Unit Test

1 Hvorfor anvende Unittest

Vi laver unittest af vores kode for at teste vores enklte funktioner. Dette gør, at vi kan opfange evt bugs i vores kode på et tidligt stadie contra hvis vi lavede big bang på vores kode, så ville det være svært at finde bugs. Det er ligeledes en god måde at dokumentere funktionaliteten af ens kode på. Det får også en til at sætte tempoet ned og tænke MENS man koder og ikke bare koder med hovedet under armen.

2 Hvordan unitteste

Til at skrive unittest anvender vi et test framework. Vi anvender Nunit bl.a. fordi:

- Detaljeret fejlrapport
- Mulighed for setup og teardown
- Nemt at tilføje nye test til systemet

Ved **navngivning** af testmetoderne gælder følgende:

1. Navn på UUT
2. Navn på metoden, som testes
3. Det forventede resultat

Når vi skriver en ny unittest er det typisk for en klasse. Syntaksen for en sådan test klasse kan ses på figur 1 herunder. Når vi tester er der også en typisk struktur som skal følges. punkterne for disse er listet herunder.

- Arrange - her opsætter vi stubs til at returner det vi ønsker. - udføres i setup
- Act - får vores UUT til at gøre det vi ønsker - udføres i testmetoden
- Assert - asserter på om vi får vores forventet resultat. - udføres i testmetoden

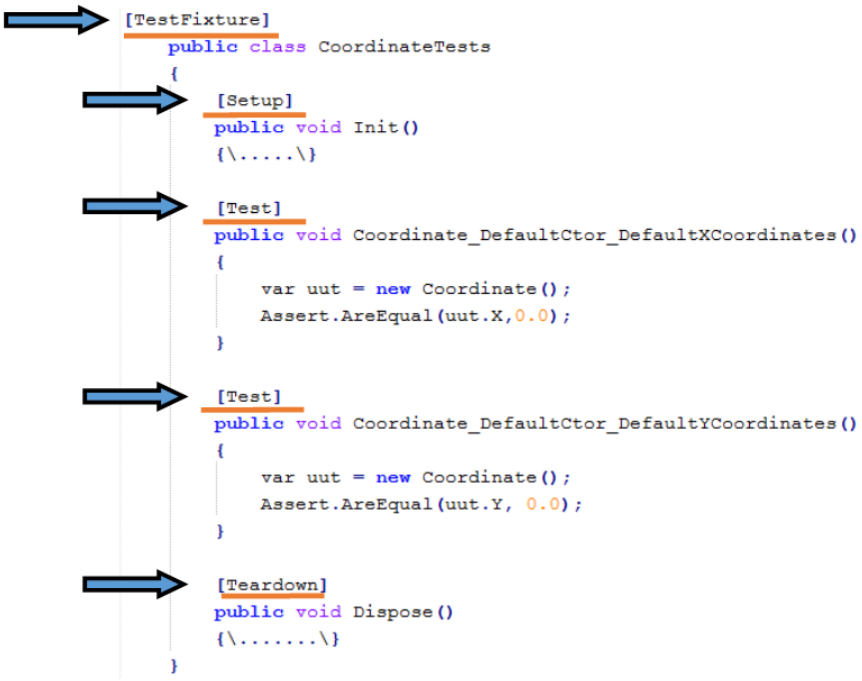

Figure 1: Unittest klasse

3 Classic assert model

Når vi ved det sidste punkt skal assert, er der to typiske måder at gøre dette på. Den første er kaldt "Classic Assert model". Classic assert anvender .AreEqual metoden. Denne tager to parameter, typisk en værdi og metoden der ønskes testet som returnerer en værdi.

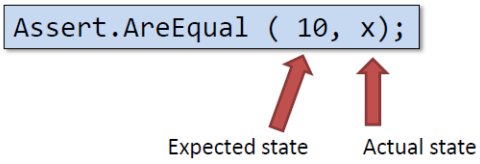

Figure 2: Classic assert model

4 Constriant-based assert model

I en constraint-based model anvedner vi metoden Assert.That. Denne bruger en varabel med den faktiske værdi og metoden Is.EqualTo("Hvad man tror den er equal to")

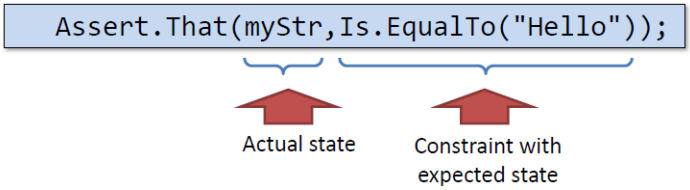

Figure 3: Constraint assert model