Proves del software

FIB

Departament d'Enginyeria de Serveis i Sistemes d'Informació
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Proves del software

"Les proves del software són l'activitat que es duu a terme per avaluar i millorar la qualitat del producte identificant defectes I problemes"

"Les proves del software consisteixen en en la verificació dinàmica del comportament d'un programa sobre un conjunt finit de casos de prova seleccionats (…) en front del comportament esperat"

Software Engineering Body Of Knowledge

Terminologia

Un error durant el desenvolupament provoca

un defecte al software que pot causar

una fallada durant l'execució del programa

Mitjançant les proves, podem provocar una fallada (el programa no es comporta com s'esperava).

Un cop identificada, haurem de trobar i corregir el defecte.

Un cop corregit el defecte, caldrà analitzar l'error que l'ha provocat

L'objectiu final és trobar els **errors**!

Classificació de les proves

Segons l'àmbit del sistema provat:

- Unitàries (una classe aïllada)
- Integració (dos o més components)
- Sistema (tot el sistema)

Segons l'objectiu de la prova

- Acceptació (acceptació del sistema)
- Alpha/Beta (abans del lliurament final)
- Conformitat/Correctesa
- Fiabilitat (trobar errors)
- Regressió (verificar que no s'han introduït nous errors)
- Rendiment/Càrrega
- Usabilitat

Classificació de les proves

Segons el coneixement sobre l'estructura interna del sistema

- Caixa negra (no la coneixem)
- Caixa blanca (si la coneixem)

Desenvolupament de software guiat per les proves

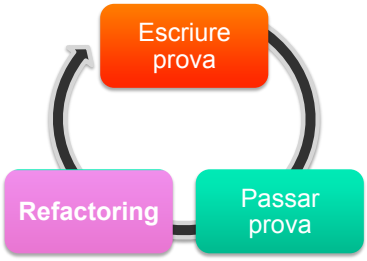
Test Driven Development (TDD)

És una tècnica de desenvolupament de software centrada en les proves unitàries

Test First: Escriure les proves abans que el codi.
Refactoring: Modificar l'estructura interna (el disseny) d'un sistema sense alterar-ne el comportament

Tres normes:

No s'escriu codi de producció fins que hi ha una prova que falla
A la prova, només s'escriu el mínim de codi necessari per fallar
No s'escriu més codi de producció que el mínim necessari per passar la prova



Test Driven Development: Beneficis

No hi ha cap responsabilitat que no tingui una prova associada

Tenim un conjunt de proves de regressió que ens permet fer **refactoring**

Hem documentat casos d'exemple dels requisits en un llenguatge no ambigu i fàcilment verificable (són executables).

Com que tenim proves de regressió, el disseny és més flexible

Ens obliga a pensar abans en els requisits que en la implementació

Disseny emergent: El disseny "emergeix" a mesura que anem afegint casos de prova

Test Driven Development: Bones pràctiques

El codi de proves s'ha de dissenyar i s'ha de mantenir. A més dels habituals, hi ha principis de disseny específics per a les proves:

F.I.R.S.T

Fast: Si no s'executen ràpidament, a la pràctica no s'executaràn

Independent: Les proves no han de dependre entre elles

Repeatable: S'han de poder executar en qualsevol entorn (desenvolupament, preproducció, etc.)

Self-Validating: El resultat d'una prova és booleà: O passa o no passa. No es necessita validació manual per determinar si ha passat

Timely: S'han d'escriure al moment adequat: Just abans que el codi que proven. Si s'escriuen després, pot passar que el codi sigui massa difícil de provar i no el provem.

Behavior Driven Development

BDD és a les proves d'acceptació el que TDD és a les proves unitàries

Els casos de prova s'escriuen en llenguatge natural però són executables

```
# language: ca
Característica: Suma
  Per evitar fer errors tontos
  Com un matemàtic idiota
  Vull saber la suma dels números

  Esquema de l'escenari: Sumar dos números
    Donat que he introduït <entrada_1> a la calculadora
    I que he introduït <entrada_2> a la calculadora
    Quan premo el <botó>
    Aleshores el resultat ha de ser <resultat> a la pantalla

  Exemples:
    | entrada_1 | entrada_2 | botó | resultat |
    | 20        | 30        | add  | 50       |
    | 2         | 5         | add  | 7        |
    | 0         | 40        | add  | 40       |
```

Behavior Driven Development

Un programa interpreta la descripció de l'escenari i el trasllada a invocacions d'operacions del sistema

```
Before do
  @calc = Calculadora.new
end

Donat /que he introduït (\d+) a la calculadora/ do |n|
  @calc.push n.to_i
end

Quan /premo el (\w+)/ do |op|
  @result = @calc.send op
end

Aleshores /el resultat ha de ser (\d+) a la pantalla/ do |result|
  @result.should == result.to_f
end
```


Bibliografia

- Astels, D; Chelimsky, D; Dennis, Z; Hellesoy, A.; Helmkamp B., North D. "*The Rspec Book*", Pragmatic Bookshelf, 2008
- Beck, K. "*Test Driven Development: By Example*", Addison-Wesley Professional, 2002