

Dia 2: Test Driven Development i cobertura

Xavier Sala Pujolar



Universitat
de Girona

Febrer 2021

Qualitat dels tests

Els tests tenen bona qualitat
si canviant el codi el test
continua funcionant com ho ha
de fer

Mai s'ha d'oblidar que la
qualitat dels tests està molt
lligada a la qualitat del codi

Continuen funcionant quan hi ha canvis

Si canvien els
requeriments és
normal que alguns
tests fallin

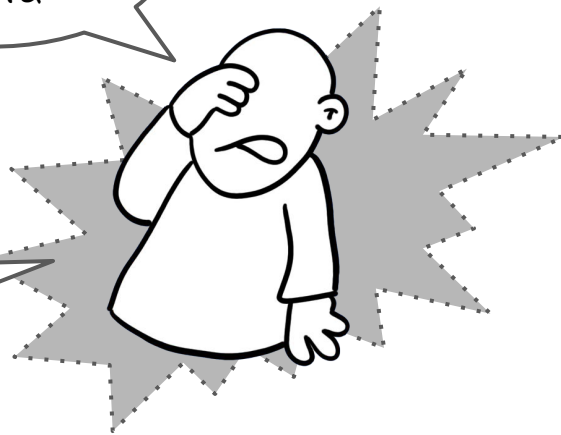
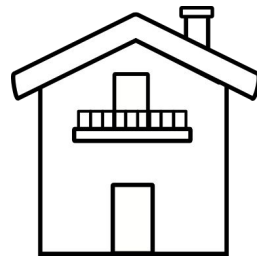


Si el test no tenia res
a veure amb el canvi,
ha de continuar
funcionant

Falla el
test d'obrir
porta

Un sol canvi no ha de
fer que fallin molts
tests

I no va la
xemeneia



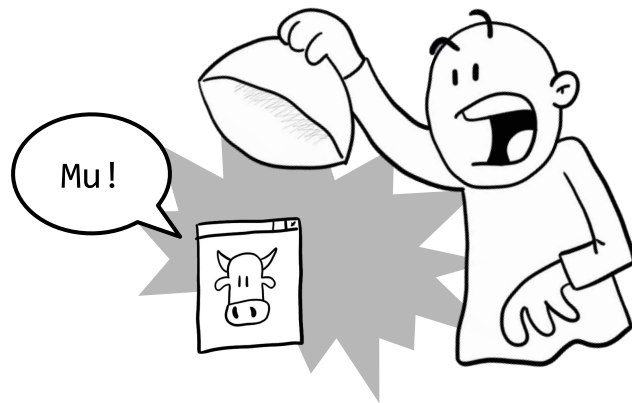
Cobreixen la part important del codi

Com a mínim el codi
de negoci s'ha de
provar



Protegeixen contra les regressions i el refactor

Han d'evitar que es
degradi el programa
al afegir funcions
noves



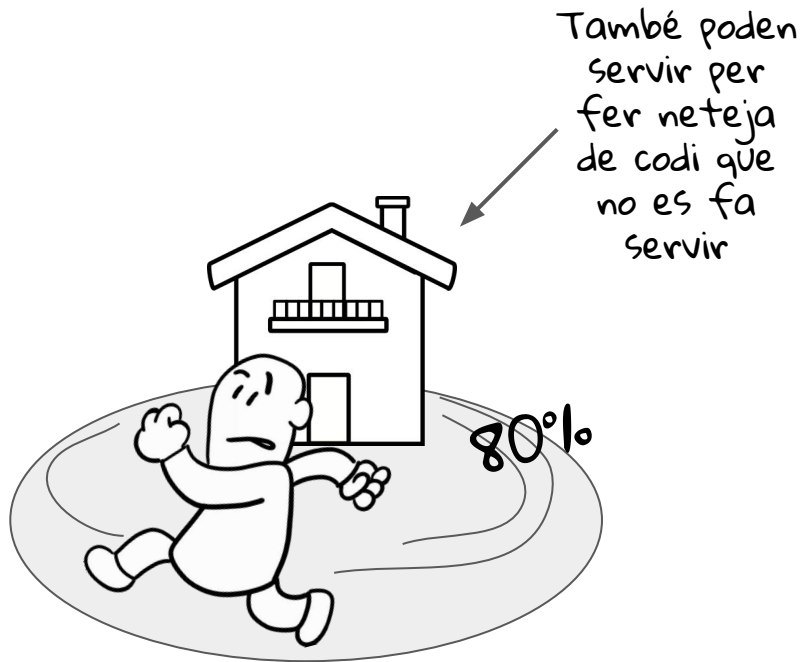
Han de ser fàcilment mantenibles

S'ha d'evitar
comprovar detalls
d'implementació



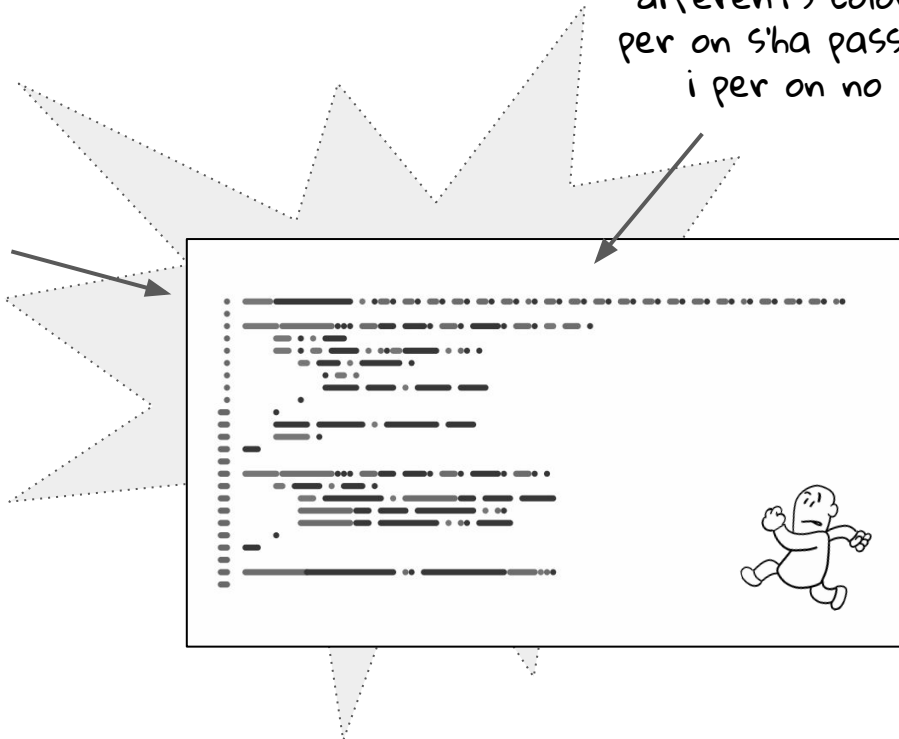
Programes de cobertura

Els programes de cobertura donen informació sobre per quines línies del codi i branques (if) passen els tests



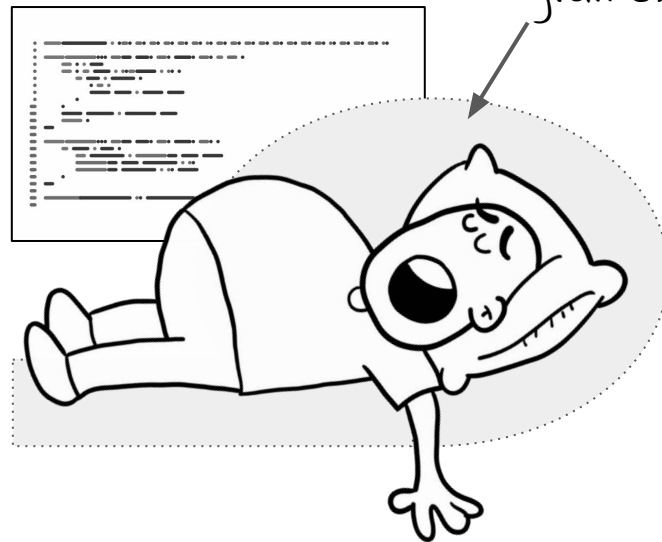
Generen
informes
amb els
resultats

Solen pintar de
diferents colors
per on s'ha passat
i per on no



Una cobertura del
100% no implica que
els tests siguin
millors

I normalment
representa un
gran esforç



TDD



El Test Driven Development és una tècnica de desenvolupament que obliga a escriure el Test abans del codi

Obliga als desenvolupadors a pensar el comportament abans d'implementar-lo



**Mai escriure codi sense un
test que falli**

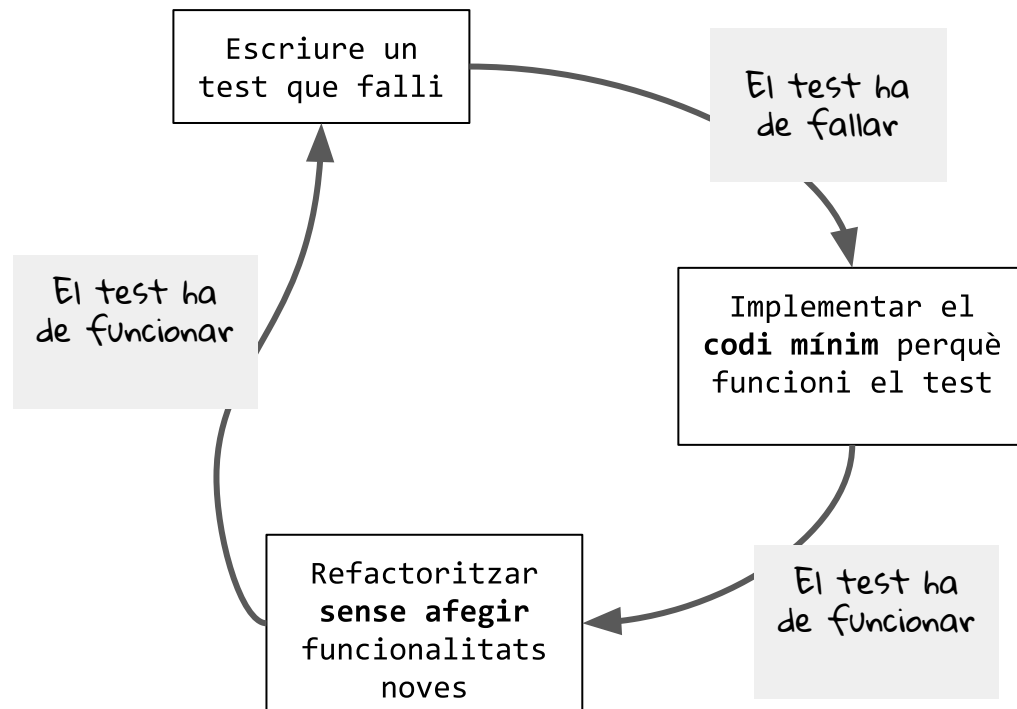
Hi hauran tests
de pràcticament
tot el codi

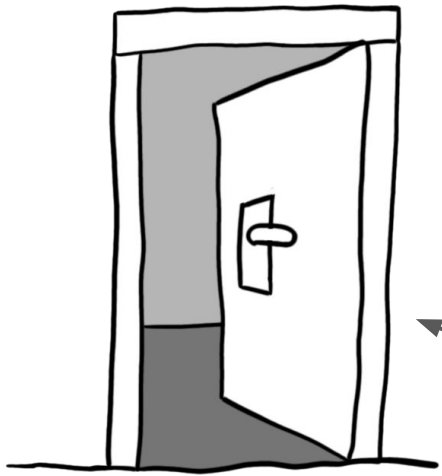
**S'ha d'escriure el codi
mínim per fer que el test
funcioni**

Hi ha la
quantitat de
codi justa

Desapareixen els
mètodes "I si
algun dia ..."

El cicle de treball amb
TDD és sempre el
mateix





Funcionalitat 1: Es pot obrir la porta

Accionar la porta
fa que s'obri

SUT: Porta

1. Escriure un test que falli

✕Unit.net

Obrir la porta

No hi ha codi de producció,
o sigui que no compilarà ...

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

    var resultat = porta.Acciona();

    Assert.True(resultat);
}
```

2. Implementar el codi mínim perquè el test passi

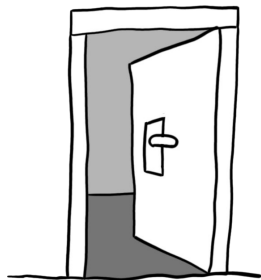
Definim l'esquema mínim



```
public class Porta {  
    public bool Acciona() {  
        return false;  
    }  
}
```




El test encara
falla ...



SUT: Porta

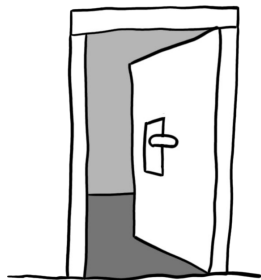
Obrir la porta

No importa
ni l'estil, ni la
forma, ...

Només que el
test passi

```
class Porta {  
  
    public bool Acciona() {  
        return true;  
    }  
  
}
```

3. Refactoritzar sense afegir funcionalitats noves

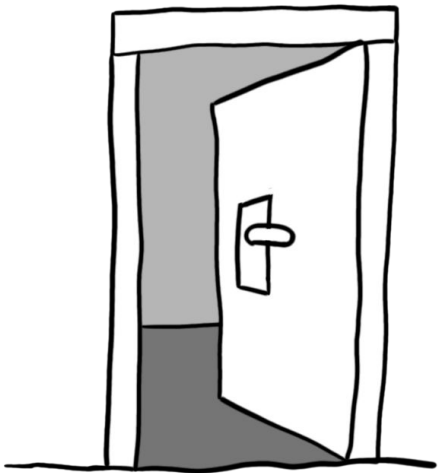


SUT: Porta

Obrir la porta

No hi ha
millores...

```
class Porta {  
  
    public bool Acciona() {  
        return true;  
    }  
  
}
```



SUT: Porta

Funcionalitat 2: La porta es pot tancar

```
class Porta {  
  
    public bool Acciona() {  
        return true;  
    }  
  
}
```

1. Escriure un test que falli

✕Unit.net

Tancar la porta

```
[Fact]
public void TestSiLaPortaTanca() {
    Porta porta = new Porta();
    porta.Acciona() // obro

    var resultat = porta.Acciona();

    Assert.False(resultat);
}
```

2. Implementar el codi mínim perquè el test passi

Tancar la porta

No importa ni
l'estil, ni la
forma, ...

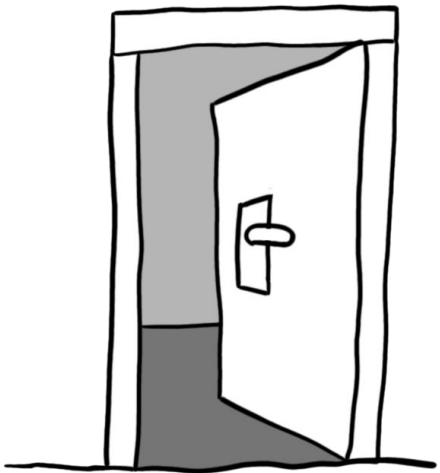
```
class Porta {  
    private bool _esOberta;  
  
    public bool Acciona() {  
        if (_esOberta) {  
            _esOberta = false;  
        }  
        else {  
            _esOberta = true;  
        }  
        return _esOberta;  
    }  
}
```

3. Refactoritzar sense afegir funcionalitats noves

```
class Porta {  
    private bool _esOberta;  
  
    public bool Acciona() {  
        if (_esOberta) {  
            _esOberta = false;  
        }  
        else {  
            _esOberta = true;  
        }  
        return _esOberta;  
    }  
}
```

Tancar la porta

```
class Porta {  
    private bool _esOberta;  
  
    public bool Acciona() {  
        _esOberta = !_esOberta;  
        return _esOberta;  
    }  
}
```



SUT: Porta

Funcionalitat 3: Saber com està la porta

```
class Porta {  
    private bool _esOberta;  
  
    public bool Acciona() {  
        _esOberta = !_esOberta;  
        return _esOberta;  
    }  
  
    public bool EsOberta() {  
        return false;  
    }  
}
```


1. Escriure un test que falla

✕Unit.net

Estat de la porta

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

    var resultat = porta.Acciona();
    Assert.Equal(porta.EsOberta(),
                  resultat);

    resultat = porta.Acciona();
    Assert.Equal(porta.EsOberta(),
                  resultat);

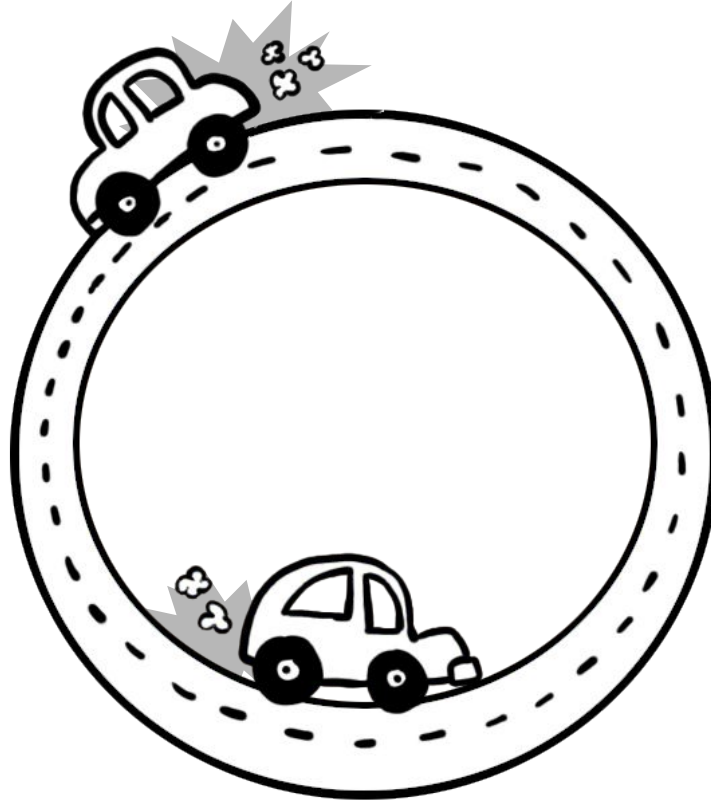
}
```

2. Implementar el codi mínim perquè el test passi

Estat de la porta

```
class Porta {  
    private bool _esOberta;  
  
    public bool Acciona() {  
        _esOberta = !_esOberta;  
        return _esOberta;  
    }  
  
    public bool EsOberta() {  
        return _esOberta;  
    }  
}
```

3. Refactoritzar sense afegir funcionalitats

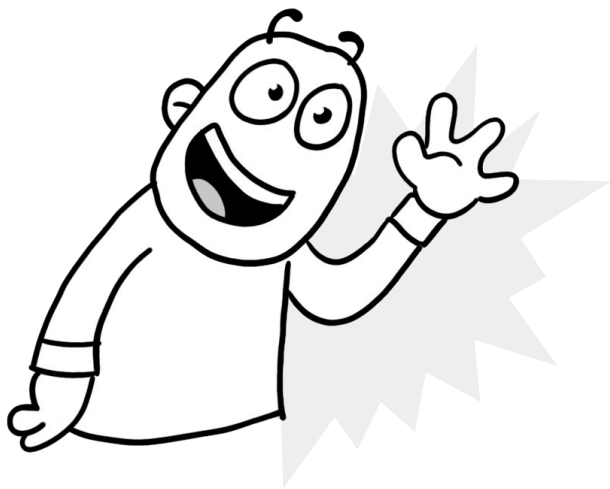


No et fa
vergonya fer
un test sobre
una porta?



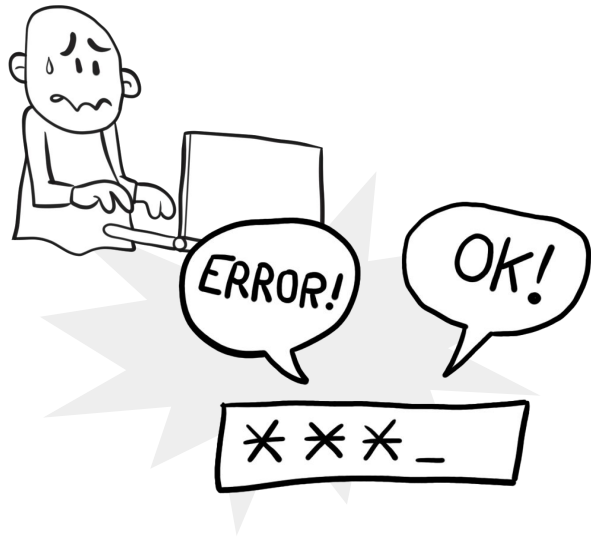
P1: Per fi anem per feina

El saludador



- Ha de rebre el nom de la persona
- Ha de comprovar que la primera lletra del nom està en majúscules
- Ha de saludar de forma diferent segons la hora que sigui:
 - De 6:00 a 12:00 ha de dir "Bon dia"
 - De 14:00 a 20:00 ha de dir "Bona tarda"
 - De 21:00 a 06:00 ha de dir "Bona nit"
- Si el nom té més de dues paraules ha d'afegir "Senyor"

Validador de contrasenyes



- Les contrasenyes han de tenir 8 caràcters de llargada com a mínim
- Han de tenir més de 2 números
- Han de tenir majúscules i minúscules
- No hi poden haver tres caràcters seguits iguals
- Si hi ha un caràcter especial els números poden ser iguals
- Si no hi ha caràcter especial els números han de ser diferents
- La contrasenya no pot ser el nom d'usuari, ni el nom de l'usuari al revés

Repository



<https://github.com/fxaviersala/Curs-Testing-Udg-2021/tree/day2>

