

# **Dia 1: Introducció i Tests unitaris**

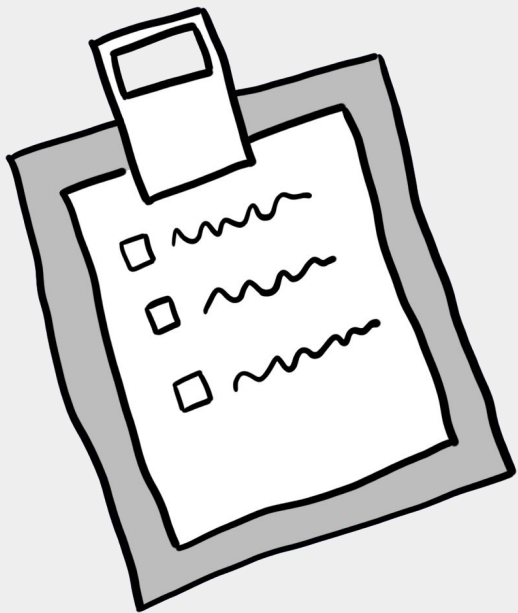
Xavier Sala Pujolar



**Universitat  
de Girona**

Febrer 2021

# Tests



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

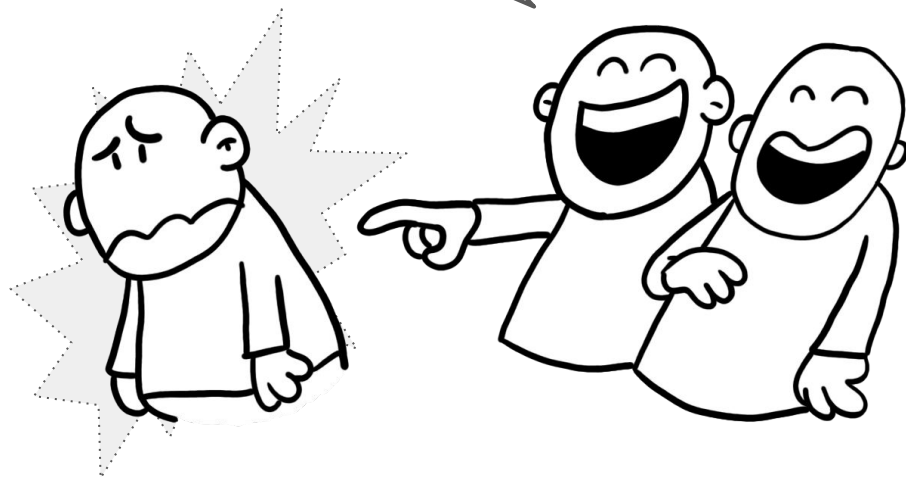
És molt important que els errors es detectin tan aviat com sigui possible i sobretot que no es repeteixin

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)

És  
Legacy Code!



L'execució dels  
tests pot ser  
lenta

Has d'escriure  
molt més codi

Poden fer-te  
canviar el disseny

A vegades els tests  
tenen més codi que el  
programa

Detecció precoç  
de bugs

Incrementen  
la confiança

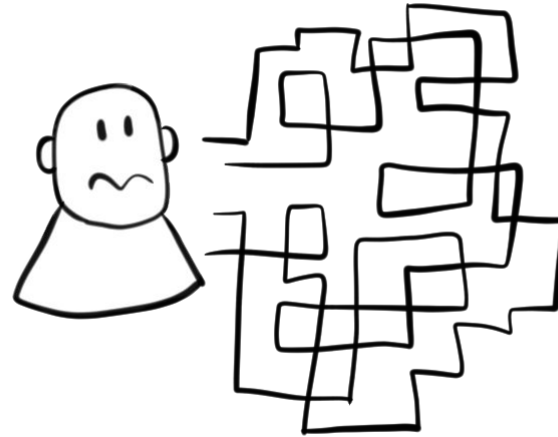
Es poden fer  
servir de  
documentació

Si el codi és  
testable, és de  
més qualitat

Permeten  
detectar errors  
en la  
refactorització

El codi **tendeix a**  
**deteriorar-se** a mesura  
que el codi canvia

Els tests permeten  
fer créixer el projecte  
de forma sostenible

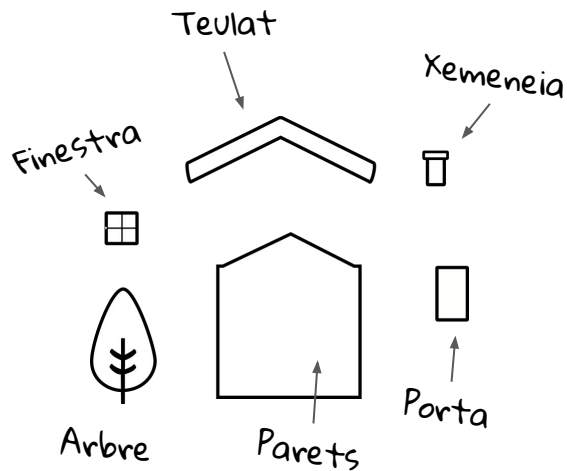


# Tipus de tests

---

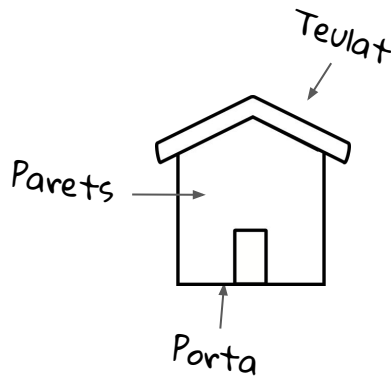
## Tests Unitaris

Provar una sola classe/funcionalitat



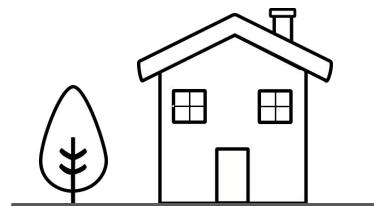
## Tests d'Integració

Provar diferents mòduls alhora



## Tests end-to-end

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



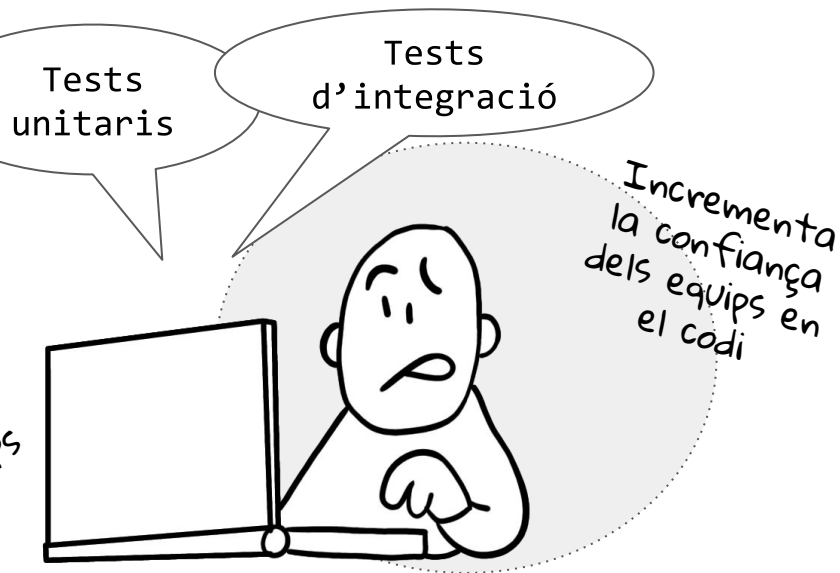
# Els han de fer els desenvolupadors?

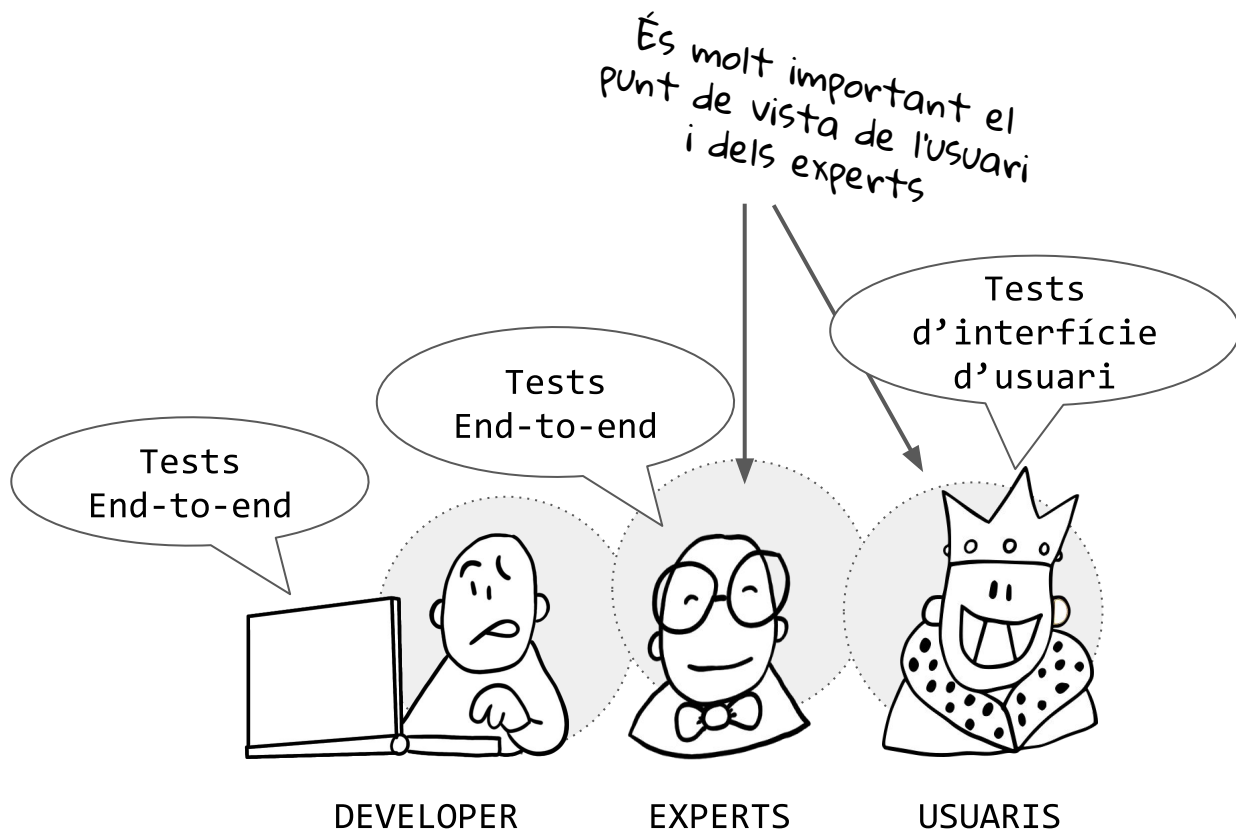
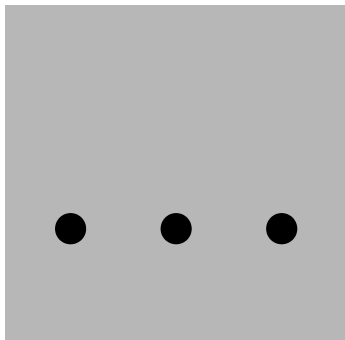
---

SI

Atrapar bugs  
aviat:

- redueix cost
- redueix temps de reparació







# Estructura d'un test

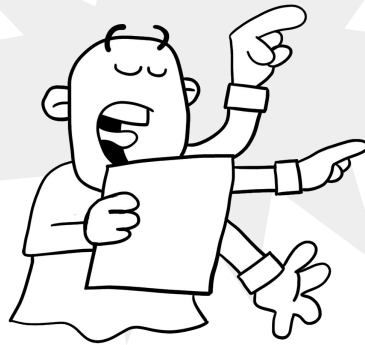


Han de ser fàcils  
d'entendre

Evitar els if

Han de ser ràpids

Han d'estar  
integrats en el  
cicle de  
desenvolupament



# Estructura d'un test?

---

Es pot dividir un  
test en tres parts



Arrange / Given

Act / When

Assert / Then

### 3 Fases

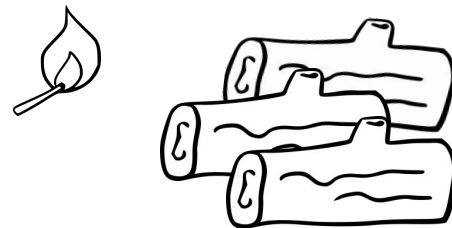
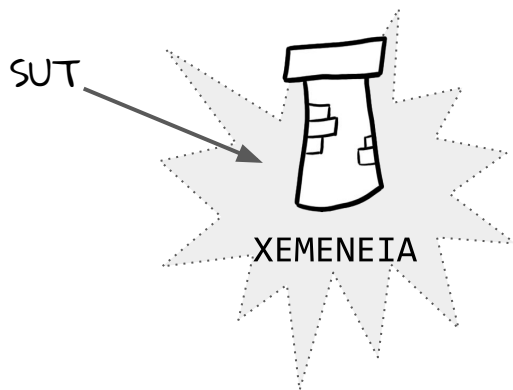
Arrange / Given

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

SUT



XEMENEIA

Executar el  
mètode

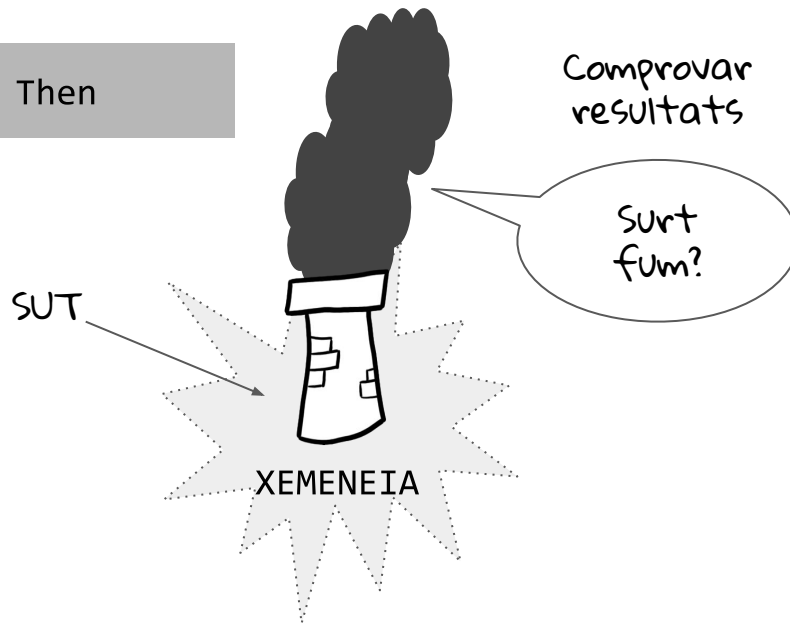


### 3 Fases

Arrange / Given

Act / When

Assert / Then



# Què comprovar en un test?

S'han de provar els casos "normals"

ex.  
Xemeneia

Està apagada  
no treu fum



Està encesa  
treu fum



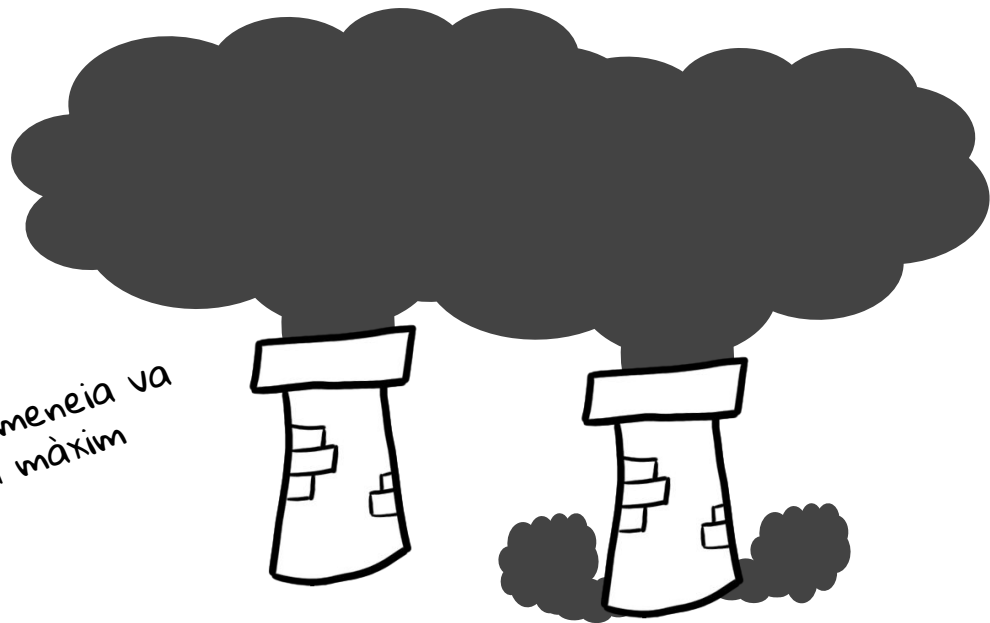
Casos d'error



S'han de comprovar els  
casos "extrems"

La xemeneia va  
una mica més del  
màxim

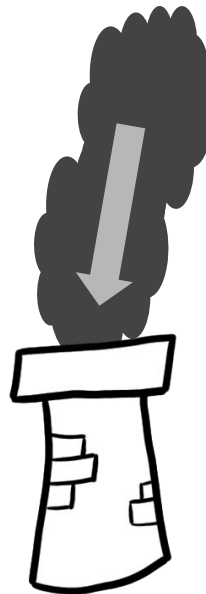
La xemeneia va  
al màxim





S'ha de provar com  
funciona en casos  
“estranys”

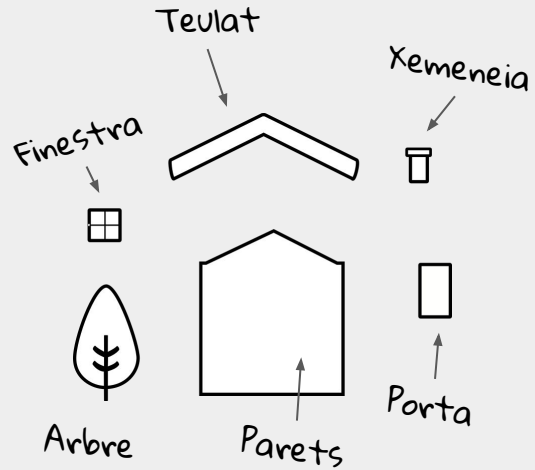
El fum entra  
en comptes de  
sortir



Entra alguna  
altra cosa



# 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

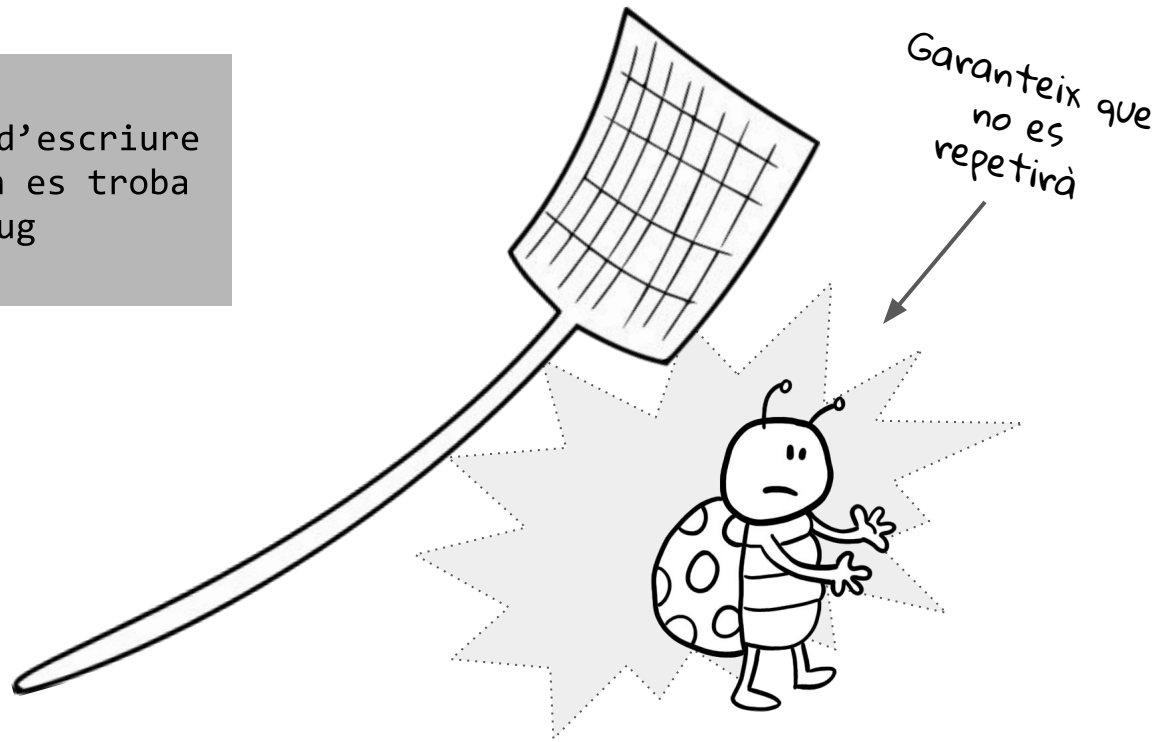
3

Han de comprovar una  
sola funcionalitat

4

Han de ser repetibles

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

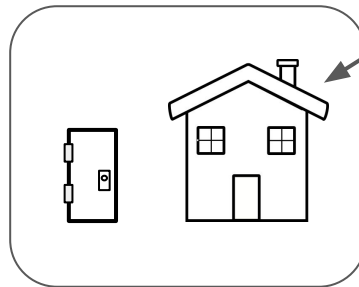


Garanteix que  
no es  
repetirà

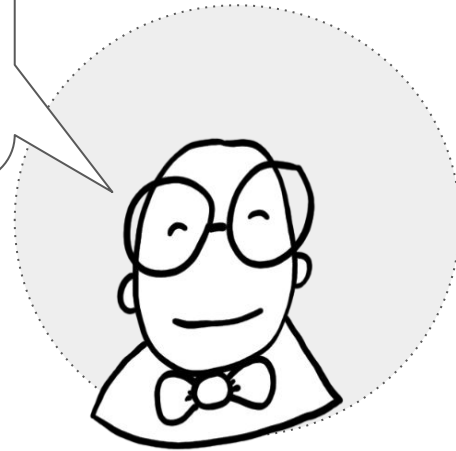
# Els tests han de comprovar de forma aïllada

---

Els tests unitaris  
s'han de fer de  
forma aïllada



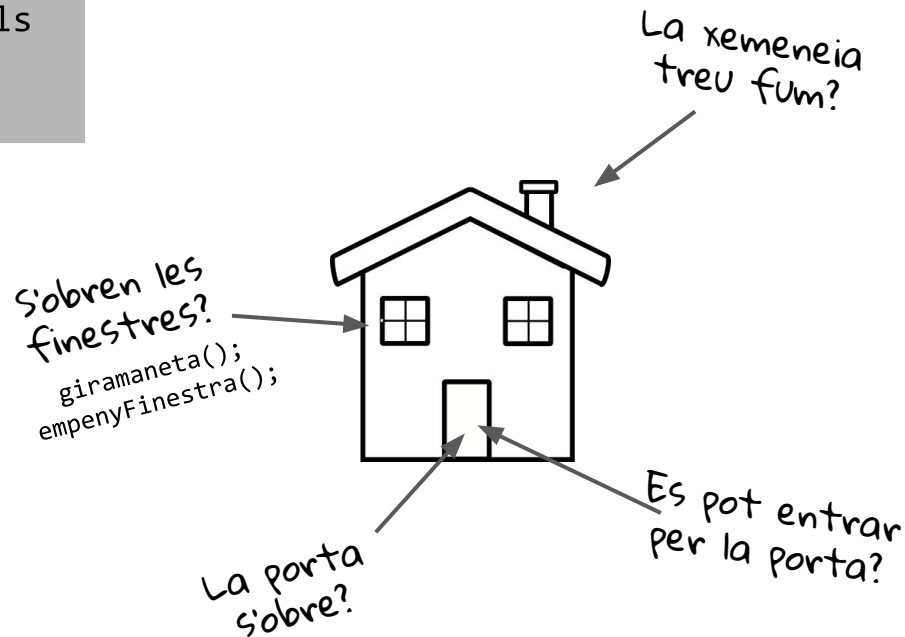
Si es prova la  
porta, la casa  
no ha  
d'interferir



# Provar una sola funcionalitat

---

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

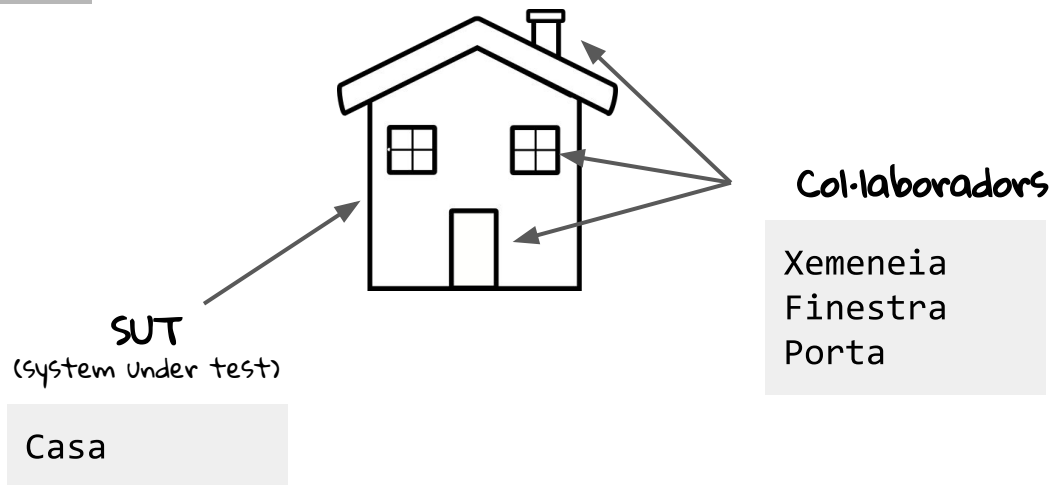


No cal fer tests dels  
mètodes que no tenen un  
mínim de funcionalitat

Hi han diverses  
opinions sobre  
aquest tema..



Sovint les classes  
en contenen  
d'altres



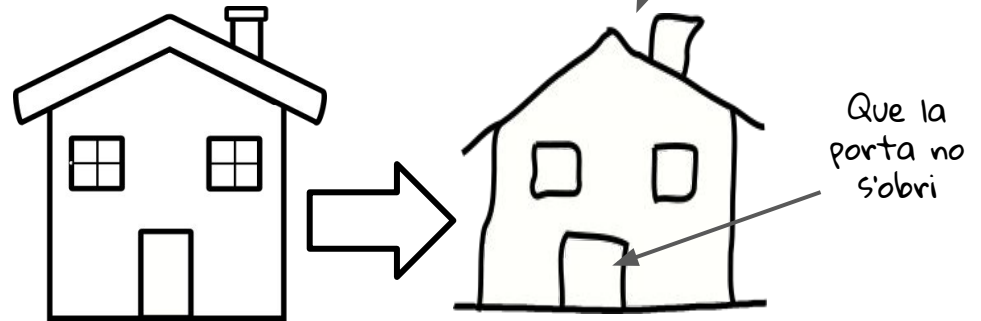


# Creació de dobles per aïllar tests

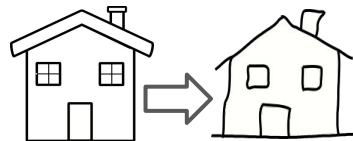
---

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

Versions simplifiades  
dels col·laboradors



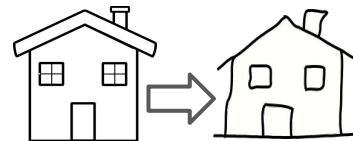
## Stubs



Es passen  
com simples  
valors però  
no se'n fa  
res

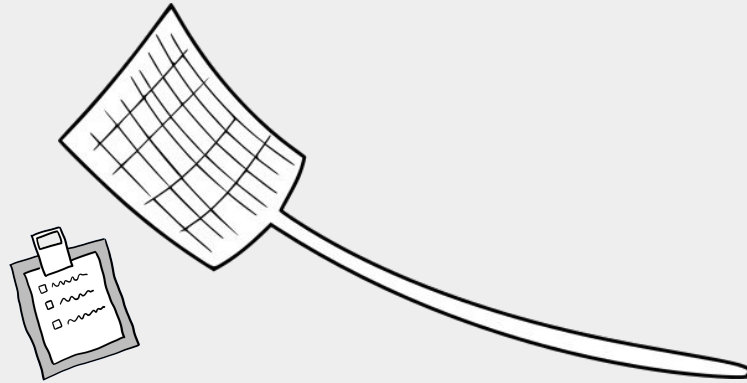
Conté dades  
que el SUT  
necessita

## Mocks



Falsifiquen  
funcionalitats dels  
col·laboradors

# Eines per fer tests



# Tipus de Testers

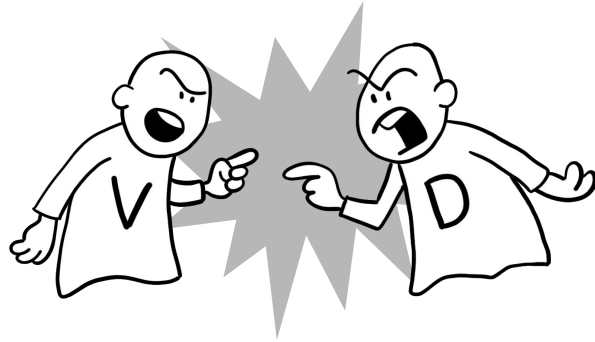


# Tipus de testers



```
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();  
    }  
}
```

Cap problema



Amb reflection  
s'injecta una  
instància de Porta

El test serà  
complicat però no  
passa res

```
public class Casa {  
  
    private Porta _porta;  
  
    public Casa() {  
        _porta = new Porta();  
    }  
  
    public Entra() {  
        _porta.obrePorta();  
        dins = true;  
        _porta.tancaPorta();  
    }  
}
```

Quina basura de  
codi

Reescriu-lo perquè el  
constructor rebi la  
instància de Porta

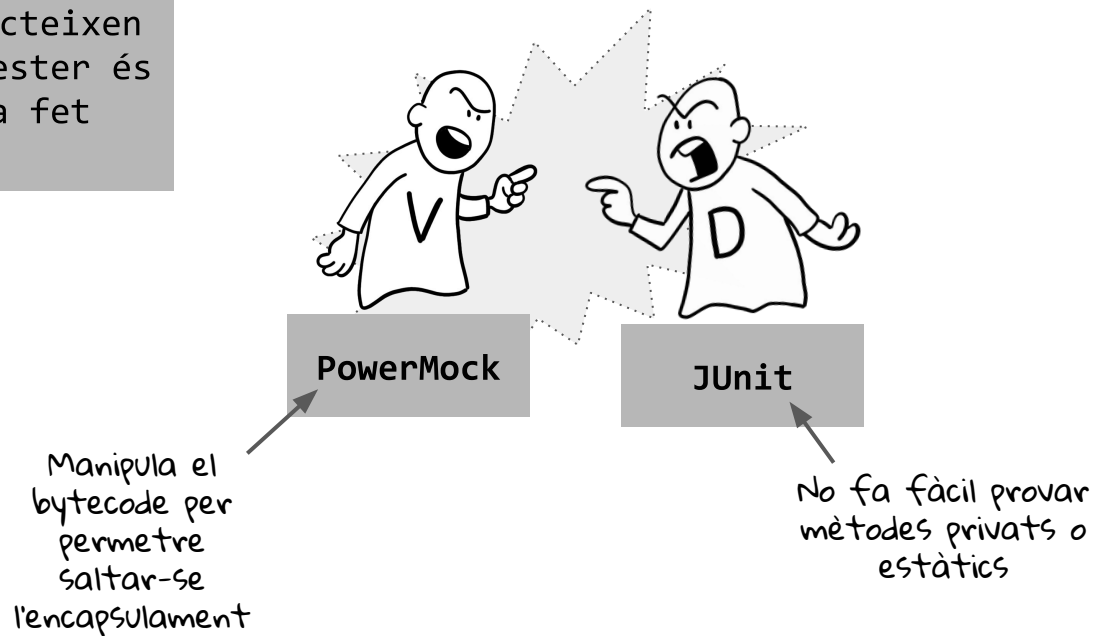


Només seria pitjor  
amb mètodes estàtics

Els tests han de  
ser senzills



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



Vols fer un  
~~puto~~  
exemple?



P1: Per fi anem per feina

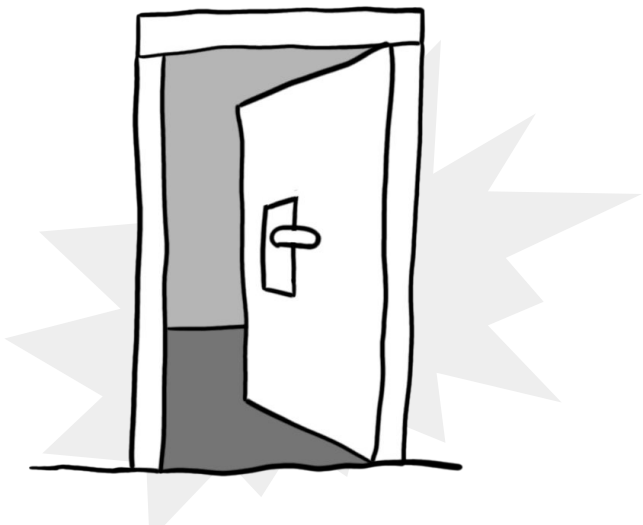
# Codi font

---



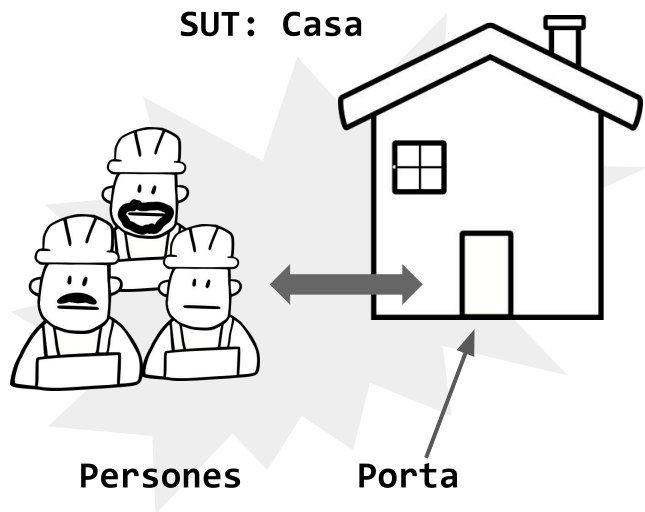
<https://github.com/fxaviersala/Curs-Testing-Udg-2021/tree/day1>

# Porta



SUT: Porta

- La porta **pot tenir clau** o ser sense clau
- S'ha de poder **obrir i tancar**
- Si té clau:
  - S'ha de poder girar la clau al pany
  - Si **està tancada amb clau** la porta **no s'obre**



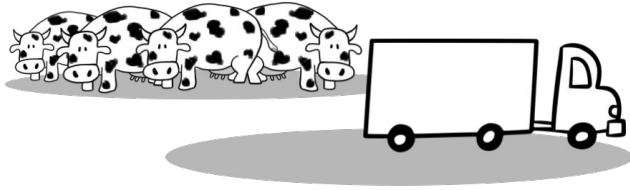
- En la casa hi entren i surten persones
  - Les persones no poden entrar, ni sortir si la porta està tancada
  - La porta es pot obrir i tancar
  - La mateixa persona no pot entrar si ja és a dins (ni pot sortir si no està a dins)

# 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

# Transport de vaques



- Comprova que es poden posar vaques en un camió però sempre complint les mesures de seguretat:
  - El camió té un pes màxim que no s'ha de superar
  - La raça de la vaca determina la quantitat de llet que produirà

Resum de les  
eines que usarem  
en les pràctiques

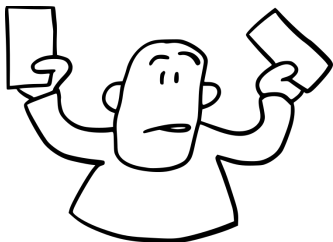




# Frameworks de tests

---

Proporcionen comandes  
específiques per fer  
comparacions



×Unit.net



Són els  
encarregats de  
comprovar els  
resultats

Assert.Equal

Assert.InRange

Assert.Null

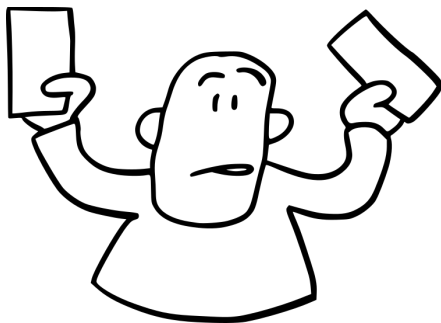
Assert.True

Assert.Same

Assert.Contains

Assert.IsType

Assert.Empty



Comprovacions a  
través d'Asserts

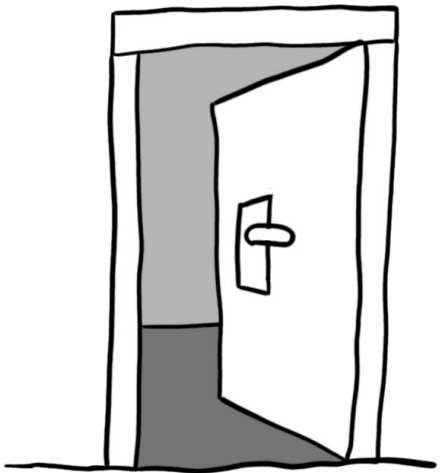


xUnit.net

```
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;  
    }  
}
```

✕Unit.net

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

    porta.Acciona();

    Assert.True(porta.EsOberta(),
        "La porta no s'ha obert");
}
```

### 3 Fases

Arrange

Act

Assert

```
Porta porta = new Porta();
```

Prepara  
l'objecte a  
provar

```
porta.Acciona();
```

Executa  
el  
mètode

```
AssertTrue(porta.EsOberta(),  
"La porta no s'ha  
obert");
```

Comprova  
que ha  
funcionat

missatge d'error en  
cas de fallar

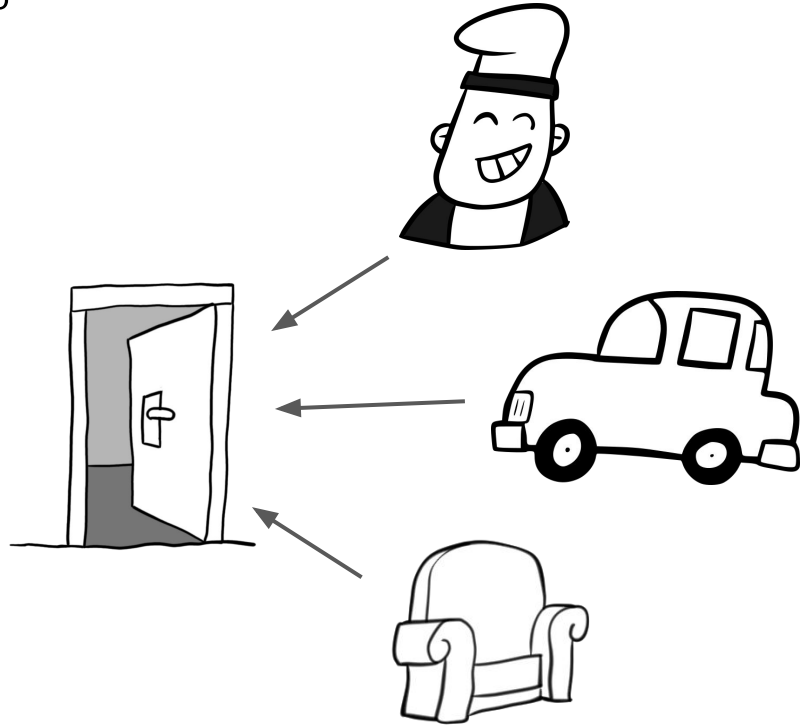


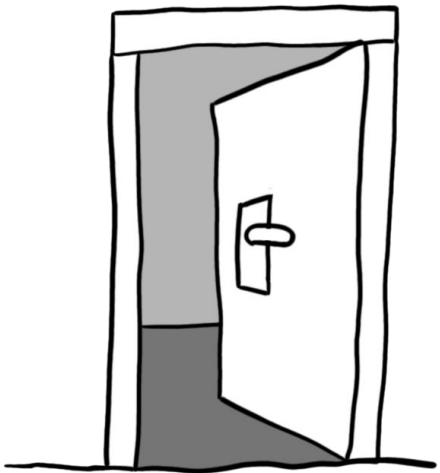
P2: Tests parametrizats

# 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();

    Assert.True(porta.EsOberta() == esperat,
        "La porta no funciona bé");
}
```



✕Unit.net

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

    porta.Acciona();

    Assert.True(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

✕Unit.net



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

    porta.Acciona();

    porta.EsOberta().Should().Be(true),
        "La porta no s'ha obert");
}
```

✕Unit.net



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

    porta.Acciona();

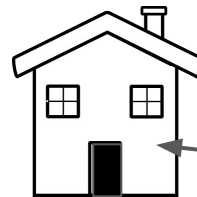
    porta.EsOberta().Should().Be(esperat,
        "La porta no funciona bé");
}
```



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 que 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



