20. Testování softwaru

Testování se provádí k odhalení chyb (ne k potvrzování bezchybnosti). Protože testy nám ukazují pouze chyby, co jsou na povrchu. Většinou proto zkoušíme více testů s trochu jinými parametry. Pokud sada testů dlouhodobě neukazuje chyby, není nutné ji opakovaně spouštět.

Nejdůležitější základní druhy testů softwaru:

Unit testy – **navrhují a provádějí programátoři**, testují vnitřní chod programu a jeho většinou povrchové chyby a bugy

Integrační testy – provádějí testeři, kteří neznají (ani nepotřebují) interní strukturu kódu (testy základního chování, spojení s daty, funkce elementů v GUI, …)

Akceptační testy – navrhují analytici testů, provádějí to koncoví uživatelé, bývá uzavřené prostředí (beta verze u her)

Regresní testy – provádějí se po změnách a opravách chyb, jestli nebylo poškozeno něco, co již fungovalo („zavlečené chyby“), provádí se často a bývají automatizované

Exploratorní (Výzkumné) testy – provádí tester, znalost přesného použití programu, snaha o reálné scénáře použití (i extrémní používání, off-line, chybějící periferie, mnohonásobné klikání, …)

Penetrační testy – pro ověření zabezpečení systému, běžné techniky pro porušení integrity systému, skrytí nejdůležitějších informací před vnějším vniknutí (hesla, podvržení identity, SQL Injection, Stack (nebo Buffer) overflow, …)

Výkonnostní (Performance) testy – test použitelnosti v reálném provozu, přijatelné odezvy i na horších strojích a při horším připojení. Pomalost chodu systému bývá jedna z nejčastějších důvodů znechucení uživatelů.

Unit testy

Pro programátory jsou důležité hlavně unit testy. Pomáhají udržovat správnost a flexibilitu kódu.

Zásady pro vytvoření unit testů jsou Rychlost, Izolovanost, Opakovatelnost, Jednoduchost, Popisnost.

Obecná konvence a schéma pro všechny unit testy:

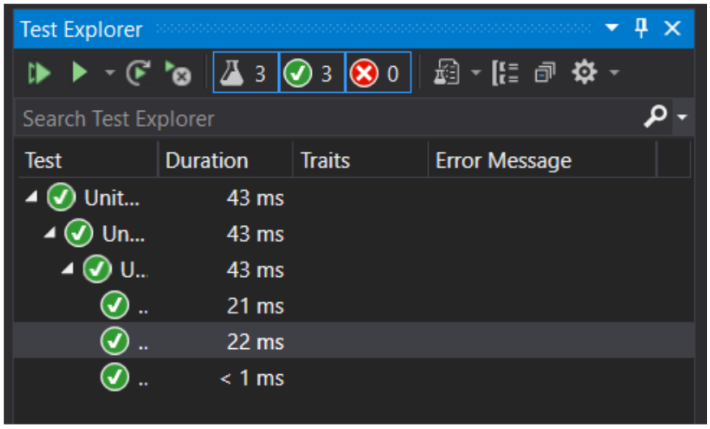
Arrange – příprava na počáteční podmínky pro test (nastavení proměnných, import dat)

Act – provedení samotného test (volání metody, vytvoření objektu)

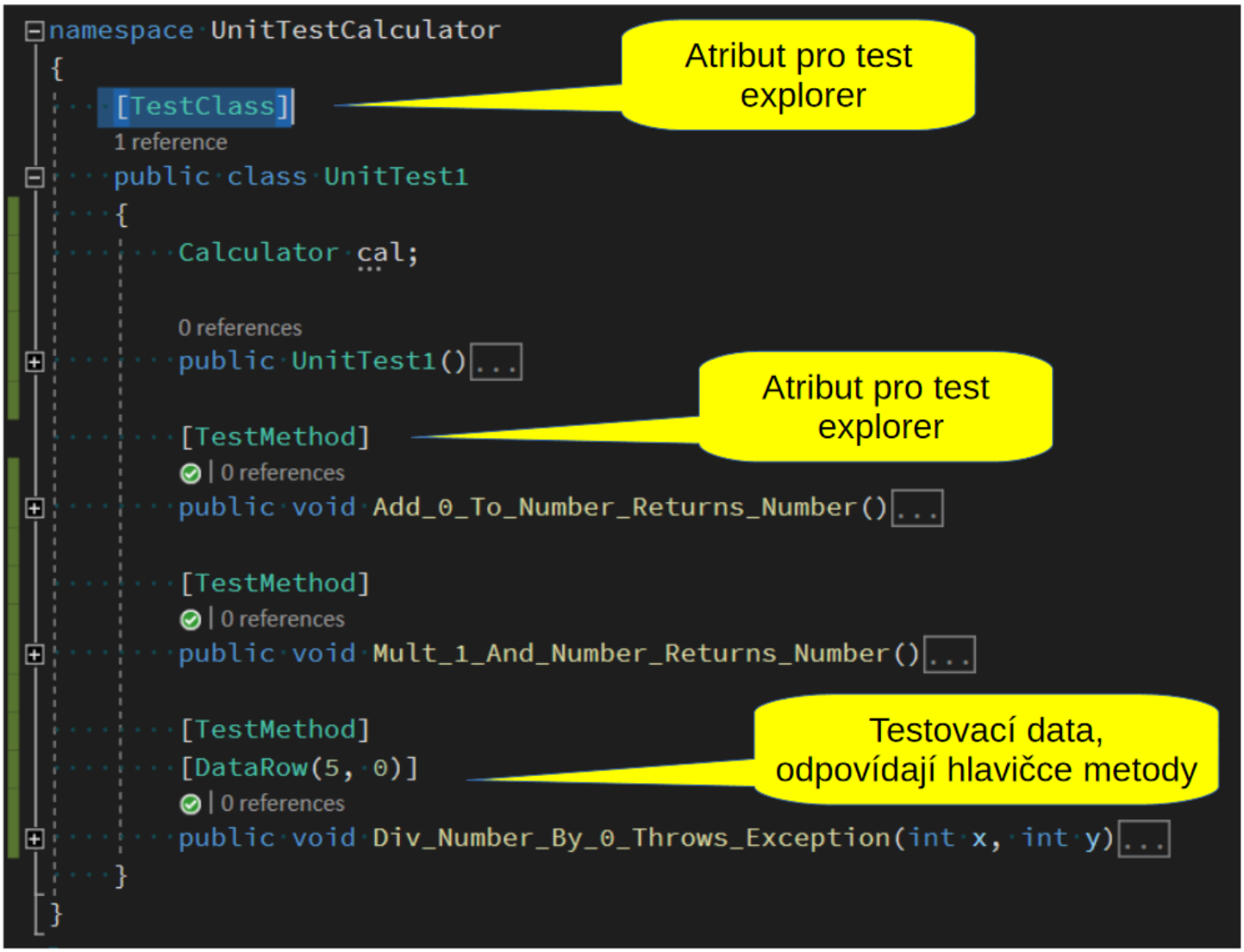
Assert – porovnání výsledku a očekávaného výsledku

Konvencí by také mělo být správné pojmenování testu tak, aby jinému programátorovi řekl co se testuje i jaký by měl být výsledek.

Ve Visual Studiu je speciální druh projektu na unit testy, který se dá jednoduše přidat do již stávajícího *Solution (řešení)*. Zde se dají napsat, zkoušet a získávat výsledky testů. Je zde i speciální statistika, jestli test prošel, za jak dlouho, kde byla chyba, čím vším to prošlo, …



Syntaxe testů

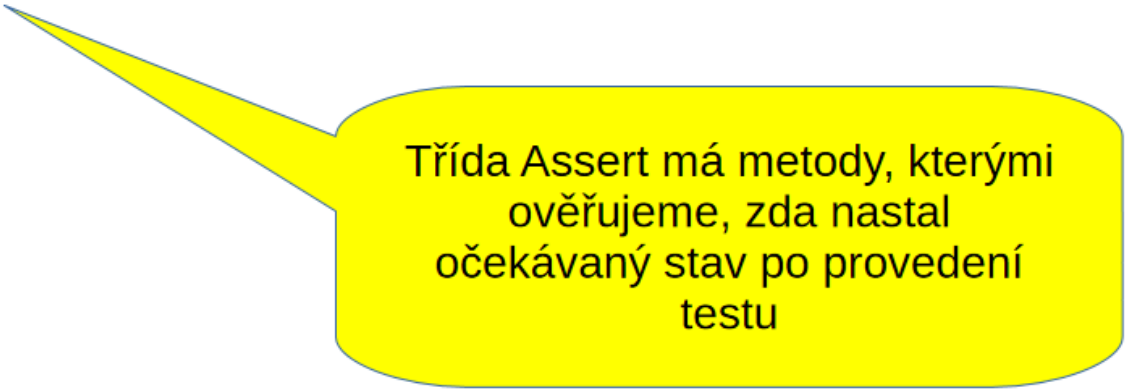


Typická testovací metoda

```
public void Add_0_To_Number_Returns_Number()
{
    // Arrange
    int x = 5, y = 0;

    // Act
    int result = cal.Add(x, y);

    // Assert
    Assert.IsTrue(result == x);
}
```



Test Driven Development

TDD je přístup k vývoji SW založen na malých krocích. Jde o techniku, kdy nejdříve se definuje funkcionalita, poté se napíše test pro tuto funkcionalitu, dále napsání kódu a pak ověření funkcionality. Jako testy se využívají právě unit testy. Jsou rychlé a automatické a můžeme je spouštět opakovaně a i během psaní nového kódu.

Vývojový cyklus podle TDD:

Definice funkce – vytyčení funkcionality, základní vlastnost, co by měla dělat (většinou v hlavě, na papír, …)

Napsat test – vytvoření testu podle toho, co by měla testovaná funkce dělat, tím se eliminuje odchýlení od funkčnosti, všechny testy by se měli spustit a žádný by neměl projít.

Napsat kód – napsání kódu, aby to vyhovovalo funkcionalitě a aby rovnou splnil test

Otestovat kód – testy mohou být i automatické, je potřeba zachytit všechny aspekty funkce s různými parametry, spuštěný může být i vícekrát

Refaktorace – snaha o jednoduší a elegantnější kód, musí pořad projít všemi testy (tudíž opětovné spuštění testů)

Přínosem je menší snaha o vytvoření něčeho víc (snaha pouze o splnění podmínek v testech), vše by mělo být správně právě díky testům, vše souvisí se vším tudíž snazší orientace v kódu

Zápornou stránkou je pomalejší vývoj, chyby mohou být i v testech (tudíž chybný kód i přes splněný test), některé části kódu a některé funkce se těžko testují a test zabere delší čas vytvořit než samotný kód.