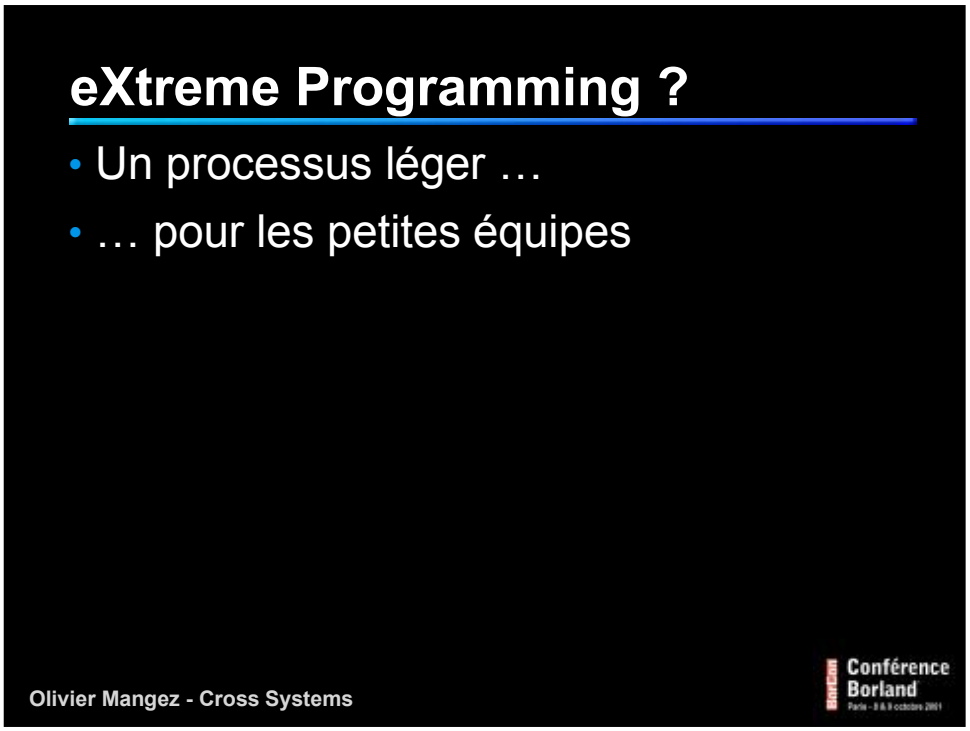
- JUnit : framework de test écrit par :
  - Erich Gamma (GoF, Design Patterns)
  - Kent Beck (XP)

Plan

- **eXtreme Programming**
- Tests
- JUnit
- Pour aller plus loin
- Et Delphi dans tout ça ?

Olivier Mangez - Cross Systems

Conférence Borland



- XP est un **processus** :
  - ensemble de rôles et de pratiques...
  - … pour mener à bien un projet
  - les pratiques prennent tout leur sens ensemble

- **But** :
  - qualité
  - respect des coûts et des délais
  - robuste au changement

- **Caractéristiques** :
  - processus léger (peu de contraintes)
  - adaptatif, pas prédictif
  - centré sur les tests
  - petite équipe
  - proche du client (le client est acteur)

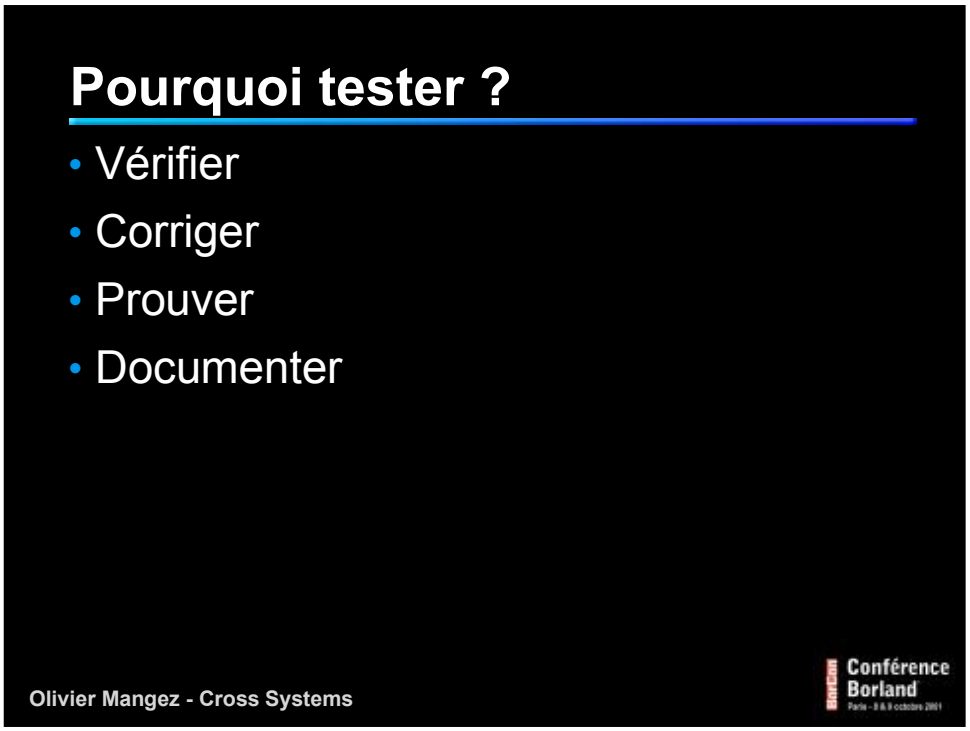
- **Planification**
  - une itération à la fois
  - adaptatif
  - faire les choses en temps voulu
- **Design**
  - rapide
  - laisser parler le code
- **Codage** : le code appartient à tout le monde
  - pour avancer/corriger plus vite
  - norme de codage
  - gestion de version
  - pair programming (- productif ? Relecture, - de surf, + motivé, code testé + tôt)
  - changer de module à chaque itération
- **Refactoring**
  - rendre le code parlant
  - factoriser
  - simplifier, supprimer
- **Tests** : à suivre

Plan

- eXtreme Programming
- **Tests**
- JUnit
- Pour aller plus loin
- Et Delphi dans tout ça ?

Olivier Mangez - Cross Systems

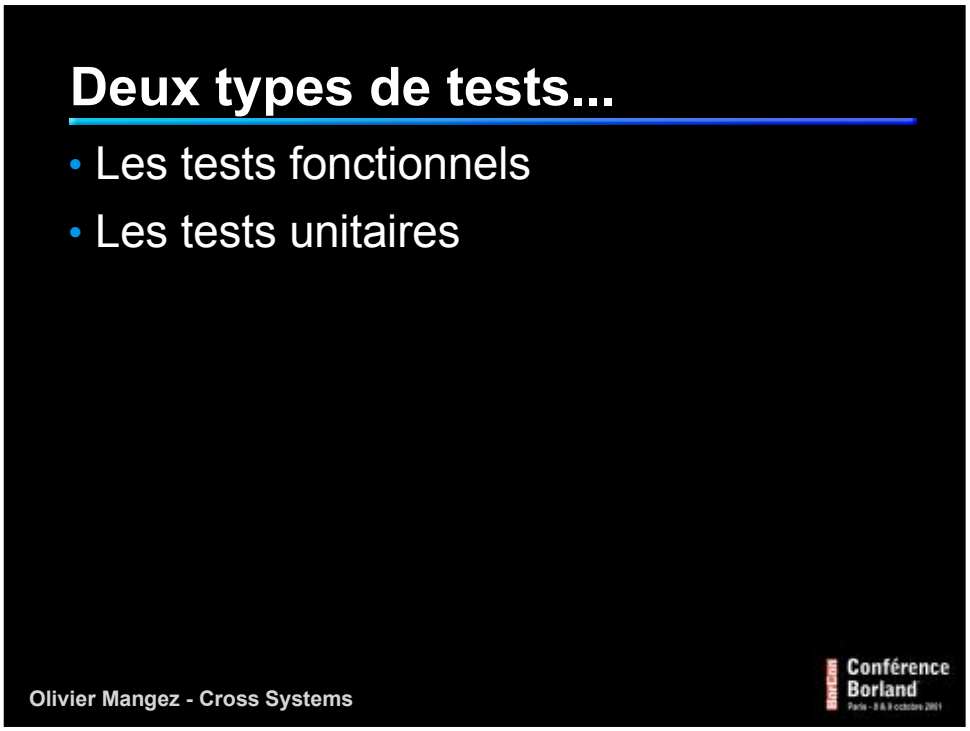
Conférence Borland



- **Vérifier**
  - que ça marche
  - que ça marche encore
    (régressions, filet pour refactoring,
    collective code ownership)

- **Corriger**
  - on découvre un bug
  - on écrit un test qui le met en évidence
  - on lance le test pour vérifier qu'il échoue
    (et donc qu'il fonctionne)
  - on corrige le bug
  - on lance le test pour vérifier qu'il réussit

- **Prouver**
  - on n'affirme pas au client que ça marche,
    on lui montre
  - une fonctionnalité n'existe que si elle a été testée

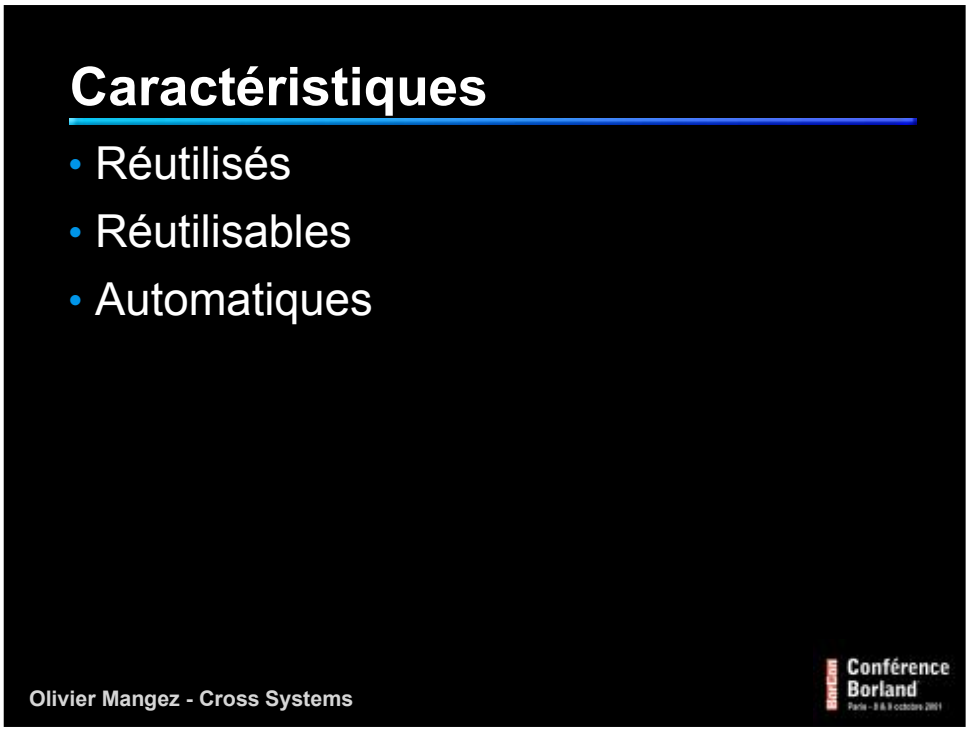


- Les **tests fonctionnels :**
  - menés par la recette, l'équipe de test
  - au niveau du GUI
  - ils répondent à la question :

    « est-ce que ce qu'on demande au code est ce que veut le client ? »

- Les **tests unitaires :**
  - menés par les développeurs
  - au niveau de la classe, de la méthode
  - ils répondent à la question :

    « est-ce que le code fait ce qu'on lui demande ? »



- **Réutiliser** ?
  - Pour éviter les régressions (évolutions, refactoring, collective code ownership)
  - Lancer tous les tests, tous les jours

- Mais réutiliser => **réutilisable**
  - le test ne doit pas envoyer des dizaines de données à interpréter
  - le test ne peut pas être codé dans une fonction main

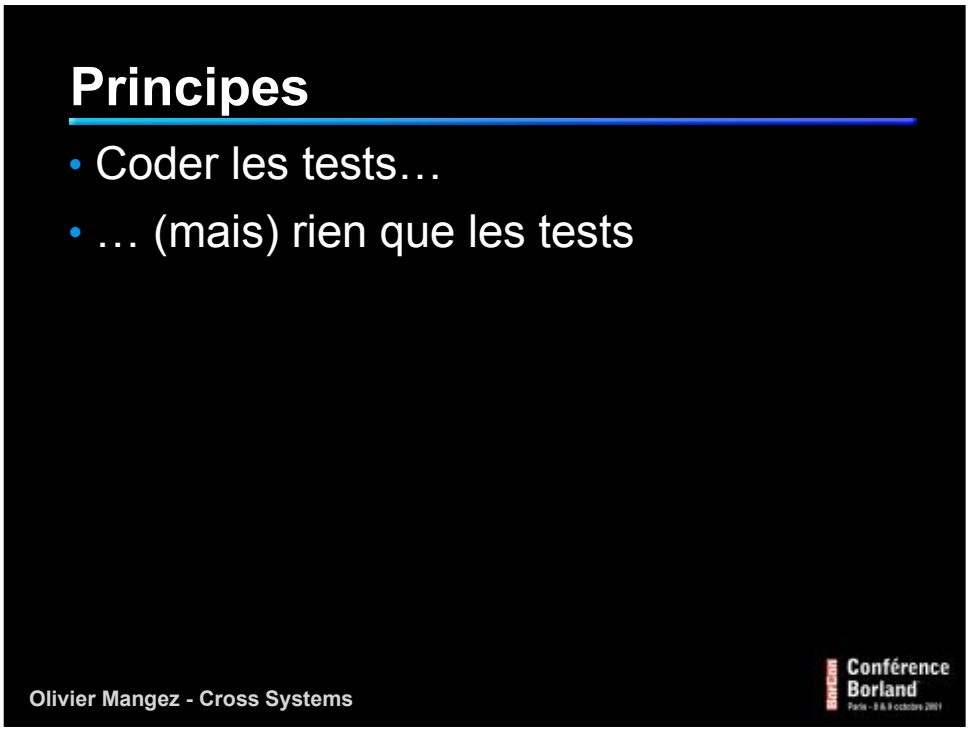
- Donc réutilisable => **automatique**
  - automatisation du lancement
  - automatisation de l'interprétation

Plan

- eXtreme Programming
- Tests
- **JUnit**
- Pour aller plus loin
- Et Delphi dans tout ça ?

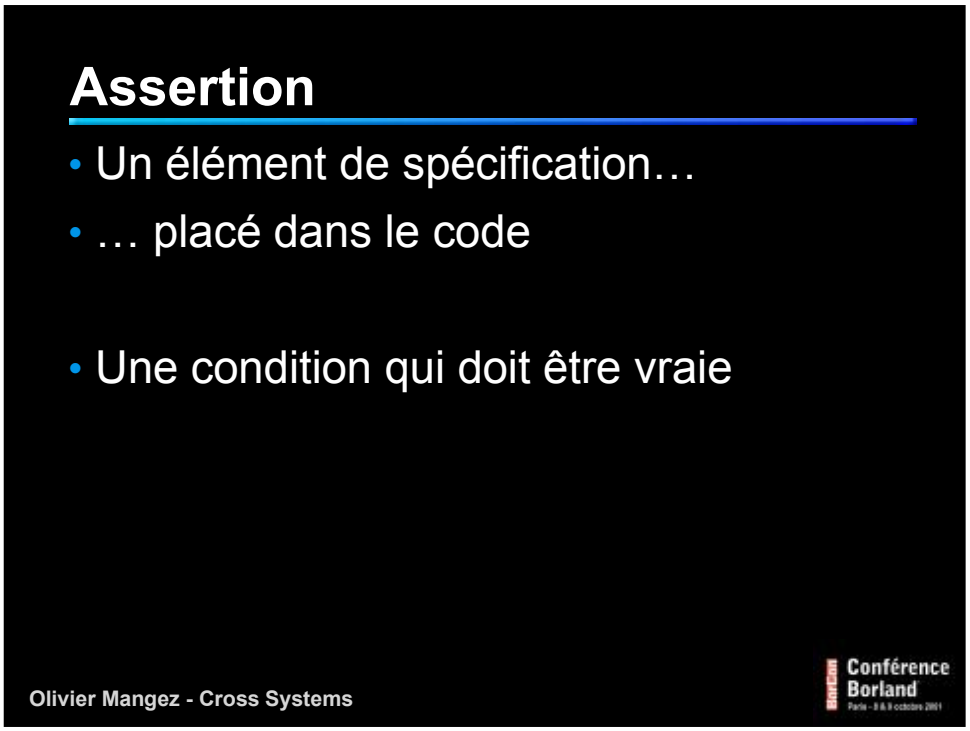
Olivier Mangez - Cross Systems

Conférence Borland



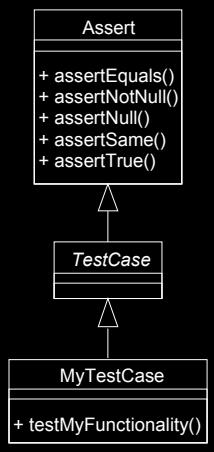
- On veut **automatiser** :
  - le lancement
  - l'interprétation

- Mais ce qui nous intéresse, ce sont les **tests**
  - il nous faut toujours les coder
  - mais c'est la seule chose à coder
  - l'automatisation est déléguée à JUnit



- Une assertion, c'est un **élément de spécification** placé au sein du code.
- Une assertion est caractérisée par une condition booléenne qui doit être vraie.
- Dans le cas d'un test, une assertion fausse entraîne l'échec du test.
- En général, on associe à chaque assertion un message d 'erreur. Si l'assertion est fausse, le message d'erreur est affiché.

Un simple test




```
public class MyTestCase
extends TestCase {

  public MyTestCase(String name) {
    super(name);
  }

  public void testMyFunctionality() {
    …
    assertTrue(…);
    …
  }
}
```

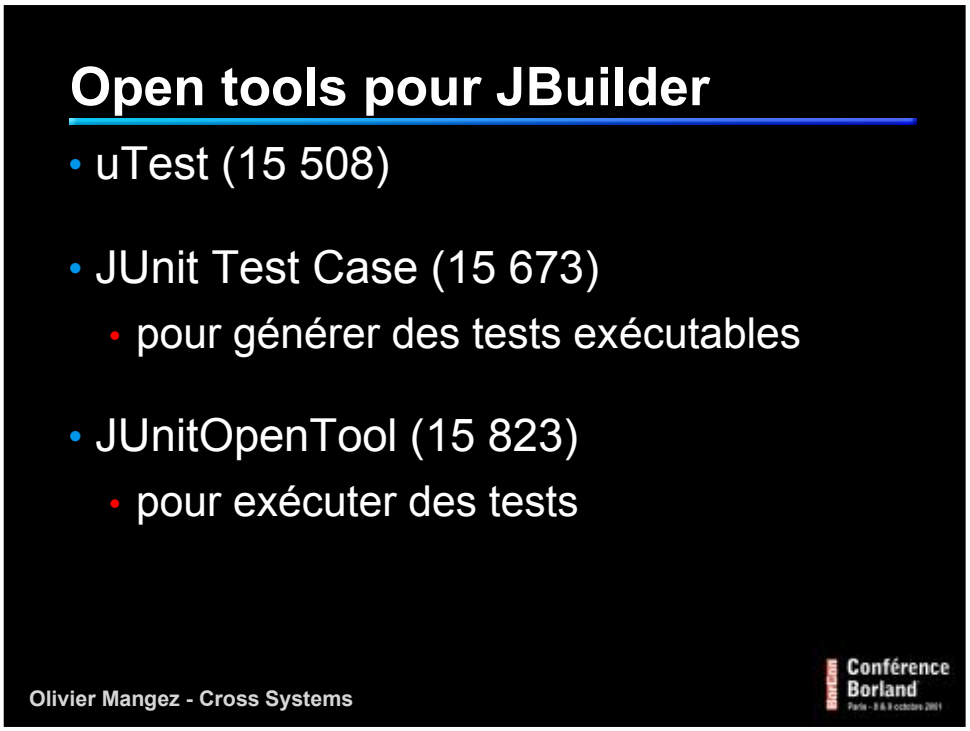
Olivier Mangez - Cross Systems

Conférence Borland

- Dans le framework JUnit, il existe une classe (**`junit.framework.TestCase`**) qui permet d'encapsuler un test. Pour créer un test, il faut donc créer une **classe dérivée** de `TestCase`.

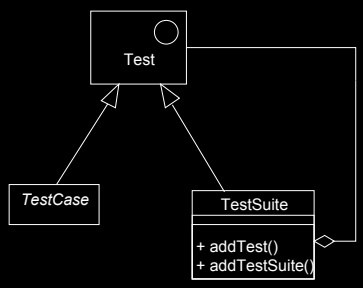
- A cette classe, il faut ajouter une méthode du type **`public void testXXX()`** dans laquelle on placera le test.

- Bien entendu, il faut indiquer les conditions dans lesquelles le test échoue. Pour cela on utilise le mécanisme d'**assertion** fournit par la classe `junit.framework.Assert`.



- JUnit = framework + plateforme de lancement
- OpenTool = module que l 'on ajoute à JBuilder pour lui ajouter des fonctionnalités (à télécharger sur http://codecentral.borland.com)
- Demo : test de Money
  - 1) test de equals
  - 2) test de add
- A une classe à tester, on associe une classe de test dans laquelle on place toutes les méthodes de test. Mais comment lancer tous les tests ?

Un ensemble de tests



Olivier Mangez - Cross Systems

Conférence Borland

- Afin de pouvoir **automatiser l'exécution** des tests, on veut pouvoir tous les exécuter en une seule opération.
- Analogie avec les répertoires :
  - TestCase = fichier
  - TestSuite = répertoire
- Pour cela, il suffit de créer une instance de **`junit.framework.TestSuite`** et de lui ajouter tous les tests à l'aide de la méthode **`addTest()`.**

Plan

- eXtreme Programming
- Tests
- JUnit
- **Pour aller plus loin**
- Et Delphi dans tout ça ?

Olivier Mangez - Cross Systems

Conférence Borland

Coupler les tests

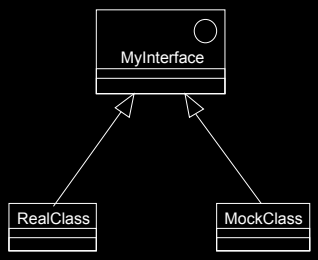
- Impossible de tester :
  - `myObject.save()`
  - `myObject.load(int id)`
- Il suffit de les tester en même temps !

Olivier Mangez - Cross Systems

Conférence Borland

Mock objects

- L’effet Canada Dry



Olivier Mangez - Cross Systems

Conférence Borland

Intégrer JUnit dans la plate-forme

- Intégrer JUnit
  - dans le moteur de servlets
  - dans le serveur d'application
  - ...

Olivier Mangez - Cross Systems

Conférence Borland

Plan

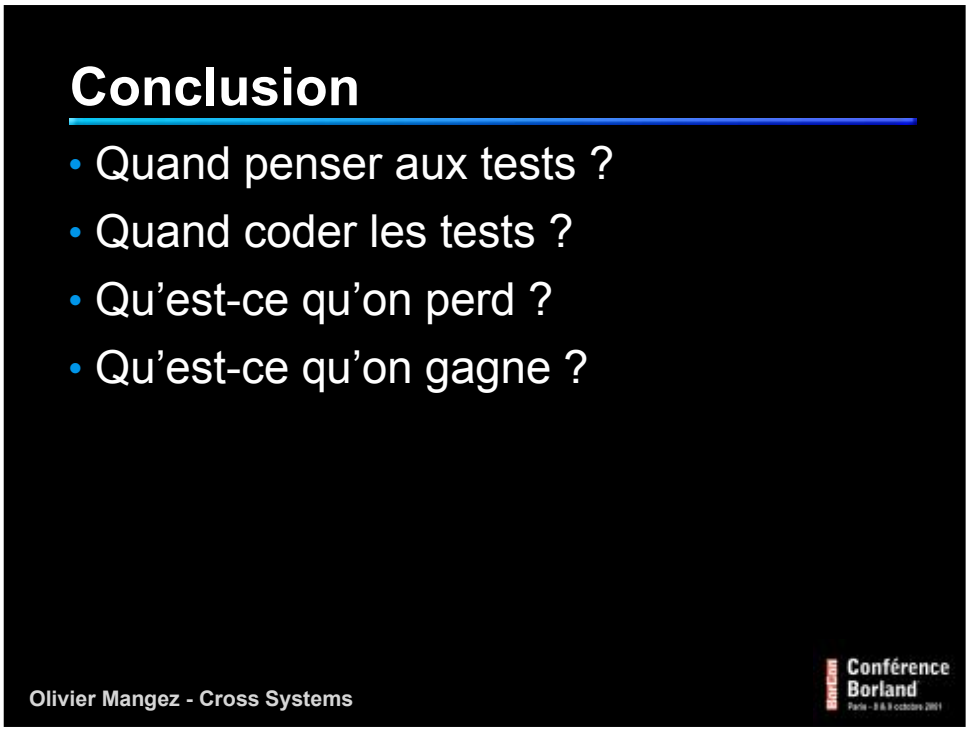
- eXtreme Programming
- Tests
- JUnit
- Pour aller plus loin
- **Et Delphi dans tout ça ?**

Olivier Mangez - Cross Systems

Conférence Borland

- Les frameworks XUnit sont équivalents à JUnit (souvent des noms très proches : TTestCase, ...)


BorCon
This is the end...
BorCon
Conférence
Borland



- Il faut **y penser** dès le départ parce que :
  - cela implique des choix de conceptions
  - cela n 'est pas transparent (interface, equals)

- Il faut **les coder** dès le départ parce que :
  - en fin de projet, on n'a jamais le temps
  - si on a le temps, c'est trop compliqué

- On **perd** un peu de temps au départ…
- … mais on en **gagne** beaucoup à l'arrivée

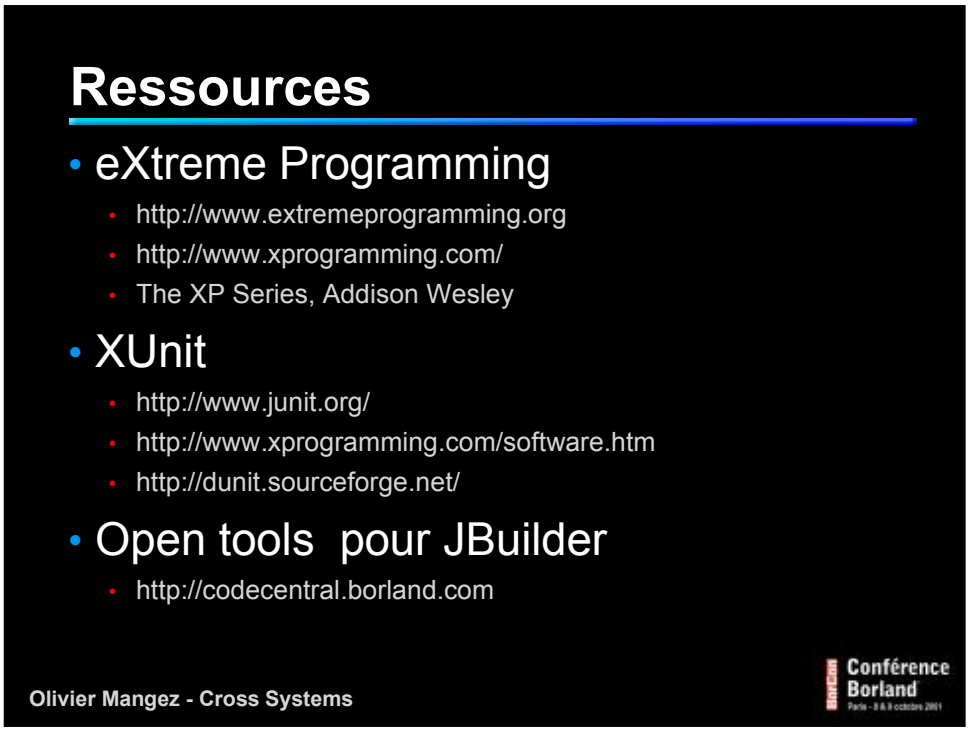
- Et surtout, on atteint son but : la **qualité**
  - parce qu 'on est assuré que le code fonctionne
  - parce que pour tester, il faut bien structurer

Sessions connexes

- **1114 : XP, mythes et réalités**
  - Jean-Louis Bénard
- **1128 : Réalisation d'une plate-forme de test unitaire compatible XP/BAS**
  - Hiram Madelaine
  - Mardi, 10h30
- **3148 : Introduction à la programmation extrême**
  - Olivier Baudin
  - Mardi, 14h45

Olivier Mangez - Cross Systems

Conférence Borland


Ressources
• eXtreme Programming
• http://www.extremeprogramming.org
• http://www.xprogramming.com/
• The XP Series, Addison Wesley
• XUnit
• http://www.junit.org/
• http://www.xprogramming.com/software.htm
• http://dunit.sourceforge.net/
• Open tools pour JBuilder
• http://codecentral.borland.com
Olivier Mangez - Cross Systems
Conférence Borland

Contact

- **Olivier Mangez :**

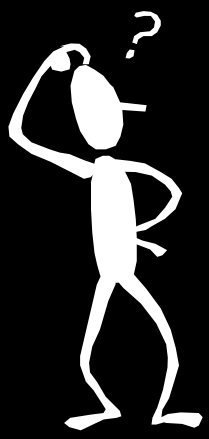
  o.mangez@cross-systems.com

- **Cross Systems :**

  13 rue Fernand Léger, 75020 Paris

  Tel : 01 43 58 61 61
  Fax : 01 43 58 62 43

A vos questions !



Olivier Mangez - Cross Systems

Conférence Borland