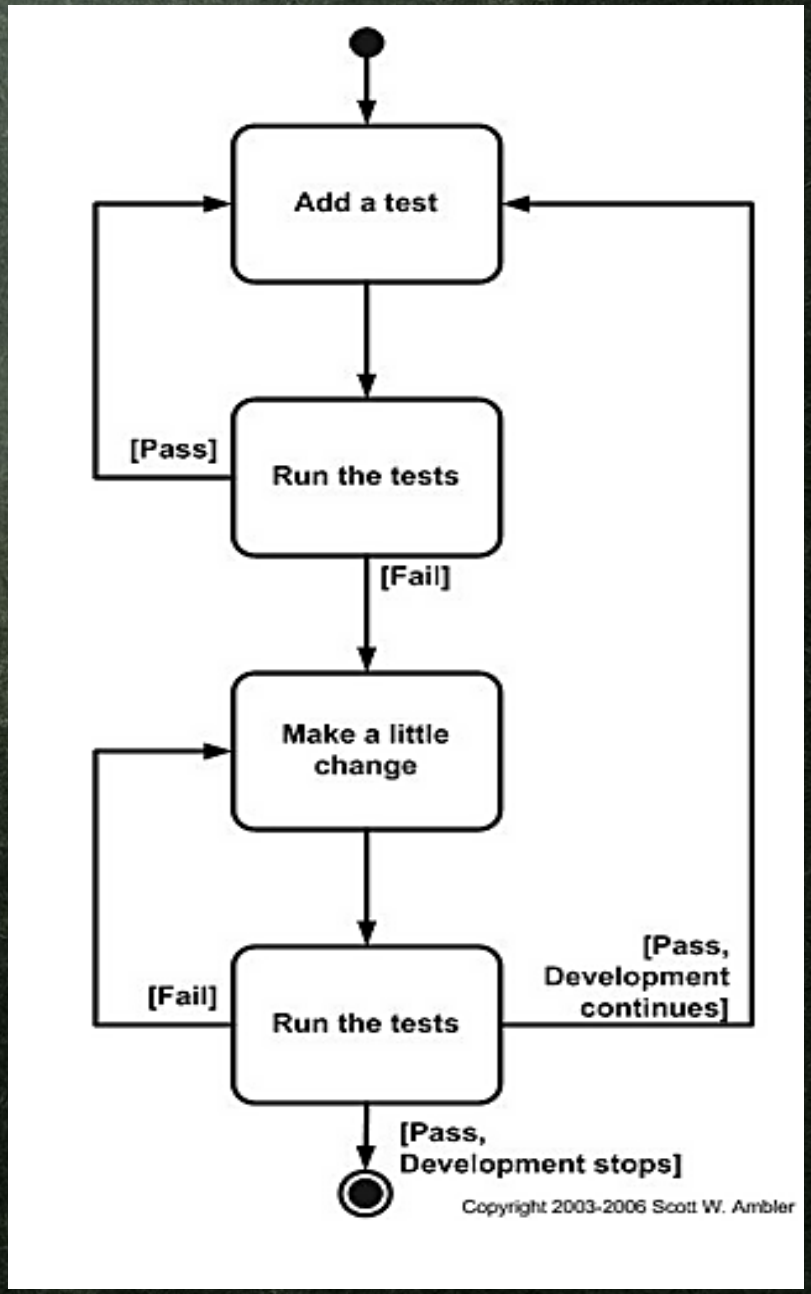
Dezvoltarea prin testare

Cuprins:

- Definitie;
- Ciclul dezvoltarii prin testare;
- Avantaje;
- Dezavantaje;

Definitie:

- TDD este una din cea mai bună metodă cunoscută de a face design incremental. Practicienii TDD codifică exemplele folosind teste care sunt apoi păstrate pentru a valida soluția completă. Prin identificarea și diminuarea similarităților din cod în pasul de refactorizare, design-ul este simplificat și îmbunătățit în continuu.



Ciclul dezvoltarii prin testare:

Dezvoltarea prin testare presupune citeva etape:

- Adaugare test;
- Rularea tuturor testelor(testele noi sa nu treaca);
- Scrierea codului;
- Rularea tuturor testelor(testele noi sa treaca);
- Refactorizarea codului;
- Repetarea ciclului;

Adaugare test:

În dezvoltarea prin testare, adăugarea unei noi funcționalităţi începe cu scrierea testului. Inevitabil, acest test nu va trece, deoarece codul corespunzător nu este încă scris. (În cazul în care testul scris a trecut, înseamnă că "noua" funcționalitate există deja, sau testul este greșit.) Pentru a scrie testul, dezvoltatorul trebuie să fi înțeles în mod clar care cerinţe trebuie îndeplinite de noile functionalitati. Se iau în considerare noile cazuril de utilizare. Noile cerințe pot aduce, de asemenea, la modificări ale testelor existente.

Rularea tuturor testelor (testele noi sa nu treaca):

- În acest stadiu, se verifică că testele scrise nu trec. Această etapă verifică, de asemenea testele: testele scrise pot trece în orice moment și, în consecință, să fie inutil. Noi teste nu ar trebui să treaca la motive lesne de înțeles. Acest lucru va spori încrederea (deși nu există nici o garanție în întregime), care test este într-adevăr de testare pentru care a fost dezvoltat.

Scrierea codului:

- În această etapă se adaugă codul astfel ca testul va trece. Acest cod nu trebuie să fie perfect. Trebuie sa fie acceptabil, asa ca treaca testul într-un fel. Pentru că pașii următori vor îmbunătăți și lustrui codul.

- Este important să se scrie cod care va face sa treacă testul. Nu adăugați funcționalitati inutile și care nu a fost testate.

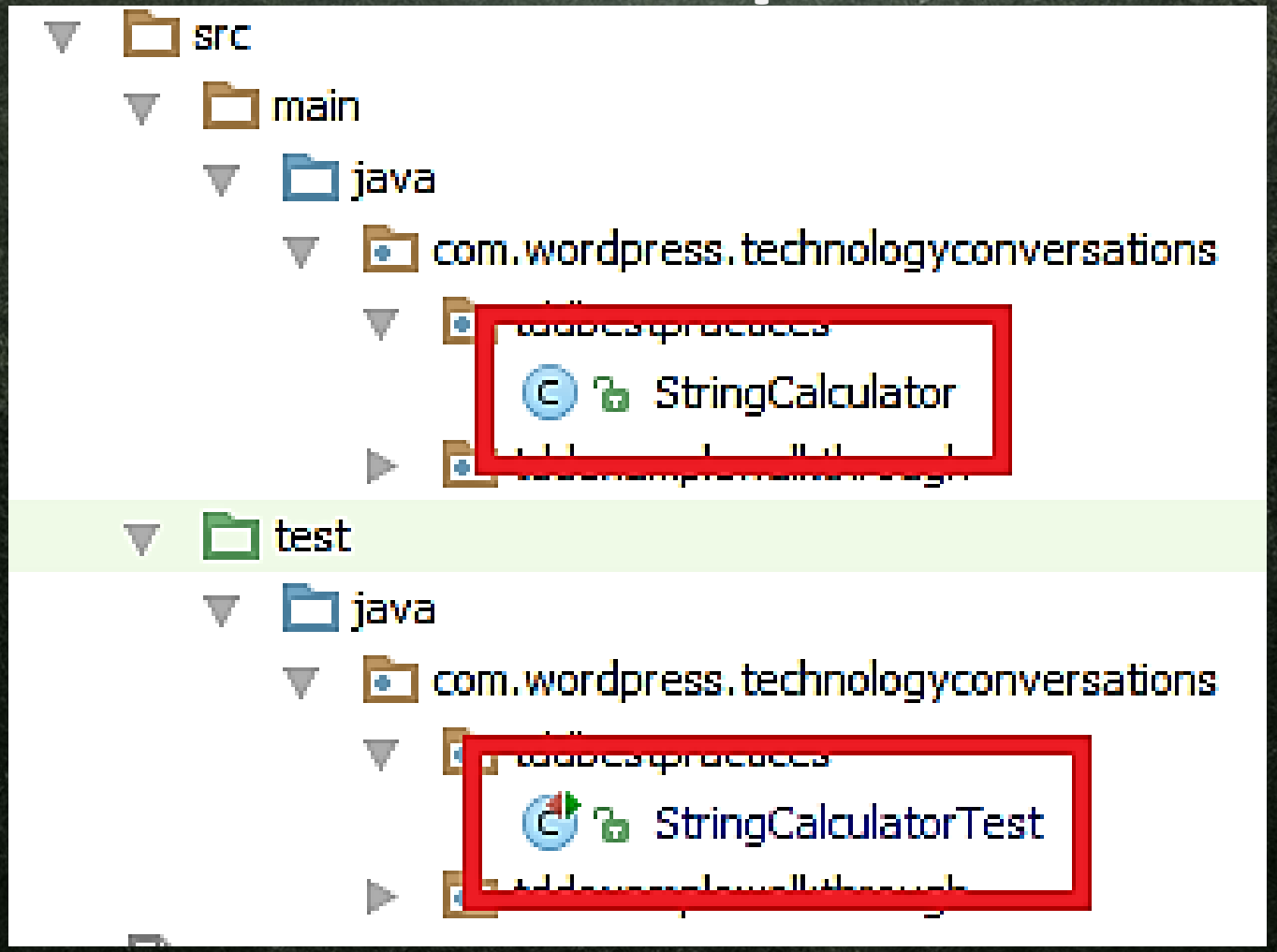
Rularea tuturor testelor (testele noi sa treaca):

- Dacă toate testele trec, programatorul poate fi sigur că codul îndeplinește toate cerințele de testare. Puteți trece apoi la etapa finală a ciclului.

Refactorizarea codului:

- Când a ajuns la funcționalitatea necesară, acest cod poate fi curățat. ***Refactorizare*** - procesul de schimbare a structurii interne a programului, nu afectează comportamentul său extern. Are scopul de a facilita înțelegerea funcționării softului, pentru a elimina dublarea de cod, pentru a facilita schimbări în viitorul apropiat.

Exemplu:


src
main
java
com.wordpress.technologyconversations
StringCalculator
test
java
com.wordpress.technologyconversations
StringCalculatorTest

Exemplu:

```
@Test
public final void
whenSemicolonDelimiterIsSpecifiedThenItIsUsedToSeparateNumbers() {
        Assert.assertEquals(3+6+15, StringCalculator.add("//;n3;6;15"));
}
@Test
public final void whenOneNumberIsUsedThenReturnValueIsThatSameNumber() {
        Assert.assertEquals(3, StringCalculator.add("3"));
}

@Test
public final void whenTwoNumbersAreUsedThenReturnValueIsTheirSum() {
        Assert.assertEquals(3+6, StringCalculator.add("3,6"));
}
```

Exemplu:

```
@Test
public final void
whenNegativeNumbersAreUsedThenRuntimeExceptionIsThrown() {
    RuntimeException exception = null;
    try {
        StringCalculator.add("3,-6,15,-18,46,33");
    } catch (RuntimeException e) {
        exception = e;
    }
    Assert.assertNotNull("Exception was not thrown", exception);
    Assert.assertEquals("Negatives not allowed: [-6, -18]",
    exception.getMessage());
}
```

Exemplu:

```
@Test
public final void whenAddIsUsedThenItWorks() {
    Assert.assertEquals(0, StringCalculator.add(""));
    Assert.assertEquals(3, StringCalculator.add("3"));
    Assert.assertEquals(3+6, StringCalculator.add("3,6"));
    Assert.assertEquals(3+6+15+18+46+33,
    StringCalculator.add("3,6,15,18,46,33"));
    Assert.assertEquals(3+6+15, StringCalculator.add("3,6n15"));
    Assert.assertEquals(3+6+15, StringCalculator.add("//;n3;6;15"));
    Assert.assertEquals(3+1000+6,
    StringCalculator.add("3,1000,1001,6,1234"));
}
```

Avantaje:

- Testele pot fi salvate ca documentatie;
- Este mai usor de realizat refactorizarea;
- Ajuta la evitarea erorilor;
- Programatorii care scriu teste sunt mai productivi;
- Se reduce timpul efectiv;

Dezavantaje:

- Exista probleme care nu pot fi rezolvate prin testare(securitate);
- Complicat de implementat cind este nevoie sa treaca testele functionale (implemetarea interfetei, aplicatii cu BD);
- Multimea de teste poate provoca prea multa incredere, ceea ce cauzeaza un control mai slab.