47. What are NuGet packages?

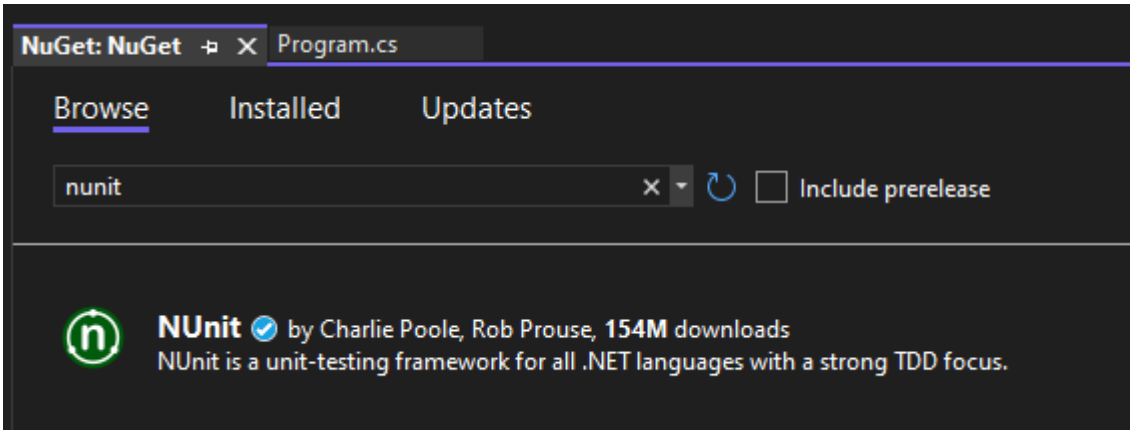
Brief summary: NuGet packages contain compiled code that someone else created, that we can reuse in our projects. The tool used to install and manage them is called NuGet Package Manager.

NuGet is a Microsoft-supported package manager, so a tool through which developers can create, share, and consume useful code.

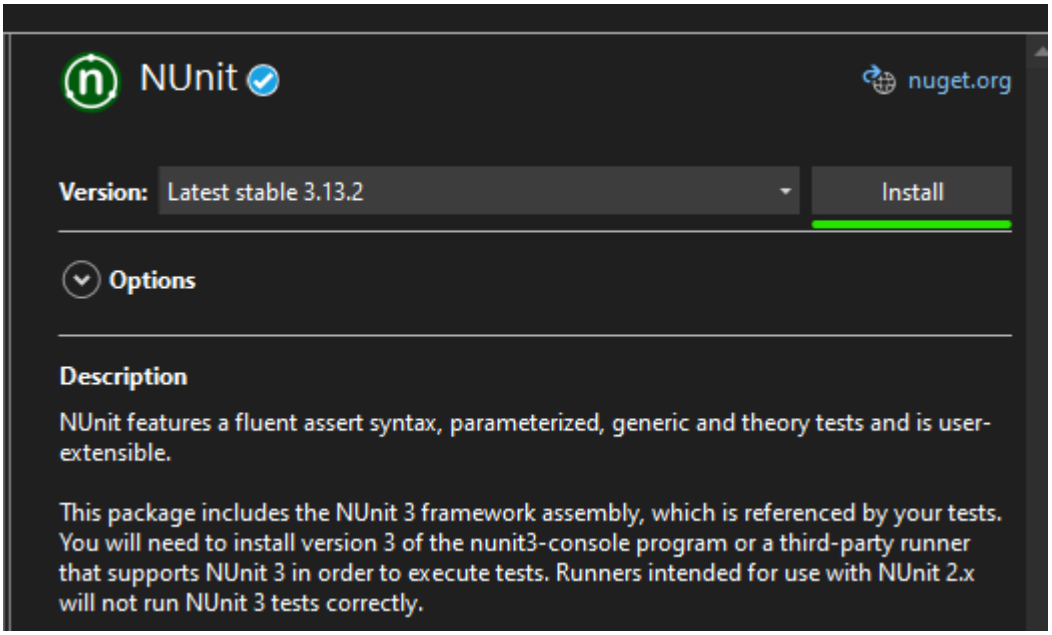
There are tons of libraries that other developers created, which we can use in our own projects. NuGet Package Manager is the tool that allows us to access them. Each package contains the dlls built from the code that someone else developed.

For instance, let's add NUnit and Moq to a project. NUnit is one of the most popular unit testing frameworks for C#, and Moq is a mocking library. Together they are two essential tools that we can use to create unit tests for our code.

The easiest way to install a NuGet package is by right-clicking on the project and selecting "Manage NuGet Packages". On the screen that opens we can search for the package that we want to install:



After selecting it, we can choose the version we want to install. Let's select the latest one.



After installing the package, we can start using it:

```
using NUnit.Framework;

[TestFixture]
0 references
public class Tests
{

}
```

There is another, sometimes even more convenient way of installing NuGet packages. We can simply start using the types from the package, and once Visual Studio complains that it doesn't know them, we can choose to install the package from the context menu:

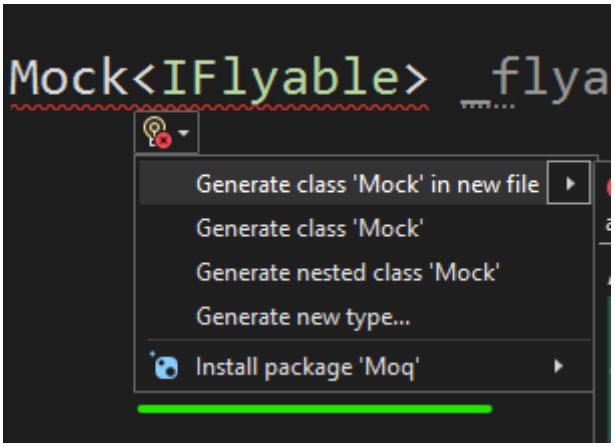
```
public interface IFlyable
{
    0 references
    public void Fly();
}

[TestFixture]
0 references
public class Tests
{
    private Mock<IFlyable> _flyableMock;
}
```

CS0246: The type or namespace name 'Mock<>' could not be found (are you missing a using directive or an assembly reference?)

Show potential fixes (Ctrl+.)

In this case, I'm trying to use Mock type from the Moq framework, which is not currently installed. I can click on the suggestion button to see that Visual Studio kindly offers to install this NuGet package for me:



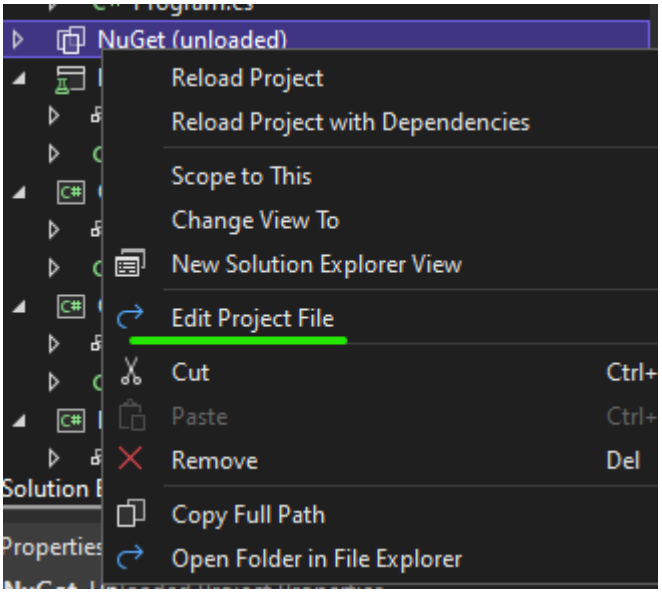
After doing so, the code compiles correctly:

```
using Moq;
using NUnit.Framework;

1 reference
public interface IFlyable
{
    0 references
    public void Fly();
}

[TestFixture]
0 references
public class Tests
{
    private Mock<IFlyable> _flyableMock;
}
```

Let's take a look at how the *.csproj file changed after installing those two packages. To see the *.csproj file of the project we must first unload it. Right-click on it and select "unload". After, you can right-click on the unloaded project again and select "Edit Project File".



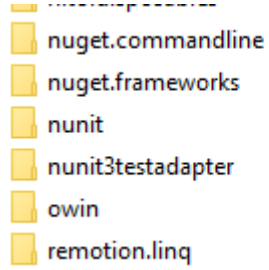
In the *.csproj file that will open we will see the entries that have been added by NuGet:

```
<ItemGroup>
  <PackageReference Include="Moq" Version="4.16.1" />
  <PackageReference Include="NUnit" Version="3.13.2" />
</ItemGroup>
```

The question is: where exactly did the packages get installed? Well, this evolved with the versions of .NET, but in .NET 6 which we use in this course, it by default gets installed in your Windows's user folder, for example in a path like this:

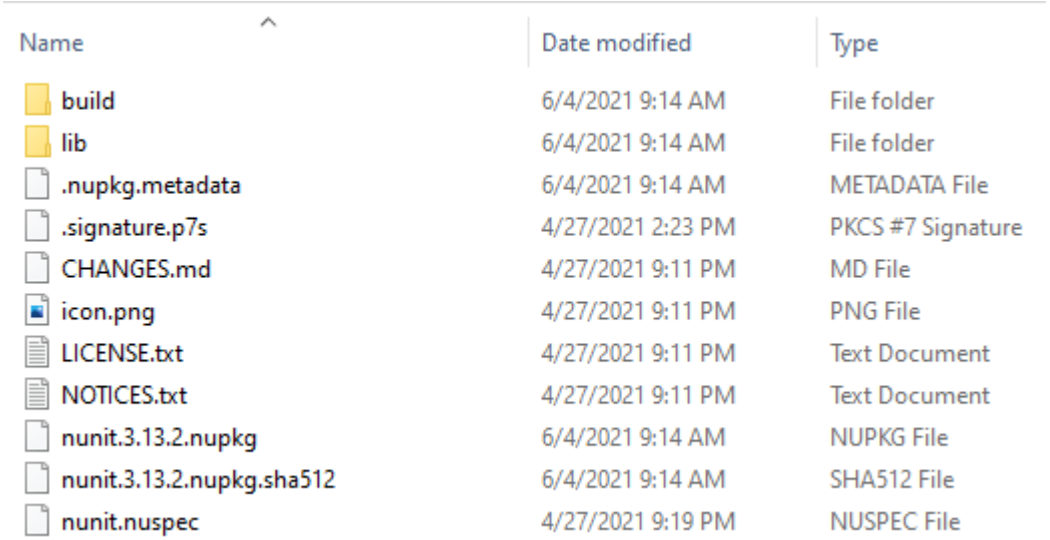
C:\Users\Krystyna\.nuget\packages

And here we can see the nunit folder:



Don't worry if for you it looks different. I have dozens of different coding projects on my computer and overall 344 NuGet packages installed.

Thanks to the fact that the package gets installed in the user folder, it can be reused between different projects. I have multiple projects using NUnit, but it only exists in a single copy on my machine. Let's take a look at what's inside such a NuGet package:



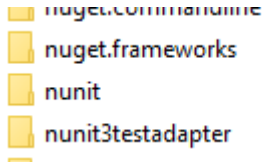
In the **lib** folder, we can find the actual dlls that get referenced from our project.

Now, I'll do something mean. I will remove the entire nunit folder from the .nuget\packages directory.

Let's see if the project will build correctly. After all, the dlls it needs have been deleted.

```
1>Done building project "NuGet.csproj".
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

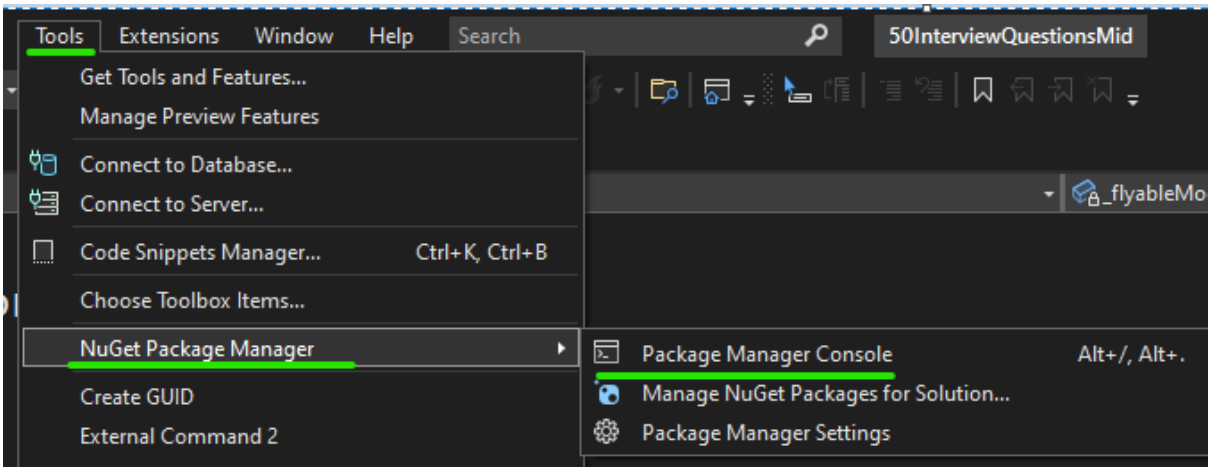
That's a bit surprising. The build was successful. Let's take a look into .nuget\packages directory again.



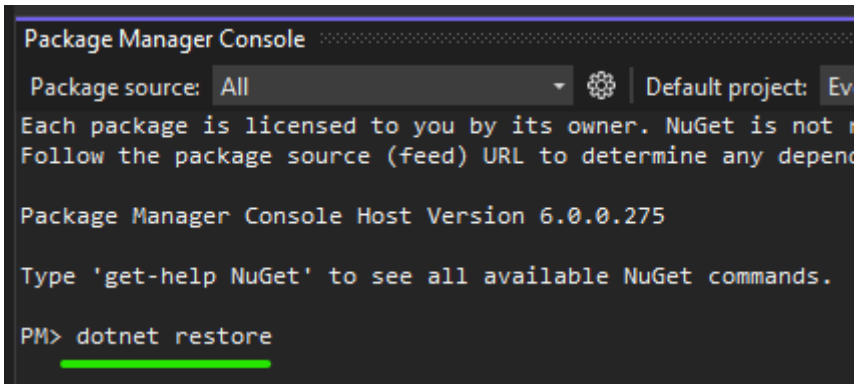
It seems like the nunit folder has "magically" reappeared.

Actually, it's no magic. The *.csproj file now clearly declares what packages it needs. Visual Studio knows that if the package is missing from the packages folder, it must simply reinstall it. This is quite convenient, especially if we share the code via some kind of repository. We only commit the code and package references to the repository, not the packages themselves. Once another programmer downloads the code and builds it, the packages get installed on his or her machine automatically.

One more thing. Sometimes Visual Studio messes something up and is not able to restore the packages. If this happens, you can always run Tools-> NuGet Package Manager -> Package Manager Console…



…and from this console, run "**dotnet restore**" command, which will restore all packages referenced in the solution.



All right. We now know how to use the packages that someone else created. I highly recommend you use this beautiful concept, and not reinvent the wheel each time you need something done. In general, when you think of something not specific to your project, but rather something that could likely be used by other developers, there is a 99% chance there is a NuGet package that already does that.

If you have some nice ideas of your own, you can always create and publish your own NuGet packages. Here is a series of articles about it:
https://docs.microsoft.com/en-us/nuget/create-packages/overview-and-workflow