

TDD et Erlang

Nicolas Charpentier - Dominic Williams

charpi.net - dominicwilliams.net

30 Mars 2009

Pourquoi certaines startups s'intéressent-elles à Erlang

Google, Amazon, Apache,
EngineYard, ...

Pensez différemment, pensez Erlang

Besoin de scalabilité?
Envie de simplicité?
Pensez Erlang

*Well, there are agile languages and agile methods.
When you put the two together you get agile squared.*

— Ward Cunningham

Tests canoniques

```
assertEquals("Hello", substring(5, "Hello_world"));  
assertEquals("orld", substring(-4, "Hello_world"));  
assertEquals("", substring(0, "Hello_world"));  
assertEquals("", substring(5, ""));  
assertEquals("", substring(5, NULL));
```

Initialisation des objets

```
public void testSelfTestDirCanBeSet() {  
    final RemoteControlConfiguration configuration;  
    final File aDirectory;  
  
    configuration = new RemoteControlConfiguration();  
    aDirectory = new File("\A_Directory_Name\  
    configuration.setSelfTestDir(aDirectory);  
    assertEquals(aDirectory,  
                  configuration.getSelfTestDir());  
}
```

Des objets comme valeur de retour

```
public void testBrowserStartCommandMatchWhenBrowserStringIsTheBrowserName () {  
    final BrowserStringParser.Result result ;  
  
    result = new BrowserStringParser ().parseBrowserStartCommand ("firefox", "firefox");  
    assertTrue (result.match ());  
    assertNull (result.customLauncher ());  
}
```

OpenQA - Selenium server

Propagation

```
public void testMatch() throws Exception {
    RowFixture fixture = new TestRowFixture();
    TypeAdapter arrayAdapter =
        TypeAdapter.on(fixture,
            BusinessObject.class.getMethod("getStrings",
                new Class[0]));
    fixture.columnBindings = new TypeAdapter[]{arrayAdapter };

    LinkedList computed = new LinkedList();
    computed.add(new BusinessObject(new String[] { "1" }));
    LinkedList expected = new LinkedList();
    expected.add(new Parse("tr", "", new Parse("td", "1", null, null), null));
    fixture.match(expected, computed, 0);
    assertEquals("right", 1, fixture.counts.right);
    assertEquals("exceptions", 0, fixture.counts.exceptions);
    assertEquals("missing", 0, fixture.missing.size());
    assertEquals("surplus", 0, fixture.surplus.size());
}
```

Cunningham & Cunningham, Inc. - Fit

Héritage, interfaces

Structure de données dynamique

```
test_self_dir_can_be_set () ->  
  "Name" = configuration: get (self_dir ,  
                                [{self_dir , "Name"}]).
```

Filtrage (*Pattern Matching*)

```
Author = "Knuth",  
"Knuth" = Author,  
Author = "Dijkstra", %% *exit*: badmatch  
...
```

```
Book = {123, Author, "The Art of Programming"},  
{123, "Knuth", "The Art of Programming"} = Book,  
{Id, _, Title} = Book,  
123 = Id,  
"The Art of Programming" = Title.
```

Programmation déclarative

Le test

```
–module( xptest ).  
–export ( [run/0] ).
```

```
run ( [] ) ->  
  [] = xptest:odd ( [] ),  
  [3] = xptest:odd ( [3] ),  
  [3,5,7,9] = xptest:odd ( [3,2,4,5,6,7,8,9] ).
```

Programmation déclarative

Le code

```
–module (xpcday) .  
–export ( [odd/1] ) .
```

```
odd( [ ] ) ->  
    [ ] ;
```

```
odd( [Value | Values] ) when Value rem 2 == 1 ->  
    [Value | odd(Values)] ;
```

```
odd( [ _ | Values ] ) ->  
    odd(Values) .
```

"Lambda, the ultimate mock"

Le test

```
-module(xpday2_test).  
-export([run/0]).
```

```
run([]) ->  
  [] = xpday2:filter([], fun(_) -> true end),  
  [3] = xpday2:filter([3], fun(3) > true end),  
  Datas = [3,2,4,5,6,7,8,9],  
  Is_odd = fun(X) when X rem 2 == 1 -> true;  
            (-) -> false end,  
  [3,5,7,9] = xpday2:filter(Datas, Is_odd).
```

"Lambda, the ultimate mock"

Le code

```
-module(xpday2).  
-export([filter/2]).  
  
filter([], _) ->  
    [];  
filter([Value|Values], Predicat) ->  
    Filter = case Predicat(Value) of  
                true -> Value;  
                _ -> []  
            end,  
    [Filter | filter(Values, Predicat)].
```

Pourquoi la concurrence

- Interactivité des IHM
- Améliorer les performances
- Multi-cœur

Threads à mémoire partagée

- Modèle dominant
- Processus légers
- Ressources partagées, notamment mémoire
- Non-déterminisme:
 - situations de compétition (*race*)
 - interblocages (*deadlocks*)

Tests de threads

- Vérifier que c'est parallèle
 - seulement possible si le thread à tester peut être "ralenti"
 - nécessite de créer des threads dans le test
- Vérifier que c'est ré-entrant. C'est le plus dur:
 - le non-déterminisme est causé par le reste du programme
 - on ne peut pas contrôler l'ordonnanceur
- Tests illisibles et sans valeur documentaire

Acteurs, ou processus communicants par messages

- Depuis 1978 et CSP (Hoare):
 - processus légers isolés
 - envoi de messages
- Parfaitement déterministe
- Simple et compréhensible

Acteurs en Erlang

- Trois primitives
 - spawn
 - ! (envoi de messages)
 - receive (réception de messages et synchronisation)
- Identique pour process locaux ou distribués
- L'écriture de tests devient simplissime

TDD de process Erlang: le test

```
run () ->  
  Counter = spawn (counter, count, [1]),  
  Counter ! {self (), next},  
  1 = next (Counter),  
  Counter ! {self (), next},  
  2 = next (Counter),  
  pass.
```

```
next (Counter) ->  
  receive {Counter, Next} -> Next  
  after 100 -> timeout  
  end.
```

TDD de process Erlang: le code

```
-module (counter).  
-export ([count/1]).  
  
count (Number) ->  
    receive {From, next} ->  
    From ! {self (), Number},  
    count (Number + 1)  
end.
```

I'm sorry that I long ago coined the term "objects" for this topic because it gets many people to focus on the lesser idea.

The big idea is "messaging".

— Alan Kay

Retrouvez plus d'information sur

- <http://dominicwilliams.net>
- <http://charpi.net>