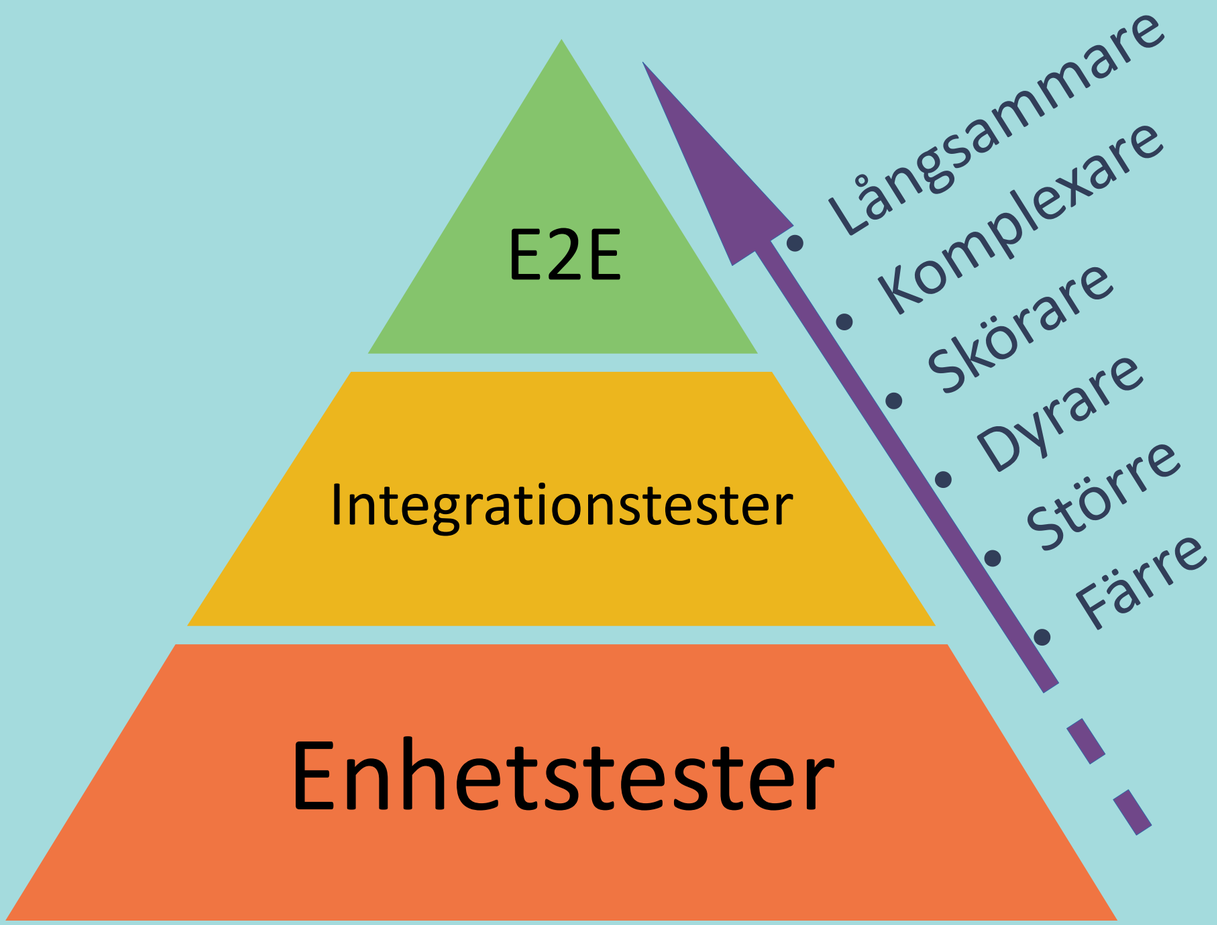
Enhetstester

Automatiska tester

- Billiga att köra
- Snabba att köra
- Konsekventa
- Fel upptäcks snabbare

Typer av automatiska tester



- End-to-endtester (E2E)
  - Testar att helheten fungerar
- Integrationstester
  - Testar att delar fungerar ihop
- Enhetstester
  - Testar att varje del fungerar

MINDFUL STACK

Enhetstester

Det finns ingen exakt definition vad ett enhetstest faktiskt är men generellt

- De är för programmerarens egen skull
- De ska vara snabba
- De är lågnivå och skrivs i samma språk och verktyg som koden

Det finns många olika definitioner av vad som är en enhet

- Inom OOP - oftast en klass, men ibland mer eller mindre
- Inom FP – En funktion eller en samling funktioner
- Du som utvecklare (eller ni som team) avgör vad som passar bäst för just ert system.

Bra artikel av Martin Fowler

MINDFUL STACK

Hur snabba ska enhetstester vara?

Tillräckligt snabba för inte avskräcka dig från att köra dem

Compile suite

- Relevanta snabba tester
- Körs när du tror koden funkar
- Flera gånger per dag
- Max några sekunder

Commit suite

- Alla tester
- Körs innan du commitar koden
- Kan även köras när du tar rast
- Max typ en kvart

Deployment suite

- Riktigt långsamma tester
- Körs i vårt CD-flöde

Läs: [Martin Fowler: Deployment Pipeline]

Enhetstester, fördelar och vinster

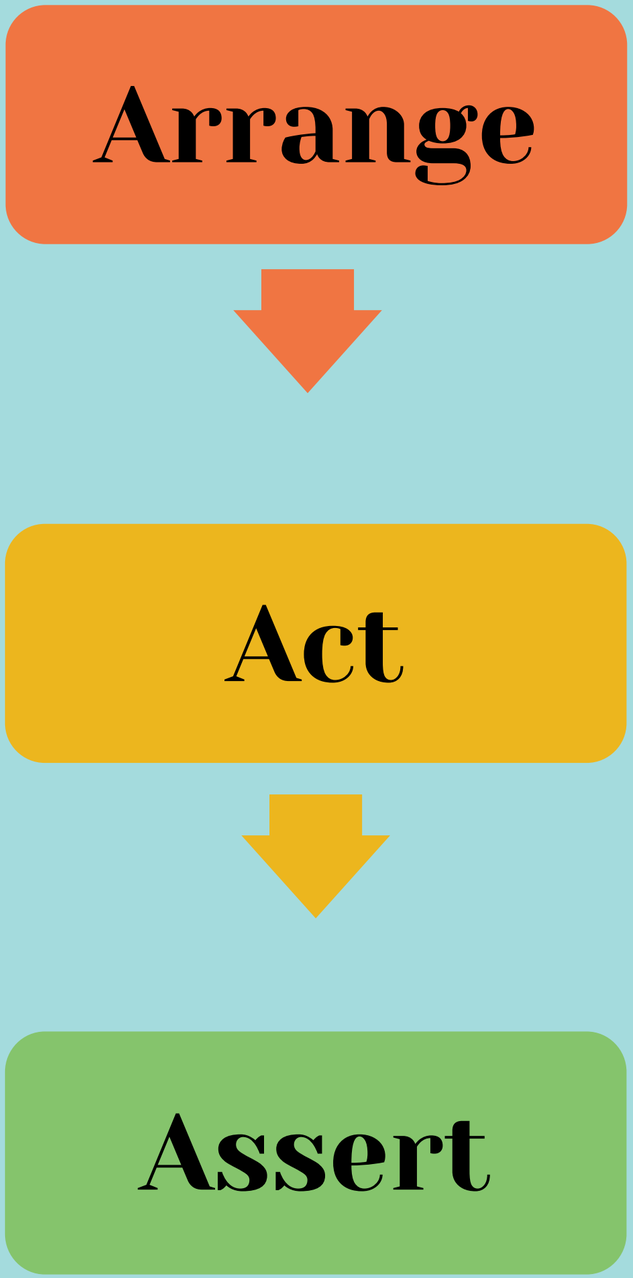
- Hittar felen redan på komponent-nivå
- Sparar tid vid testning
- Ändringar och uppgraderingar sent i utvecklingen blir enklare
- Tester skapar trygghet
- Tester dokumenterar funktionalitet

MINDFUL STACK

Enhetstester, ”nackdelar” och begränsningar

- Tester kan inte hitta alla fel, omöjligt att testa alla scenarion.
- Enhetstester täcker inte allt, krävs även integrationstester m.m.
- Svårt att skriva bra tester, tar tid att lära sig.
- Tar tid att skriva tester
- Kan skapa falsk trygghet
- Mer att skriva om när funktionaliteten ska ändras

MINDFUL STACK



Arrange–Act–Assert (AAA, 3A)

- Arrange - Förbered objektet som ska testas
- Act - Utför testet
- Assert - Kolla om resultatet stämmer med förväntningarna

Kuriosa

- Ibland talas det om ett fjärde steg för att rensa upp efter sig
- Eller en till assert innan act (också kallat affirm)

Läs: xp123.com : arrange-act-assert

MINDFUL STACK

Ramverk

- Slippa boilerplate kod
- Snabba och optimerade
- Mycket hjälp för krångliga saker
- Exempel
  - MS-Test
  - NUnit
  - xUnit

Xunit

- Gratis
- Open source
- Skrivet av skaparen av NUnit v2
- Modern teknik för att göra enhetstester

- Lite dålig dokumentation men mycket annat material

Guide GitHub

MINDFUL STACK

Xunit

Nuget packages:

- xUnit
- xUnit.runner.console
- xunit.runner.visualStudio
- dotnet new xunit

Körs av en test-runner

- VS Test Explorer
- .NET Core CLI
- Third party

Guide GitHub

MINDFUL STACK

```
[Fact]
public void TwoPlusTwoReturnsFour()
{
    // Arrange
    var sut = new Calculator();
    var expected = 4;

    // Act
    var actual = sut.Add(2, 2);

    // Assert
    Assert.Equal(expected,actual);
}
```

Test-struktur

[Fact] – Deklarerar att metoden är ett test

Sut – **S**ystem **U**nder **T**est[1]

Assert – analyserar testet

- Pass/Fail
- Skickar resultat till testrunner
- Oftast vill vi ha en assert per test men ibland 2-3

Läs: xUnit Patterns

MINDFUL STACK

Demo!

- *Skapa ett enhetstest*

MINDFUL STACK

```
Assert.Equal(expected, actual);

Assert.Equal(expected, actual,
precision);

Assert.InRange(actual, min, max);
```

Asserts (numbers)

- Equal
  - Tar valfri parameter “precision” för att udvika floating point errors
- InRange
  - Kontrollerar att värdet är inom ett **inkluderande** intervall

MINDFUL STACK

```
Assert.Equal( expected, actual)

Assert.Equal( expected, actual,
ignoreCase: true);

Assert.Contains(expected, actual);

Assert.StartsWith(expected, actual);

Assert.Matches(pattern, actual);
```

Asserts (strings)

- Equal
  - Tar optional parameter “ignoreCase”
- Contains
- StartsWith
- EndsWith
- Matches (Regex)

MINDFUL STACK

```
Assert.Contains( expected, collection);

Assert.Empty(collection);

Assert.All( list,
 item => Assert.True(item.IsAwesome()));
```

Asserts (Collections)

- Contains
  - DoesNotContain
- Empty
  - NotEmpty
- All

```
Assert.Null(obj);

Assert.True(condition);

Assert.Same(expected, actual);

Assert.IsType<T>(obj);

Assert.Throws<SpectacularCrashException>
( () => sut.Metod());
```

Asserts (Övrigt)

- Null
  - NotNull
- True
  - False
- Same
  - Not same
- IsType<T>
  - IsNotType<T>
- Throws<T>
- Raises<EventArgs>

Övningar

1. Ord- och bokstavsräknare
2. PaketPrisBeräknare

Ord- och bokstavsräknare

Skapa en klass som räknar antalet ord och bokstäver i en sträng, skriv enhetstester med xUnit för att verifiera klassens funktionalitet.

1. Skapa ett nytt C#-konsollprojekt med namnet "WordLetterCounter".
2. Lägg till ett nytt klassbibliotek med namnet "WordLetterCounter.Tests" i samma solution.
3. Lägg till en referens från "WordLetterCounter.Tests" till "WordLetterCounter"-projektet.
4. Installera xUnit NuGet-paketet i "WordLetterCounter.Tests"-projektet.
5. Skapa en ny klass med namnet "TextAnalyzer" i "WordLetterCounter"-projektet med följande metoder:
   - CountWords(string text): Returnerar antalet ord i den angivna strängen.
   - CountLetters(string text): Returnerar antalet bokstäver (exklusive blanksteg och skiljetecken) i den angivna strängen.
6. Implementera metoderna i "TextAnalyzer"-klassen.
   - Tips: Använd string.Split()-metoden för att dela upp strängen i ord baserat på blanksteg.
   - Tips: Använd en loop för att iterera över varje tecken i strängen och räkna bokstäverna.
7. Skapa en ny klass med namnet "TextAnalyzerTests" i "WordLetterCounter.Tests"-projektet.
8. Skriv följande enhetstester i "TextAnalyzerTests"-klassen:
   - TestCountWords: Verifiera att CountWords-metoden returnerar korrekt antal ord för olika indata.
   - TestCountLetters: Verifiera att CountLetters-metoden returnerar korrekt antal bokstäver för olika indata.
   - TestEmptyString: Verifiera att både CountWords och CountLetters returnerar 0 för en tom sträng.

PaketPrisBeräknare

Skapa ett paket-hanteringsprogram

Programmet ska hantera två typer av försändelser:

- Paket (rätblock)
- Rör (cylindrar)

Användaren matar in dimensioner och vikt.
Programmet räknar ut volym och pris.

Maximal tillåten vikt är 20 kg.
Om användaren matar in en vikt under 2 kg, används 2 kg vid beräkning.

Alla beräkningsfunktioner ska vara enhetstestade med xUnit.

Prisberäkning:

Cylinder:
Pris = omkrets (cm) x längd (cm) x vikt (kg).
Minsta vikt vid beräkning är 2 kg.
Resultatet anges i öre

Paket:
Kortaste sidan (cm) x längsta sidan (cm) x vikt (kg) + 10 000
Resultatet anges i öre

Specialpris för små paket:
Om längsta sidan < 30 cm gäller fasta priser:
upp till 2 kg => 29 kr,
och 2-10 kg, => 49 kr,
och 10-20 kg, => 79 kr