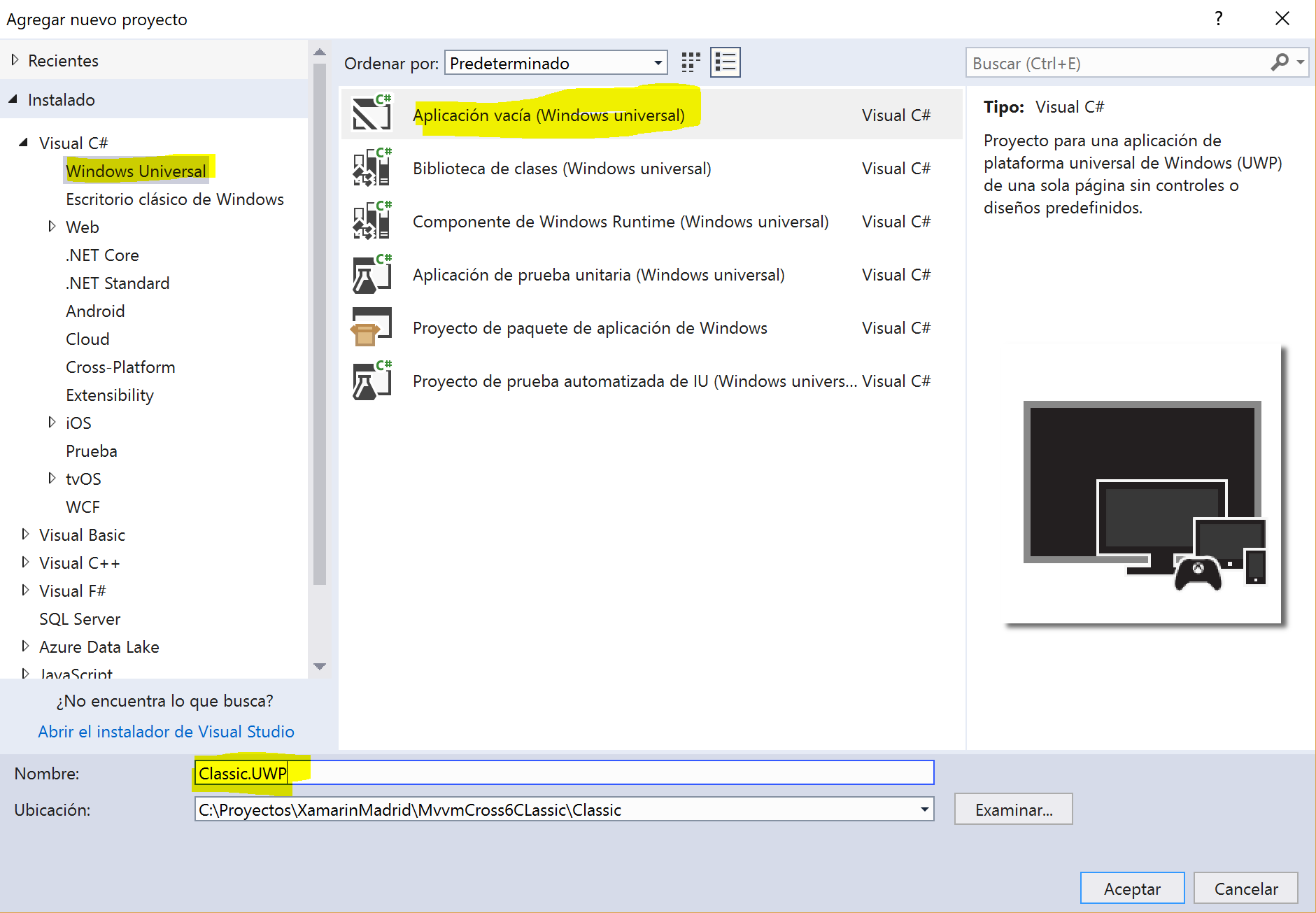
En este tutorial vamos a crear un proyecto desde 0 para UWP, IOS y Android usando MvvmCross 6 beta, que hace uso de Net Standard.

Como vamos a crear para UWP, lo haremos desde Windows y Visual Studio 2017, en mi caso estoy usando la última versión la 15.6.0

Observación:

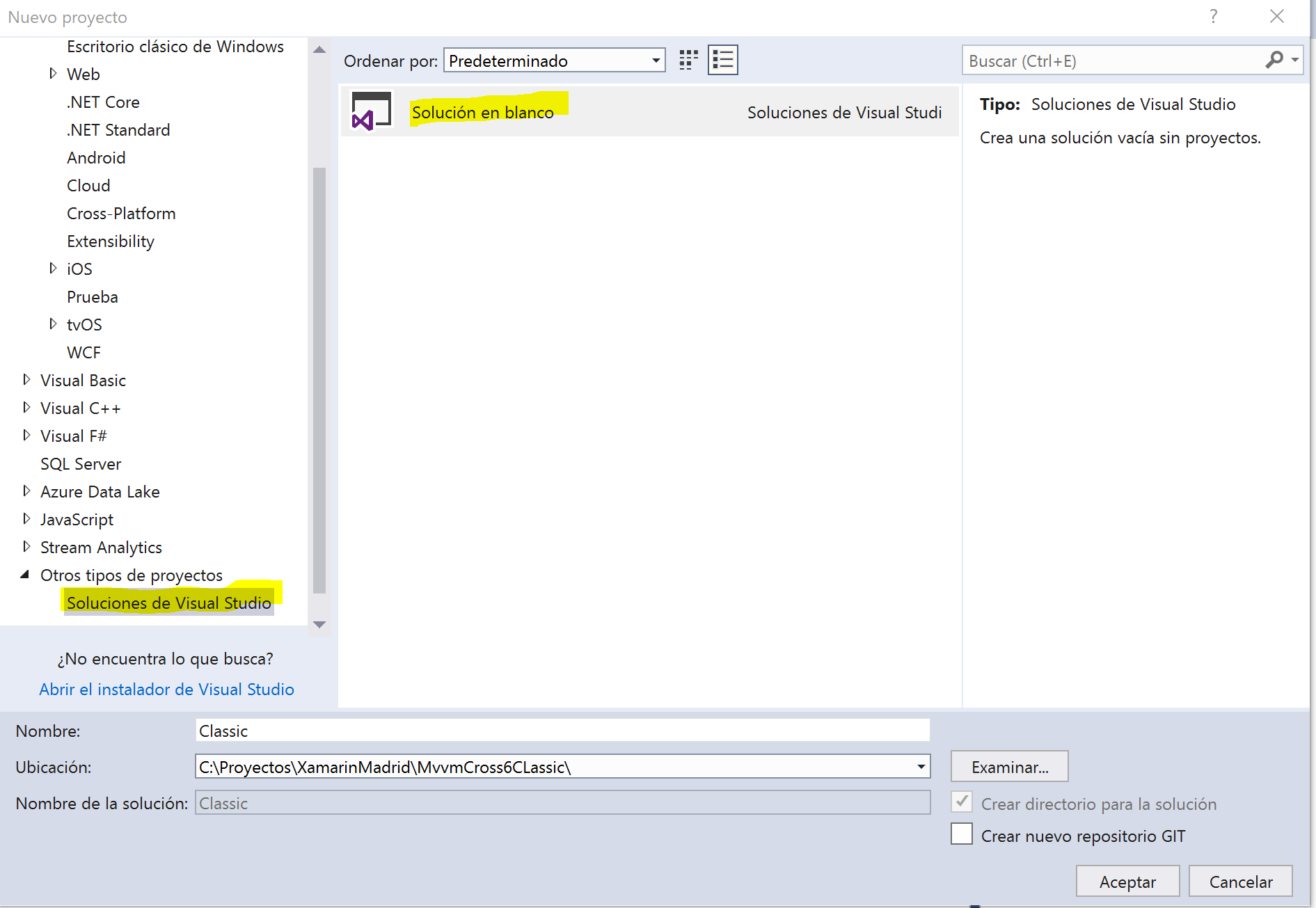
Xamarin Classic (native) es más complejo que Xamarin Forms, requiere tener conocimiento sobre todas las plataformas, explicaremos de la forma más detallada posible.

¡Empezamos!

**Paso 1: Crear una solución vacía:**

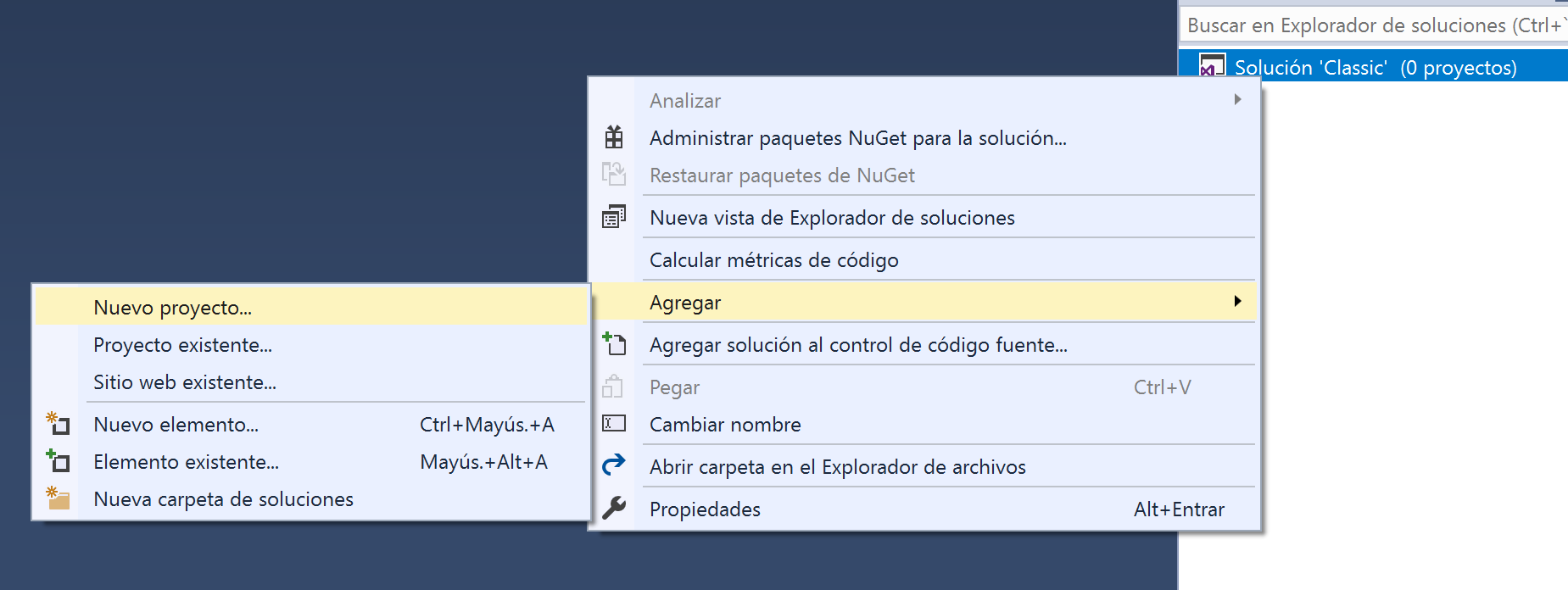
Abrimos visual studio 2017, en el menú superior seleccionamos => Nuevo => Proyecto.

Elegimos la opción “Otros tipos de proyectos”, y creamos un proyecto **solución en blanco**.



**Paso 2: Agregar los proyectos para las plataformas**

Seleccionamos con el 2º botón la solución vacía y elegimos, **agregar** => **nuevo proyecto**.

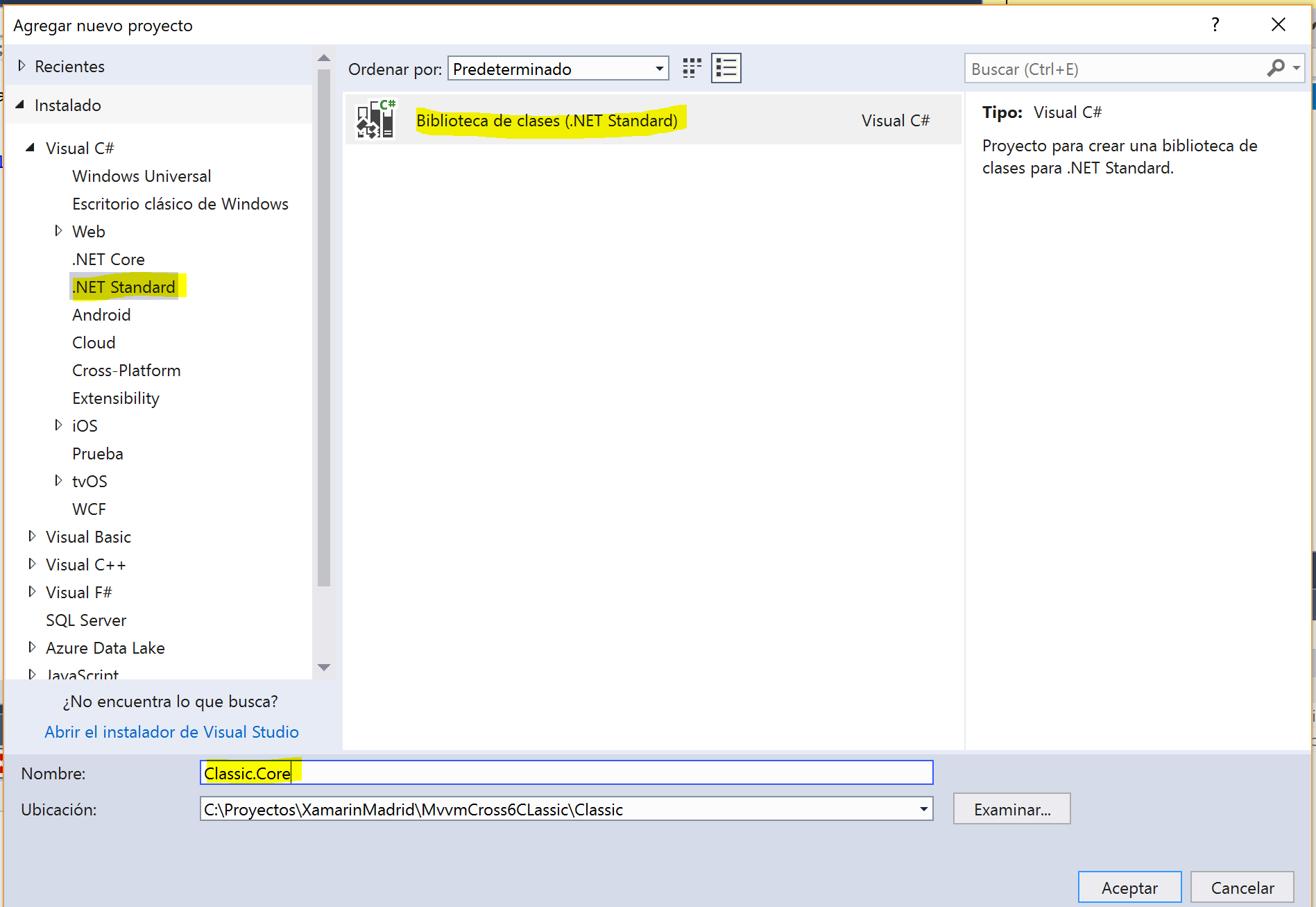


Debemos agregar 4 proyectos:

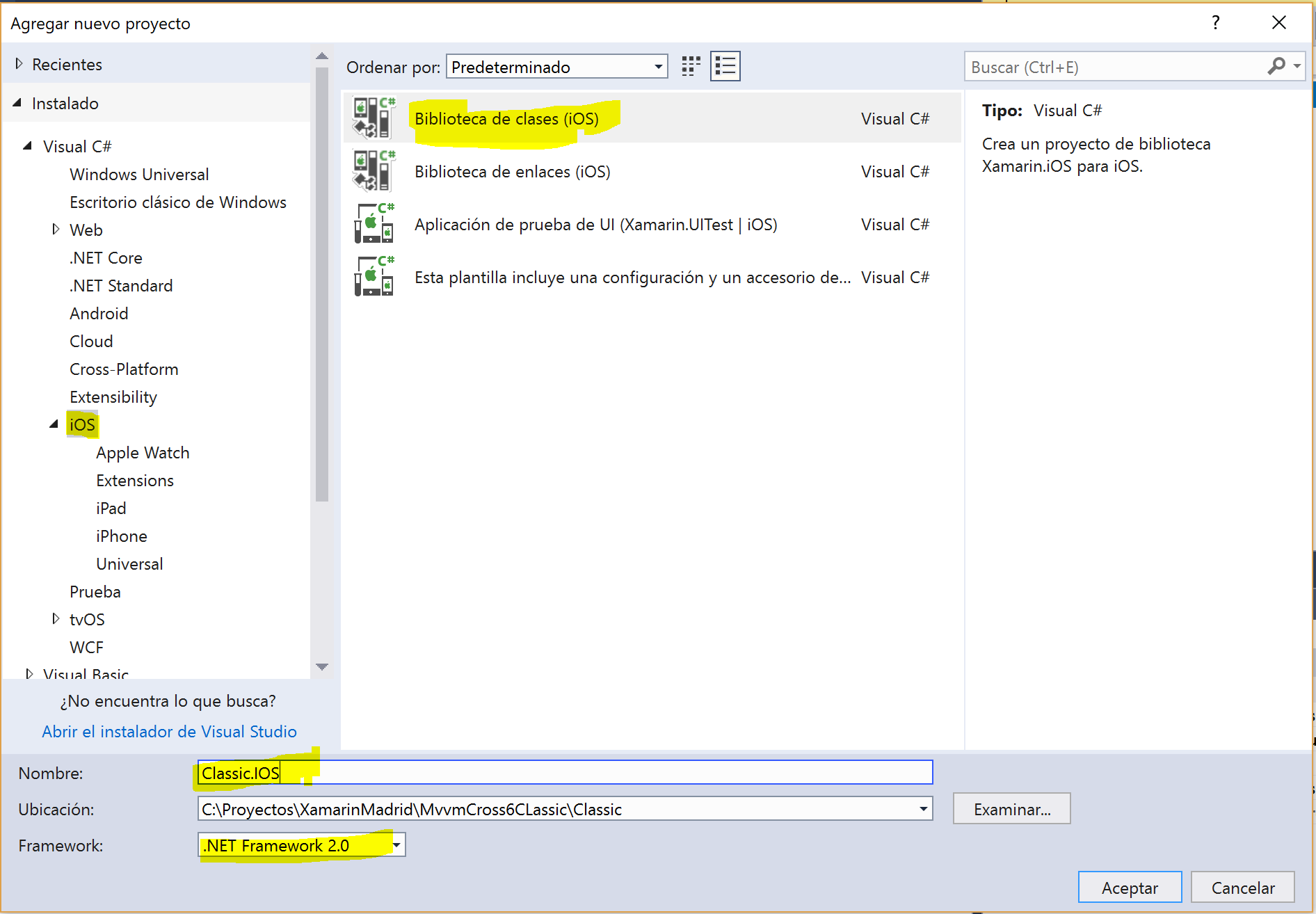
* Core,
* Android
* IOS
* UWP

Todos los proyectos de plataforma deben tener el proyecto de Core como.

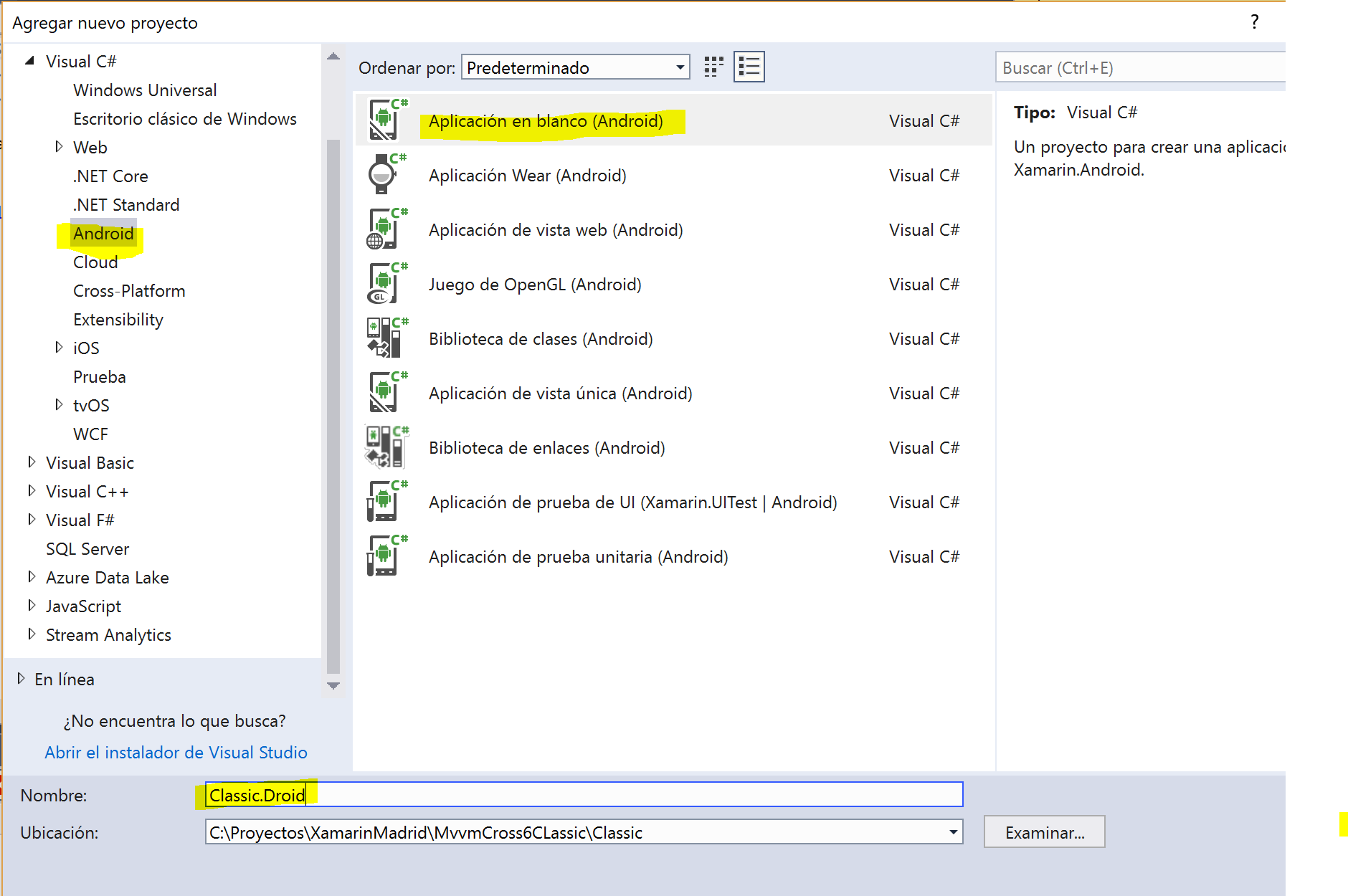
Agregando el proyecto de Core:



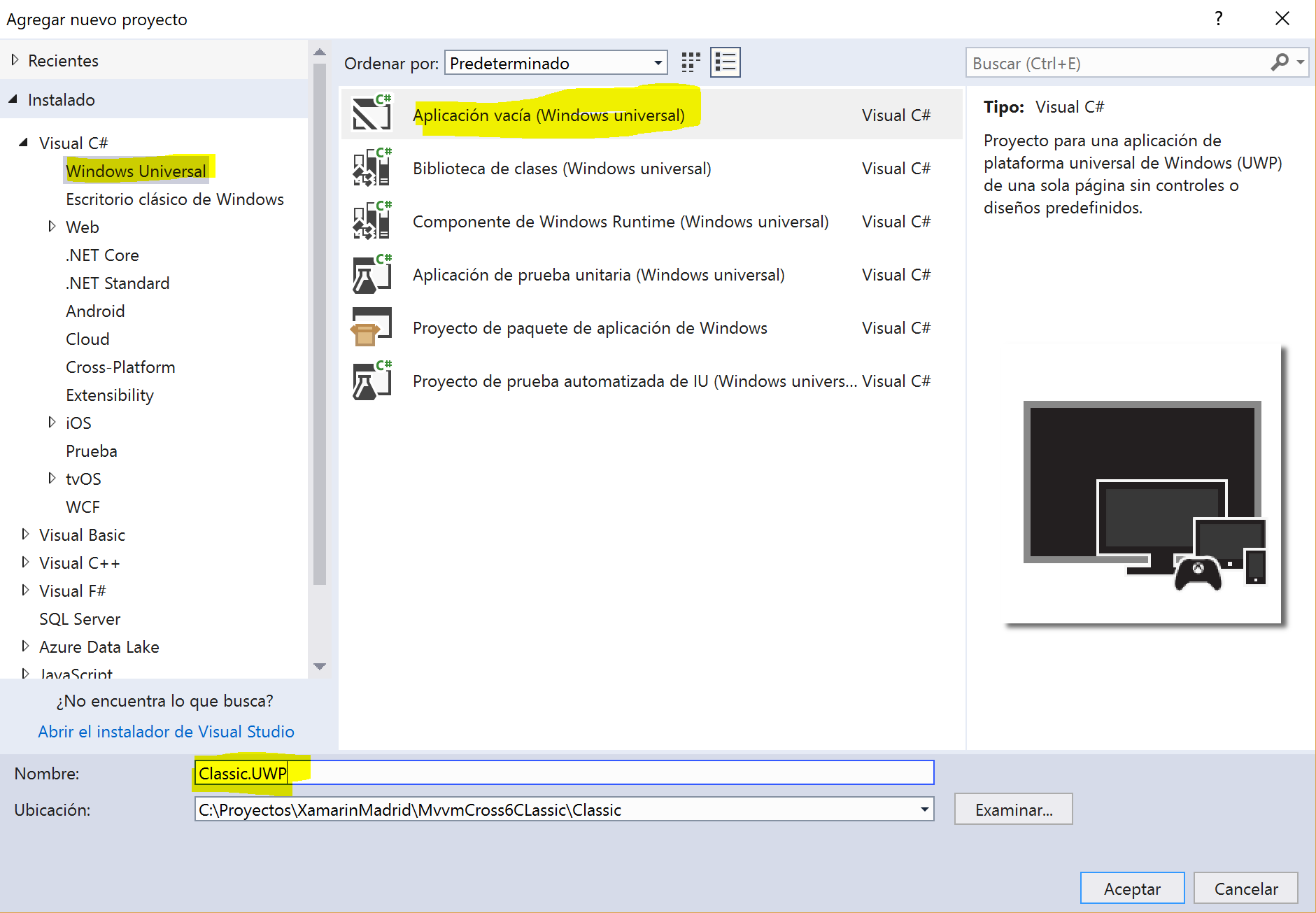
Agregando el proyecto de IOS:



Agregando el proyecto de Android:

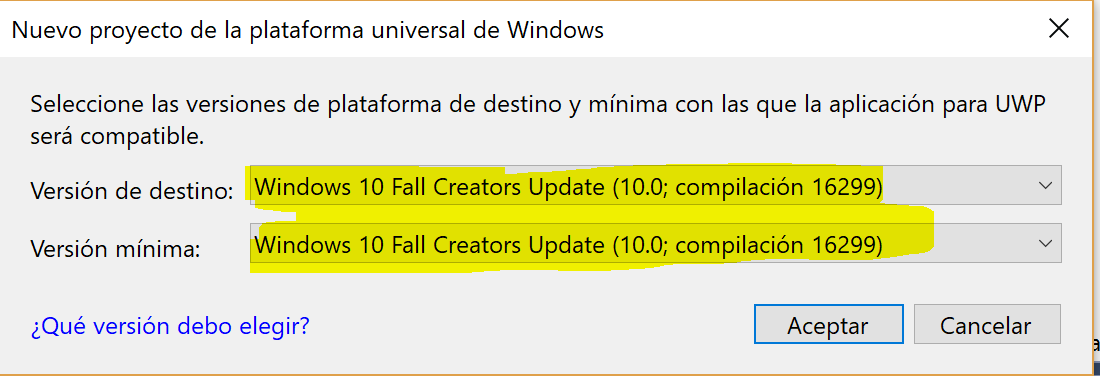


Agregando el proyecto de UWP.

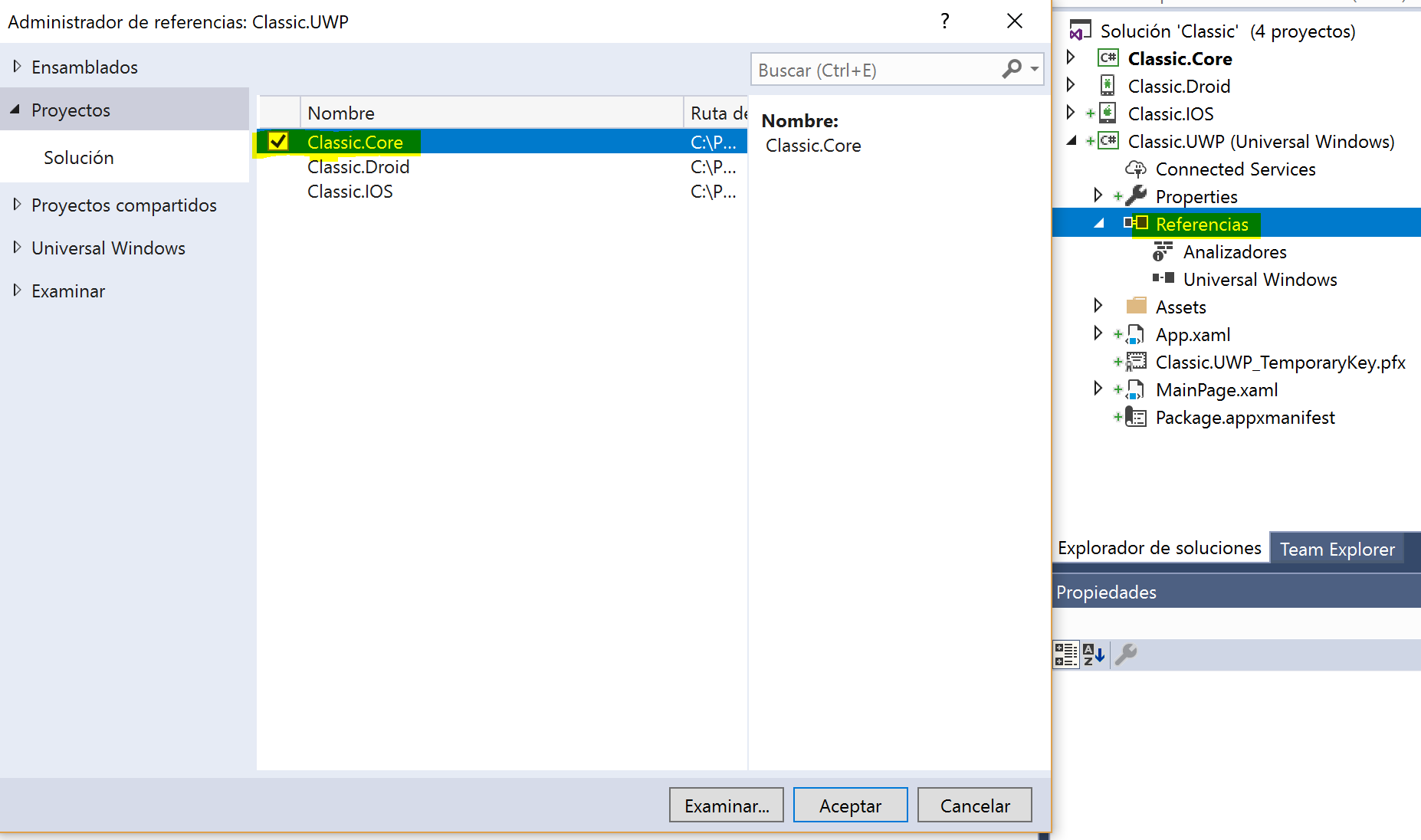


No todas las versiones de UWP son compatible con Net Standard, se debe tener como mínimo la versión “**Fall Creators**”.

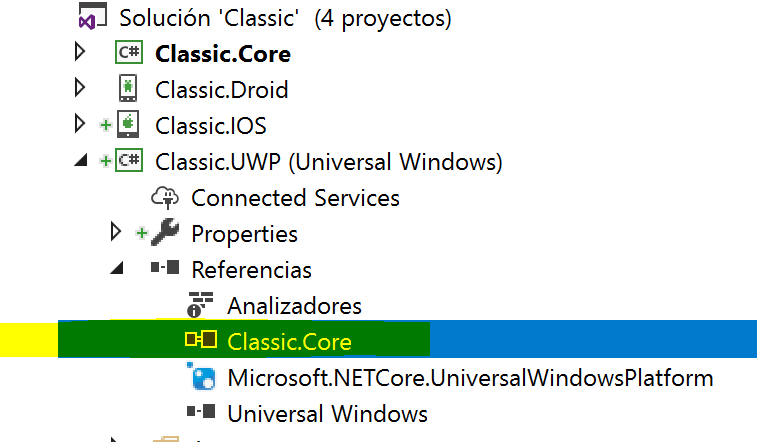
Se puede ver más detalles aquí: https://blogs.msdn.microsoft.com/dotnet/2017/10/10/announcing-uwp-support-for-net-standard-2-0/



Una vez creado todos los proyectos debemos agregar la referencia al proyecto de Core, para esto tenemos que seleccionar con el 2º botón sobre referencia, en el menú que se muestra elegir la opción “**agregar referencia**”, dentro de solución se debe seleccionar el proyecto **Classic.Core** y hacer clic en el botón **aceptar**.



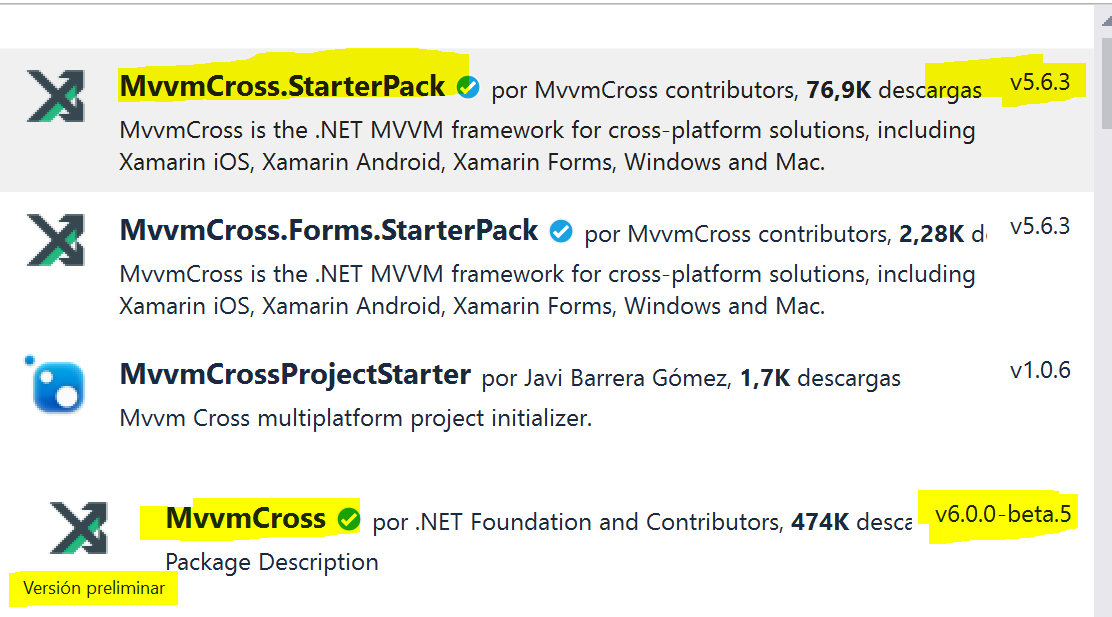
Debemos realizar la misma operación para Android y IOS. El resultado final debe ser igual a este:



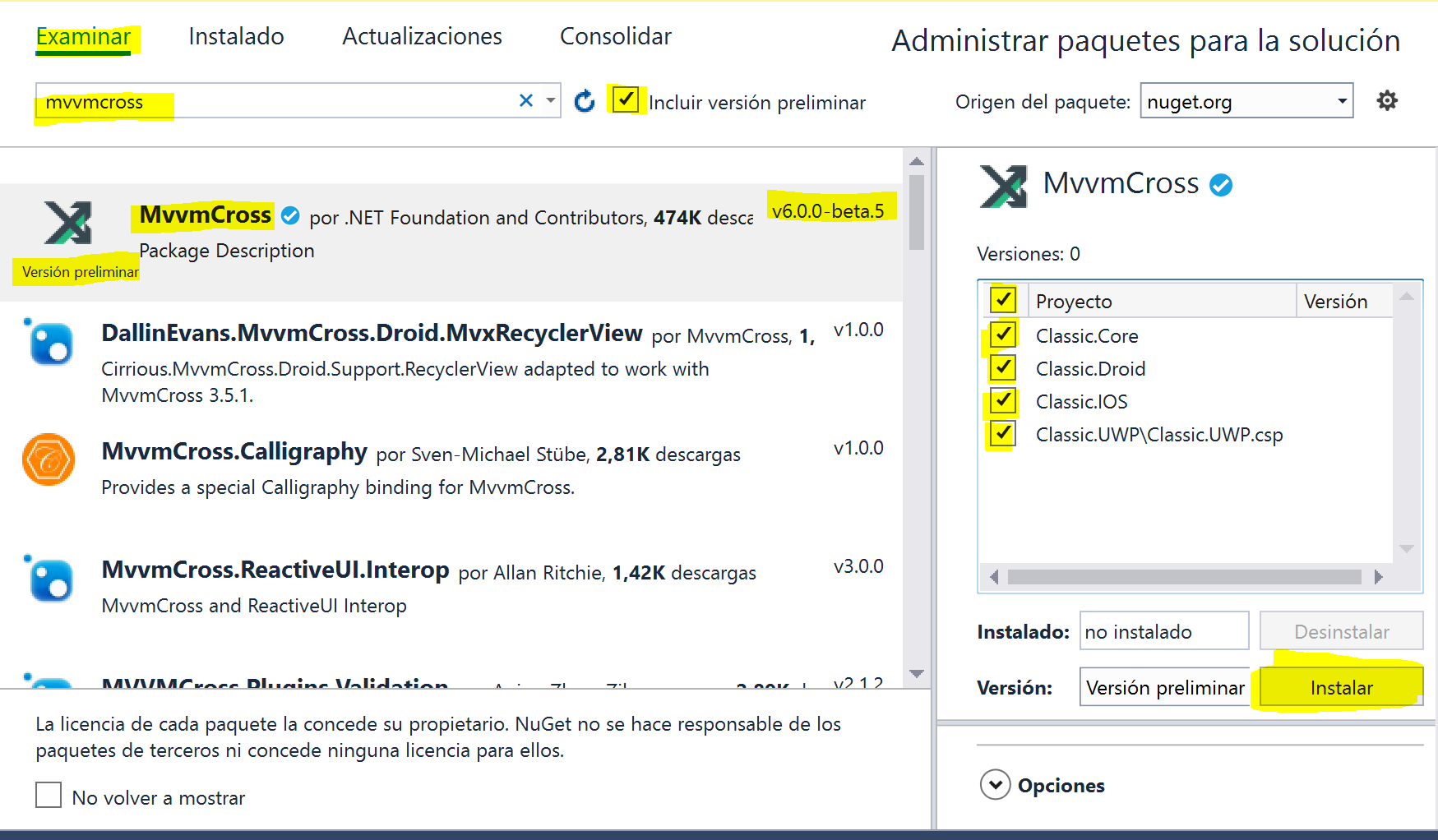
**Paso 3: Agregar MvvmCross 6.0.5 (beta)**

Al ser una versión de beta, el paquete que configura casi todo de forma automática no existe en nuget. Vamos a tener que configurar de forma manual.

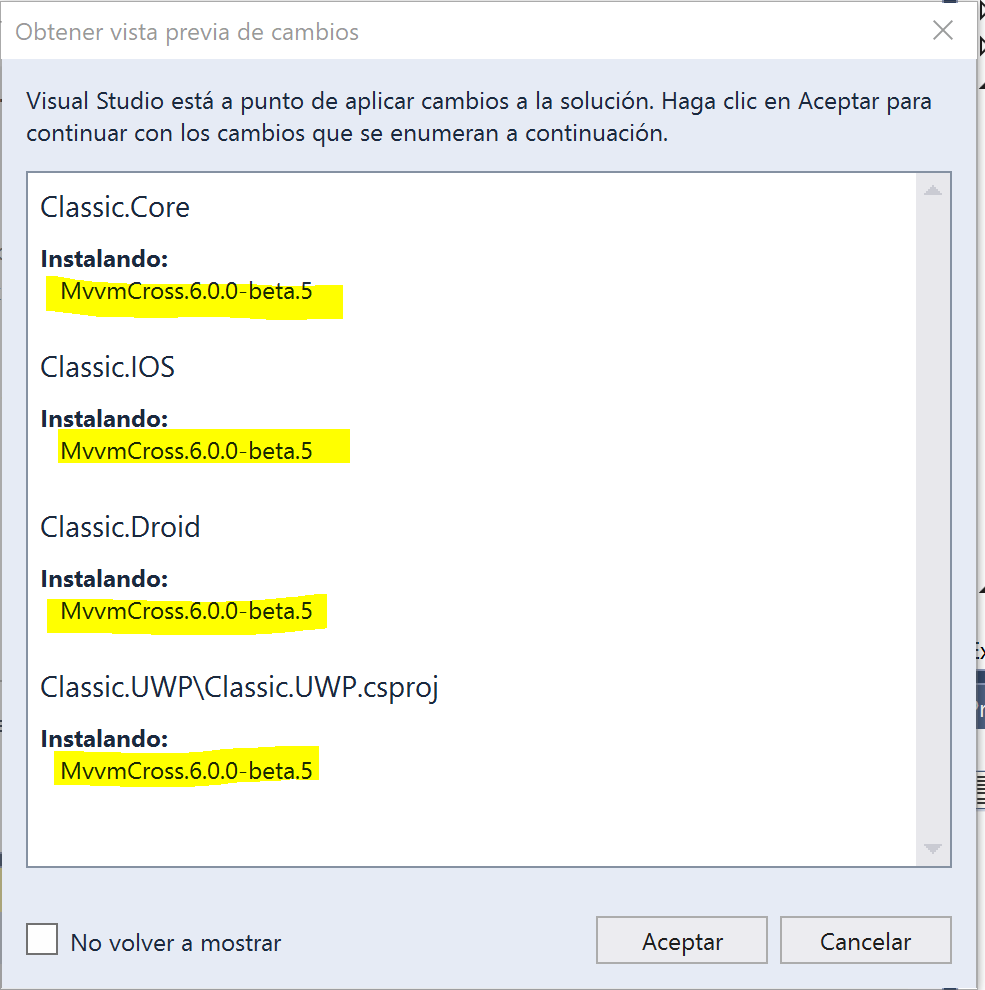
Pero de cara a futuro, revise si el paquete MvvmCross.StarterPack se encuentra disponible para la versión 6.



Para instalar **MvvmCross** debemos seleccionar la solución con el 2º botón y elegir la opción “**administrar paquetes de nuget para la solución**”, el siguiente paso es buscar por MvvmCross con la opción “incluir versión preliminar” habilitada.



Una vez elegido la versión de MvvmCross 6.0.5, debemos indicar que queremos instalar en todos los proyectos de nuestra solución. El último paso es hacer clic en instalar.



Aceptamos y toca esperar que finalice la instalación.

**Paso 4: Configurar MvvmCross en Core**

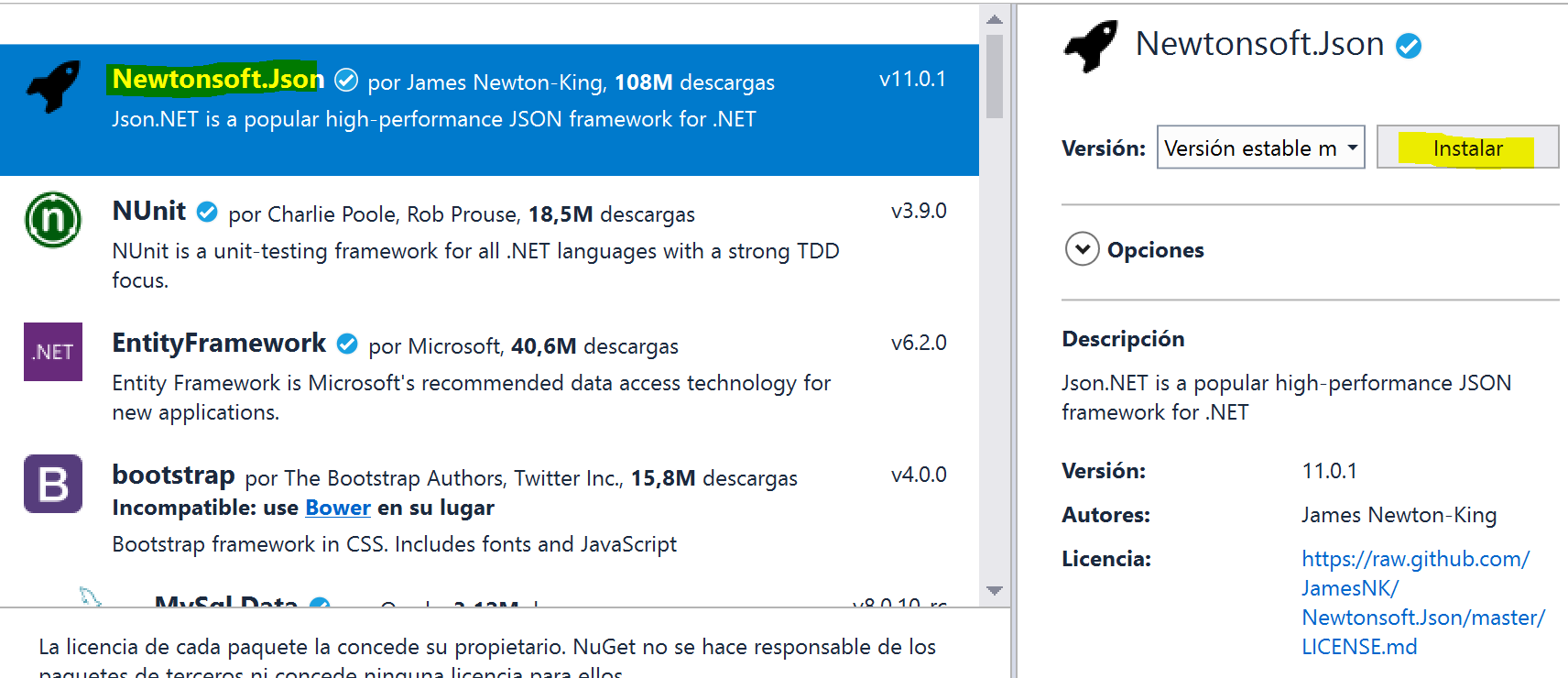
Antes de empezar a configurar debemos crear las siguientes carpetas dentro del proyecto Core:

* Constants
* Converters
  + Json
* Models
  + Base
  + Movie
* Services
  + Connectores
  + WebServices
* ViewModels
  + Base

**Paso 4.1 Agregar paquetes nuget**

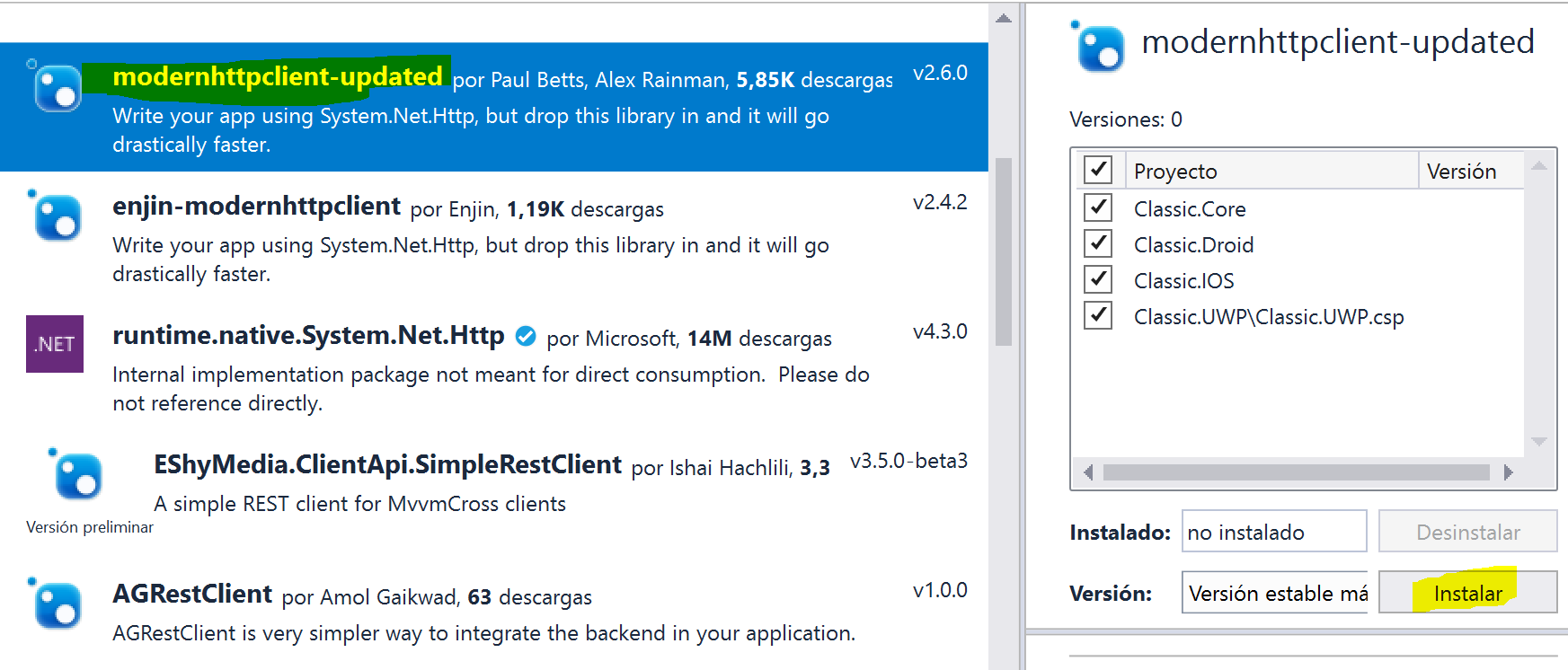
Nuestra aplicación usará la Api de Themoviedb para esto es necesario crear una cuenta en esta página: <https://www.themoviedb.org> y obtener la key para consumir datos de la API.

El siguiente paso es instalar el paquete Newtosoft.Json en el proyecto Core que será el responsable de deserializar el json del servicio.



El último paquete que vamos a instalar en el proyecto de Core, solo tiene sentido si vamos a dar soporte a versiones inferior a 5 de Android, sirve para trabajar con el httpClient nativo de cada plataforma mejorando los tiempos de respuesta. \*Se puede hacer lo mismo configurando el httpClient desde el proyecto de plataforma.

ModernhttpClient-update , se debe instalar en todos los proyectos.



**Paso 4.2 : Agregar el archivo de configuración**

Dentro de la carpeta Constants, vamos a crear una clase con el siguiente nombre ConfigConstants. Usaremos este archivo para guardar la configuración de la aplicación, se puede ver el código aquí:

[Código]

**Paso 4.3: Crear los modelos**

Vamos a crear 3 modelos:

* BaseResponse: Nos ayudará a comunicar si la petición al servicio salió bien o no.
* MoviResponse y ResultMovie: El resultado de las últimas películas de la API. Para agilizar el proceso, copie la respuesta del JSON en esta página: <http://json2csharp.com> , ella genera todas las propiedades de forma automática.

[Json del servicio]

[Código BaseResponse]

[Código MoviResponse]

[Código ResultMovie]

**Paso 4.4: Crear los Servicios**

Vamos a crear una interface para registrar el conector httpclient de la forma más simples posible.

[Código de la interfaz]

[Código de la implementación]

Creamos una nueva interfaz para agregar todos los servicios de la API de Themoviedb.

[Código de la interfaz]

[Código de la implementación]

En breve explicaré como usar.

**Paso 4.5: Crear los ViewModels**

Dentro de la carpeta ViewModels => Base, vamos a crear el archivo BaseViewModel. Todas las propiedades comunes de los viewmodels van en este archivo.

[Código aquí]

En BaseViewModel podemos destacar 3 elementos:

* Se herenda de MvxViewModel<object>
* Los servicios se inicializa por el constructor
* Contiene 2 propiedades
  + IsBusy para indicar la carga
  + Title para pintar el titulo de la pantalla.

En la carpeta ViewModels, vamos agregar nuestra página principal: HomeViewModel

En este archivo podemos destacar:

* Herenda de BaseViewModel
* Al cargar la página (ViewAppeared), se llama el servicio y el resultado se agrega en una propiedad publica que notifica la vista.

[Código aquí]

**Paso 4.6: Agregar el arranque de la aplicación**

Debemos crear el archivo App.cs, este archivo es el responsable por arrancar la aplicación en el viewmodel indicado, aparte de registrar los servicios.

Todos los servicios terminados por service, se registra de forma automática por MvvmCross. Este es el caso de nuestros 2 servicios (IWebClientService y IMovieWebService).

[Código aquí]

**Paso 5: Configurar UWP**

Lo primero que debemos crear es un archivo Setup.cs dentro del proyecto Classic.UWP.

[Código aquí]

Este archivo es el responsable por arrancar MvvmCross.

El siguiente es eliminar el archivo MainPage.Xaml, debemos crear páginas que herende directamente de MvvmCross.

Próximo para es una carpeta con el nombre de Views, agregaremos nuestras páginas en esta carpeta.

El último paso en la configuración es modificar el archivo App.xaml.cs, debemos agregar la configuración de MvvmCross para controla la navegación.

[Código aquí]

Explicar el uso del converter y como usar las páginas