C++ Korea 4th Seminar

C++ 프로젝트 ~처음 만난 세계~

Visual C++에서 유닛 테스트 하기

Com2us

최흥배

1.유닛 테스트?

2.Visual C++에서 유닛 테스트 하기

3.사례 – 계산기, 서버 개발에서 유닛 테스트

4. 마무리

유닛 테스트 왜 해요?

"TDD는 죽었다" - Rails를 만든 DHH의 글

2014-04-25

- Test-first fundamentalism(테스트 우선 근본주의)는 마치 금욕을 하라고만 하는 성교육과 같다.
  - 자기 혐오에 빠져있는 사람을 위한 비현실적이고 실효성없는 도덕교육 같은 것이다.

그래서 우리는 어디로 가야 하나?

- 첫 단계로, 문제가 있다는 것은 인정하자.
  - 두번째 단계는, unit 부터 system 사이의 테스팅 스펙트럼의 균형을 잡는 것이다.
  - 현재의 광신적인 TDD 운동은 unit 테스트를 포커스하는 경향이 있다.

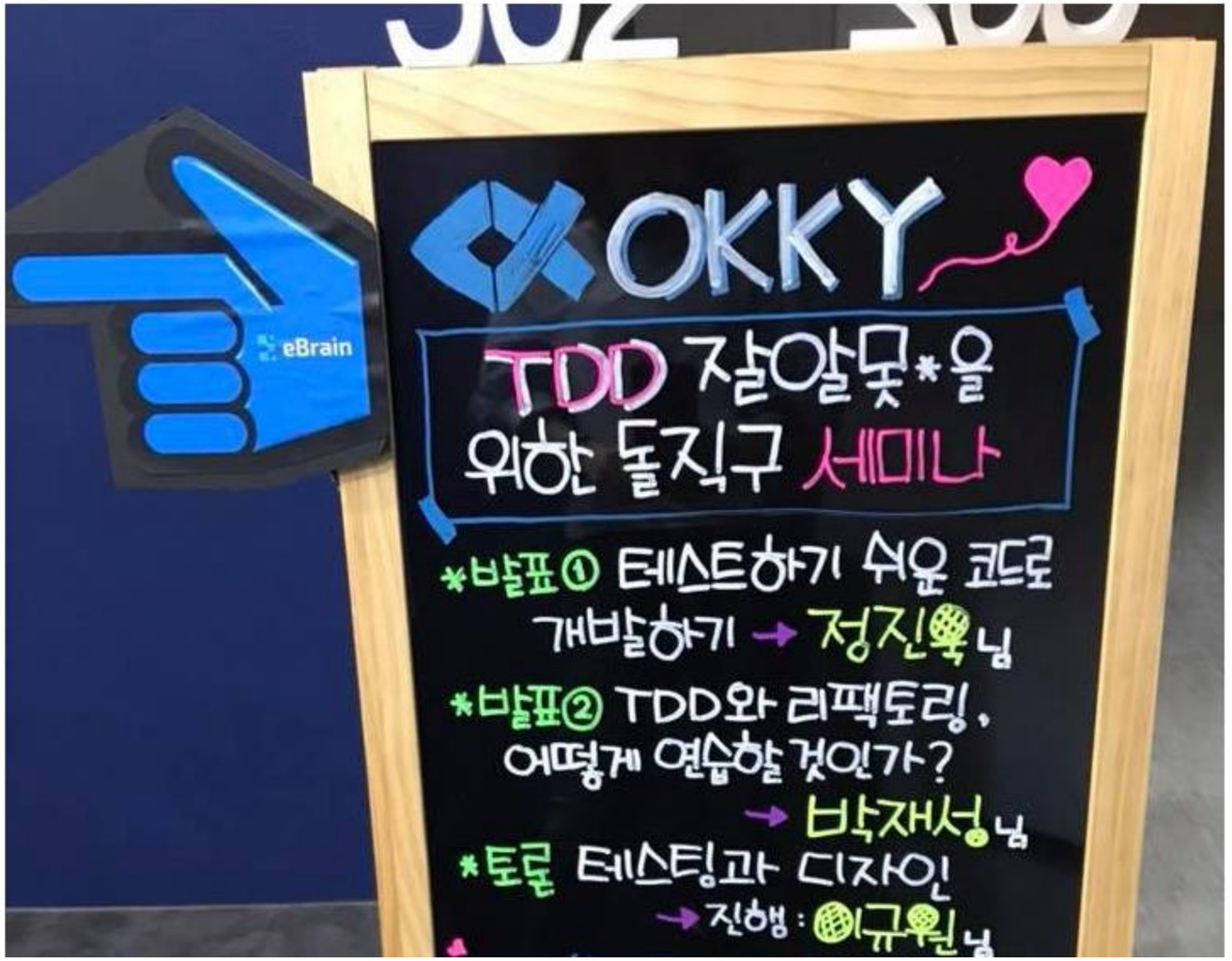
출처: https://sangwook.github.io/2014/04/25/tdd-is-dead-long-live-testing.html

TDD 잘알못을 위한 돌직구 세미나 참석 후기

세미나 · 2018.06.22 08:32

OKKY 최단시간 마감 세미나!
TDD 잘알못을 위한 돌직구 세미나에 다녀왔습니다.

- 세미나 링크



http://jojoldu.tistory.com/306

실내운전연습장

개발 잘 하기, 시간 절약....

개발 일정 계산 실수

프로그램 개발 시 생각 이상으로 디버깅에서 많은 시간 소요

경험이 부족하면 개발 일정을 계산 할 때 디버깅
시간을 계산하지 못해서 일정 계산에 실수 발생

그렇다고 디버깅 시간을 계산하려고 하니....

코드에 버그가 있을지? 없을지?

있다면 얼마나 있을지?

알기 힘들다....

코드에 버그가 없으면 디버깅 시간도 없어지겠죠...

그래서 유닛 테스트...

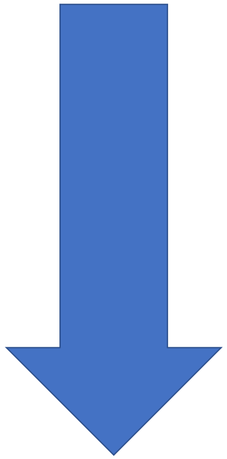
디버깅과 시간

수정 비용이 커진다

프로그래머 자신이 하는 테스트

QA에서 잡은 버그

유저가 보고한 버그



신념 만으로는...

버그는 만들지 않겠다 라는 신념 만으로는 잡을 수 없음

체계화된 시스템

기술이 필요

유닛 테스트

버그를 사전에 잡을 수 있는 방법 중의 하나

프로그래머 단계에서 꽤 많은 수의 버그를 잡을 수 있음

그러나....

유닛 테스트는 공짜가 아님
개발하기에도 바쁜데....

개발 코드에 수정이 발생하면 그 이상으로 유닛 테스트 코드에도 수정 발생
중복 코드는 악!!!

유닛 테스트는 해당 기능의 배움보다 경험이 꽤 중요

또, 유닛 테스트 만으로는 부족

유닛 테스트를 잘 했다고 해서 버그가 0은 결코 아니다.

유닛 테스트는 전체 보다는 부분 부분을 테스트 하는 것
부분 부분이 정확하다면 전체도 정확할 **확률이 높음**

AutoTest와 QA 테스트도 필요

성공 보다는 실패

보통 기능의 성공에 대한 부분은 유닛 테스트 없이 테스트가 가능.

그러나 **실패에 대한 테스트는 하기 어렵고, 잘 하지 않음**.
정말 중요 보통 시연, QA 테스트에서는 성공에 대한 것만 테스트
하지 실패 상황에 대한 테스트를 하기 어려움

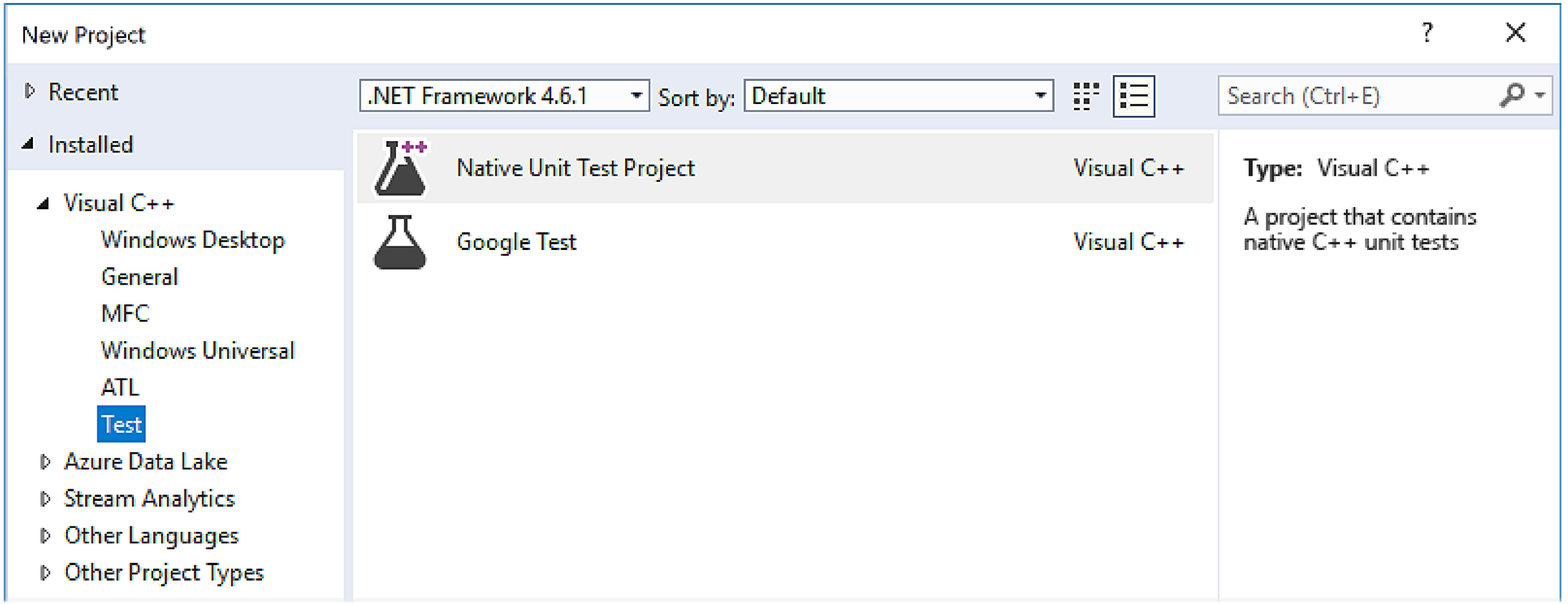
유닛 테스트는 실패에 대한 테스트를 다양하게 자주 할 수 있다.

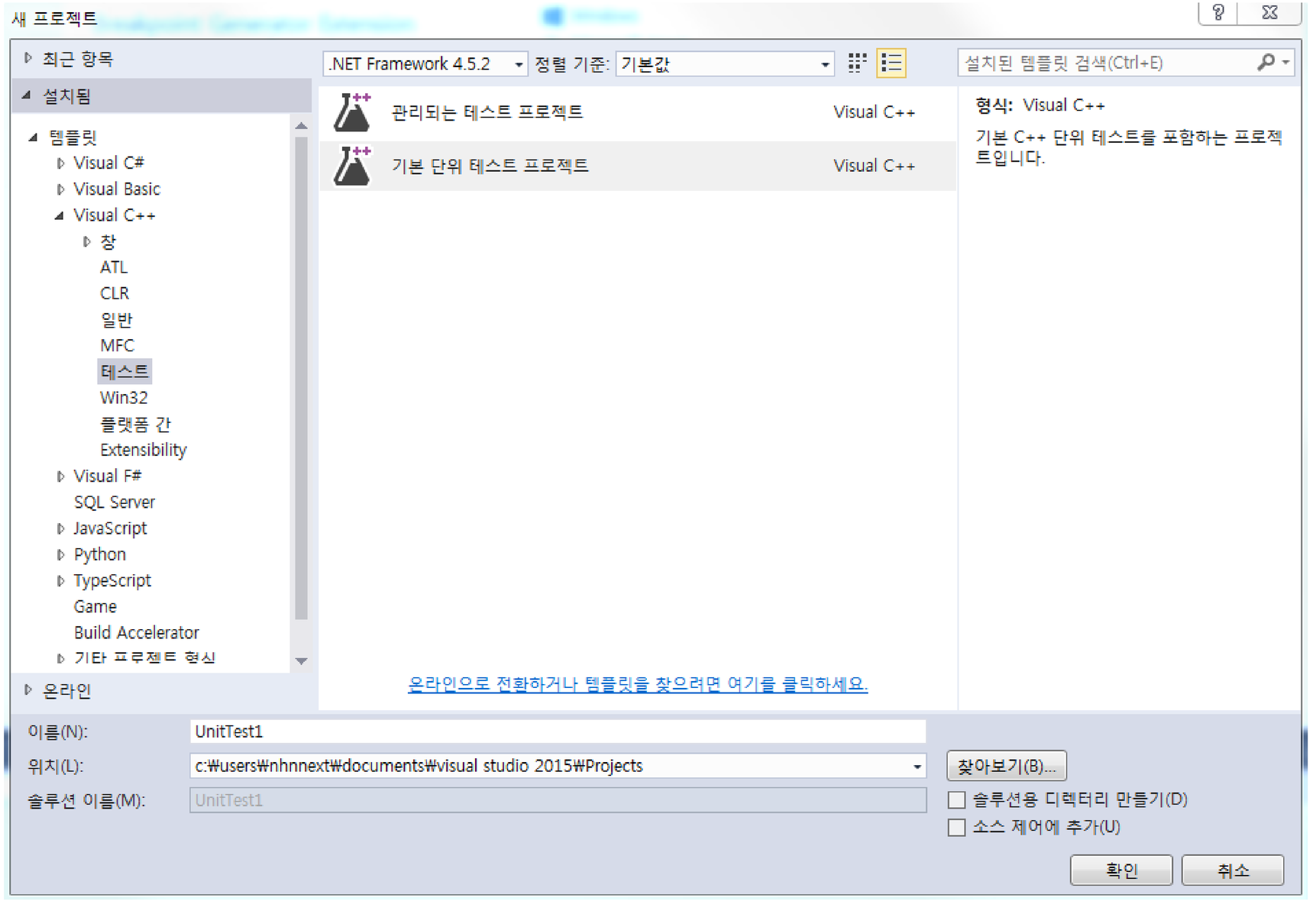
좋은 함수를 만들 수 있다

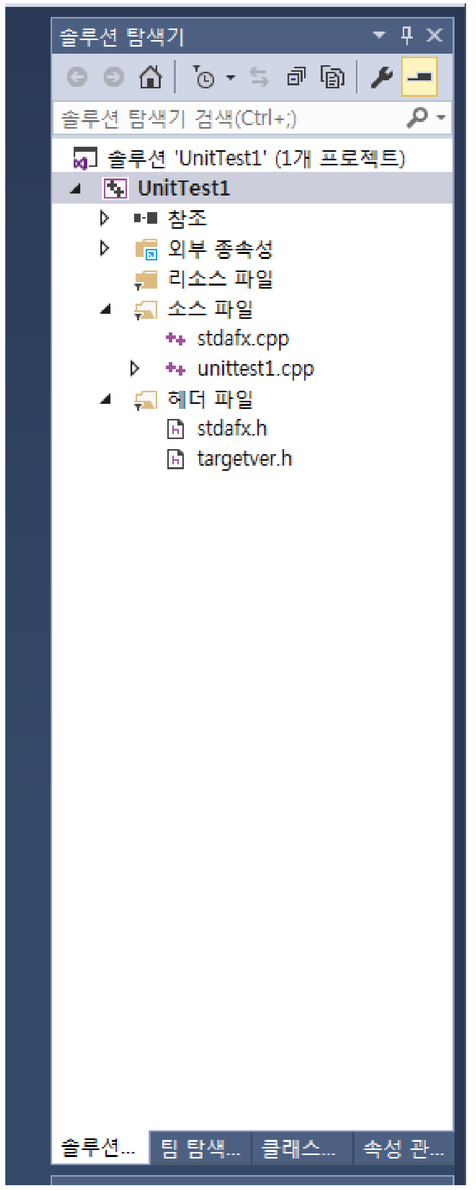
크기가 큰 함수, 복수 기능이 있는 함수는 유닛 테스트 어려움.

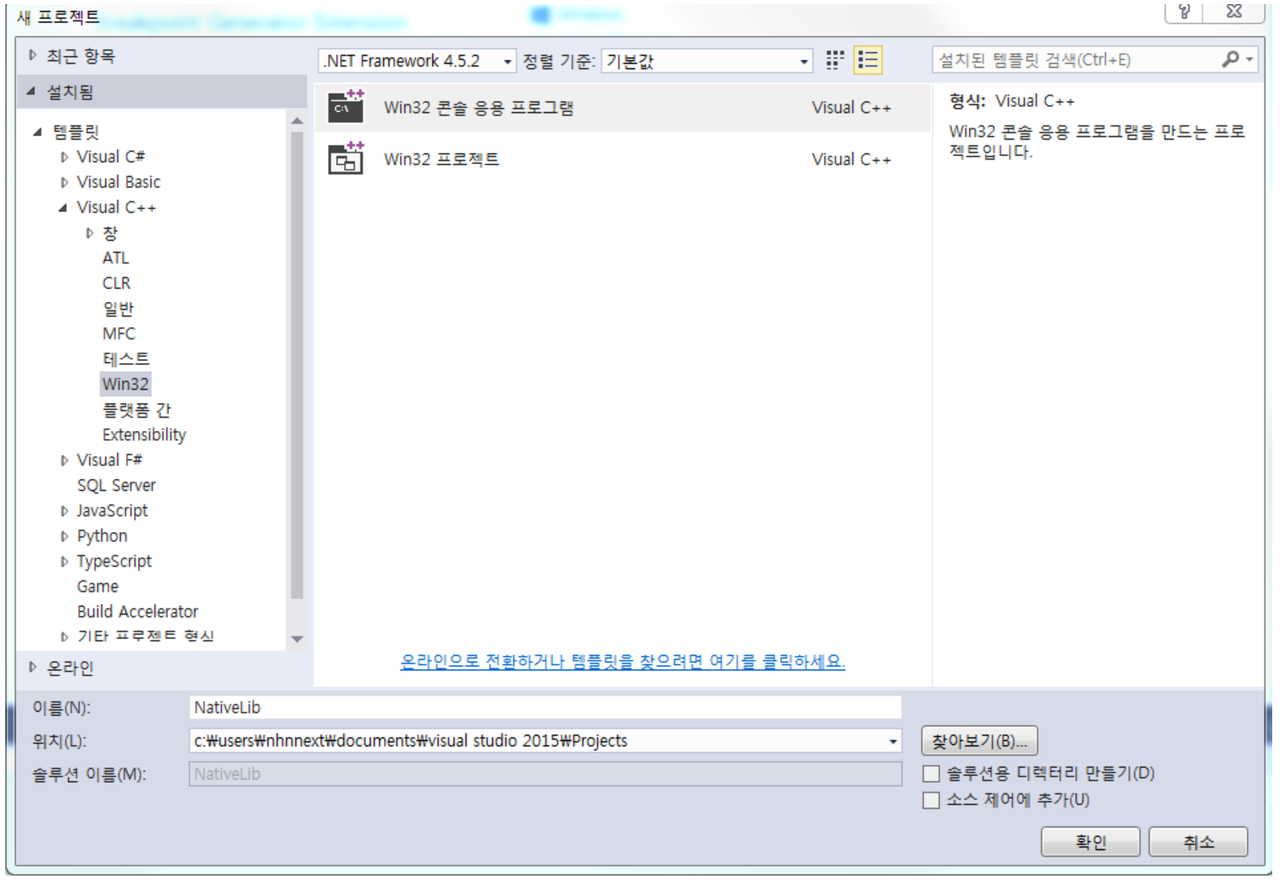
리팩토링 공부

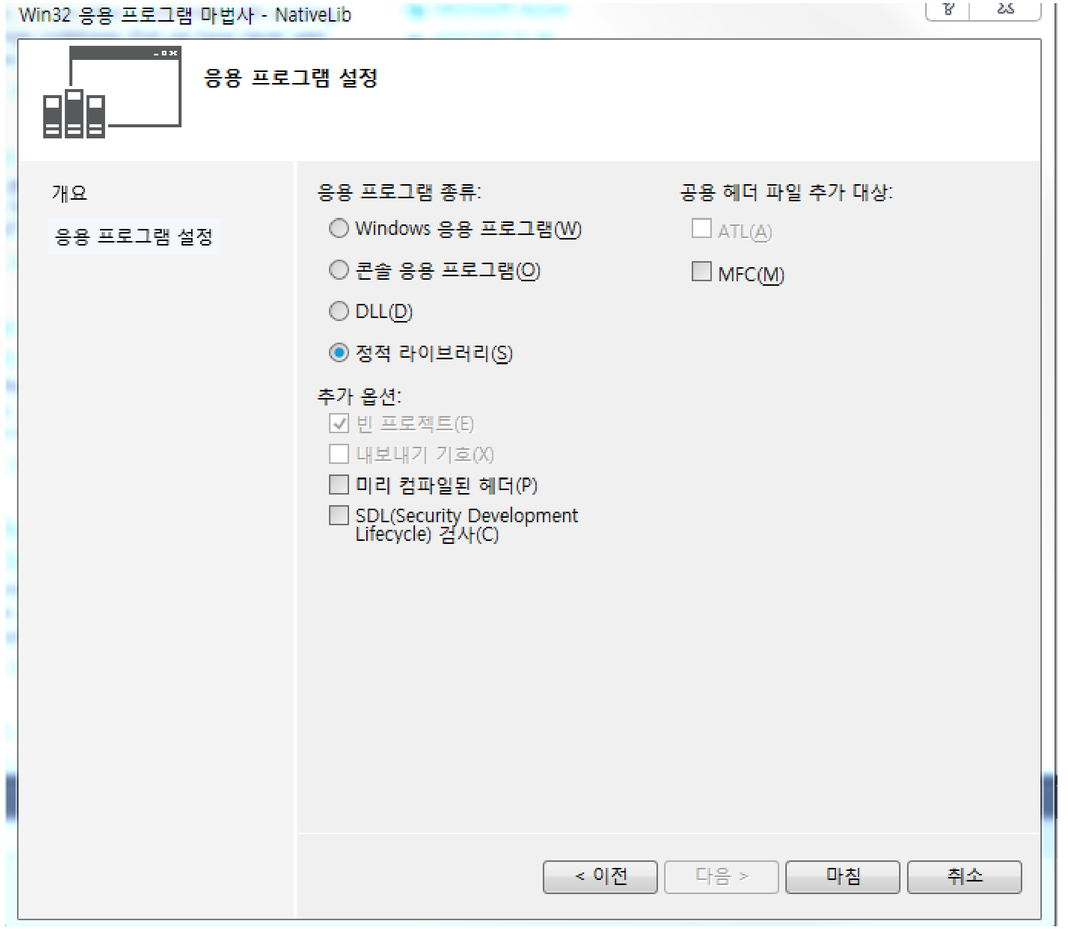
Visual Studio의 (순정)C++ 유닛 테스트

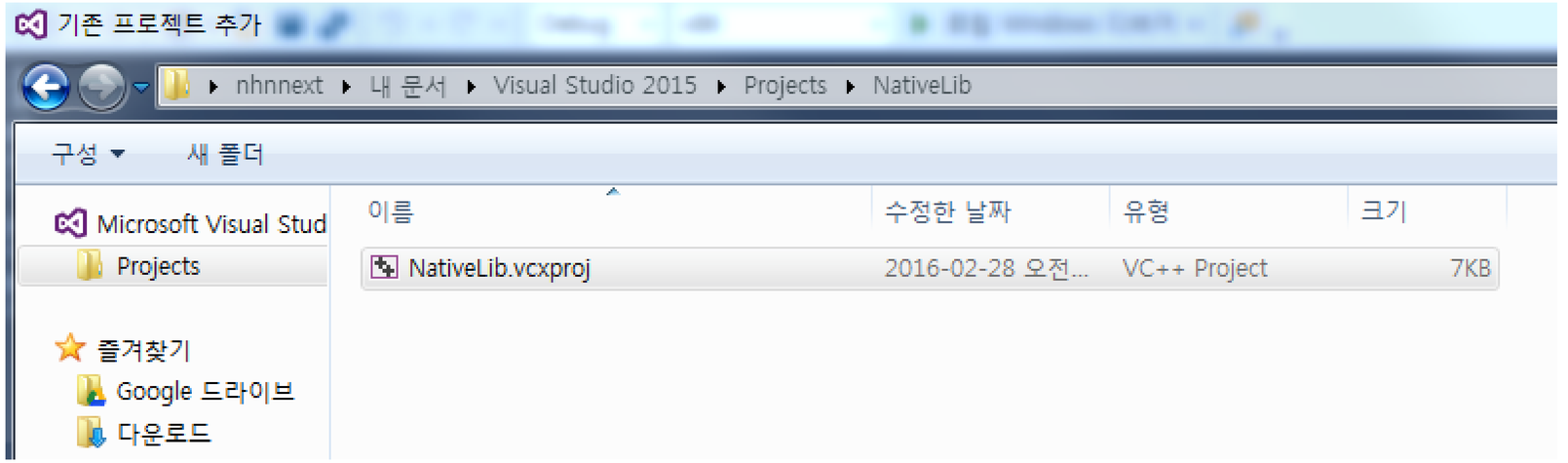

New Project
Recent
Installed
Visual C++
Windows Desktop
General
MFC
Windows Universal
ATL
Test
Azure Data Lake
Stream Analytics
Other Languages
Other Project Types
.NET Framework 4.6.1
Sort by:
Default
Search (Ctrl+E)
Native Unit Test Project
Visual C++
Google Test
Visual C++
Type: Visual C++
A project that contains native C++ unit tests

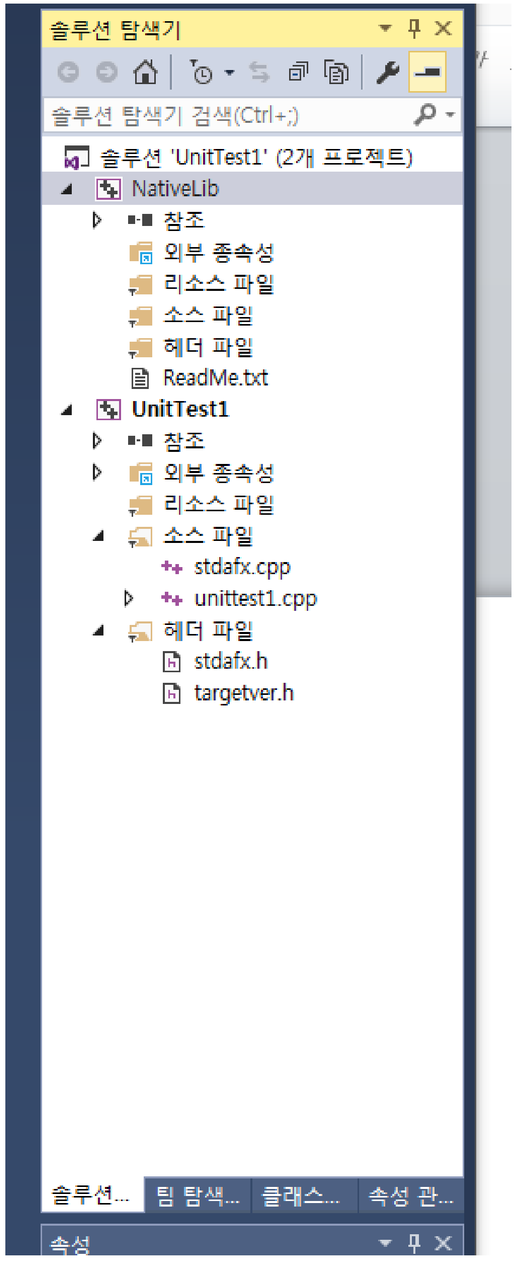

새 프로젝트
최근 항목
설치됨
템플릿
Visual C#
Visual Basic
Visual C++
창
ATL
CLR
일반
MFC
테스트
Win32
플랫폼 간
Extensibility
Visual F#
SQL Server
JavaScript
Python
TypeScript
Game
Build Accelerator
온라인
.NET Framework 4.5.2
정렬 기준:
기본값
설치된 템플릿 검색(Ctrl+E)
관리되는 테스트 프로젝트
Visual C++
기본 단위 테스트 프로젝트
Visual C++
형식: Visual C++
기본 C++ 단위 테스트를 포함하는 프로젝트입니다.
온라인으로 전환하거나 템플릿을 찾으려면 여기를 클릭하세요.
이름(N):
UnitTest1
위치(L):
c:₩users₩nhnnext₩documents₩visual studio 2015₩Projects
찾아보기(B)...
솔루션 이름(M):
UnitTest1
솔루션용 디렉터리 만들기(D)
소스 제어에 추가(U)
확인
취소

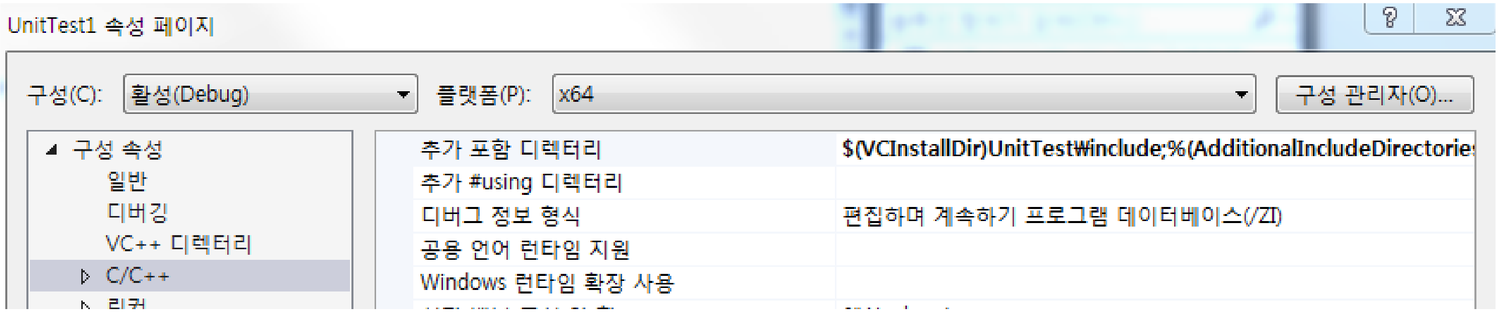

솔루션 탐색기
솔루션 탐색기 검색(Ctrl+;)
솔루션 'UnitTest1' (1개 프로젝트)
UnitTest1
참조
외부 종속성
리소스 파일
소스 파일
stdafx.cpp
unittest1.cpp
헤더 파일
stdafx.h
targetver.h
솔루션...
팀 탐색...
클래스...
속성 관...

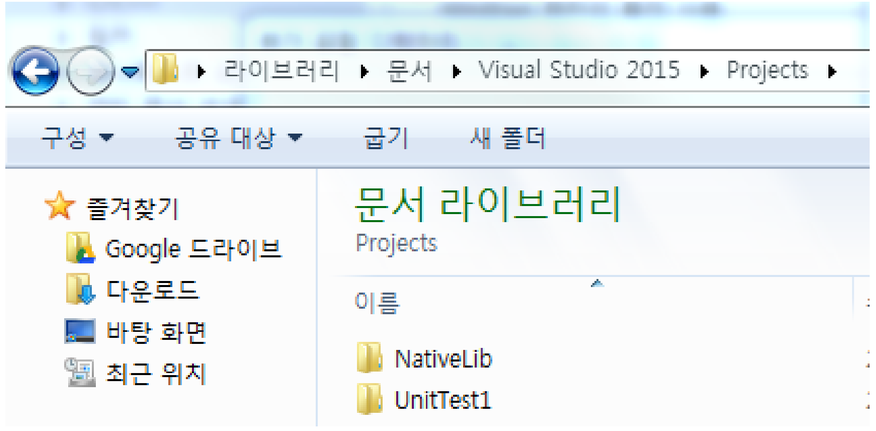

새 프로젝트
최근 항목
설치됨
템플릿
Visual C#
Visual Basic
Visual C++
창
ATL
CLR
일반
MFC
테스트
Win32
플랫폼 간
Extensibility
Visual F#
SQL Server
JavaScript
Python
TypeScript
Game
Build Accelerator
온라인
.NET Framework 4.5.2
정렬 기준: 기본값
설치된 템플릿 검색(Ctrl+E)
Win32 콘솔 응용 프로그램 Visual C++
Win32 프로젝트 Visual C++
형식: Visual C++
Win32 콘솔 응용 프로그램을 만드는 프로젝트입니다.
온라인으로 전환하거나 템플릿을 찾으려면 여기를 클릭하세요.
이름(N): NativeLib
위치(L): c:₩users₩nhnnext₩documents₩visual studio 2015₩Projects
찾아보기(B)...
솔루션 이름(M): NativeLib
솔루션용 디렉터리 만들기(D)
소스 제어에 추가(U)
확인
취소

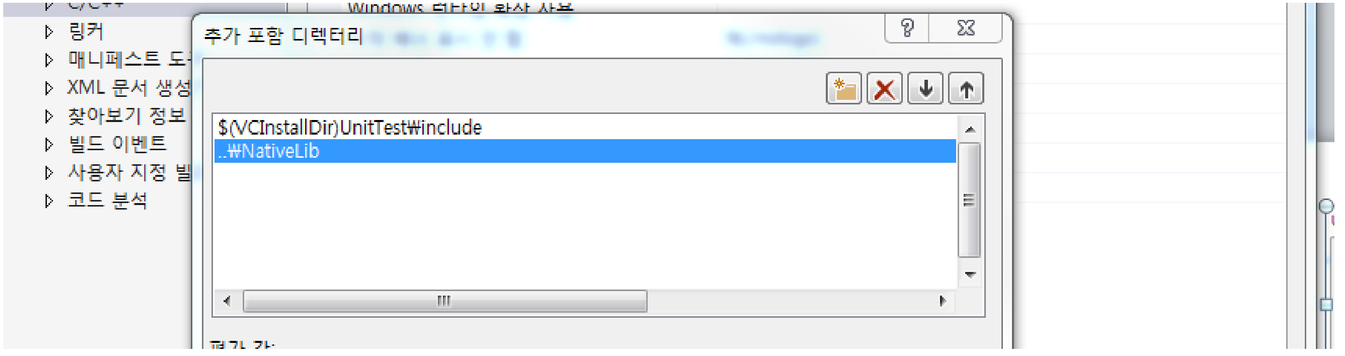

Win32 응용 프로그램 마법사 - NativeLib
응용 프로그램 설정
개요
응용 프로그램 설정
응용 프로그램 종류:
Windows 응용 프로그램(W)
콘솔 응용 프로그램(O)
DLL(D)
정적 라이브러리(S)
추가 옵션:
빈 프로젝트(E)
내보내기 기호(X)
미리 컴파일된 헤더(P)
SDL(Security Development Lifecycle) 검사(C)
공용 헤더 파일 추가 대상:
ATL(A)
MFC(M)
< 이전
다음 >
마침
취소

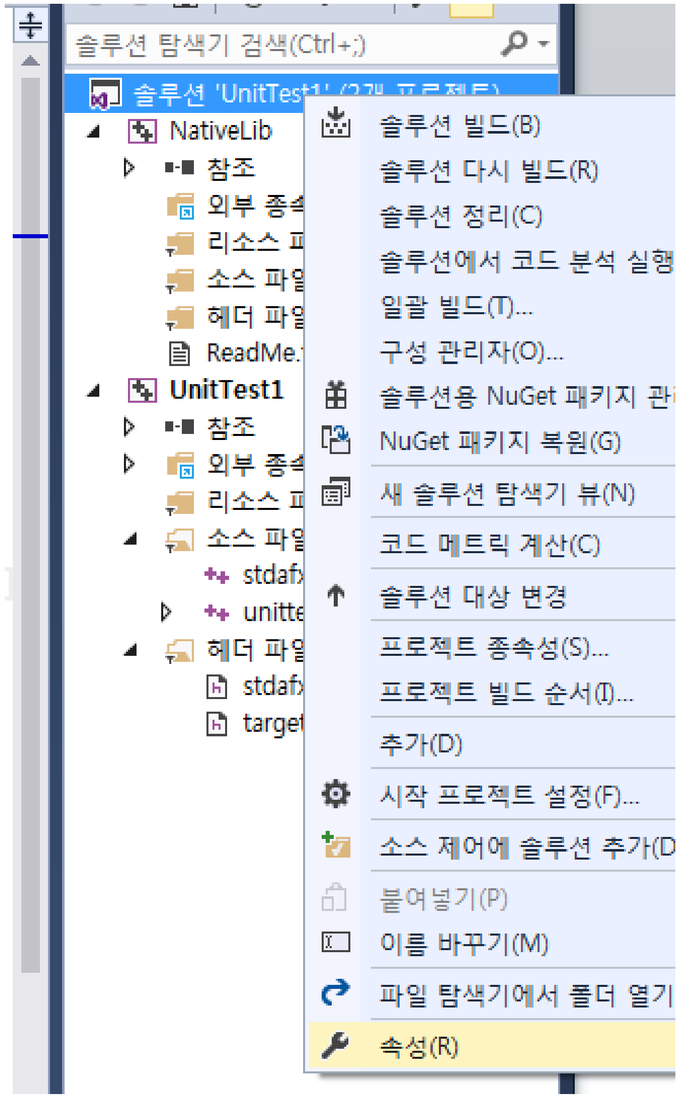

기존 프로젝트 추가
nhnnext ▸ 내 문서 ▸ Visual Studio 2015 ▸ Projects ▸ NativeLib
구성
새 폴더
Microsoft Visual Stud
Projects
즐겨찾기
Google 드라이브
다운로드
이름
수정한 날짜
유형
크기
NativeLib.vcxproj
2016-02-28 오전...
VC++ Project
7KB

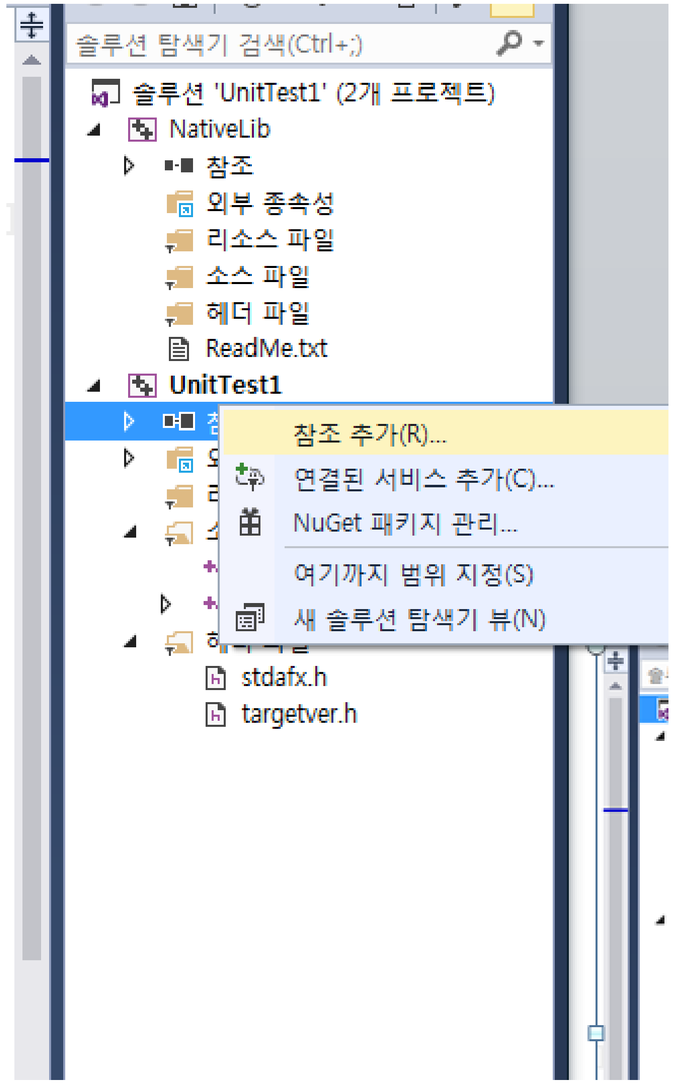

솔루션 탐색기
솔루션 탐색기 검색(Ctrl+;)
솔루션 'UnitTest1' (2개 프로젝트)
NativeLib
참조
외부 종속성
리소스 파일
소스 파일
헤더 파일
ReadMe.txt
UnitTest1
참조
외부 종속성
리소스 파일
소스 파일
stdafx.cpp
unittest1.cpp
헤더 파일
stdafx.h
targetver.h
솔루션...
팀 탐색...
클래스...
속성 관...
속성


UnitTest1 속성 페이지
구성(C): 활성(Debug)
플랫폼(P): x64
구성 관리자(O)...
구성 속성
일반
디버깅
VC++ 디렉터리
C/C++
추가 포함 디렉터리
$(VCInstallDir)UnitTest₩include;%(AdditionalIncludeDirectorie
추가 #using 디렉터리
디버그 정보 형식
편집하며 계속하기 프로그램 데이터베이스(/ZI)
공용 언어 런타임 지원
Windows 런타임 확장 사용

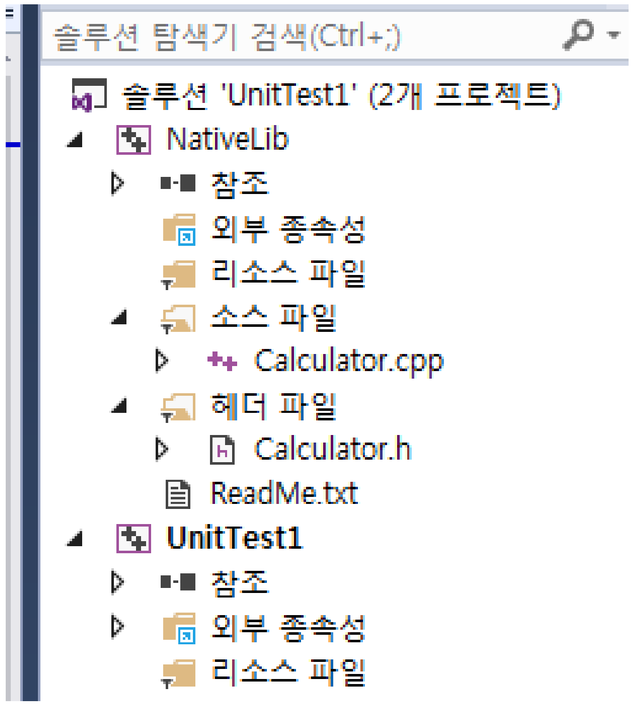

라이브러리 ▸ 문서 ▸ Visual Studio 2015 ▸ Projects ▸
구성
공유 대상
굽기
새 폴더
즐겨찾기
Google 드라이브
다운로드
바탕 화면
최근 위치
문서 라이브러리
Projects
이름
NativeLib
UnitTest1


링커
매니페스트 도
XML 문서 생성
찾아보기 정보
빌드 이벤트
사용자 지정 빌
코드 분석
추가 포함 디렉터리
$(VCInstallDir)UnitTest₩include
..₩NativeLib


솔루션 탐색기 검색(Ctrl+;)
NativeLib
참조
외부 종속
리소스 파
소스 파일
헤더 파일
ReadMe.
UnitTest1
참조
외부 종속
리소스 파
소스 파일
stdafx
unitte
헤더 파일
stdafx
target
솔루션 빌드(B)
솔루션 다시 빌드(R)
솔루션 정리(C)
솔루션에서 코드 분석 실행
일괄 빌드(T)...
구성 관리자(O)...
솔루션용 NuGet 패키지 관
NuGet 패키지 복원(G)
새 솔루션 탐색기 뷰(N)
코드 메트릭 계산(C)
솔루션 대상 변경
프로젝트 종속성(S)...
프로젝트 빌드 순서(I)...
추가(D)
시작 프로젝트 설정(F)...
소스 제어에 솔루션 추가(D
붙여넣기(P)
이름 바꾸기(M)
파일 탐색기에서 폴더 열기
속성(R)


솔루션 탐색기 검색(Ctrl+;)
솔루션 'UnitTest1' (2개 프로젝트)
NativeLib
참조
외부 종속성
리소스 파일
소스 파일
헤더 파일
ReadMe.txt
UnitTest1
참조 추가(R)...
연결된 서비스 추가(C)...
NuGet 패키지 관리...
여기까지 범위 지정(S)
새 솔루션 탐색기 뷰(N)
stdafx.h
targetver.h


참조 추가
프로젝트
솔루션
공유 프로젝트
프로젝트 검색(Ctrl
이름
경로
NativeLib
C:₩Users₩nhnnext₩Doc
이름:
NativeLib



Calculator.cpp | Calculator.h | unittest1.cpp

NativeLib

```
1   #pragma once
2
3
4   class Calculator
5   {
6   public:
7       Calculator() {}
8       ~Calculator() {}
9
10      int Add(int a, int b);
11  };
```

Calculator.cpp | Calculator.h | unittest1.cpp

NativeLib

```
1   #include "Calculator.h"
2
3
4
5   int Calculator::Add(int a, int b)
6   {
7       return a + b;
8   }
```

Calculator.cpp | Calculator.h | unittest1.cpp

UnitTest1 | UnitTest1

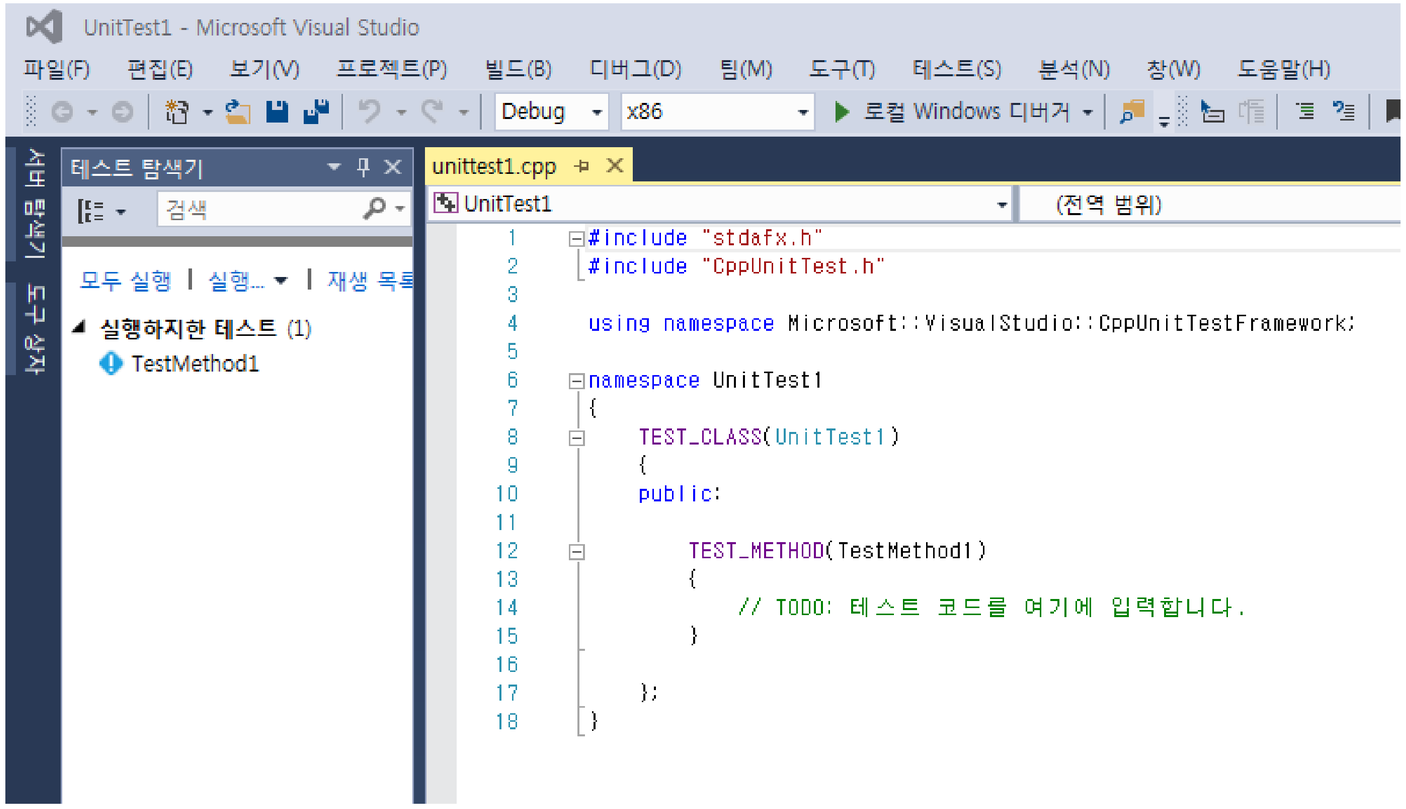
```
1  #include "stdafx.h"
2  #include "CppUnitTest.h"
3
4  #include "..₩NativeLib₩Calculator.h"
5
6
7  using namespace Microsoft::VisualStudio::CppUnitTestFramework;
8
9  namespace UnitTest1
10 {
11     TEST_CLASS(UnitTest1)
12     {
13     public:
14
15         TEST_METHOD(TestMethod1)
16         {
17             Calculator cal;
18             Assert::AreEqual(3, cal.Add(1, 2));
19         }
20
21     };
22 }
```

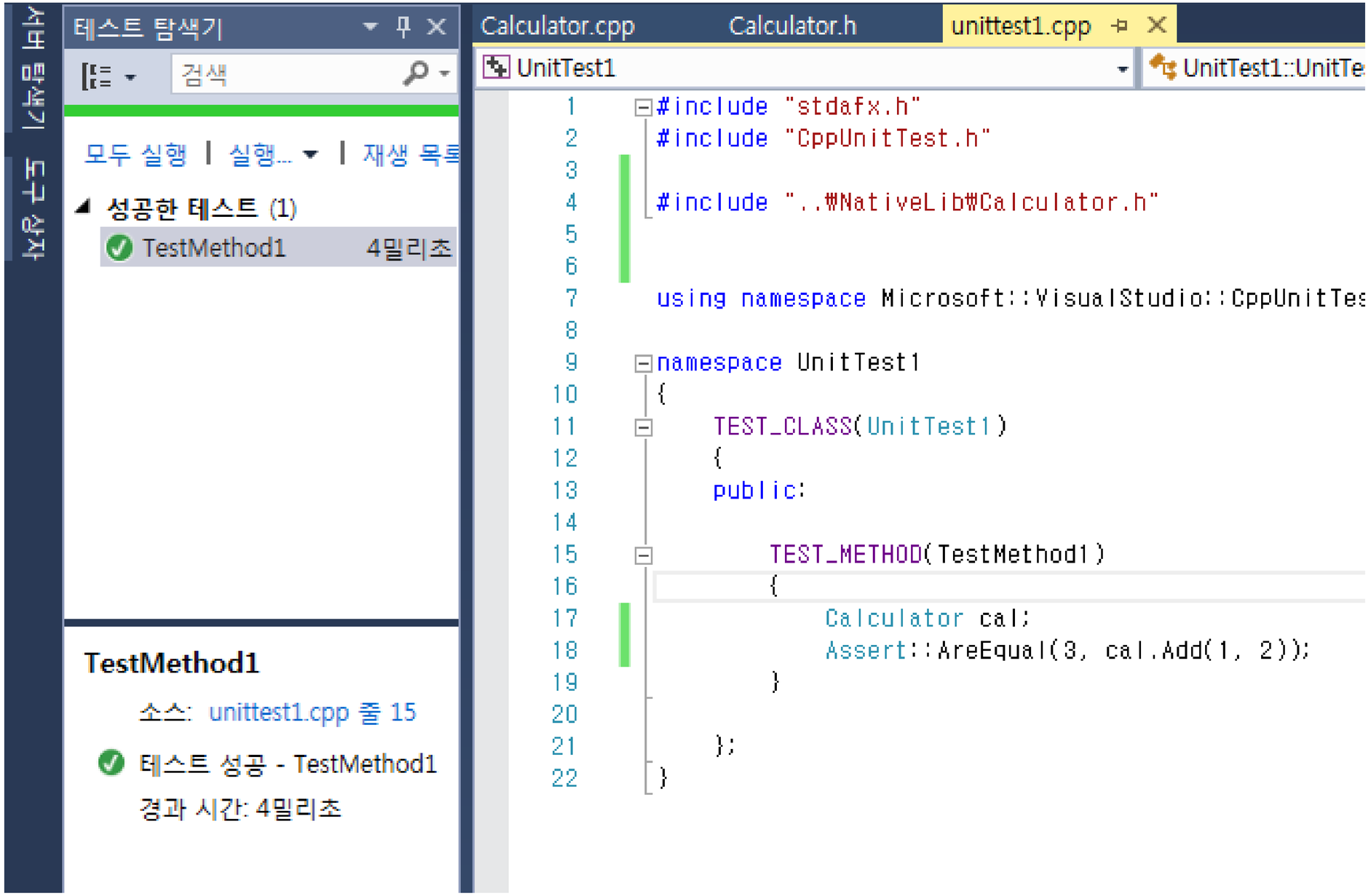
솔루션 탐색기

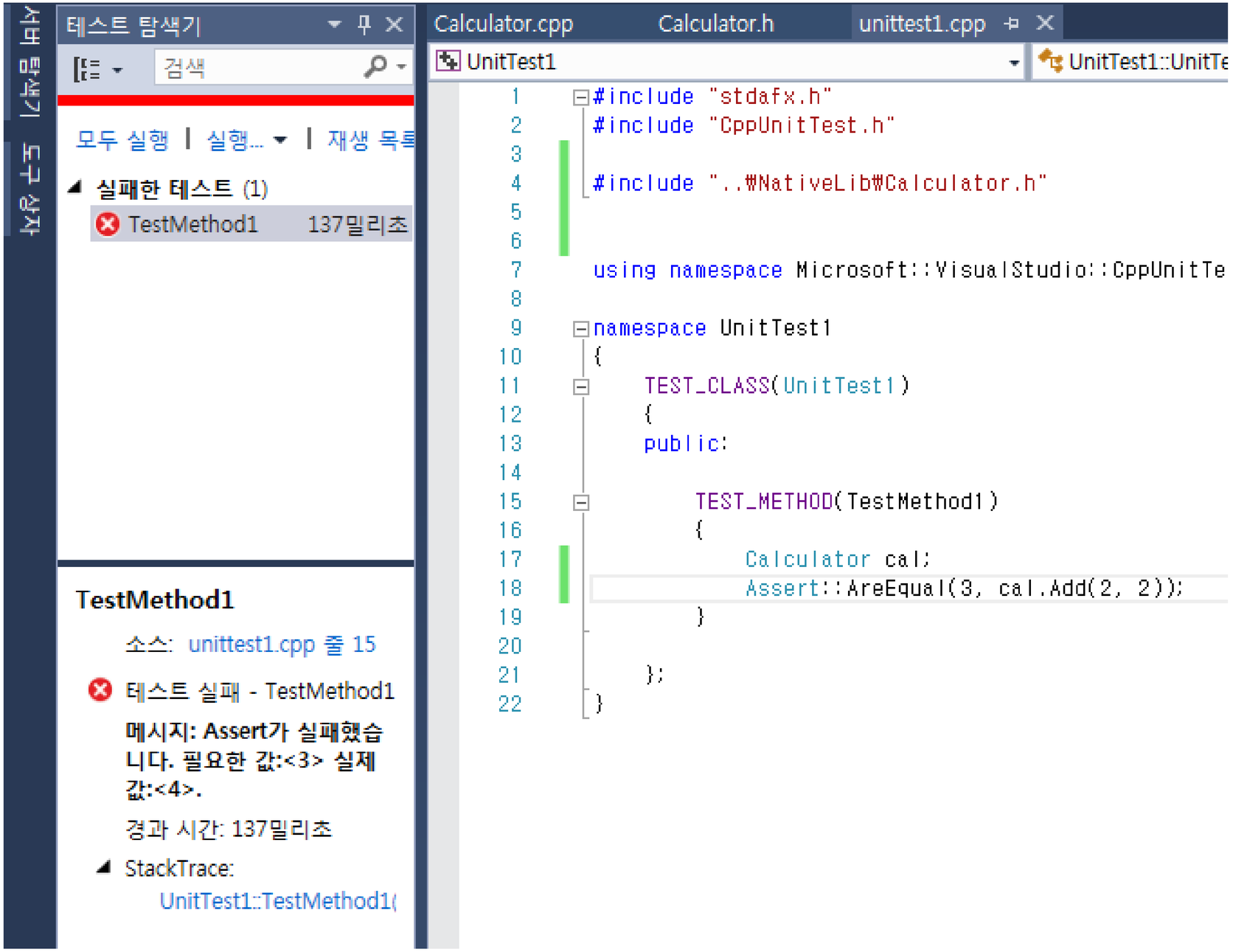
솔루션 탐색기 검색(Ctrl+;)

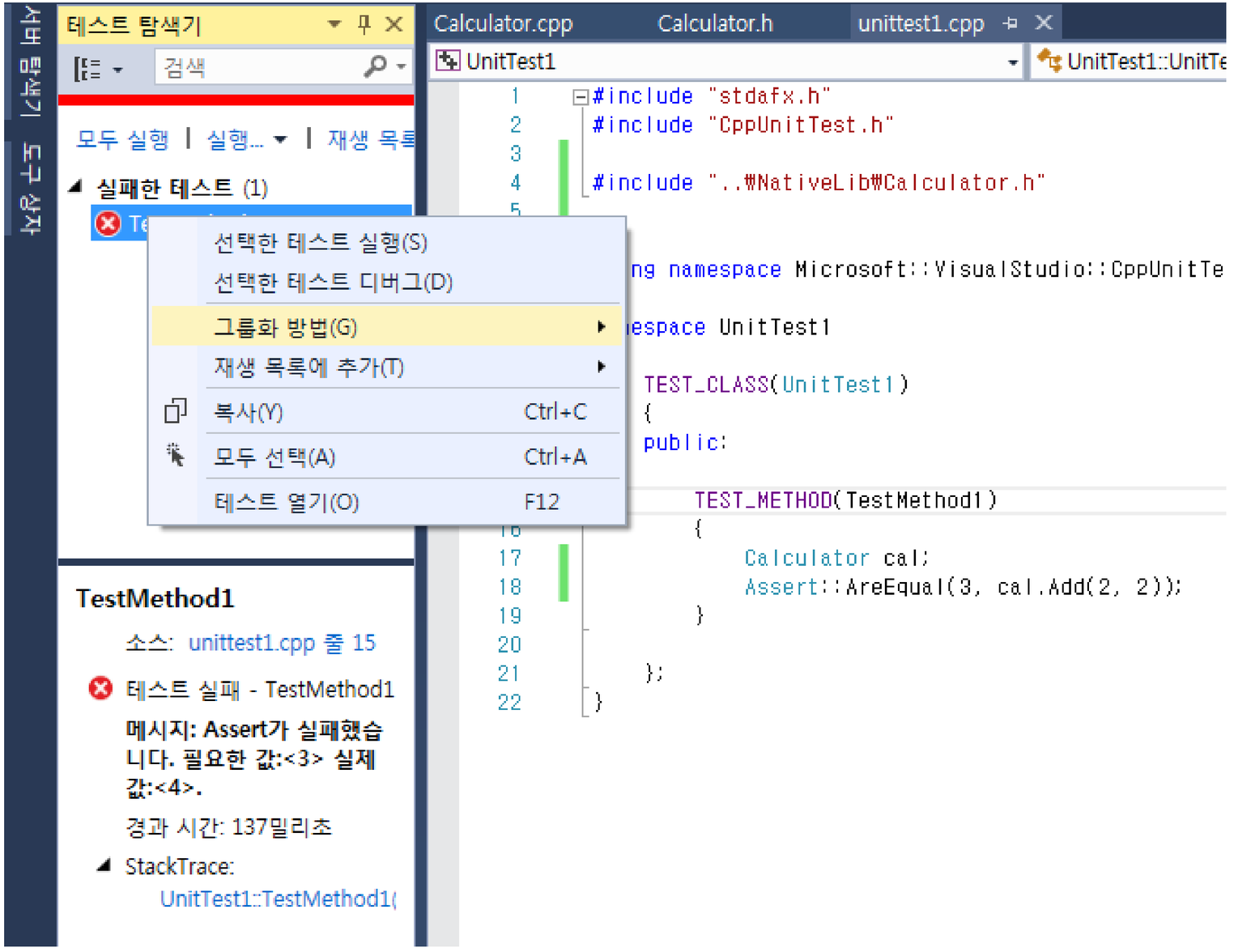
- 솔루션 'UnitTest1' (2개 프로젝트)
  - NativeLib
    - 참조
    - 외부 종속성
    - 리소스 파일
    - 소스 파일
      - Calculator.cpp
    - 헤더 파일
      - Calculator.h
    - ReadMe.txt
  - **UnitTest1**
    - 참조
    - 외부 종속성
    - 리소스 파일
    - 소스 파일
      - stdafx.cpp
      - unittest1.cpp
    - 헤더 파일
      - stdafx.h
      - targetver.h


D)
팀(M)
도구(T)
테스트(S)
분석(N)
창(W)
도움말(H)
실행(R)
디버그(D)
재생 목록(P)
테스트 설정(S)
창(W)
테스트 탐색기(T)

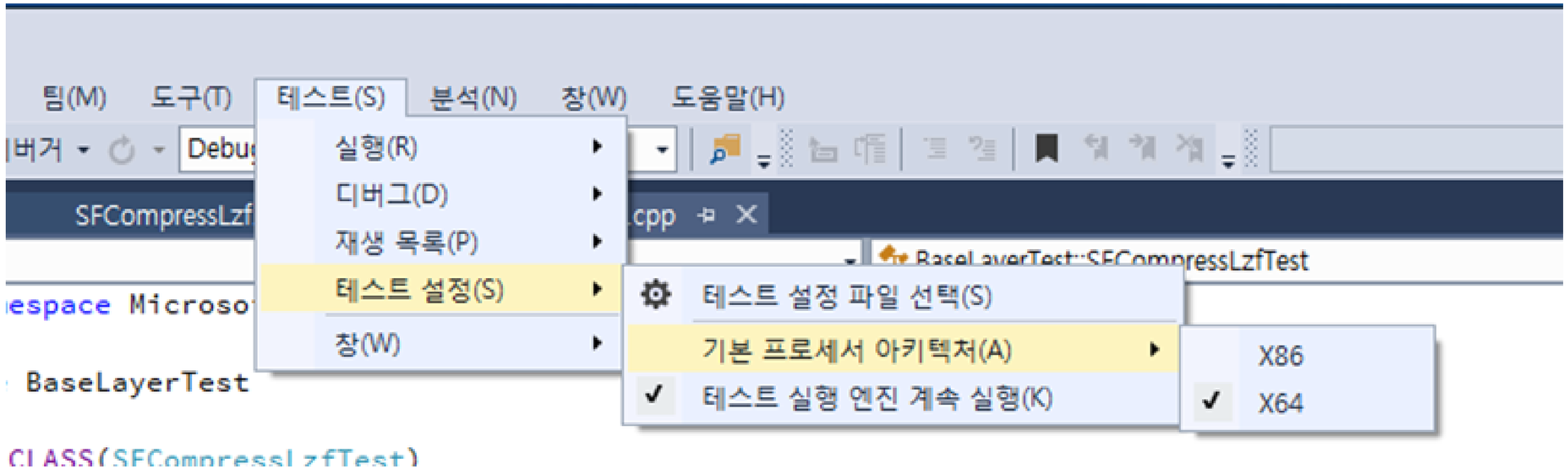

UnitTest1 - Microsoft Visual Studio
파일(F)
편집(E)
보기(V)
프로젝트(P)
빌드(B)
디버그(D)
팀(M)
도구(T)
테스트(S)
분석(N)
창(W)
도움말(H)
Debug
x86
로컬 Windows 디버거
테스트 탐색기
검색
모두 실행
실행...
재생 목록
실행하지한 테스트 (1)
TestMethod1
서버 탐색기
도구 상자
unittest1.cpp
UnitTest1
(전역 범위)
#include "stdafx.h"
#include "CppUnitTest.h"
using namespace Microsoft::VisualStudio::CppUnitTestFramework;
namespace UnitTest1
{
TEST_CLASS(UnitTest1)
{
public:
TEST_METHOD(TestMethod1)
{
// TODO: 테스트 코드를 여기에 입력합니다.
}
};
}


서버 탐색기
도구 상자
테스트 탐색기
검색
모두 실행 | 실행... | 재생 목록
성공한 테스트 (1)
TestMethod1 4밀리초
TestMethod1
소스: unittest1.cpp 줄 15
테스트 성공 - TestMethod1
경과 시간: 4밀리초
Calculator.cpp
Calculator.h
unittest1.cpp
UnitTest1
UnitTest1::UnitTe
#include "stdafx.h"
#include "CppUnitTest.h"
#include "..₩NativeLib₩Calculator.h"
using namespace Microsoft::VisualStudio::CppUnitTes
namespace UnitTest1
{
TEST_CLASS(UnitTest1)
{
public:
TEST_METHOD(TestMethod1)
{
Calculator cal;
Assert::AreEqual(3, cal.Add(1, 2));
}
};
}


서버 탐색기
도구 상자
테스트 탐색기
검색
모두 실행 | 실행... | 재생 목록
실패한 테스트 (1)
TestMethod1 137밀리초
TestMethod1
소스: unittest1.cpp 줄 15
테스트 실패 - TestMethod1
메시지: Assert가 실패했습니다. 필요한 값:<3> 실제 값:<4>.
경과 시간: 137밀리초
StackTrace:
UnitTest1::TestMethod1(
Calculator.cpp
Calculator.h
unittest1.cpp
UnitTest1
UnitTest1::UnitTe
#include "stdafx.h"
#include "CppUnitTest.h"
#include "..₩NativeLib₩Calculator.h"
using namespace Microsoft::VisualStudio::CppUnitTe
namespace UnitTest1
{
TEST_CLASS(UnitTest1)
{
public:
TEST_METHOD(TestMethod1)
{
Calculator cal;
Assert::AreEqual(3, cal.Add(2, 2));
}
};
}


테스트 탐색기
검색
모두 실행 | 실행... | 재생 목록
실패한 테스트 (1)
선택한 테스트 실행(S)
선택한 테스트 디버그(D)
그룹화 방법(G)
재생 목록에 추가(T)
복사(Y) Ctrl+C
모두 선택(A) Ctrl+A
테스트 열기(O) F12
TestMethod1
소스: unittest1.cpp 줄 15
테스트 실패 - TestMethod1
메시지: Assert가 실패했습니다. 필요한 값:<3> 실제 값:<4>.
경과 시간: 137밀리초
StackTrace:
UnitTest1::TestMethod1(
Calculator.cpp
Calculator.h
unittest1.cpp
UnitTest1
UnitTest1::UnitTe
#include "stdafx.h"
#include "CppUnitTest.h"
#include "..₩NativeLib₩Calculator.h"
ng namespace Microsoft::VisualStudio::CppUnitTe
espace UnitTest1
TEST_CLASS(UnitTest1)
{
public:
TEST_METHOD(TestMethod1)
{
Calculator cal;
Assert::AreEqual(3, cal.Add(2, 2));
}
};
}

64비트 테스트 하기


팀(M)
도구(T)
테스트(S)
분석(N)
창(W)
도움말(H)
실행(R)
디버그(D)
재생 목록(P)
테스트 설정(S)
창(W)
테스트 설정 파일 선택(S)
기본 프로세서 아키텍처(A)
테스트 실행 엔진 계속 실행(K)
X86
X64
BaseLayerTest

VC++ 유닛 테스트 기능 소개

http://codezine.jp/article/detail/6464
내용을 번역, 정리

TEST_CLASS(class_name){...};
TEST_METHOD를 모은 것이 TEST_CLASS이다.
테스트 대상 별로 TEST_CLASS를 정의하는 것이 기본이지만, 상황 혹은 기호시(예를 들면 정상 계열과 이상 계열 등)세분화해도 상관 없다. 그 때는 단일 소스에 복수의 TEST_CLASS를 적어 나가도 좋고, 추가/새로운 항목/C++ 유닛 테스트 클래스로 또 다른 소스에 나눠도 좋음.

TEST_METHOD(method_name){...}
테스트 각 항목에 해당한다. 테스트 대상을 동작시키고 그 결과를 검증한다.

TEST_MODULE_INITIALIZE(method_name){...}
TEST_MODULE_CLEANUP(method_name){...}
모든 테스트의 실행 직전/직후에 각각 한번만 호출한다. 이름 그대로 테스트 모듈의 전 준비/후 처리를 위한 것. 환경 변수/글로벌 변수를 설정하거나 DLL을 LoadLibarary/FreeLibrary 하거나 COM을 초기화/해방하는 등.

http://codezine.jp/article/detail/6464

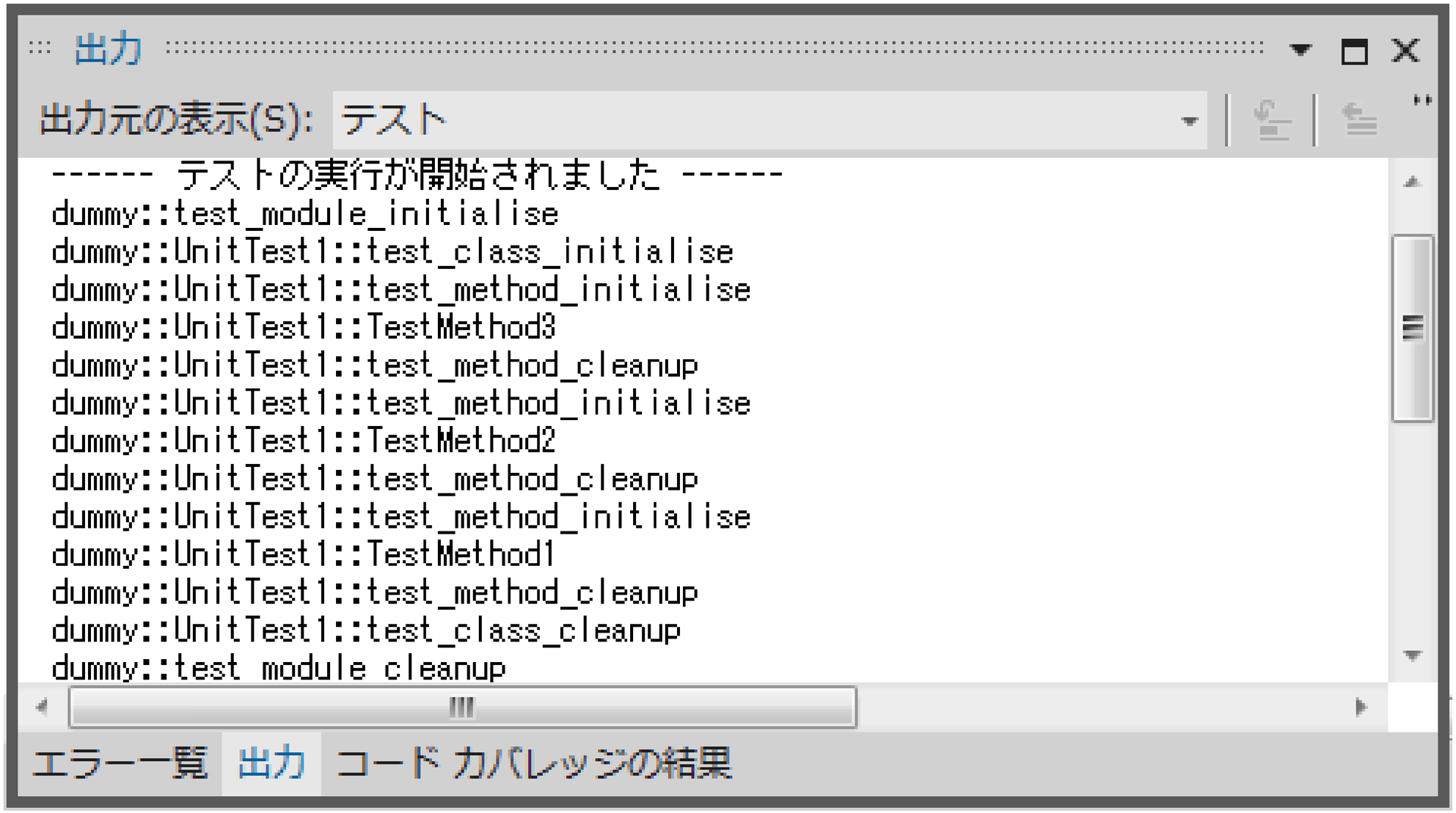
TEST_CLASS_INITIALIZE(method_name){...}
TEST_CLASS_CLEANUP(method_name){...}
TEST_CLASS 내 일련의 TEST_METHOD 실행 직전/직후에 한번만 하는 전 준비/뒤처리를 정의한다. 예를 들면 데이터베이스와 Socket과의 연결/절단 이라든가.

TEST_METHOD_INITIALIZE(method_name){...}
TEST_METHOD_CLEANUP(method_name){...}
각 TEST_METHOD의 실행 직전/직후에 호출된다.
테스트 별로 준비/뒤처리를 하는 것이 목적이다.

http://codezine.jp/article/detail/6464

```
#include "stdafx.h"
#include "CppUnitTest.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace dummy {

 TEST_MODULE_INITIALIZE(test_module_initialize) { Logger::WriteMessage(__FUNCTION__); }
 TEST_MODULE_CLEANUP(test_module_cleanup)      { Logger::WriteMessage(__FUNCTION__); }

 TEST_CLASS(DummyTest) {
 public:
  TEST_CLASS_INITIALIZE(test_class_initialize) { Logger::WriteMessage(__FUNCTION__); }
  TEST_CLASS_CLEANUP(test_class_cleanup)      { Logger::WriteMessage(__FUNCTION__); }

  TEST_METHOD_INITIALIZE(test_method_initialize) { Logger::WriteMessage(__FUNCTION__); }
  TEST_METHOD_CLEANUP(test_method_cleanup)      { Logger::WriteMessage(__FUNCTION__); }

  TEST_METHOD(TestMethod1) { Logger::WriteMessage(__FUNCTION__); }
  TEST_METHOD(TestMethod2) { Logger::WriteMessage(__FUNCTION__); }
  TEST_METHOD(TestMethod3) { Logger::WriteMessage(__FUNCTION__); }
 };

}
```

http://codezine.jp/article/detail/6464



http://codezine.jp/article/detail/6464

검증 함수	설명
AreEqual<T>(const T& expected, const T& actual)	expected == actual
AreEqual(double expected, double actual, double tolerance)	|expected-actual| <= |tolerance|
AreEqual(float expected, float actual, float tolerance)	
AreEqual(const char* expected, const char* actual, bool ignoreCase =false)	문자열로서 등가
AreEqual(const wchar_t* expected, const wchar_t* actual, bool ignoreCase =false)	
AreSame<T>(const T& expected, const T& actual)	&expected == &actual
AreNotEqual<T>(const T& notExpected, const T& actual)	!(notExpected == actual)
AreNotEqual(double notExpected, double actual, double tolerance)	|notExpected-actual| > |tolerance|
AreNotEqual(float notExpected, float actual, float tolerance)	

http://codezine.jp/article/detail/6464

검증 함수	설명
AreNotEqual(const char* notExpected, const char* actual, bool ignoreCase =false)	문자열로서 등가가 아니다
AreNotEqual(const wchar_t* notExpected, const wchar_t* actual, bool ignoreCase =false)	
AreNotSame<T>(const T& notExpected, const T& actual)	!(&notExpected == &actual)
IsNull<T>(const T* actual)	actual == nullptr
IsNotNull<T>(const T* actual)	!(actual == nullptr)
IsTrue(bool condition)	condition == true
IsFalse(bool condition)	condition == false
Fail()	실패
AreEqual<T>(T^ expected, T^ actual)	expected->Equals(actual)
AreEqual(string^ expected, string^ actual, bool ignoreCase =false)	문자열로서 등가
AreSame<T>(T% expected, T% actual)	%expected == %actual

http://codezine.jp/article/detail/6464

검증 함수	설명
AreNotEqual<T>(T^ notExpected, T^ actual)	!expected->Equals(actual)
AreNotEqual(string^ notExpected, string^ actual, bool ignoreCase =false)	문자열로서 등가가 아니다
AreNotSame<T>(T% notExpected, T% actual)	!(%notExpected == %actual)
IsNull<T>(T^ actual)	actual == nullptr
IsNotNull<T>(T^ actual)	!(actual == nullptr)
ExpectedException<Exception,Functor>(Functor functor)	functor()이 Exception을 throw 한다
ExpectedException<Exception,ReturnType>(ReturnType (*func)())	func()이 Exception을 throw 한다

http://codezine.jp/article/detail/6464

Visual C++ Team Blog

C++ tutorials, C and C++ news, and information about the C++ IDE Visual Studio from the Microsoft C++ team.

C++ Unit Testing in Visual Studio

April 19, 2017 by Augustin Popa // 18 Comments

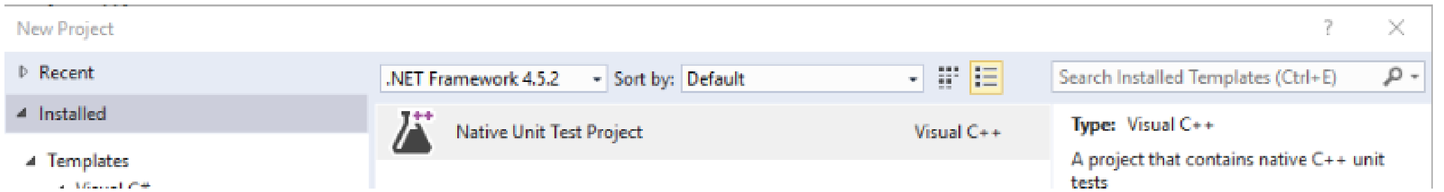
Testing is an increasingly important part of a software development workflow. In many cases, it is insufficient to test a program simply by running it and trying it out – as the scope of the project gets more involved, it becomes increasingly necessary to be able to test individual components of the code on a structured basis. If you're a C++ developer and are interested in unit testing, you'll want to be aware of Visual Studio's unit testing tools. This post goes through just that, and is part of a series aimed at new users to Visual Studio. This blog post goes over the following concepts:

1. Setting Up Unit Testing
2. The Microsoft Native C++ Unit Test Framework
3. Using the Test Explorer to Run Tests in the IDE
4. Determining Unit Test Code Coverage

Setting Up Unit Testing

The easiest and most organized way to set up unit tests is to create a separate project in Visual Studio for your tests. You can create as many test projects as you want in a solution and connect them to any number of other Visual Studio projects in that solution that contain the code you want to test. Assuming you already have some code that you want to test, simply follow these steps to get yourself set up:

1. Right-click your solution and choose *Add* > *New* > *Project*. Click the *Visual C++* category, and choose the *Test* sub-category. Select *Native Unit Test Project*, give the project a descriptive name, and then click *OK*.



https://blogs.msdn.microsoft.com/vcblog/2017/04/19/cpp-testing-in-visual-studio/

Visual Studio에서 C/C++에 대한 단위 테스트 작성

2017. 11. 04. • 읽는 데 8분 • 참가자

이 문서의 내용

다른 언어처럼 **테스트 탐색기** 창을 사용하여 C++ 단위 테스트를 작성하여 실행할 수 있습니다. **테스트 탐색기**에 대한 자세한 내용은 테스트 탐색기를 사용하여 단위 테스트 실행을 참조하세요.

> **참고**
>
> Live Unit Testing, Coded UI Tests 및 IntelliTest 등의 일부 기능은 C++에서 지원되지 않습니다.

Visual Studio에는 다음 C++ 테스트 기능이 포함되어 있으며 추가 다운로드가 필요하지 않습니다.

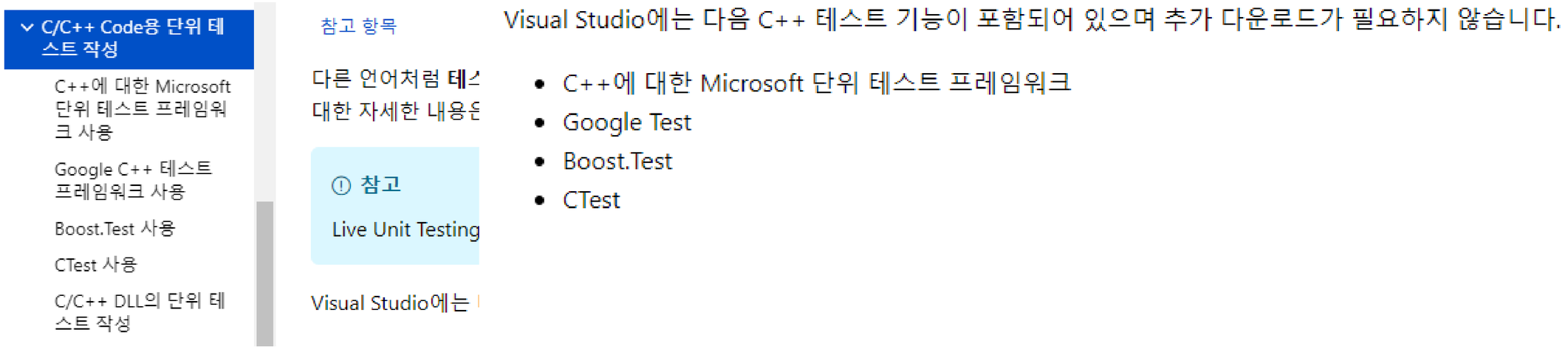
- C++에 대한 Microsoft 단위 테스트 프레임워크
- Google Test
- Boost.Test
- CTest

다른 C++ 유닛테스트 프레임워크

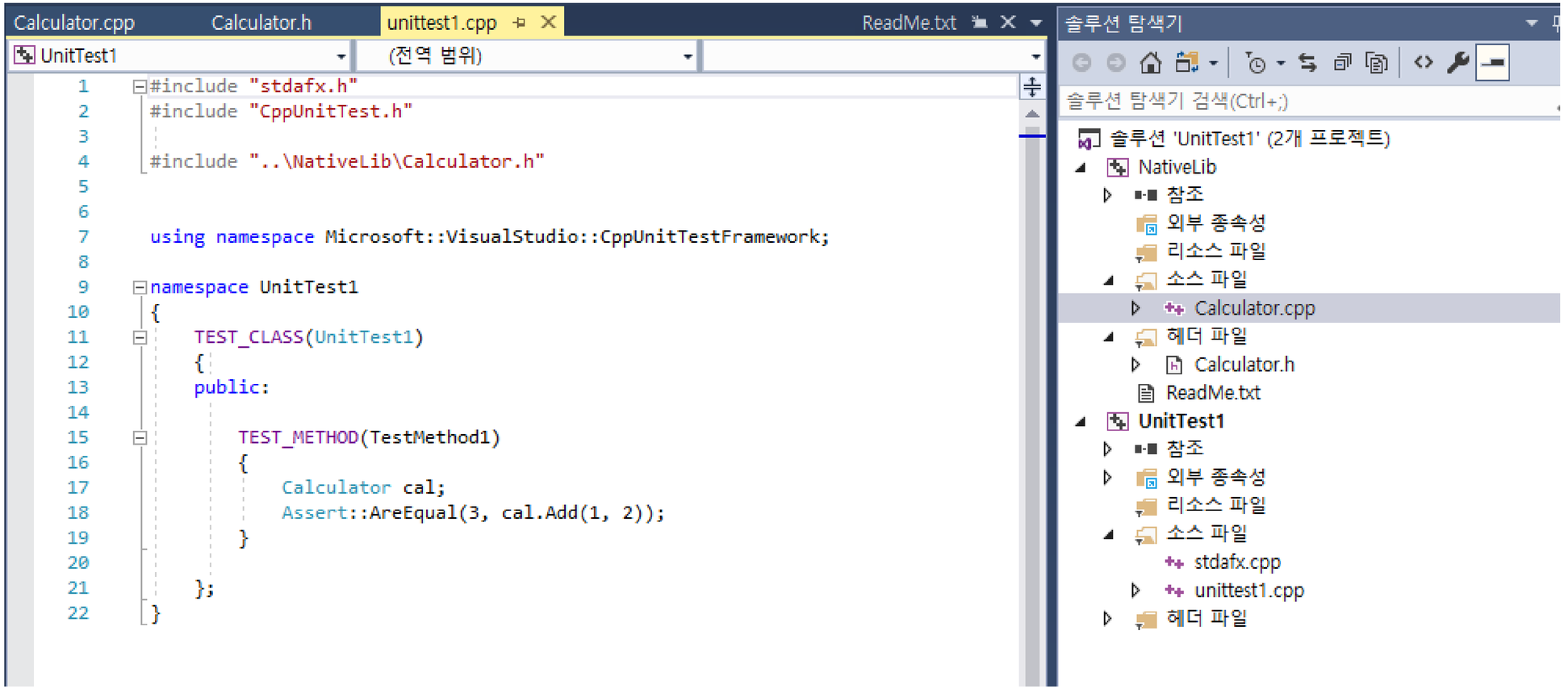
GTest
https://github.com/google/googletest

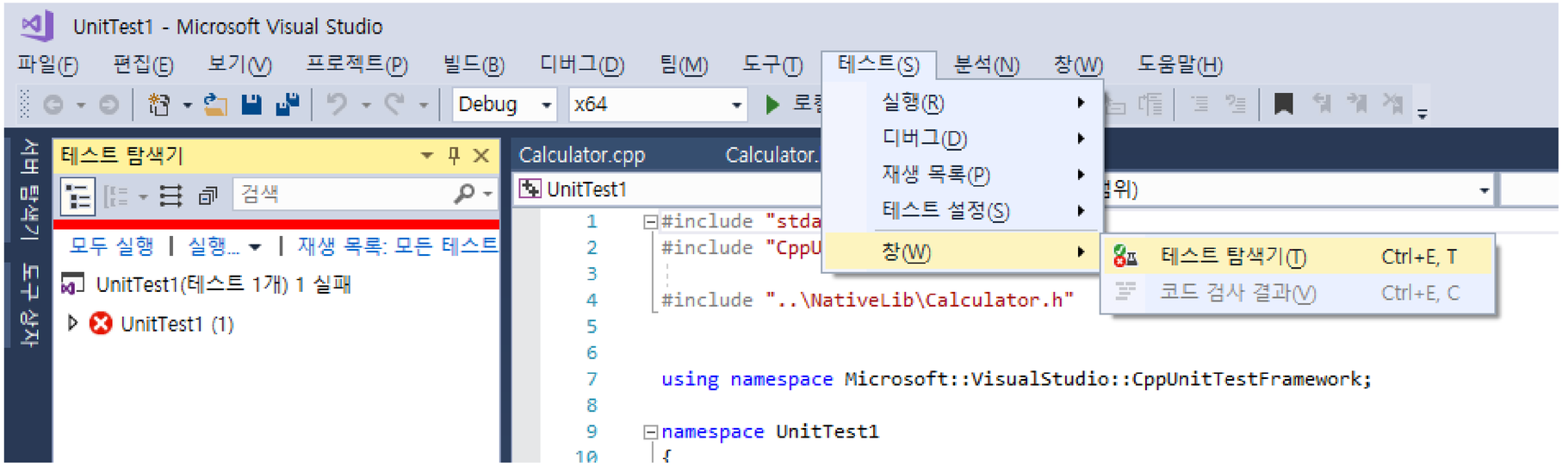
Boost.Test
http://www.boost.org/doc/libs/1_60_0/libs/test/doc/html/index.html

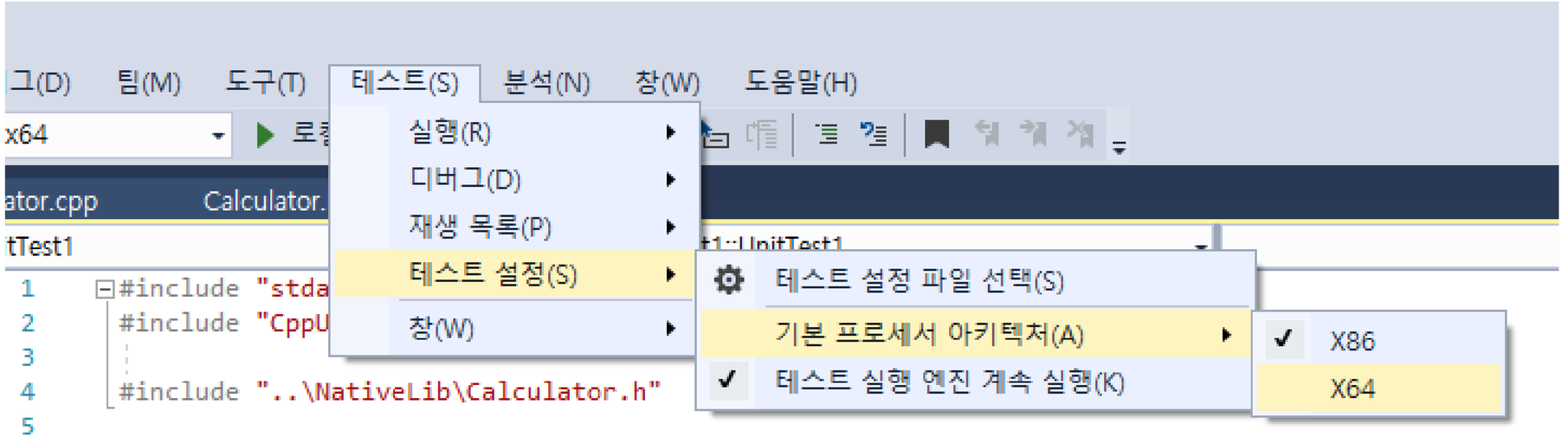
Catch
https://github.com/philsquared/Catch



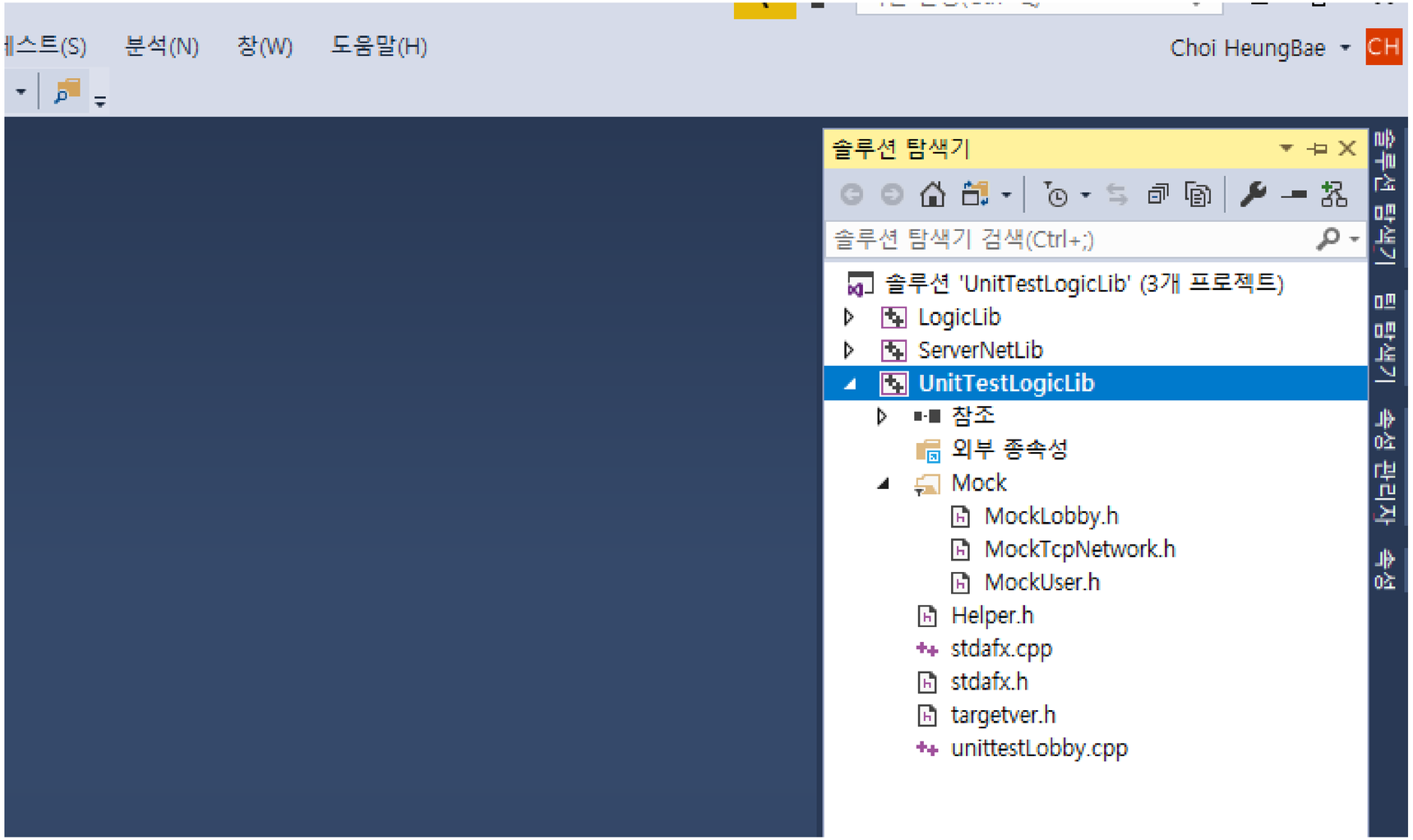
Demo – 자주 보는(?) 계산기

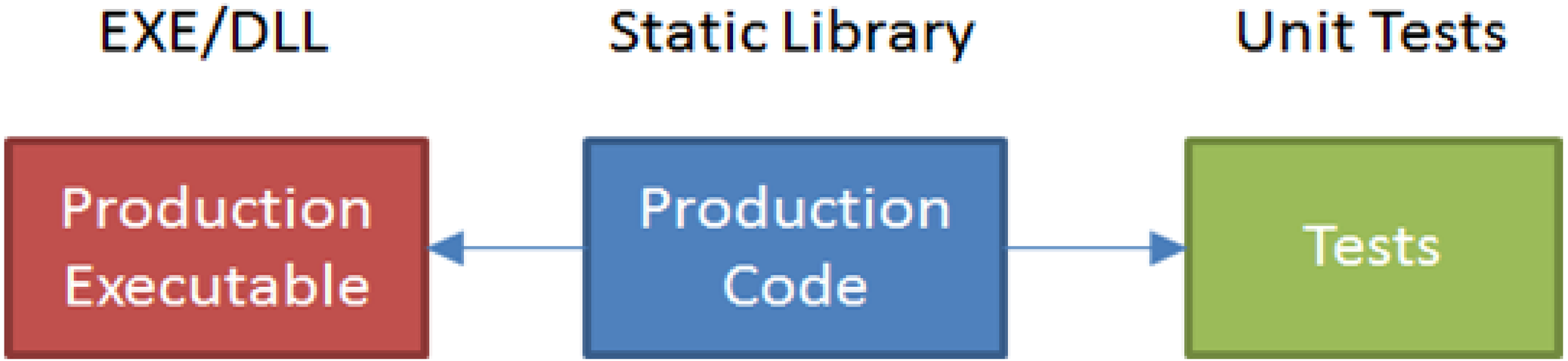



UnitTest1 - Microsoft Visual Studio
파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 팀(M) 도구(T) 테스트(S) 분석(N) 창(W) 도움말(H)
Debug
x64
실행(R)
디버그(D)
재생 목록(P)
테스트 설정(S)
창(W)
테스트 탐색기(T) Ctrl+E, T
코드 검사 결과(V) Ctrl+E, C
테스트 탐색기
검색
모두 실행 | 실행... | 재생 목록: 모든 테스트
UnitTest1(테스트 1개) 1 실패
UnitTest1 (1)
Calculator.cpp
UnitTest1
#include "stda
#include "CppU
#include "..\NativeLib\Calculator.h"
using namespace Microsoft::VisualStudio::CppUnitTestFramework;
namespace UnitTest1


그(D) 팀(M) 도구(T) 테스트(S) 분석(N) 창(W) 도움말(H)
x64
실행(R)
디버그(D)
재생 목록(P)
테스트 설정(S)
창(W)
테스트 설정 파일 선택(S)
기본 프로세서 아키텍처(A)
테스트 실행 엔진 계속 실행(K)
X86
X64
#include "stda
#include "CppU
#include "..\NativeLib\Calculator.h"

Demo – 채팅 서버


테스트(S)
분석(N)
창(W)
도움말(H)
Choi HeungBae
CH
솔루션 탐색기
솔루션 탐색기 검색(Ctrl+;)
솔루션 'UnitTestLogicLib' (3개 프로젝트)
LogicLib
ServerNetLib
UnitTestLogicLib
참조
외부 종속성
Mock
MockLobby.h
MockTcpNetwork.h
MockUser.h
Helper.h
stdafx.cpp
stdafx.h
targetver.h
unittestLobby.cpp
솔루션 탐색기
팀 탐색기
속성 관리자
속성



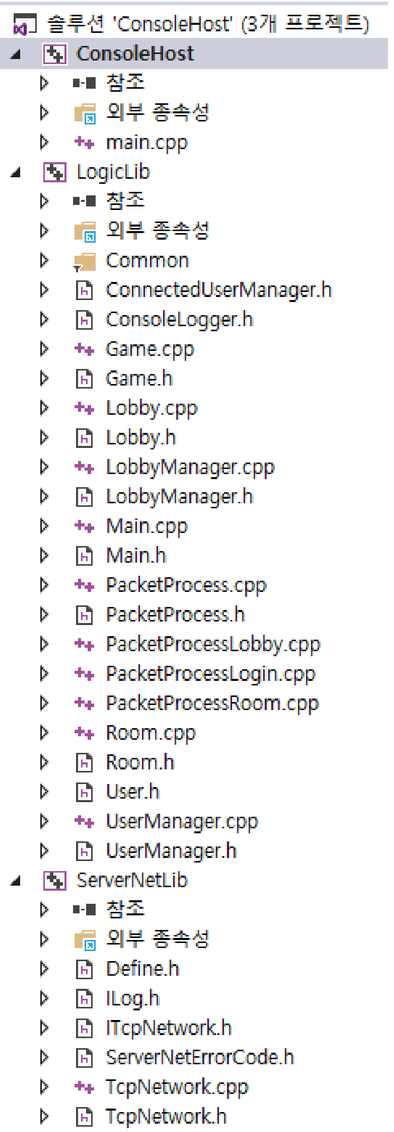
http://blogs.msdn.com/b/jsocha/archive/2010/11/19/writing-unit-tests-in-visual-studio-for-native-c.aspx

클라이언트/서버 공통 라이브러리 프로젝트

네트워크 라이브러리 프로젝트

게임 로직 라이브러리 프로젝트

플랫폼 의존 실행 파일 프로젝트



유닛테스트를 할 때는 해당 기능만 테스트 해야 한다

```
class class MembserManager
{
    ……
    void AddMember(Player* pNewPlayer)
    {
        // 조건 조사
        // 컨테이너에 추가
        // 네트워크로 다른 멤버들에게 통보
    }
};
```

테스트에서 제외

인터페이스를 사용한 클래스 정의, mock이 필요

```
class INetwork
{
};

class Network : public INetwork
{
};
```

```
class MembserManager
{
public:
  MembserManager(INetwork* pNetwork);
  {
    m_pNetwork = pNetwork;
  }

private:
  INetwork* m_pNetwork;
};
```

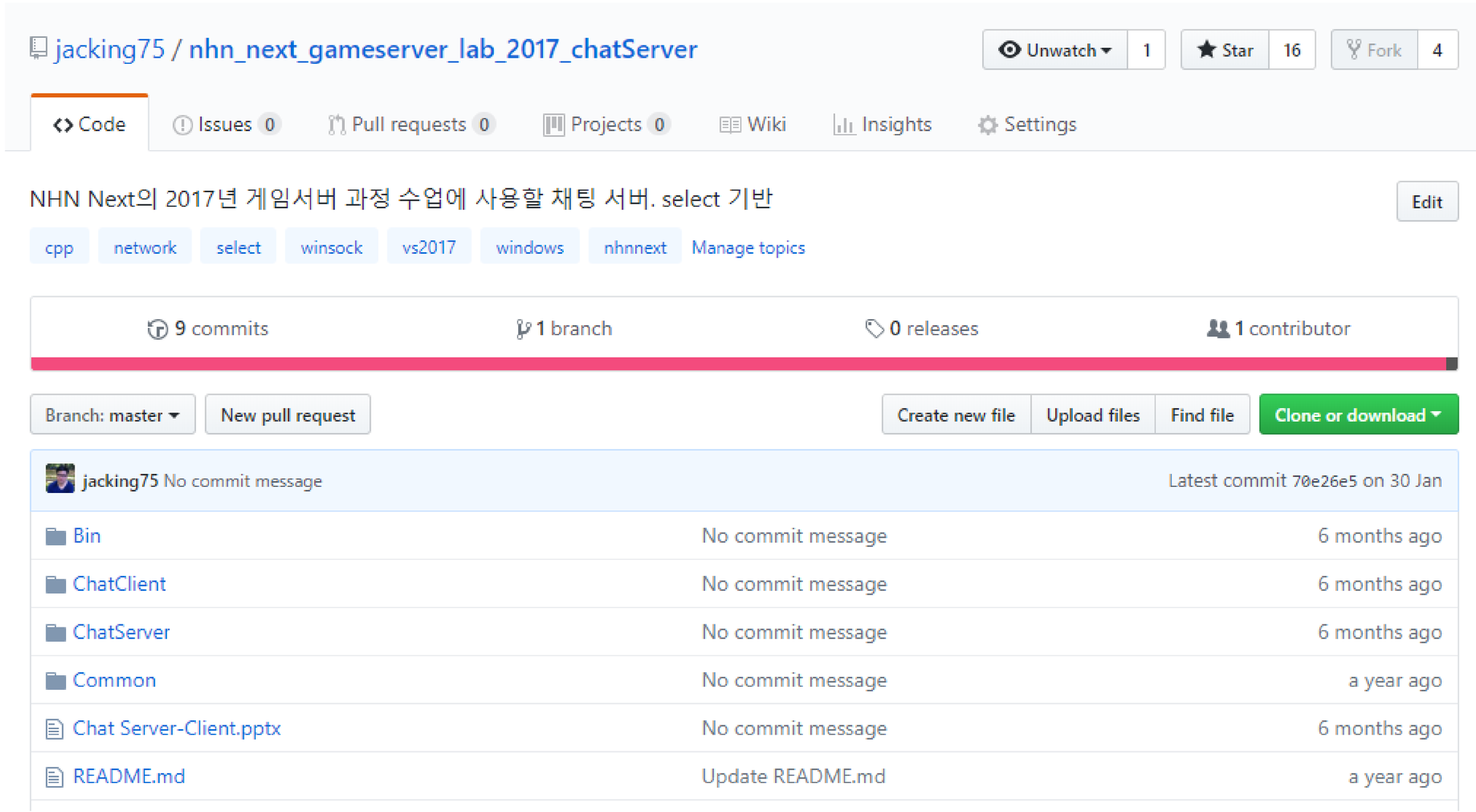
```
class MockNetwork : public INetwork
{
public:
  virtusl Send(char* pData) override
  {
    ....
    m_SendDatas.push_back(packet);
  }

  std::vector<Packet> m_SendDatas;
};
```

```
auto pMockNetwork = new MockNetwork();
auto MemberMgr =
MembserManager(pMockNetwork);
```

전역 변수
static 변수

사용하지 않는 것이...

채팅 서버 유닛 테스트



https://github.com/jacking75/nhn_next_gameserver_lab_2017_chatServer

마무리...

유닛 테스트 효용성

유닛 테스트가 절대적인 것은 아니다.

유닛 테스트가 없는 것보다는 훨씬 좋다.

유닛 테스트를 해보지 않고 비판하지 말자.

유닛 테스트를 하지 않으면 대안을 제시하고 실천하자.

유닛 테스트를 하면 코드 리팩토링도 얻을 수 있다

JavaScript, Ruby, Python

C#, Java

C++

C++은 유닛 테스트가 다른 언어에 비해 어려움.
C#의 경우 유닛 테스트에서 private도 쉽게 접근 가능

좋은 함수가 아니면

유닛 테스트도 힘들다

리팩토링

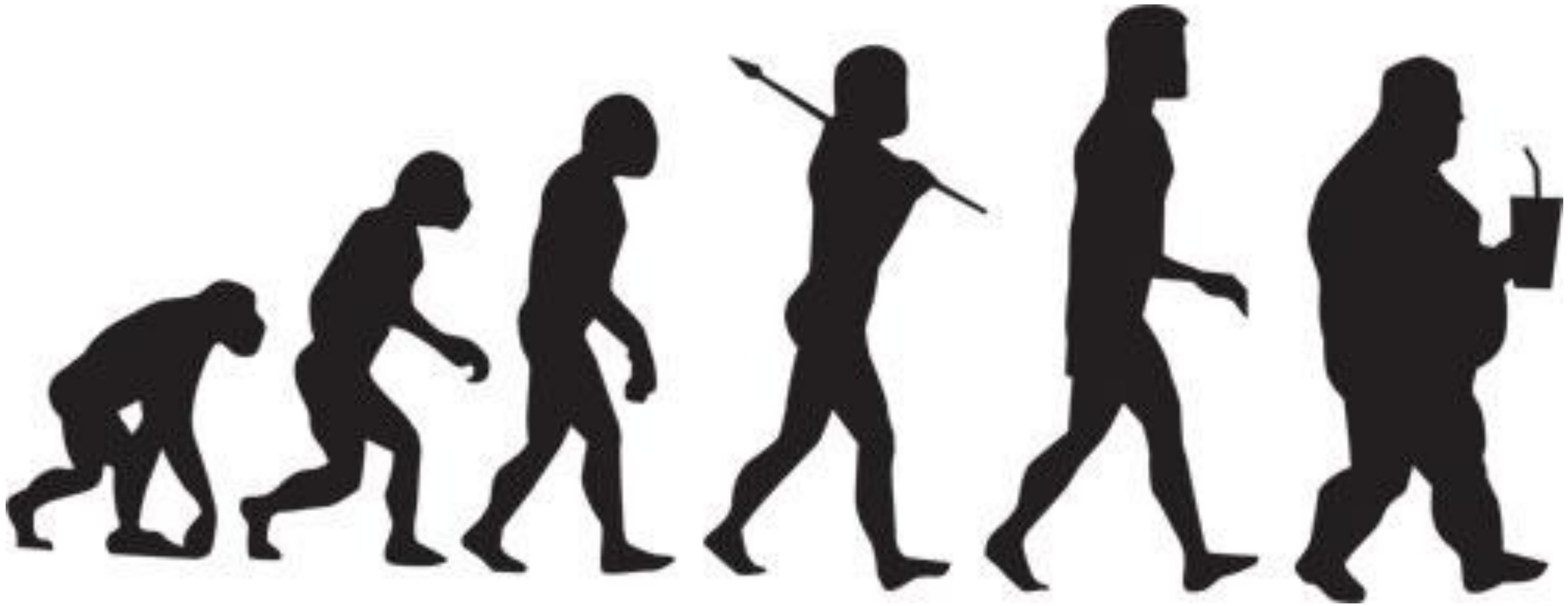
빨리 끝내야 하는 일

1) 개발하고 바로 시연 테스트 하기

2) 유닛 테스트 만든 후 시연 테스트 하기

아마 1번의 유혹에 많이 빠질 듯(사실 저도... ;;;;)

유닛 테스트 좋은지는 알겠는지 습관을 들이기 힘들다
나와 주위의 도움이 필요



게임 서버는 대부분 분산 서버 구조,
테스트 중 버그가 발생하면 서버 수정하고,
배포하고, 다시 시작.
너무 많은 시간을 소비하고 피곤함

유닛 테스트를 먼저해서
시연 테스트를 빨리 끝내야 한다

꽤 진행된 프로젝트,
누군가의 반대(특히 리더급)

너무 힘들다 ㅠㅠ

추천 하고 싶은 문서

생산적인 개발을 위한 지속적인 테스트

http://www.slideshare.net/birdkr/ss-2183466

감사합니다.