Testování, Unit testování a Dokumentace zdrojového kódu

Téma není složité, všichni jsme to už prošli. Myslím, že bude stačit, aby si každý normálně přečetl dokument jednou a poté vyzkoušel základní praktický příklad s Unit testy. V nejhorším případě mluvte něco o dokumentaci

## Testování softwaru

Testování softwaru je součástí vývoje softwaru. Jedná o zkoumání kvality a odhalování chyb v programech. Co to ale kvalita je? Kvalita je stupeň shody s požadavky.

Většina intuitivně tuší, že ověřit funkčnost je důležité. Dimenzí kvality je ale podstatně více a na mnohé z nich se zapomíná. Zde jsou:

* **Funkčnost** (Functionality) – správné chování funkcí systému, jak je definováno funkční specifikací
* **Použitelnost** (Usability) – zda vůbec a jak lze dosáhnout požadovaného cíle, zda je systém uživatelsky přívětivý, zda se s ním dobře pracuje
* **Spolehlivost** (Reliability) – zda se chová stejně za všech okolností, zvláště po přetížení, nebo po výpadku či chybě, zda tyto stavy umí detekovat a hlásit
* **Výkon** (Performance) – zda systém není pomalý a zvládne větší množství současně pracujících uživatelů, nebo naopak zda si i při splnění všech požadavků na obsluhu uživatelů nebere příliš systémových zdrojů
* **Podpora** (Supportability) – zda se systém dobře instaluje, nemá problémy s cílovými hardwarovými a softwarovými konfiguracemi a další vlastnosti související s údržbou systému a upgradovatelností
* **Bezpečnost** (Security) – zda jsou systém i jeho data bezpečné a mohou jej používat pouze oprávněné osoby
* **Kompatibilita** (Compatibility) – zda je možné jej používat i s ostatními programy a systémy
* **Přenositelnost** (Portability) – zda je možné jej přenášet například na jiný OS

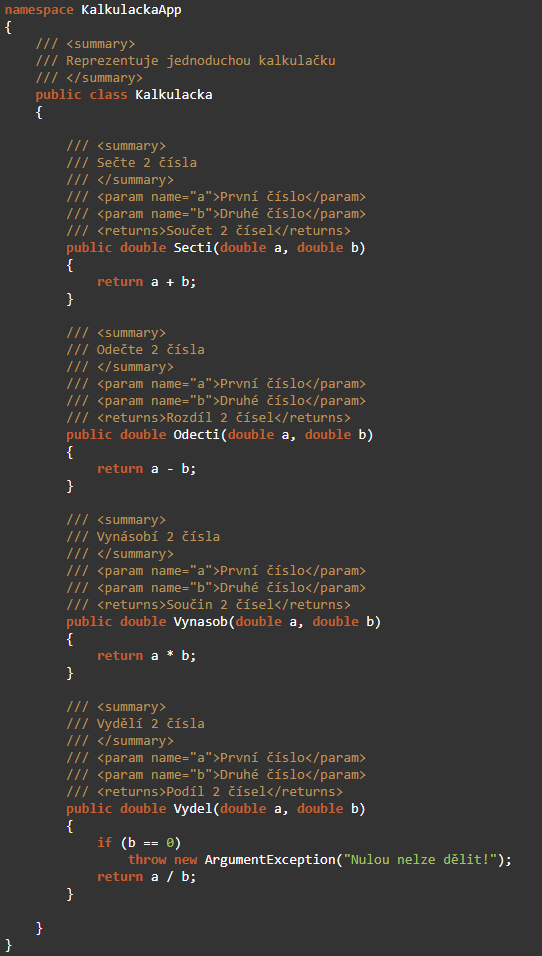
Další způsoby dělení testování jsou podle:

* **Fáze testování**
  + Unit testy
  + Modul Testy
  + Testy komponent
  + Integrační testy
  + Funkční testy
  + Systémové testy
  + Akceptační testy
* **Znalosti kódu**
  + White Box
  + Black Box
* **Způsobu realizace testů**
  + Manuální
  + Automatizované
  + Exploratory testing

**Unit testování**

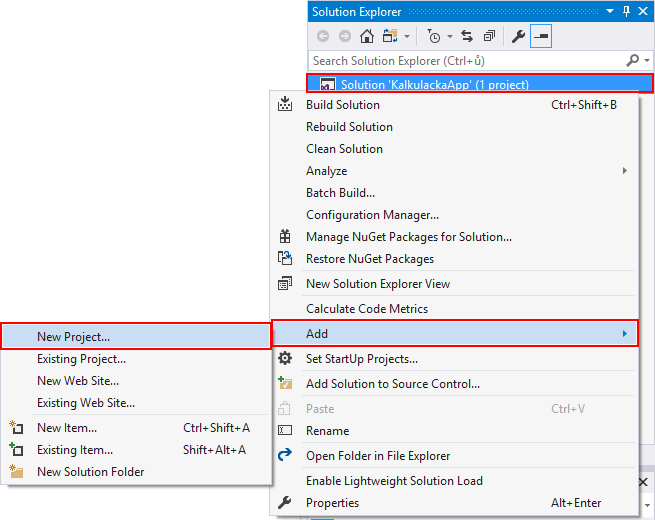
Každý jednotkový test by měl testovat pouze jednu operaci jedné jednotky. Test by neměl ověřovat stav a interakci příliš mnoha nesouvisejících součástí (to by zvyšovalo závislost testu na konkrétní implementaci). Testy by se neměly vzájemně překrývat v tom, co testují.

V C# .NET se unit testy píší pomocí nástrojů ve jmenném prostoru *Microsoft.VisualStudio.TestTools.UnitTesting*. Visual Studio poskytuje plnou podporu těchto testů a ke své aplikaci je přidáme jako další projekt do solution. Testy tedy budou od projektu úplně oddělené, což je návrhově velká výhoda, pouze nesmíme zapomenout projekty propojit příslušnými referencemi.



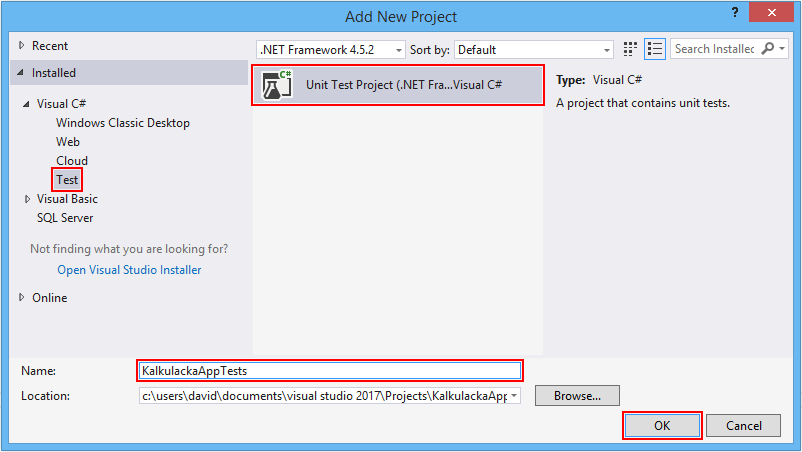
**1**

V Solution Exploreru klikneme na solution pravým tlačítkem a zvolíme Add -> New Project....



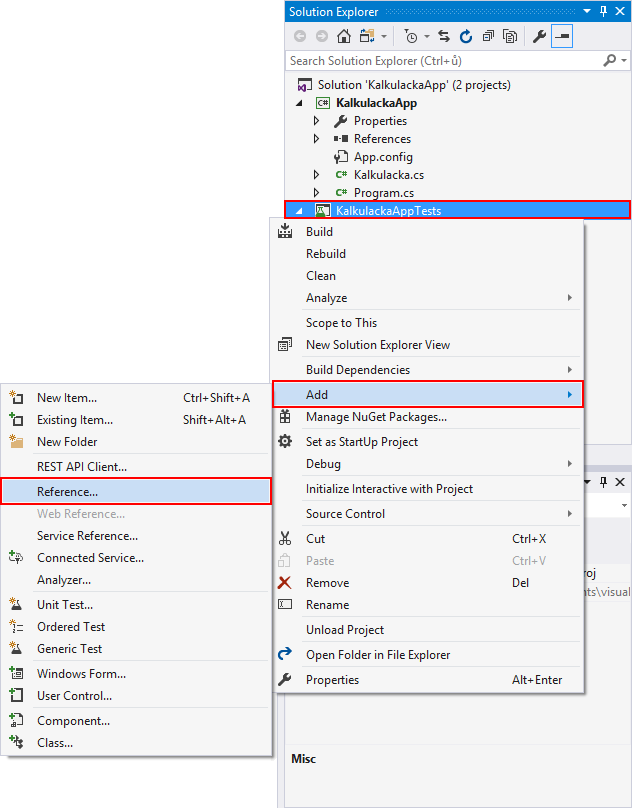
**2**

Název projektu s testy se zpravidla sestavuje jako název projektu aplikace + slovo "Tests", v našem případě tedy "KalkulackaAp­pTests":



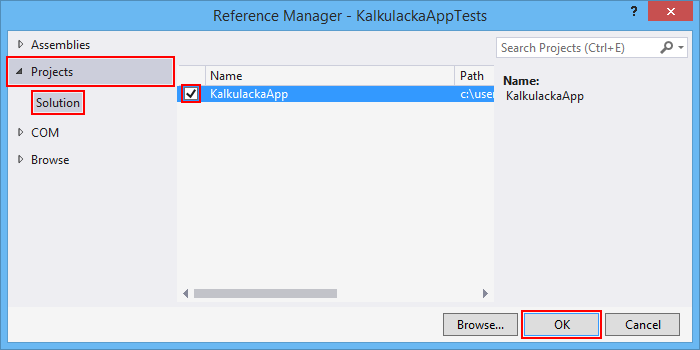
**3**

Do testovacího projektu nyní musíme přidat referenci na projekt s aplikací, abychom mohli přistupovat k příslušným třídám. To provedeme kliknutím pravým tlačítkem na projekt KalkulackaAppTests a zvolením Add -> Reference...

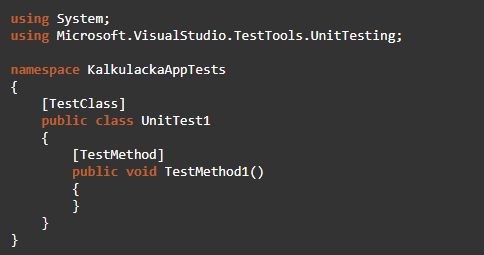


**4**

V následujícím formuláři vybereme záložku Projects -> Solution a zaškrtneme projekt KalkulackaApp. Dialog potvrdíme a tím si zpřístupníme třídu Kalkulacka.



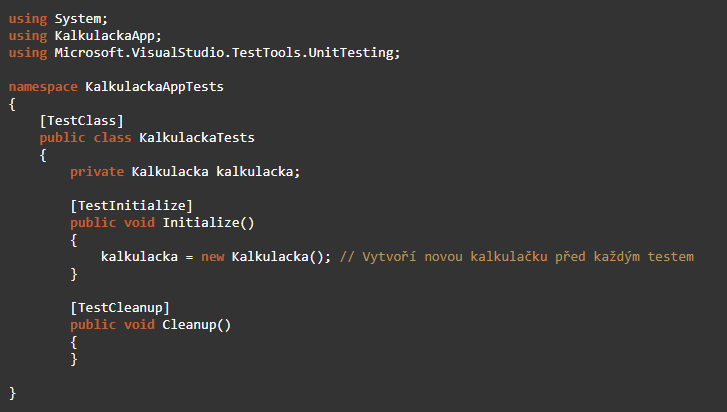
V projektu KalkulackaAppTest se nám vygeneroval nový soubor UnitTest1 s následujícím kódem:



Asi vás v objektovém programování v C# nepřekvapí, že je test třídy (scénář) reprezentovaný také třídou a jednotlivé testy metodami. S atributy (hranatými závorkami nad metodami a třídami). [TestClass] zde označuje testovací scénář, pomocí [TestMethod] jsou označené metody, které reprezentují jednotlivé testy (budou automaticky spuštěné Visual Studiem). Třídu (i její soubor) si přejmenujte na KalkulackaTests.

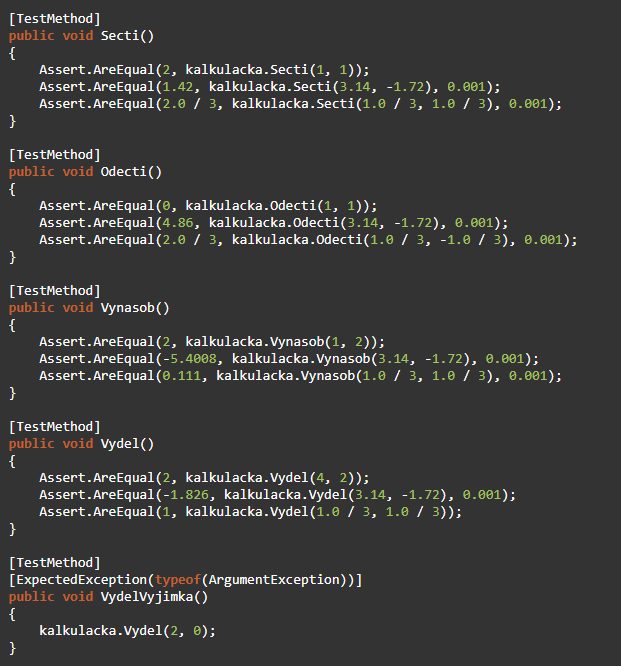
**5**

Pokrytí třídy testy. V unit testech můžeme použít ještě několik atributů. My nyní využijeme [TestInitialize] a pro názornost i [TestCleanup], čímž můžeme označit metody, které se zavolají před, resp. po každém testu v této třídě. Obvykle tedy před každým testem připravujeme znovu to samé prostředí, aby se vzájemně vůbec neovlivňovaly. Do třídy si přidejme atribut kalkulacka a v metodě s anotací [TestInitialize] v něm vždy vytvořme čerstvě novou kalkulačku pro každý test. Pokud by ji bylo ještě třeba dále nastavovat nebo bylo třeba vytvořit další závislosti, byly by také v této metodě. Metodu TestMethod1() odstraníme. Pro kalkulačku přidáme using KalkulackaApp;



**6**

Jednotlivé metody budou vždy označené atributem [TestMethod] a budou testovat jednu konkrétní metodu z třídy Kalkulacka, typicky pro několik různých vstupů. Pokud vás napadá proč metody označujeme atributy, umožňuje nám to vytvořit si i pomocné metody, které můžeme v daném testu využívat a které nebudou pokládány za testy. Visual Studio nám totiž testy (metody s anotací [TestMethod]) automaticky spustí a vypíše jejich výsledky. Přidejme následujících 5 metod do třídy KalkulackaTests:



K porovnávání výstupu metody s očekávanou hodnotou používáme statické metody v třídě Assert. Nejčastěji asi použijeme metodu AreEqual(), která přijímá jako první parametr očekávanou hodnotu a jako druhý parametr hodnotu aktuální. Desetinná čísla jsou v paměti počítače reprezentována binárně, a to způsobí určitou ztrátu jejich přesnosti a také určité obtíže při jejich porovnávání. Proto musíme v tomto případě zadat i třetí parametr a to je delta, tedy kladná tolerance, o kolik se může očekávaná a aktuální hodnota lišit, aby test stále prošel. Všimněte si, že zkoušíme různé vstupy. Sčítání netestujeme jen jako 1 + 1 = 2, ale zkusíme celočíselné, desetinné i negativní vstupy, odděleně, a ověříme výsledky.

Poslední test ověřuje, zda metoda Vydel() opravdu vyvolá výjimku při nulovém děliteli. Jak vidíte, nemusíme se zatěžovat s try-catch bloky, stačí nad metodu přidat atribut [ExpectedException] a uvést zde typ výjimky, která se očekává. Pokud výjimka nenastane, test selže. Pro testování více případů vyvolání výjimky můžeme použít další assert metody.

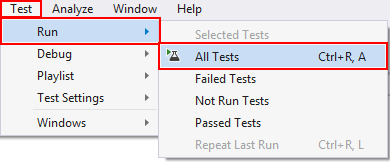
**6.1**

Kromě metody AreEqual() můžeme použít ještě mnoho dalších, určitě se snažte použít tu nejvíce vyhovující metodu, zpřehledňuje to hlášky při selhání testů a samozřejmě i následnou opravu.

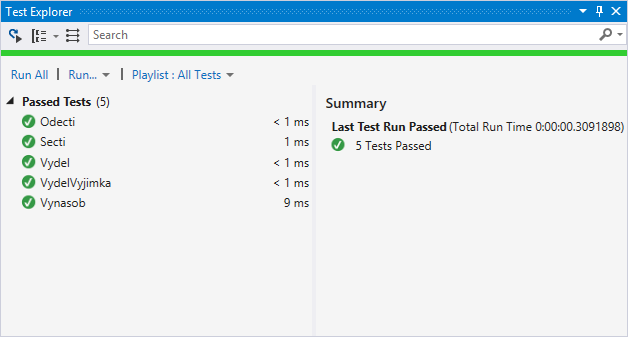
* **AreNotEqual()** - Používáme pokud chceme ověřit, že se 2 objekty neshodují. Další metody s Not zde již nebudeme zbytečně zmiňovat.
* **AreSame()** - Zkontroluje, zda 2 reference ukazují na stejný objekt (porovnává pomocí ==)
* **Equals()** - Používáme v případě, když chceme ověřit 2 objekty pomocí metody Equals() a zjistit, zda jsou stejné. Nepoužíváme pro ověření hodnoty místo AreEqual()
* **Fail()** - Způsobí selhání testů, obvykle vkládáme za nějakou podmínku a doplňujeme o volitelné parametry, kterými jsou chybová hláška a parametry
* **Inconclusive()** - Funguje podobně jako Fail(), vyvolá výjimku signalizující neprůkaznost testu
* **IsFalse()** - Ověří, zda je daný výraz nepravdivý
* **IsInstanceOfType()** - Ověří, zda je objekt instancí daného typu
* **IsNull()** - Ověří, zda je hodnota null
* **IsTrue()** - Ověří, zda je daný výraz pravdivý
* **ReplaceNullChars()** - Nahradí nullové znaky ("\0") za "\\0", využíváme zejména u diagnostických výpisů řetězců s těmito znaky
* **ThrowsException()** - Spustí předaný delegát a ověří, že vyvolává výjimku předanou jako generický argument. Metoda má také asynchronní verzi ThrowsExceptionAsync()

**7**

Spuštění testů. Testy spustíme z menu Test -> Run -> All Tests:



Uvidíme výsledky, které vypadají nějak takto:



## Dokumentace

Vnější – mimo zdrojový kód

* definice problému, požadavky, architektura
* popis rozhraní, tutoriál, FAQ
* popis návrhu, postupy, rozhodnutí, konvence
* uživatelská dokumentace

Vnitřní – součástí zdrojového kódu

* styl zápisu zdrojového kódu
* komentáře k souborům, třídám, funkcím, blokům
* popis rozhraní

*Myslím, že každý chápe, jaká kódová dokumentace je v samostatném souboru nebo uvnitř programu pomocí komentářů. A myslím, že každý ví, proč je to nutné a jak je napsat. Pokud máte pochybnosti, zde jsou vybrané informace k tomuto tématu:*

[*https://d3s.mff.cuni.cz/f/teaching/nprg043/13-documentation.html*](https://d3s.mff.cuni.cz/f/teaching/nprg043/13-documentation.html)