Dia 1: Introducció

i

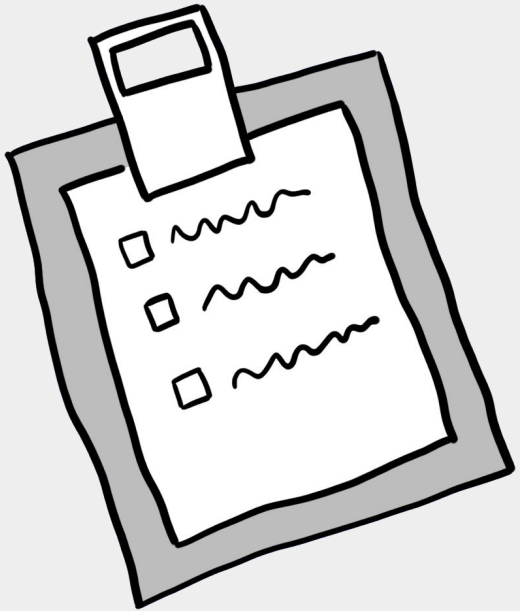
Tests unitaris

Xavier Sala Pujolar

Universitat de Girona

Febrer 2021

Tests



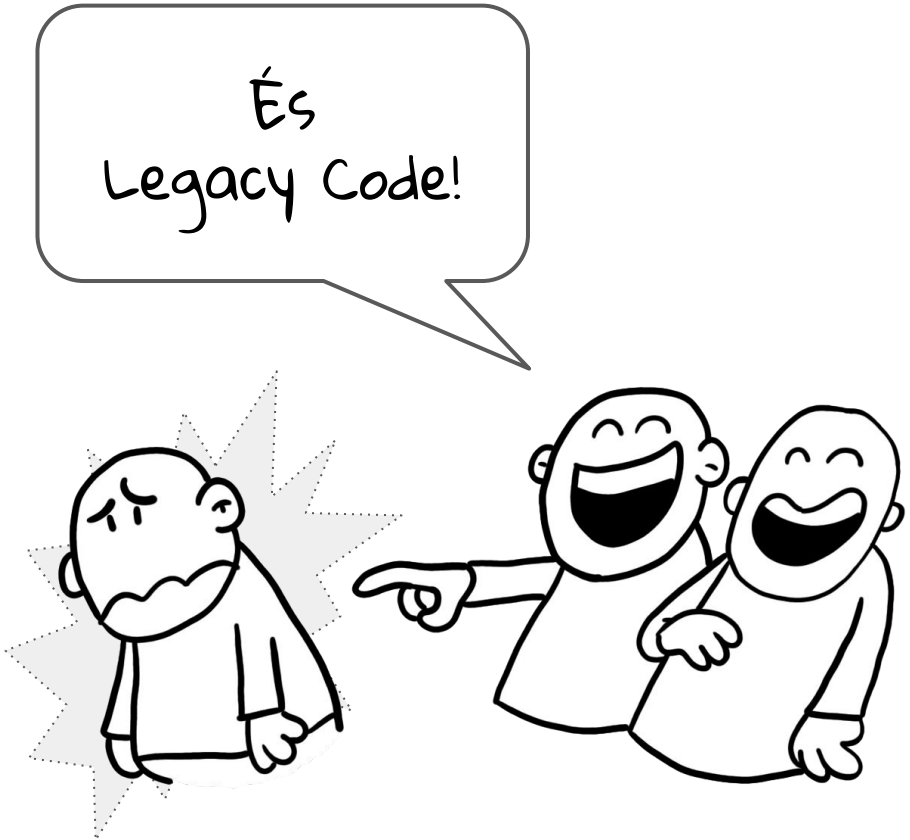
És molt difícil fer programes complexos que no tinguin cap error.

És molt important que els errors es detectin tan aviat com sigui possible i sobretot que no es repeteixin

Els testos formen part de la qualitat en el codi

El codi sense tests és mal codi. No importa que estigui molt ben escrit.

Michael Feathers (2004)



Has d'escriure molt més codi

Poden fer-te canviar el disseny

L'execució dels tests pot ser lenta

A vegades els tests tenen més codi que el programa

Incrementen la confiança

Detecció precoç de bugs

Permeten detectar errors en la refactorització

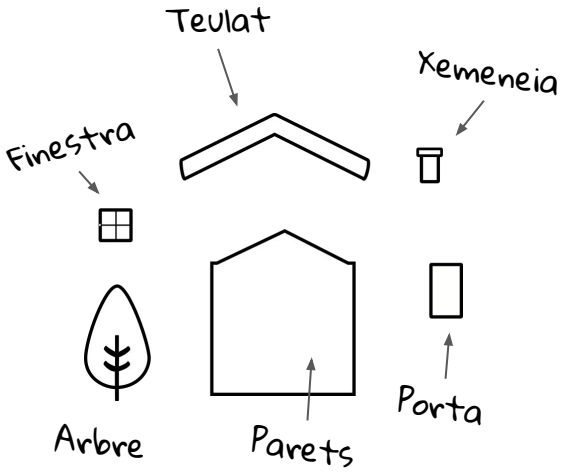
Es poden fer servir de documentació

Si el codi és testable, és de més qualitat

Tipus de tests

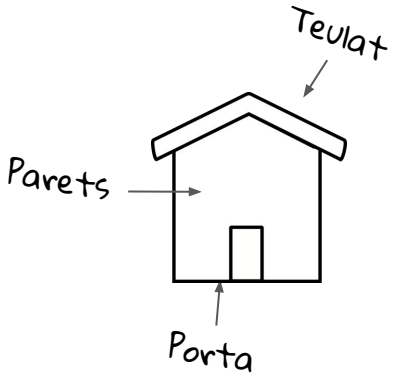
Tests Unitaris

Provar una sola classe/funcionalitat



Tests d'Integració

Provar diferents mòduls alhora

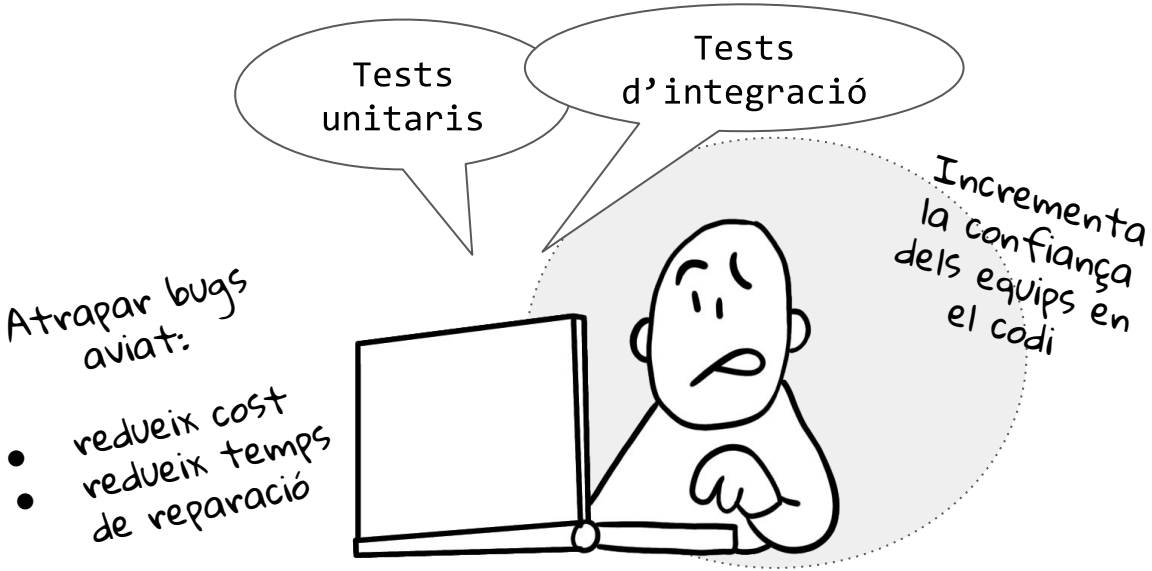


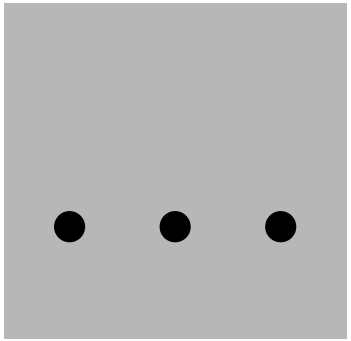
Tests end-to-end

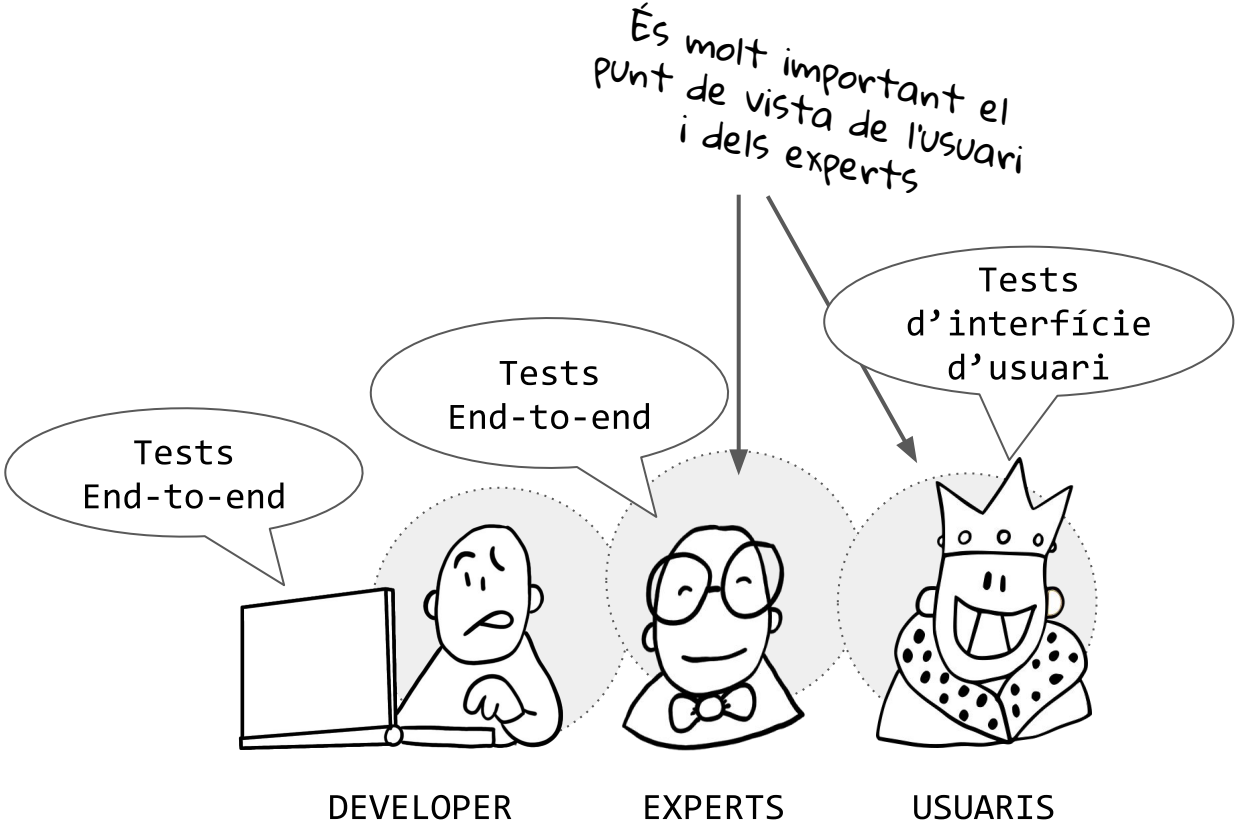
Provar l'aplicació des del punt de vista de l'usuari

Els han de fer els desenvolupadors?

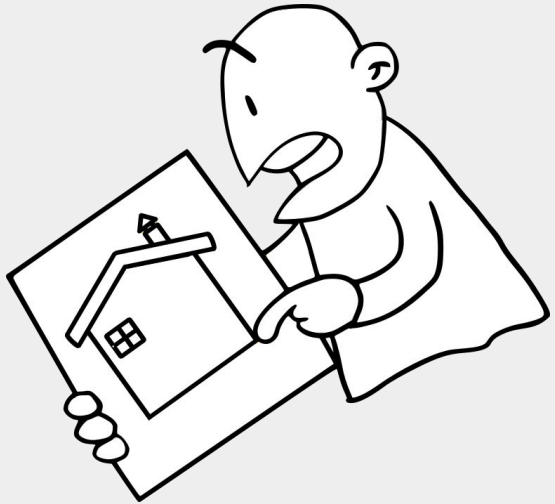
SI






És molt important el
punt de vista de l'usuari
i dels experts
Tests
End-to-end
Tests
End-to-end
Tests
d'interfície
d'usuari
DEVELOPER
EXPERTS
USUARIS

Estructura d'un test



Estructura d'un test?

Es pot dividir un test en tres parts

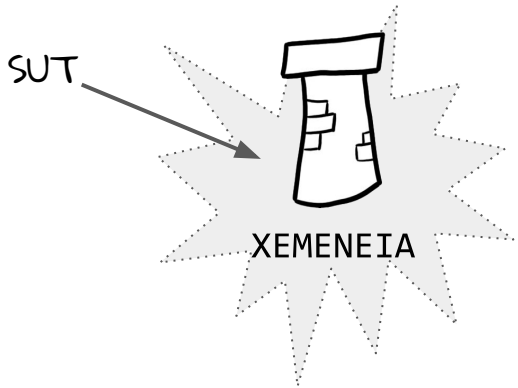
- Arrange / Given
- Act / When
- Assert / Then

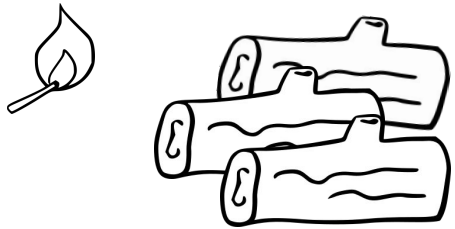
3 Fases

- Arrange / Given
- Act / When
- Assert / Then

Preparar tot el que li cal al test

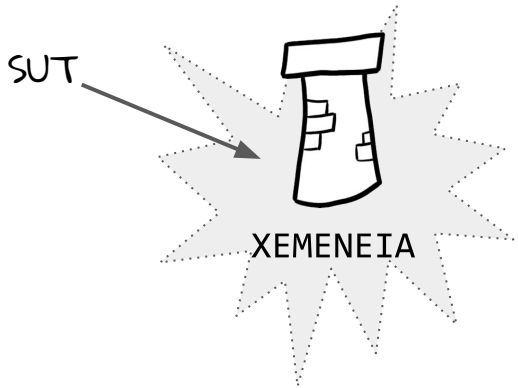
Assegurar-se de que cada cop les condicions són les mateixes

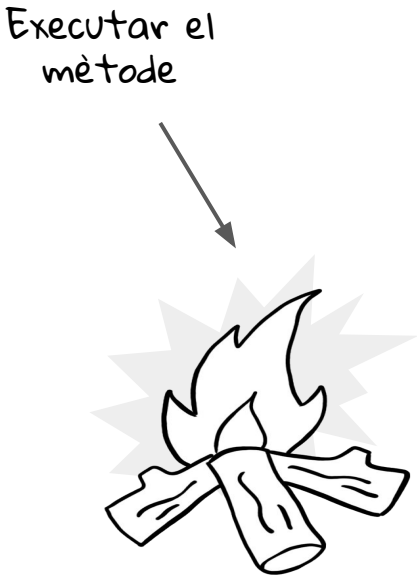




3 Fases

- Arrange / Given
- Act / When
- Assert / Then



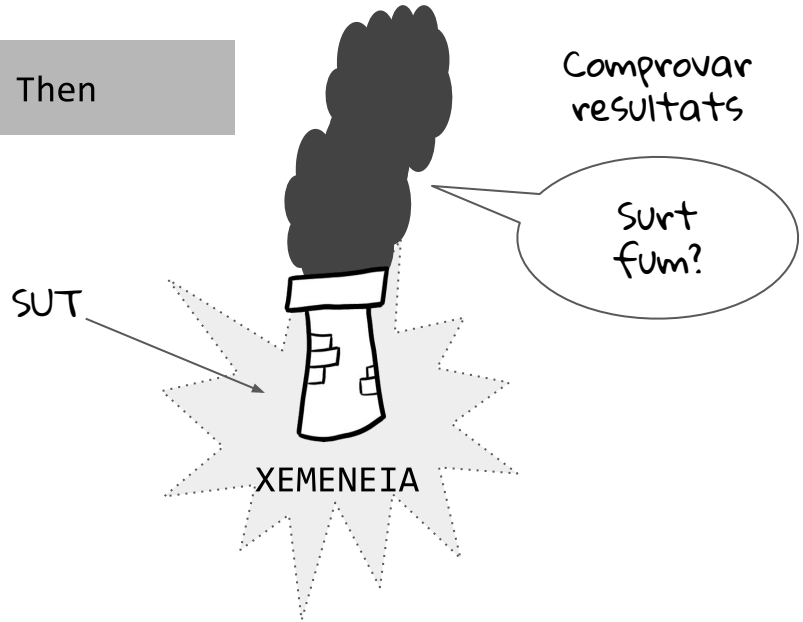


3 Fases

Arrange / Given

Act / When

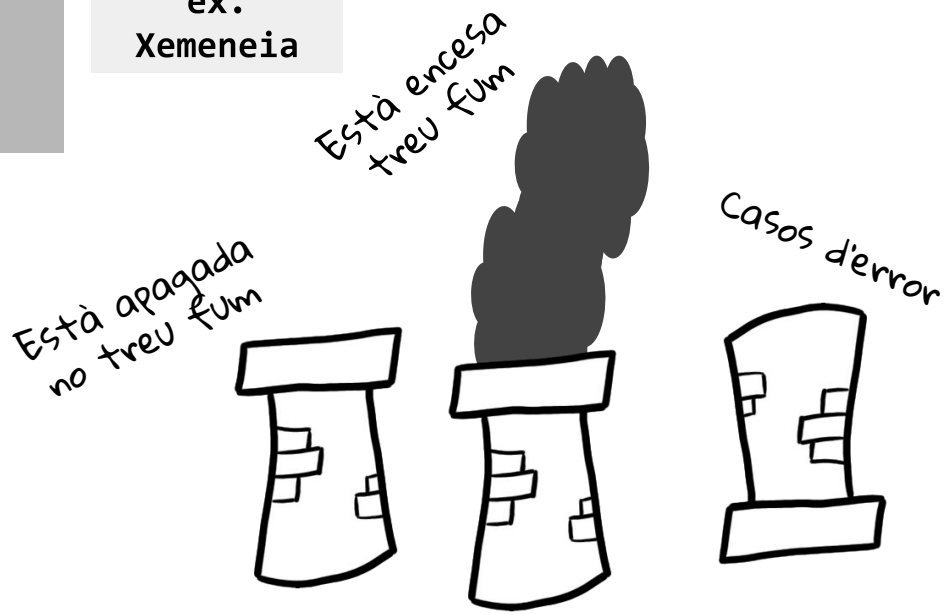
Assert / Then



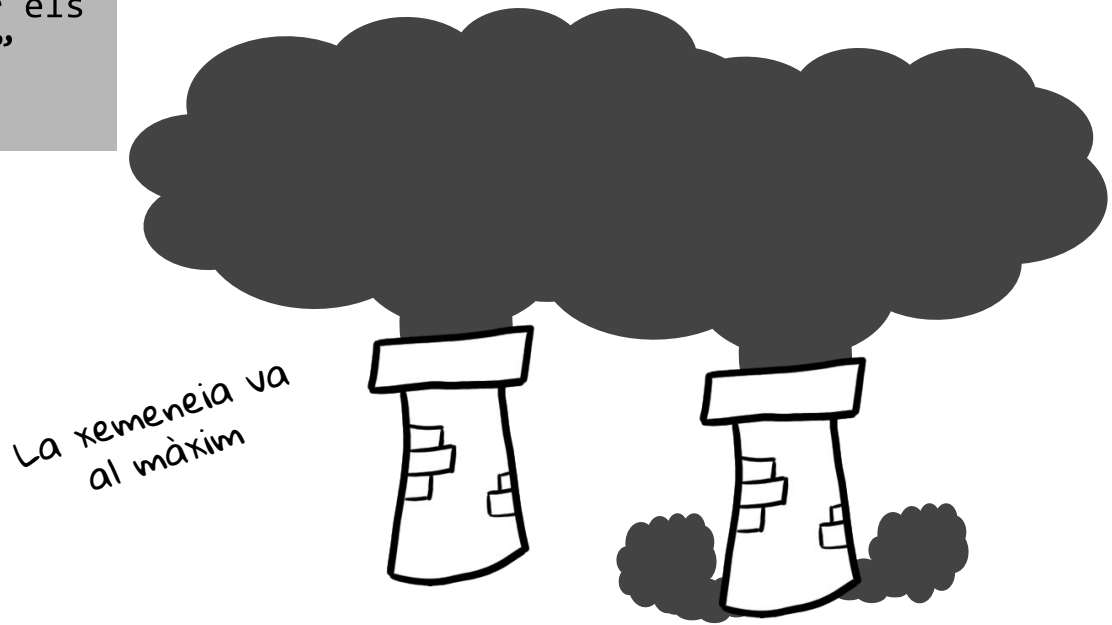
Què comprovar en un test?

S'han de provar els casos "**normals**"

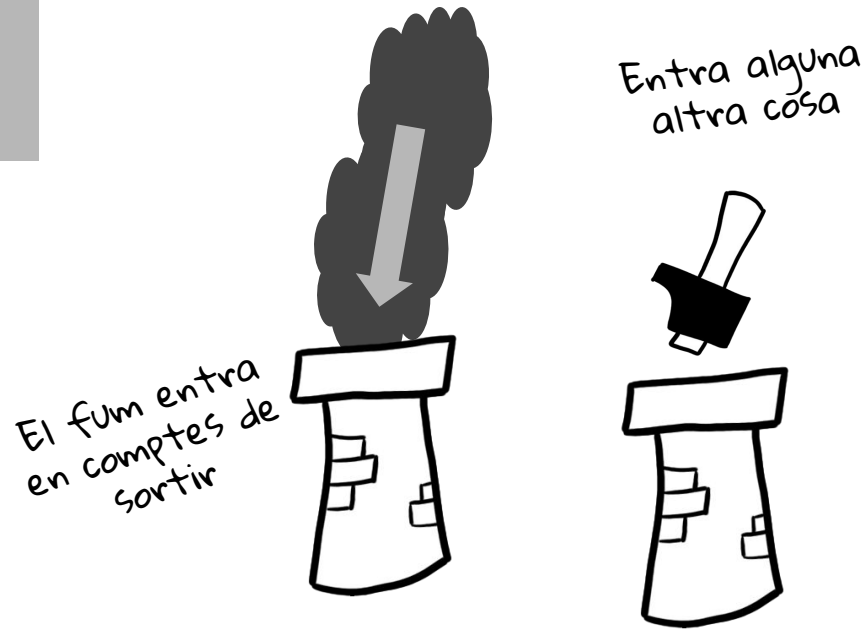
ex. Xemeneia



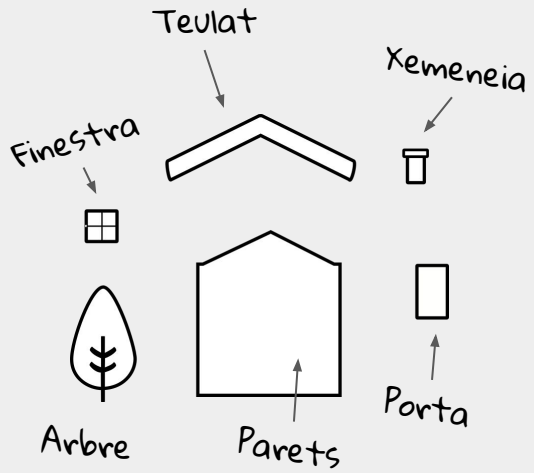
S’han de comprovar els casos **“extrems”**



S’ha de provar com funciona en casos **“estranys”**



Tests unitaris



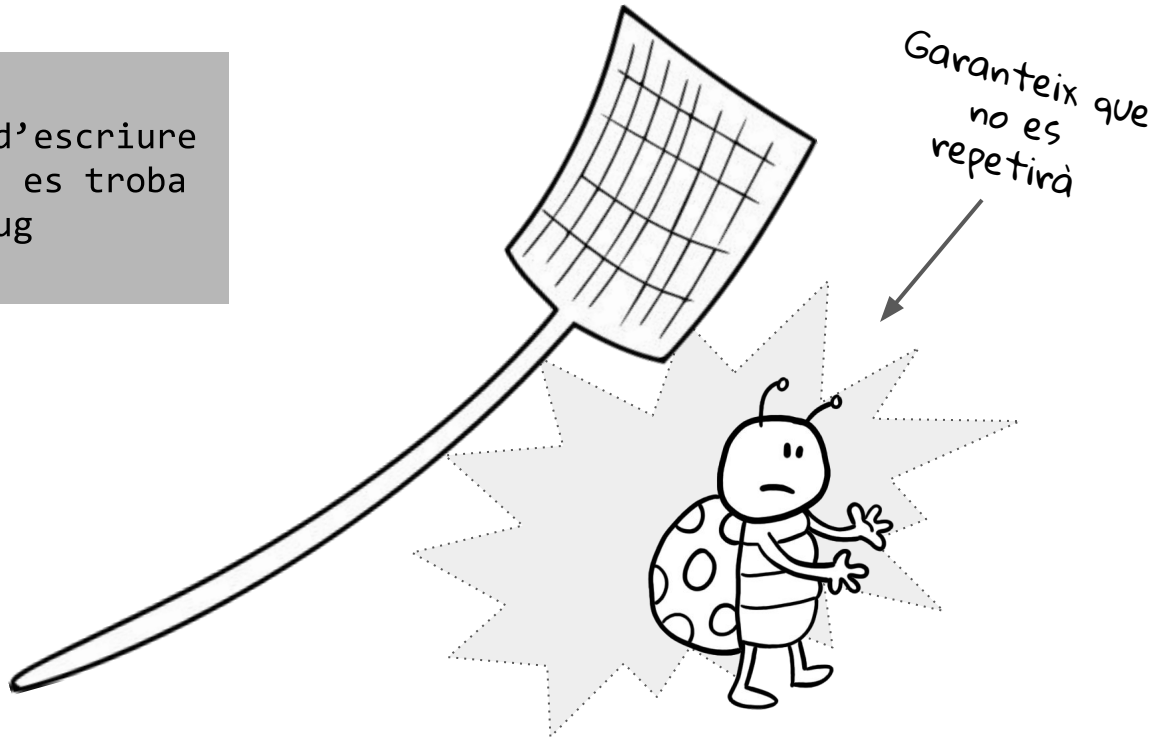
Es prova de forma aïllada una sola classe del sistema que s'està desenvolupant

S'haurien de provar totes les possibilitats

L'objectiu és assegurar-se de que la classe sempre funciona correctament

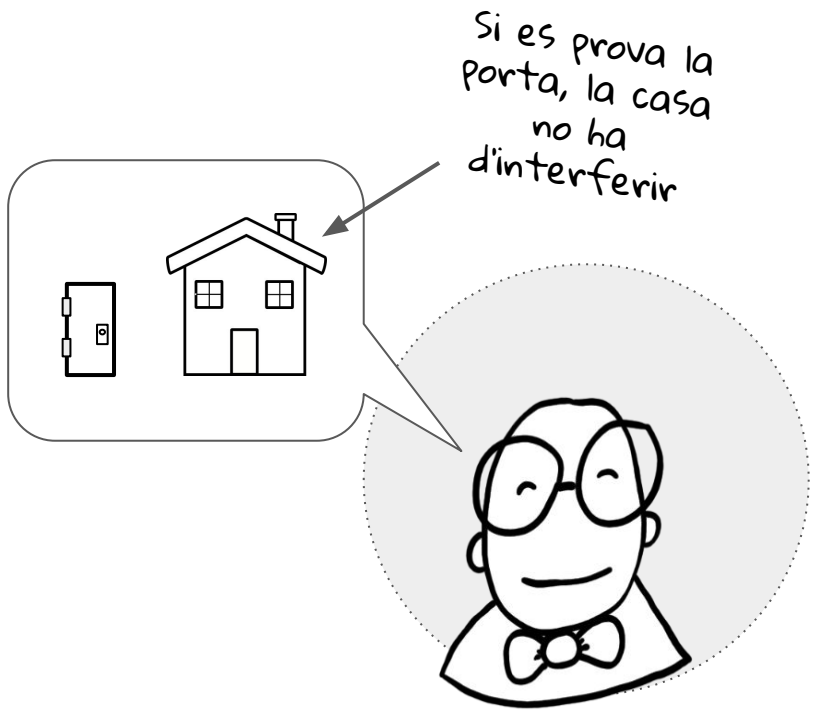
1. Han de ser ràpids
2. Els tests unitaris s’han de fer de forma aïllada
3. Han de comprovar una sola funcionalitat
4. Han de ser repetibles

SEMPRE s'ha d'escriure un test quan es troba un bug



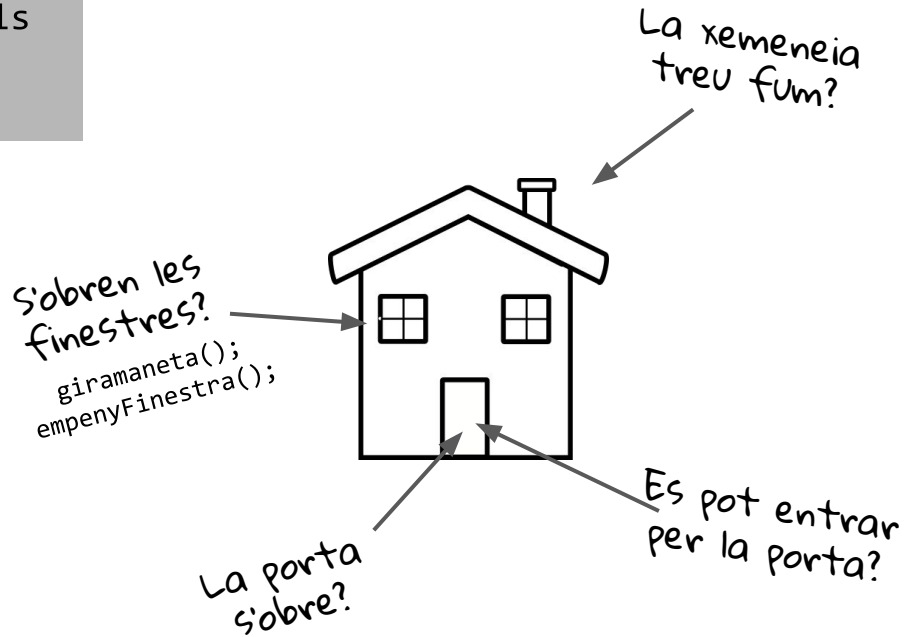
Els tests han de comprovar de forma aïllada

Els tests unitaris s'han de fer de forma aïllada



Provar una sola funcionalitat

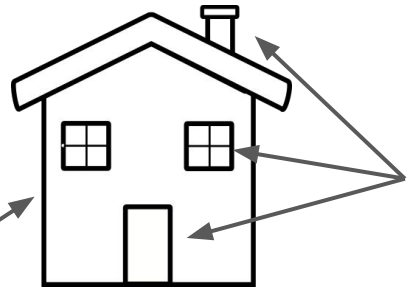
Un test ha de provar **funcionalitat** no els mètodes



No cal fer tests dels mètodes que no tenen un mínim de funcionalitat

Hi han diverses opinions sobre aquest tema..

Sovint les classes
en contenen
d'altres



Col·laboradors

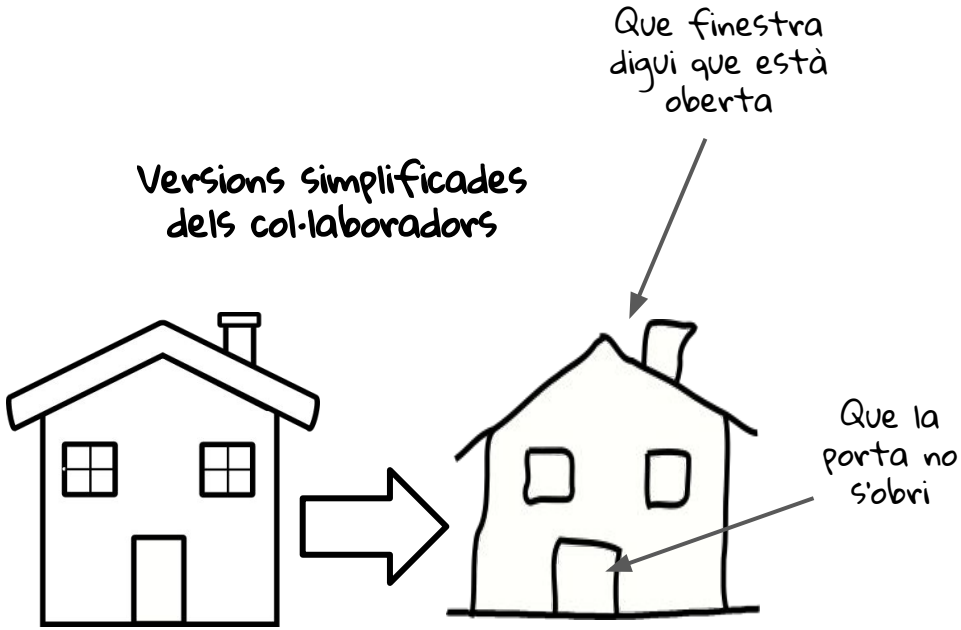
Xemeneia
Finestra
Porta

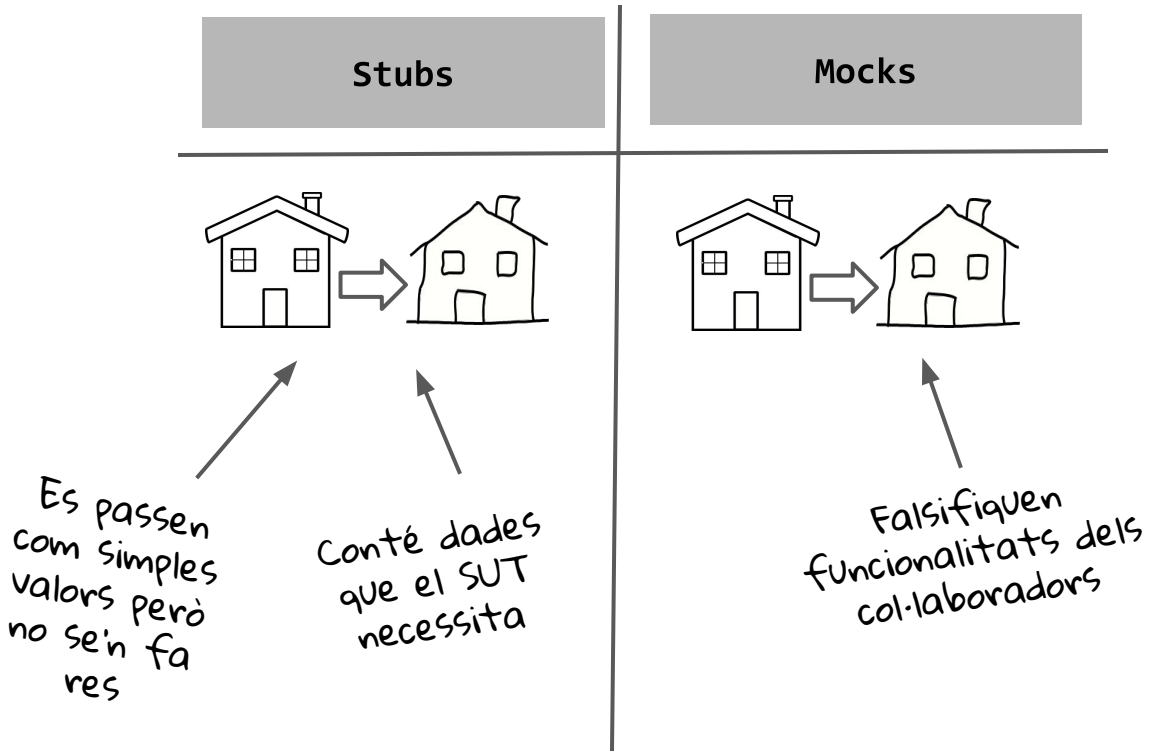
SUT
(system under test)

Casa

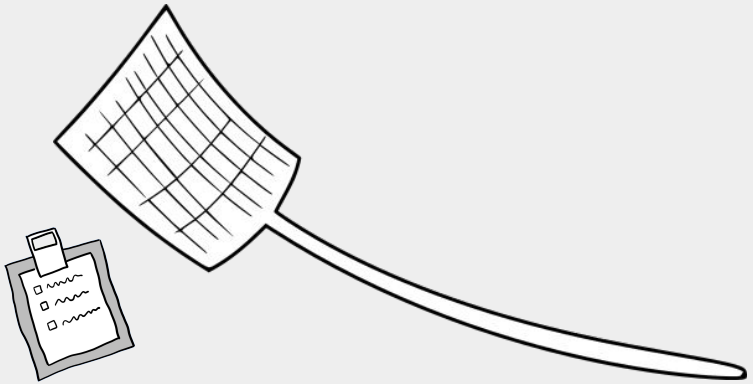
Creació de dobles per aïllar tests

Per poder-les comprovar de forma aïllada es poden crear **dobles**




Stubs
Mocks
Es passen com simples valors però no se'n fa res
Conté dades que el SUT necessita
Falsifiquen funcionalitats dels col·laboradors

Eines per fer tests

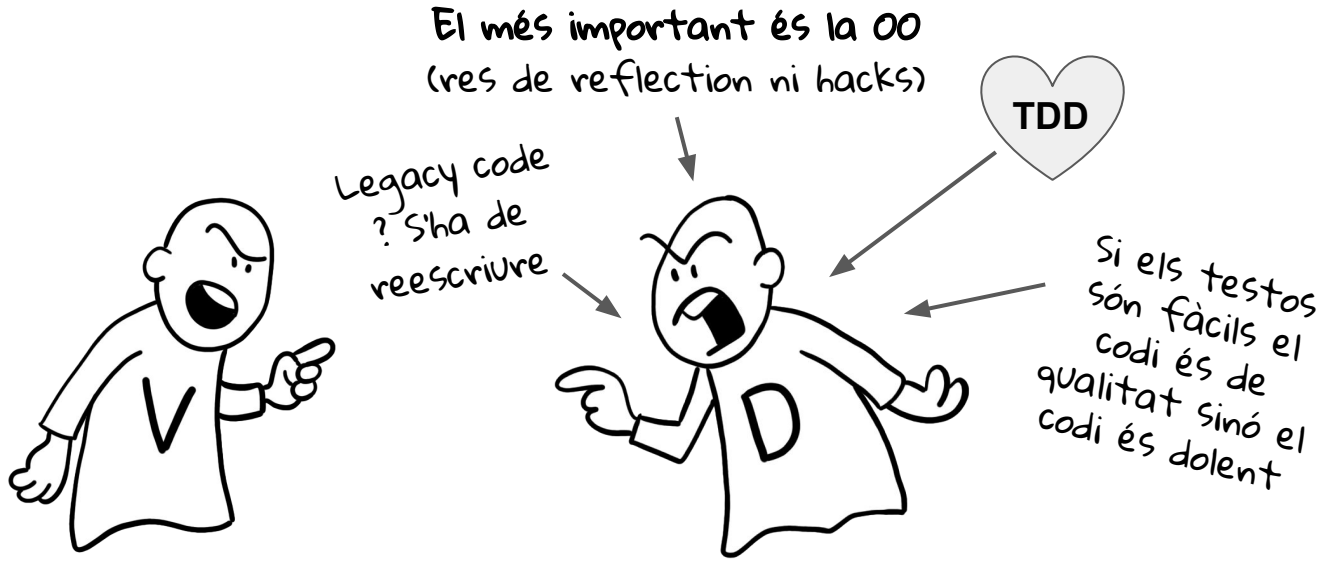


Tipus de Testers

Verificadors

Dissenyadors

Tipus de testers

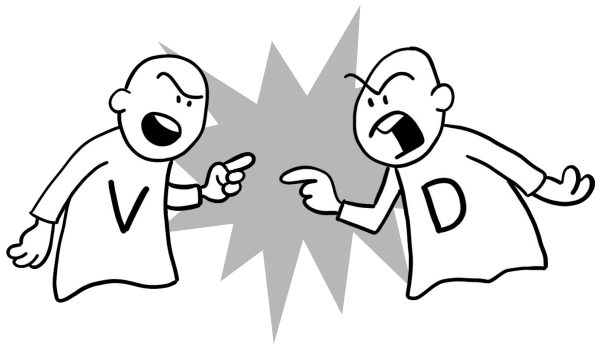


Verificadors

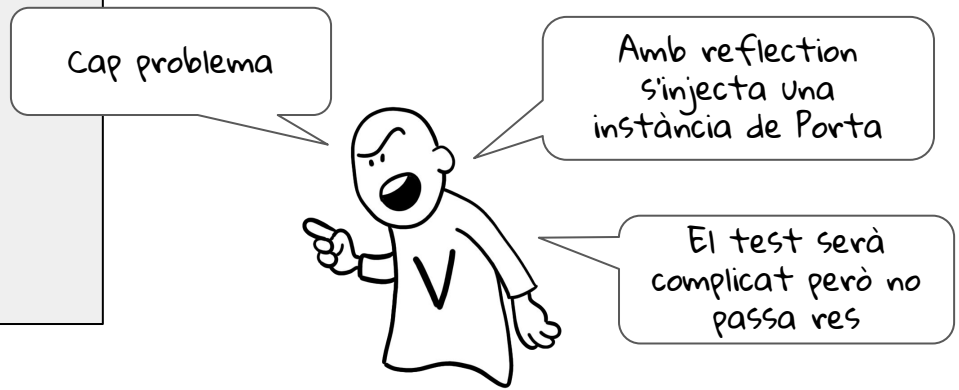
Dissenyadors

```
public class Casa {

   private Porta _porta;

   public Casa() {
       _porta = new Porta();
    }

  public Entra() {
     _porta.obrePorta();
     dins = true;
     _porta.tancaPorta();
  }
}
```

Creació de objectes privats dins d'una classe

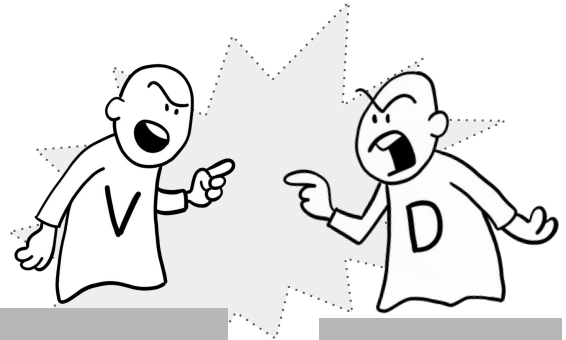


```
public class Casa {

   private Porta _porta;

   public Casa() {
       _porta = new Porta();
    }

  public Entra() {
     _porta.obrePorta();
     dins = true;
     _porta.tancaPorta();
  }
}
```



```
public class Casa {

   private Porta _porta;

   public Casa() {
       _porta = new Porta();
    }

  public Entra() {
     _porta.obrePorta();
     dins = true;
     _porta.tancaPorta();
  }
}
```

Les eines reflecteixen quin tipus de tester és el que les ha fet



PowerMock

Manipula el bytecode per permetre saltar-se l'encapsulament

JUnit

No fa fàcil provar mètodes privats o estàtics

Vols fer un ~~puto~~ exemple?

P1: Per fi anem per feina

Cistella compra online

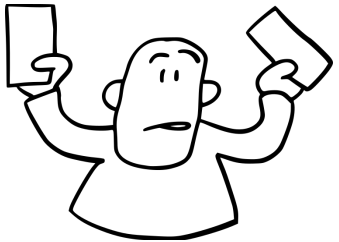
- S’han de poder **afegir** i **treure** quantitats de productes i **buidar** la cistella
- Ha de saber la quantitat d’articles
- Hi haurà un preu de **transport base**
  - El preu del transport s’incrementarà en ***1€ per cada 5 Kg*** *de pes*
  - Si el total és ***superior a 50€*** el transport és **gratuït**
  - Si es compren ***més de quatre*** unitats del mateix producte es **rebaixa un 5% en el preu del producte**
  - Els **usuaris VIP** tenen transport gratuït
- Ha de calcular el total a pagar amb transport inclòs

Resum de les
eines que usarem
en les pràctiques

2021

Frameworks de tests

Proporcionen comandes específiques per fer comparacions



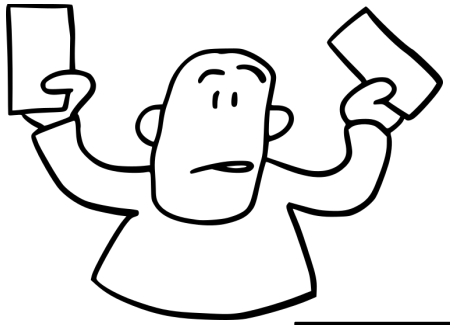
xUnit.net

nunit

MSTest

Són els encarregats de comprovar els resultats

- Assert.Equal
- Assert.InRange
- Assert.True
- Assert.Null
- Assert.Contains
- Assert.Same
- Assert.Empty
- Assert.IsType



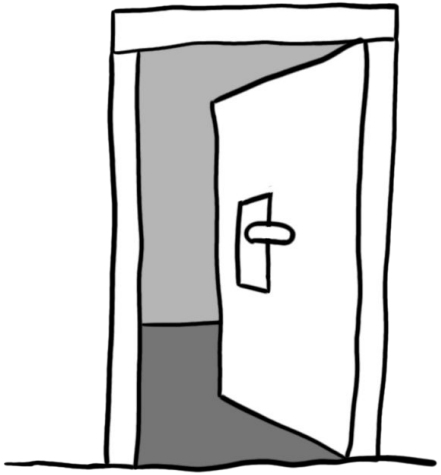
Comprovacions a través d'Asserts

xUnit.net

```
Assert.Equal(expected, result)
```

```
Assert.True(result)
```

```
Assert.Assert.Throws<Exception>(
      () => throw new Exception("name"));
```



SUT: Porta

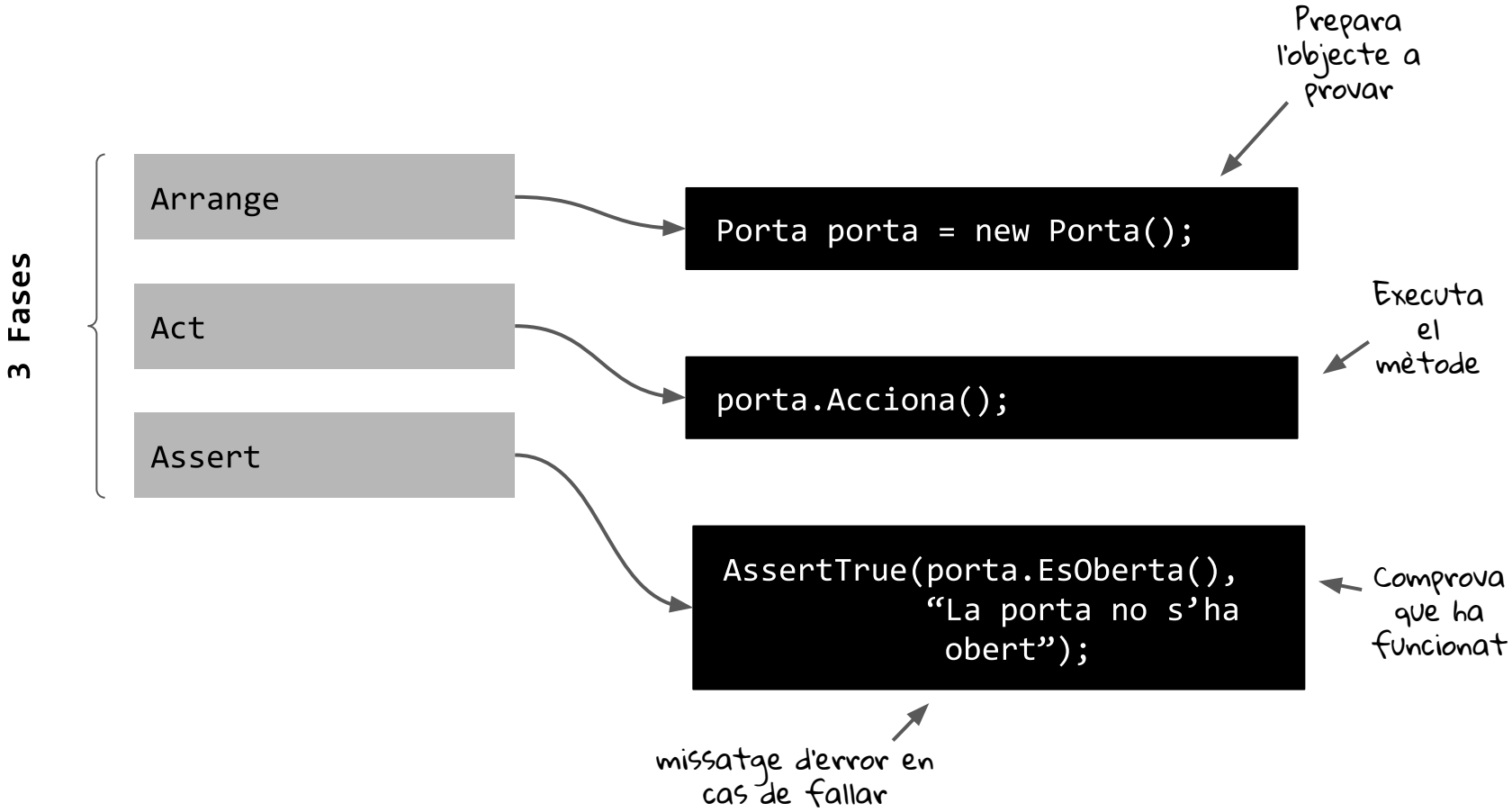
Funcionalitats:

- La porta s'obre i es tanca

```
class Porta {
    private bool _esOberta = false;

    public void Acciona() {
        _esOberta = !_esOberta;
    }

    public bool EsOberta() {
         return false;
    }
}
```

xUnit.net

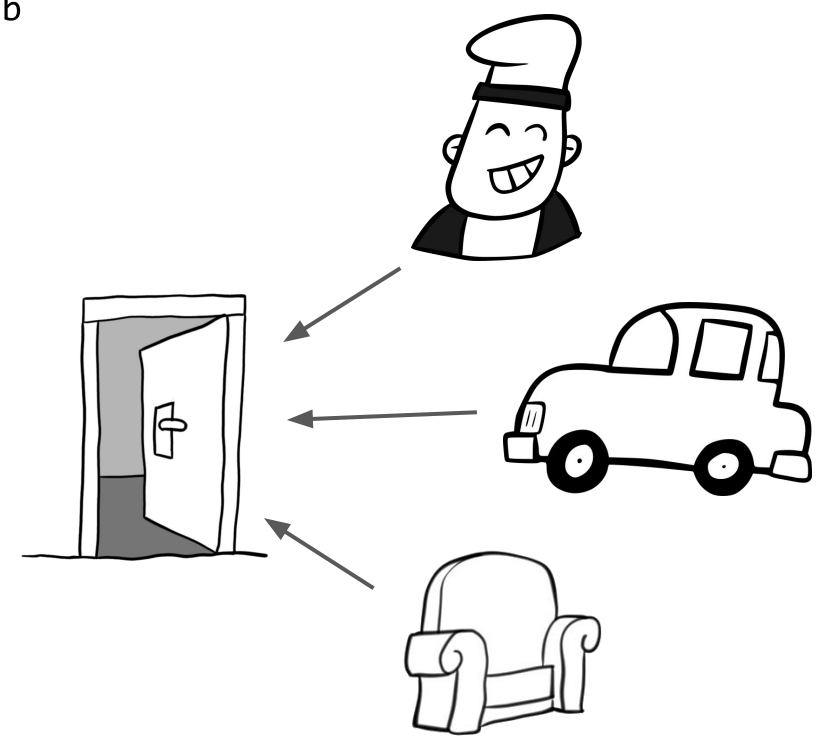
```
[Fact]
public void TestSiLaPortaObre() {
   Porta porta = new Porta();


   porta.Acciona();

   AssertTrue(porta.EsOberta(),
      “La porta no s’ha obert”);
 }
```

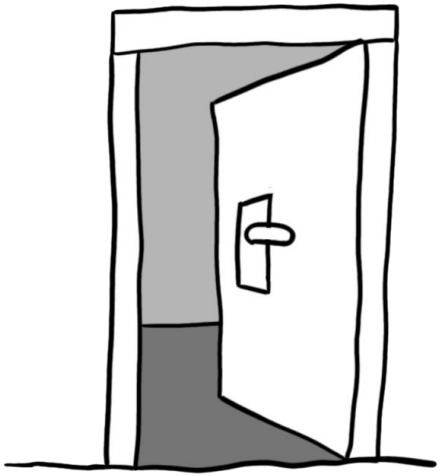

3 Fases
Arrange
Act
Assert
Porta porta = new Porta();
porta.Acciona();
AssertTrue(porta.EsOberta(),
"La porta no s'ha
obert");
Prepara
l'objecte a
provar
Executa
el
mètode
Comprova
que ha
funcionat
missatge d'error en
cas de fallar

P2: Tests parametritzats

Tests parametritzats

Sovint cal repetir una prova amb dades diferents per provar una determinada funcionalitat





SUT: Porta

```
[Theory]
[InlineData(true, false)]
[InlineData(false, true)]
public void TestSiLaPortaObre(
    bool estatPorta,
    bool esperat)
{
   Porta porta = new Porta(estatPorta);

   porta.Acciona();

   AssertTrue(porta.EsOberta() == esperat,
      “La porta no funciona bé”);
 }
```

xUnit.net

```
[Theory]
[InlineData(true, false)]
[InlineData(false, true)]
public void TestSiLaPortaObre(
    bool estatPorta,
    bool esperat)
{
   Porta porta = new Porta(estatPorta);

   porta.Acciona();

   AssertTrue(porta.EsOberta() == esperat,
      “La porta no funciona bé”);
 }
```

P3: Eines de matching

Fa que les comprovacions dels tests semblin més “naturals” i fàcils d’entendre

Simplifica algunes comprovacions

És extensible

xUnit.net


```
[Fact]
public void TestSiLaPortaObre() {
   Porta porta = new Porta();


   porta.Acciona();

   porta.EsOberta().Should().Be(true),
      “La porta no s’ha obert”);
 }
```

xUnit.net


```
[Theory]
[InlineData(true, false)]
[InlineData(false, true)]
public void TestSiLaPortaObre(
    bool estatPorta,
    bool esperat)
{
   Porta porta = new Porta(estatPorta);

   porta.Acciona();

   porta.EsOberta().Should().Be(esperat,
      “La porta no funciona bé”);
 }
```

P4: Classes composades

```
public class Casa {
   private Porta porta;
   private int gent;

   public Casa(Porta p) {
       porta = p;
       gent = 0;
    }

  public bool Entra() {
     int abans = gent;
     if (porta.IsOberta()) {
        gent++;
     }
     return gent != abans;
  }
}
```

Provar la casa de forma aïllada

Es defineix una porta falsa

```
Porta falsa = mock(Porta.class);
```

```
when(falsa.IsOberta())
       .thenReturn(false);
```

A la que li podem dir què volem que retorni

```
@Test
public void SiLaPortaEsTancadaNoEntra() {

    // Arrange
    Porta falsa = mock(Porta.class);
   when(falsa.IsOberta())
         .thenReturn(false);
    Casa = new Casa(falsa);

    // Act
    int resultat = casa.Entra();

   // Assert
   assertFalse(resultat);
}
```

Crea la Casa amb la porta falsa

El resultat ha de ser l'esperat

2021