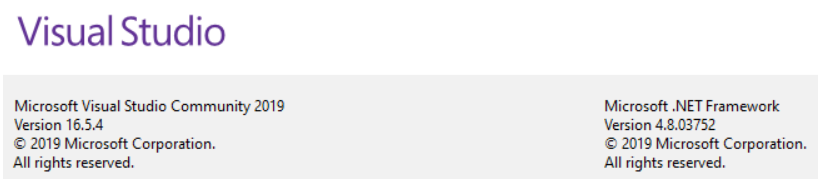
Testning för C++ i Visual Studio – GoogleTest

Guiden beskriver hur du sätter upp ett projekt med testningsmiljö i Visual Studio.
Totalt kommer vi skapa tre olika projekt i en och samma ”Solution”:

- Software : Vårt program, här använder vi endast definierade klasser.
  Av någon typ, Command Line i detta exempel.
- Software_lib : Vår bibliotek, här definierar vi klasser som vårt system kommer bestå av.
  Av Typen Static Library.
- Software_test : Vår testmiljö, alla tester.
  Av typen Google Test.



Guiden är skriven för Visual studio 2019

tldr; Guiden går igenom följande;

1. Skapa Software Projektet
2. Skapa Software_lib Projektet (Static Library), som delprojekt
3. Referera till Software_lib från Software projektet
4. Skapa Software_test Projektet (Google Test), som delprojekt
5. Referera till Software_lib från Software_test projektet

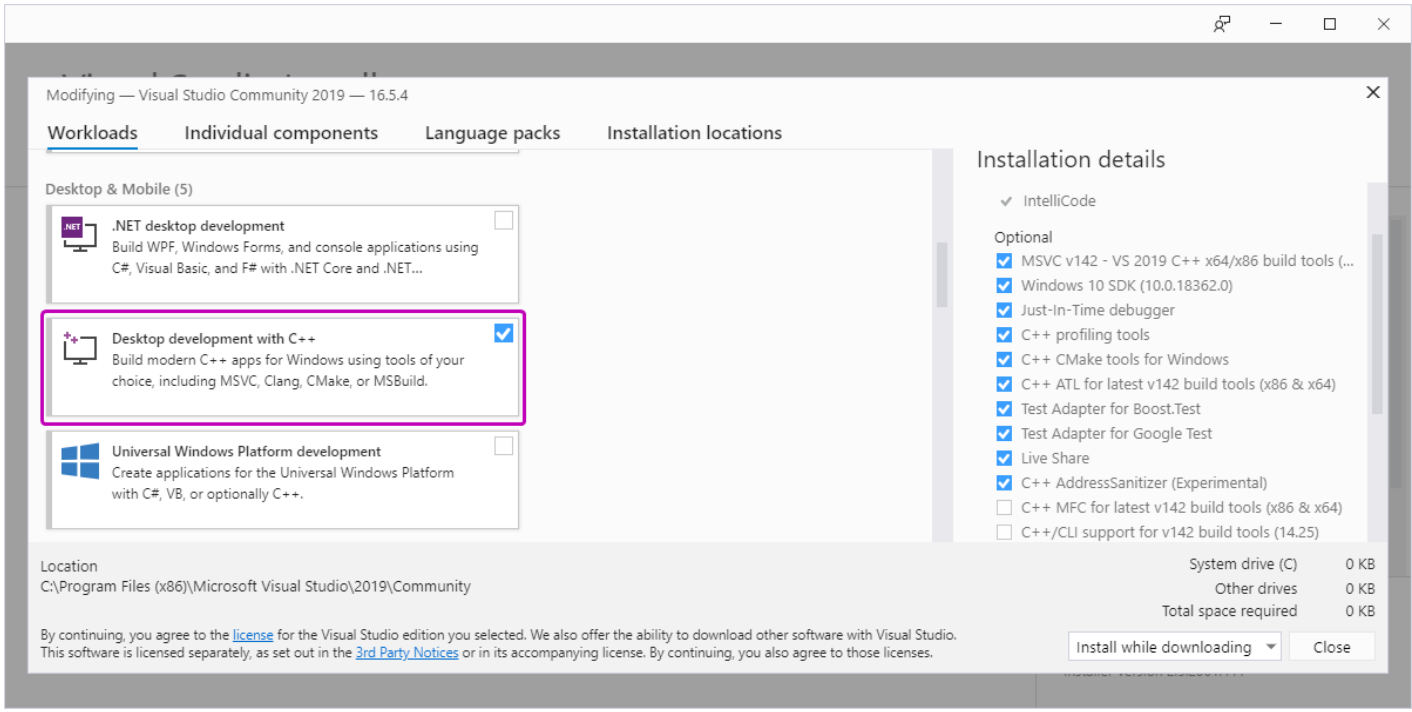
- Sista sidorna innehåller en kort demo för hur man jobbar mellan projektet samt en cheatsheet för GoogleTest

Krav

Google Tests är Thread-safe om pthreads används, ***Google Test är inte Thread-safe på Windows platformar***.

Enda kravet är att paketet/workload ”Desktop development with C++” är installerat.
Om det saknas eller om du är osäker kan du enkelt installera/kontrollera det genom

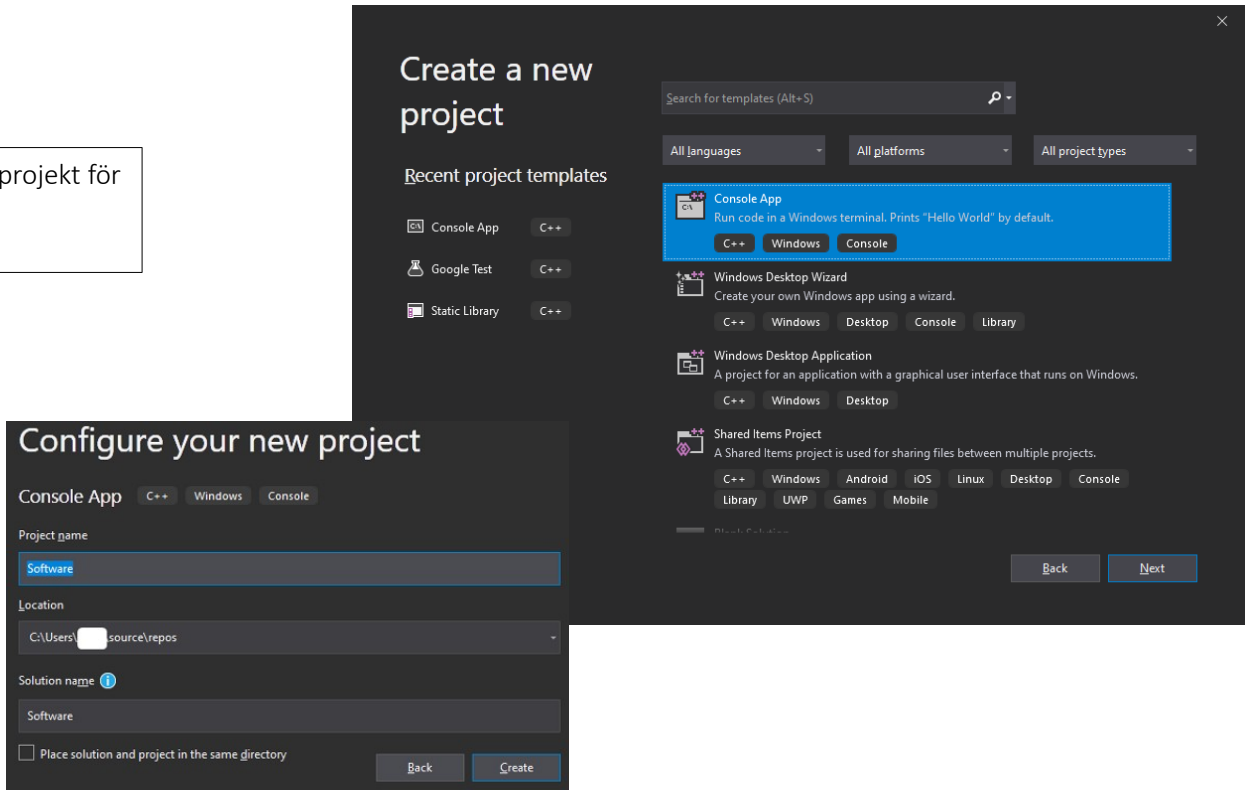
Visual Studio → Tools → Get Tools and Features...



Ställa in Visual Studio – Testning med GoogleTest

Skapa Software Projeket

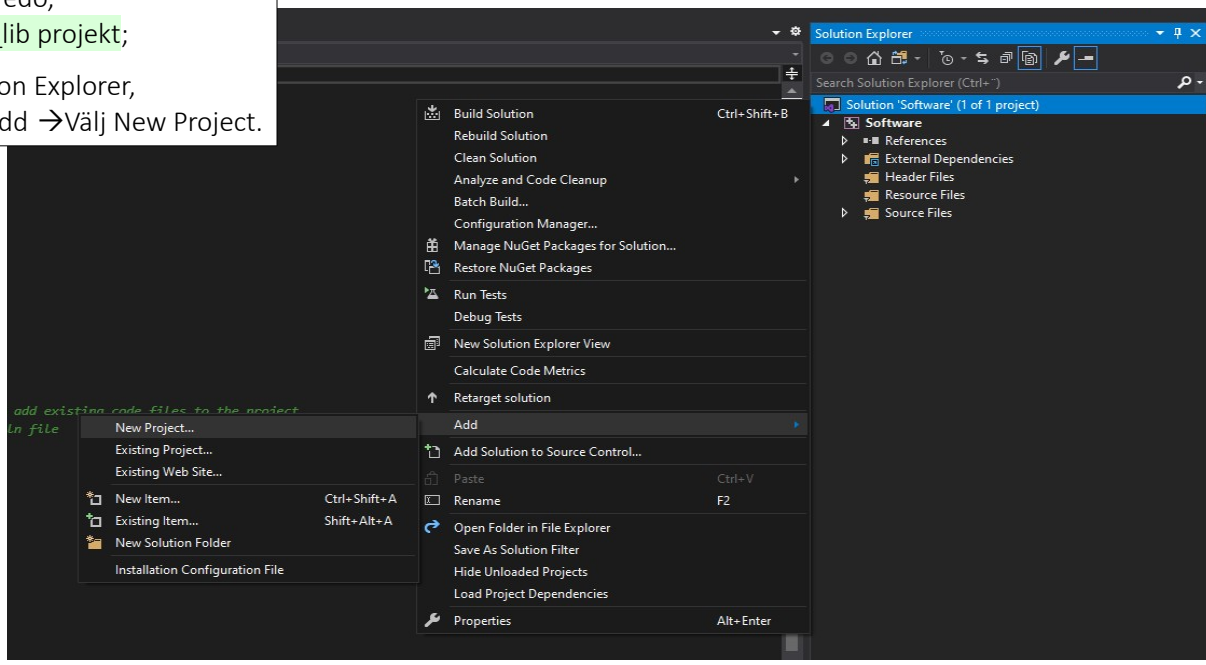
Börja med att skapa ett nytt projekt för systemet du ska bygga.



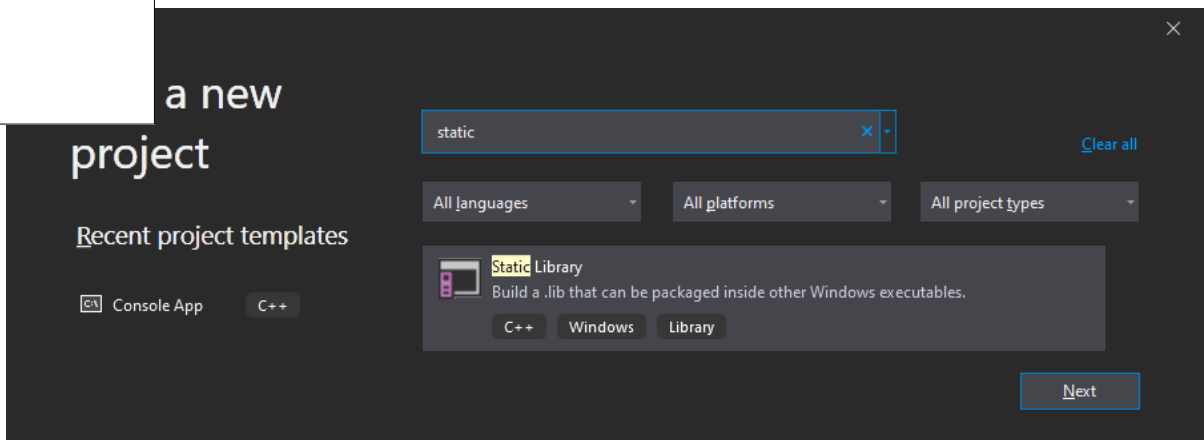
Skapa Software_lib Projeket

Nu är fösta projeket (Software) redo,
härnäst skapar vi vårt Software_lib projekt;

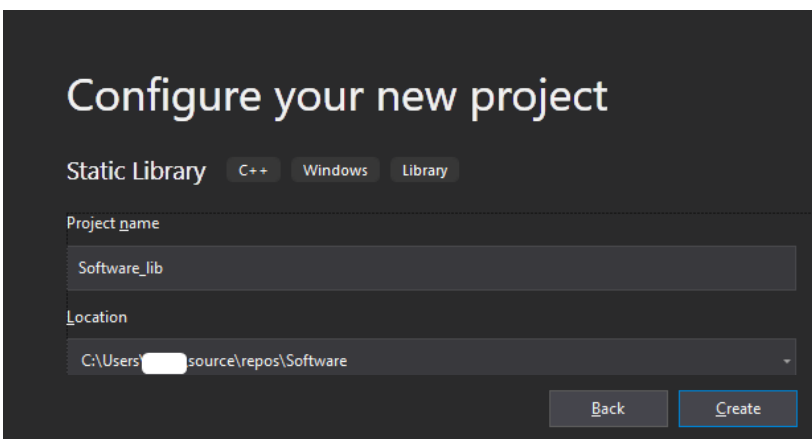
När projektet är öppnat, i Solution Explorer,
högerklicka På Solution →Välj Add →Välj New Project.



Sök efter Static Library,
Välj ***Static Library***,
Tryck Next



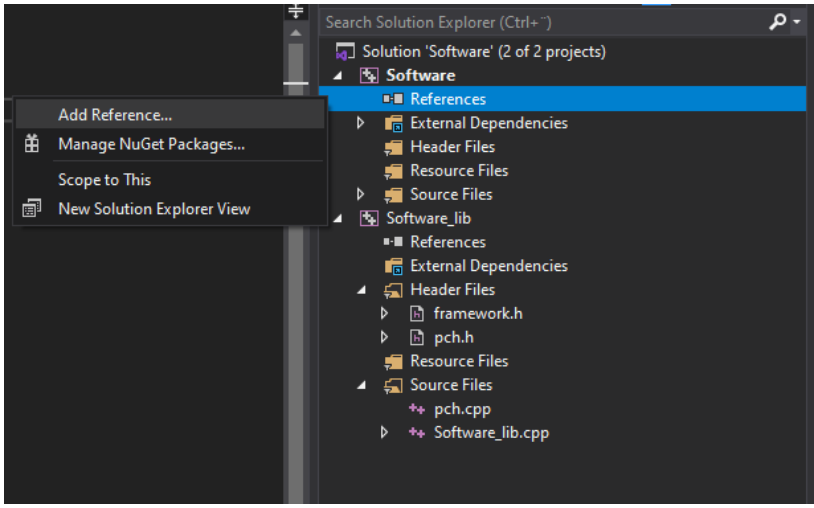
Välj ett namn,
Tryck Create



För Software Projektet, Lägg Till Software_lib Som Referens

Refera till Software_lib från Software.

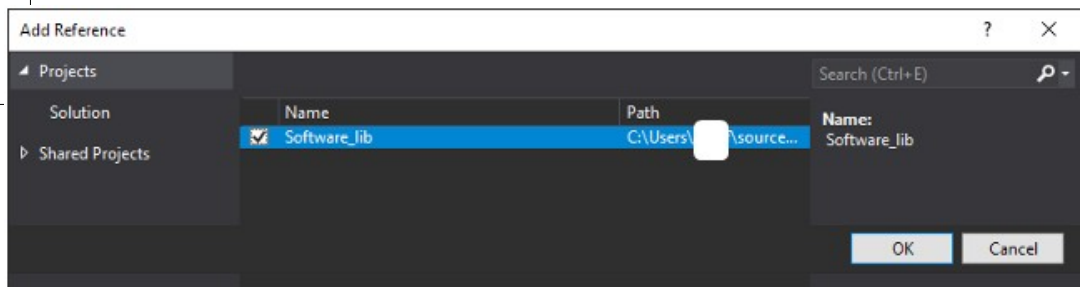
Solution Explorer
→ under Software
→ högerklicka på ***References***
→ välj Add Reference...



Bocka i Software_lib som referens.

Tryck OK.

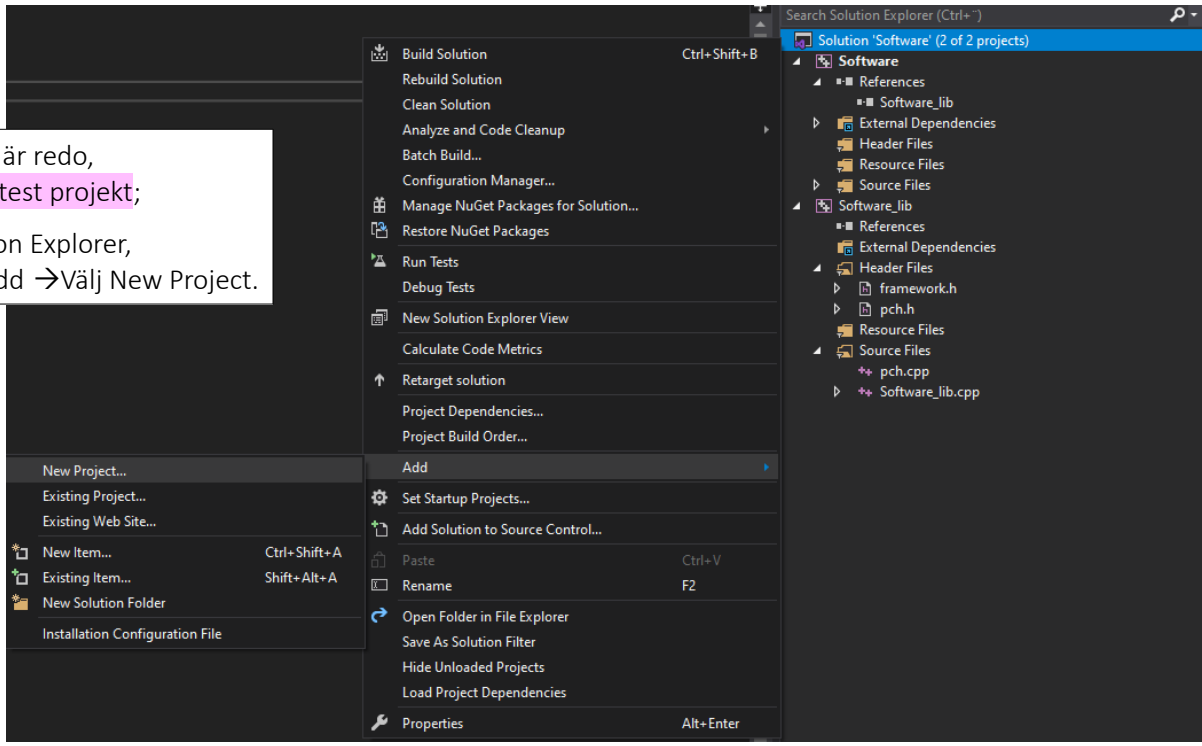
Software_lib bör nu ligga som en referens i Software Projektet, vilket även visas i Solution Explorer. under projektets referenser...



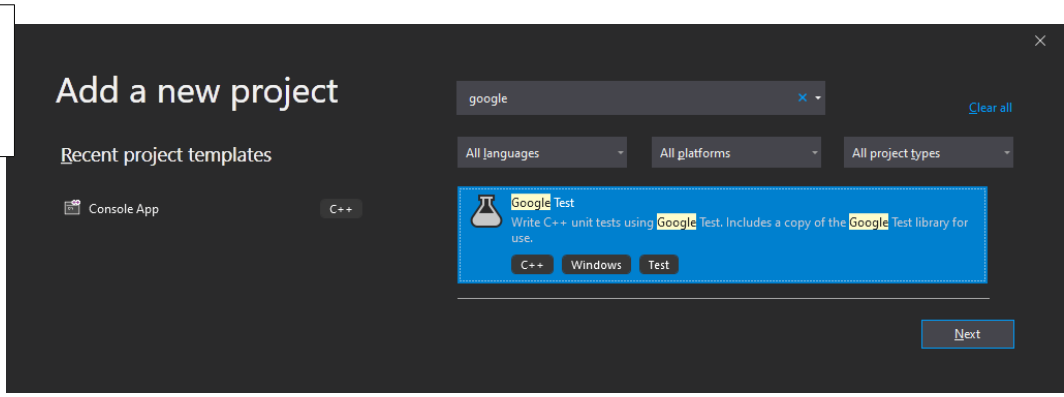
Skapa Software_test Projeketet

Både Software och Software_lib är redo,
härnäst skapar vi vårt Software_test projekt;

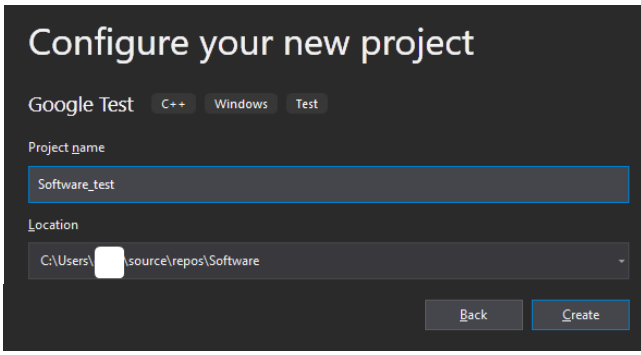
När projektet är öppnat, i Solution Explorer,
högerklicka På Solution →Välj Add →Välj New Project.



Sök efter Google Test,
Välj Google Test,
Tryck Next



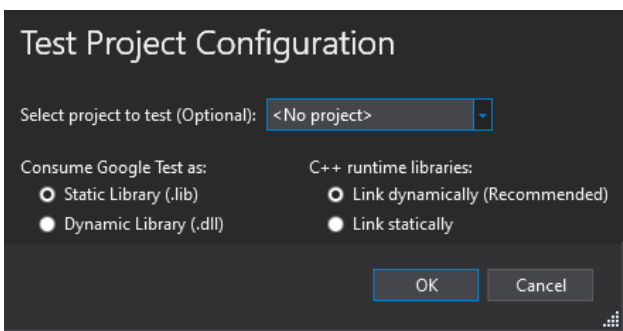
Välj ett namn,
Tryck Create



Ett fönster kommer upp efter att du tryckt create.
Följande inställningar är rekommenderade;

Project to test : <No project>
Consume Static Library : Static Library (.lib)
C++ runtime libraries : Link dynamically (Recommended)

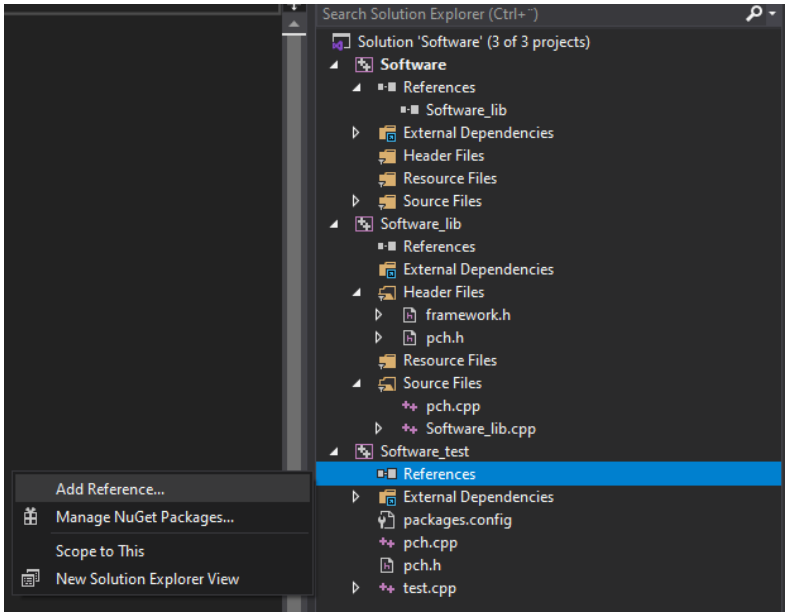
Tryck OK.



För Software_test Projektet, Lägg Till Software_lib Som Referens

Refera till Software_lib från Software_test.

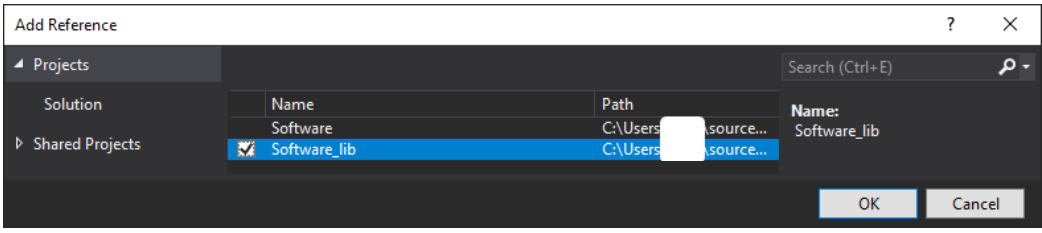
Solution Explorer
→ under Software_test
→ högerklicka på ***References***
→ välj Add Reference...



Bocka i Software_lib som referens.

Tryck OK.

Software_lib bör nu ligga som en referens i Software_test Projektet.



Sammanfattning av setup

Totalt har vi skapat 3 Projekt, varav alla befinner sig i samma Solution.
Varav Software och Software_test har angett Software_lib som referens.

- Software var det ursprungliga projektet det skapades som
  Projekt av typen Console Application, *då målet i guiden vara skapa en Console Applikation*.

- Software_test
  Projekt av typen Google Test.

- Software_lib
  Projekt av typen Static Library.

Att skriva Tester i Google Test

Följande del går igenom hur vi skriver tester. Vi utgår från att du följt guiden och har 3 projekt i samma solution.

Hur Vi Använder Google Test I Visual Studio

Visual Studios Test Explorer

Test explorer är ett verktyg i Visual Studio som listar alla dina tester och låter dig köra dem.
Du finner Test Explorer under:

Test → Test Explorer
(*alt. shortcut: CTRL+E, T*)

När du väl definierat ett test så kommer det dyka upp i Test Explorers lista av test så fort du trycker ”Run all tests”.
Du kan välja att köra ett specifikt test eller om alla definierade tester ska köras.

Software_test projektet

När ditt projekt av typen Google Test är skapat bör en *test.cpp-fil* ha skapats,
denna beskriver hur ett enkelt test kommer se ut.

Du kan skapa nya .cpp-filer för varje samling test som ska utföras, se till att:

- Varje .cpp-fil du skapar måste du se till att inkludera ”pch.h”.
- När du anger en headerfil för en klass definierad i ditt Static Library projekt måste du ändra mapp, använd:
  - #include ”../Some_lib/someClass.h” ← RÄTT : *anger att filen finns i en katalog över den aktuella...*
  - #include ”someClass.h” ← FEL : *påstår att filen finns i samma katalog som testerna...*

Testens kropp

Alla tester består av tre faser; Arrange, Act, Assert. (*förkortas ofta AAA*)

I **Arrange** fasen sätter vi upp miljön vi ska utföra testet i, det innebär att man behöver deklarera vilka variabler som behövs samt mata dessa med relevant data vi vill testa.

I **Act** fasen utför vi funktionaliteten som ska testas, denna del kommer antingen returnera ett värde vi kan testa eller indirekt ändra ett värde i något annat objekt. Det viktiga är att Act fasen ska leda till ett resultat som vi ska kunna kontrollera.

I **Assert** fasen kontrollerar vi om resultatet från Act fasen är som förväntat.
Detta görs med någon av Assert funktionerna som medföljer Google Test biblioteket (*Se sista sidan för cheatSheet*).

Simpla Tester

Ett vanligt test definieras genom macrot ”**TEST(*TestSuiteName*, *TestName*)**”,

Första parametern:	Namnet för den Enhet vi testar. Namnet för vår Testsuit.
Andra parametern:	Namnet på själva testet. Vi beskriver vilken funktionalitet av enheten vi testar.

För dessa tester kommer alla delarna av AAA befinna sig i testets kropp.

Fixtures Tester

Vi kan använda något som heter Fixtures för att slippa återupprepa kod. I princip kan man säga att Fixtures låter oss *bl.a*. definiera **Arrange** fasen i en klass som kan användas av andra test.

För att definera ett test som använder Fixtures använder vi Macrot:

”**TEST_F**(*FixtureClass*, *TestName*)”

Första parametern av **TEST_F** beskriver vilken Fixture som ska användas. Vi behöver skapa en Fixture klass, vilket görs genom att skapa en klass som ärver från **testing::Test**.

```
#include "pch.h"
#include "../Software_lib/Cat.h"

class CatTests_fixture :  public testing::Test {
protected:          // <-- OBS, not Private, use Protected!
   Cat  myCat;

   void SetUp()       override {/*Arrange code here*/}
   void TearDown()    override {/*Release allocated resources*/}
public:
   CatTests_fixture() {/*Or Arrange code here*/ }
   ~CatTests_fixture(){/*Or Release allocated resources*/ }
};

TEST_F(CatTests_fixture, def_constr_test) {

   ASSERT_EQ(myCat.getAge(), 42); // If cats age is 42, do not assert...
}
```

Exemplet till höger demonstrerar en fixture klass samt ett fixture som använder sig av klassen, **Notera** att vi *använder* **protected** access modifier istället för **private**.

Varje Fixture har en SetUp och TearDown metod som kan överskridas, de har samma syfte är samma som klassens konstruktor och destruktor;

Att sätta upp klassinstansen, ev. allokera data,
Att att plocka ner klassinstansen, ev. frigöra data.

Ett Fixture tests delar utförs i denna ordning;

Fixture Constructor
Fixture SetUp
Fixture Body ← *Definierad i TEST_F macrots kropp…*
Fixture TearDown
Fixture Destructor

Skillnad mellan SetUp/TearDown och konstruktor/destruktor:

- konstruktor/destruktor

  Använd dessa om följande är viktigt för dig:

  - Tillåter dig att definiera medlems variabler som const.
  - Underlättar om vi vill ärva från denna fixture då basklassens Construktor och Destruktor automatiskt körs.

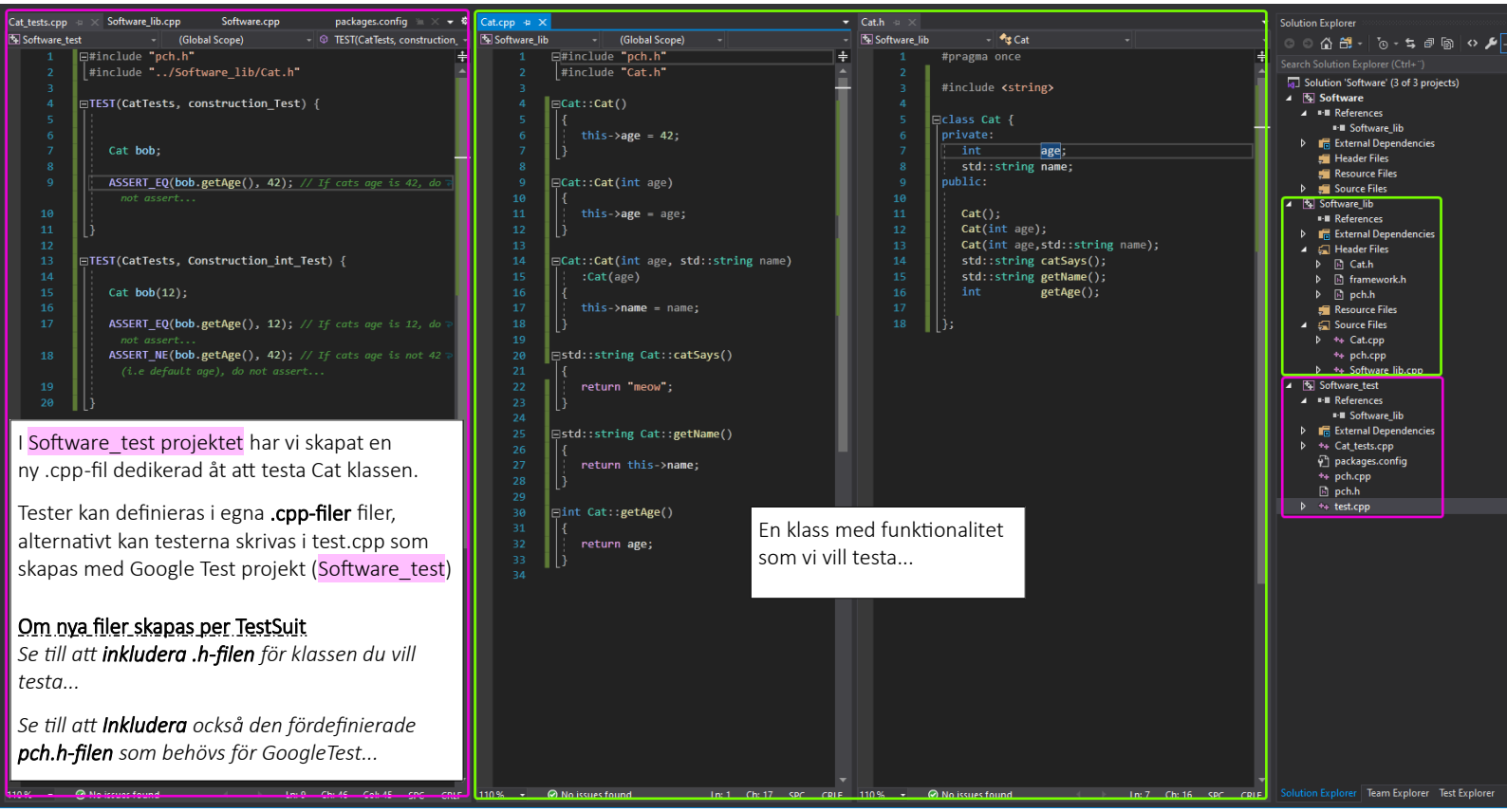
- SetUp/TearDown

  Använd dessa om följande är viktigt för dig:

  - Tillåter dig att anropa överskridna virtuella funktioner.
  - Tillåter dig att använda ASSERT_xx macros.
  - Tillåter dig att kasta exceptions.

En Fixture kan användas av flera tester, de delarna av testerna som varierar kan antingen definieras direkt i Test kroppen eller om flera tester behöver samma variation så kan en ny fixture klass skapas genom att ärva från den redan definierade.

Mer info om GoogleTest finns här....
https://github.com/google/googletest/blob/master/googletest/docs/primer.md



I Software_test projektet har vi skapat en ny .cpp-fil dedikerad åt att testa Cat klassen.

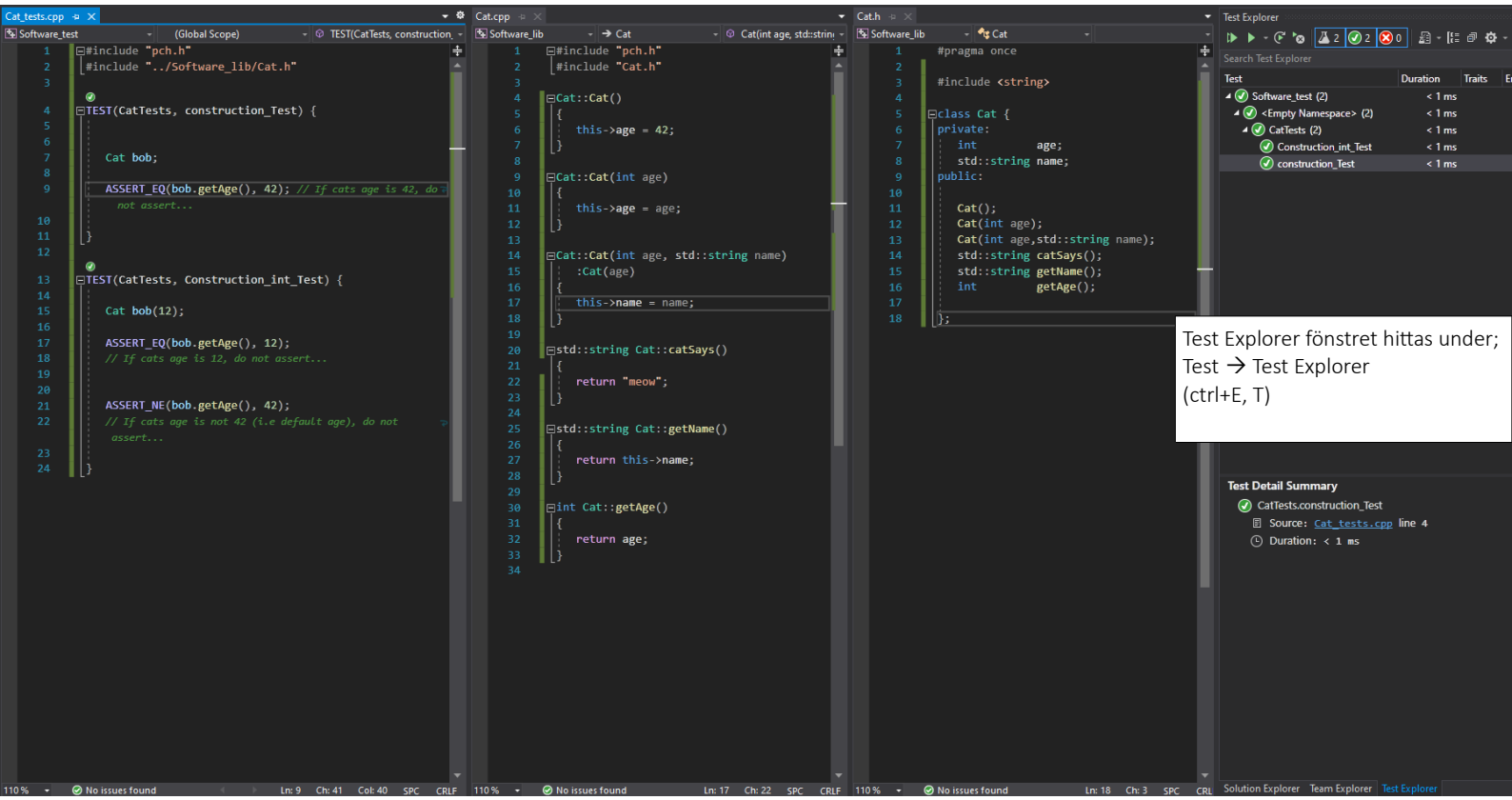
Tester kan definieras i egna **.cpp-filer** filer, alternativt kan testerna skrivas i test.cpp som skapas med Google Test projekt (Software_test)

Om nya filer skapas per TestSuit

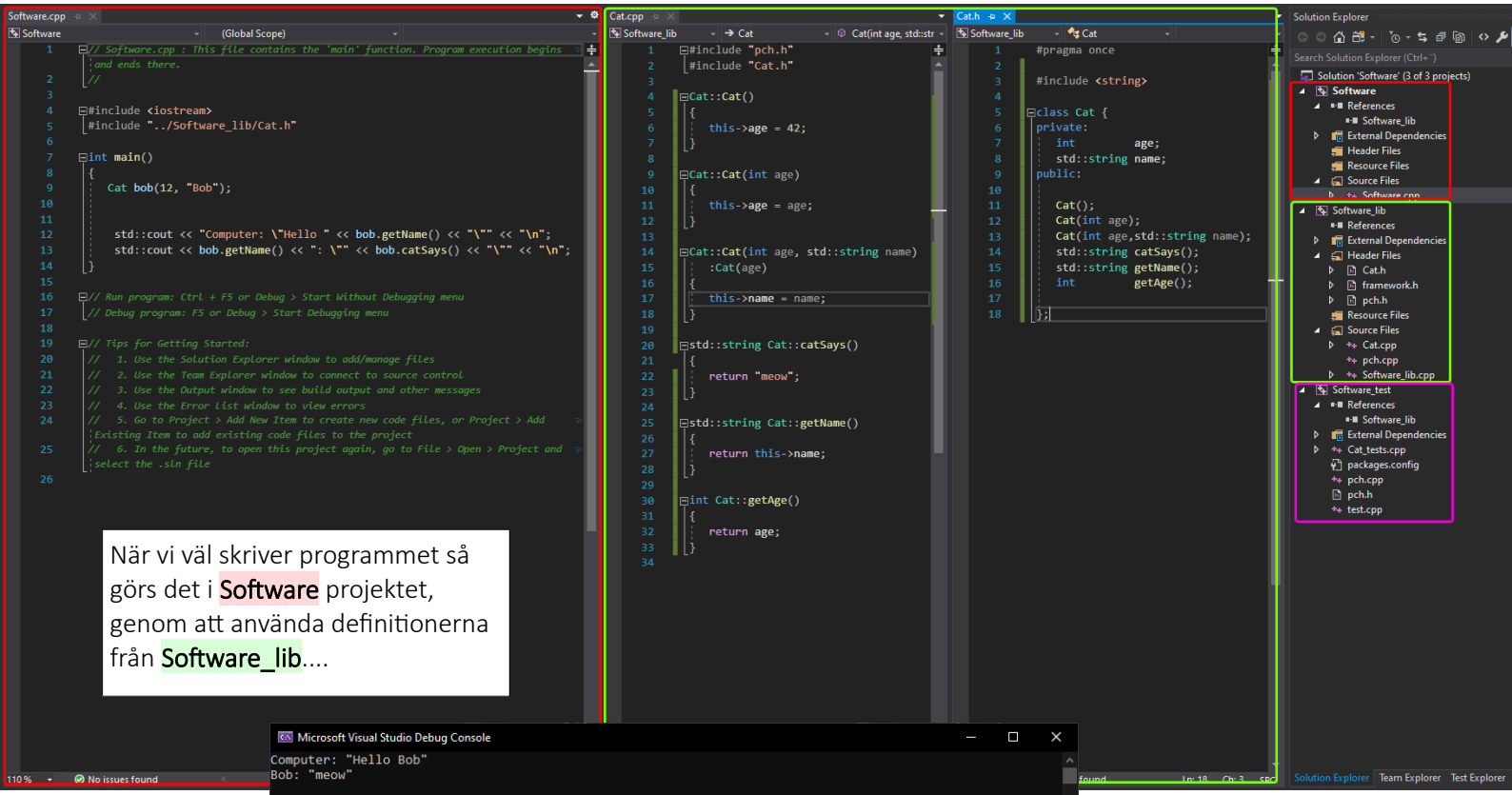
*Se till att **inkludera .h-filen** för klassen du vill testa...*

*Se till att **Inkludera** också den fördefinierade **pch.h-filen** som behövs för GoogleTest...*

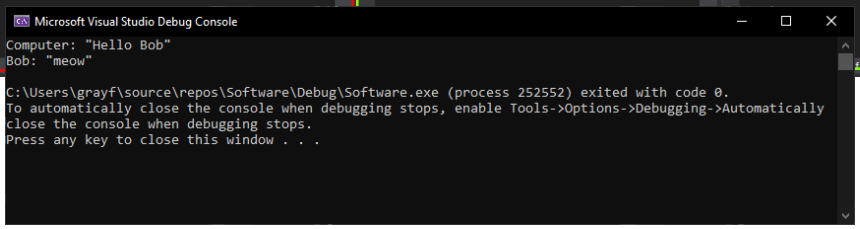
En klass med funktionalitet som vi vill testa...



Test Explorer fönstret hittas under;
Test → Test Explorer
(ctrl+E, T)



När vi väl skriver programmet så görs det i **Software** projektet, genom att använda definitionerna från **Software_lib**....



Några Olika Asserts och Exceptions som kan användas;

- Fatal assertion Typer
  - Boolean
    - ASSERT_TRUE(*condition*)
    - ASSERT_FALSE(*condition*)
  - Equal
    - ASSERT_EQ(*input, expected*)
    - ASSERT_NE(*input, expected*)
  - Equal for Strings
    - ASSERT_STREQ(*input, expected*)
    - ASSERT_STRNE(*input, expected*)
    - ASSERT_STRCASEEQ(*input, expected*)
    - ASSERT_STRCASENE(*input, expected*)
  - is Lower
    - ASSERT_LT(*input, expected*)
    - ASSERT_LE(*input, expected*)
  - is Greater
    - ASSERT_GT(*input, expected*)
    - ASSERT_GE(*input, expected*)
- Non-Fatal Assertion Typer
  - Boolean
    - EXPECT_TRUE(*condition*)
    - EXPECT_FALSE(*condition*)
  - Equal
    - EXPECT_EQ(*input, expected*)
    - EXPECT_NE(*input, expected*)
  - Equal for Strings
    - EXPECT_STREQ(*input, expected*)
    - EXPECT_STRNE(*input, expected*)
    - EXPECT_STRCASEEQ(*input, expected*)
    - EXPECT_STRCASENE(*input, expected*)
  - is Lower
    - EXPECT_LT(*input, expected*)
    - EXPECT_LE(*input, expected*)
  - is Greater
    - EXPECT_GT(*input, expected*)
    - EXPECT_GE(*input, expected*)

Asserts och Exceptions för att testa om Exceptions har kastats...;

- Fatal assertion Typer
  - ASSERT_THROW(*statement, exception*)
  - ASSERT_ANY_THROW(*statement*)
  - ASSERT_NO_THROW((*statement*)
- Non-Fatal Assertion Typer
  - EXPECT_THROW(*statement, exception*)
  - EXPECT_ANY_THROW(*statement*)
  - EXPECT_NO_THROW(*statement*)

Mer info om GoogleTest finns här....
https://github.com/google/googletest/blob/master/googletest/docs/primer.md