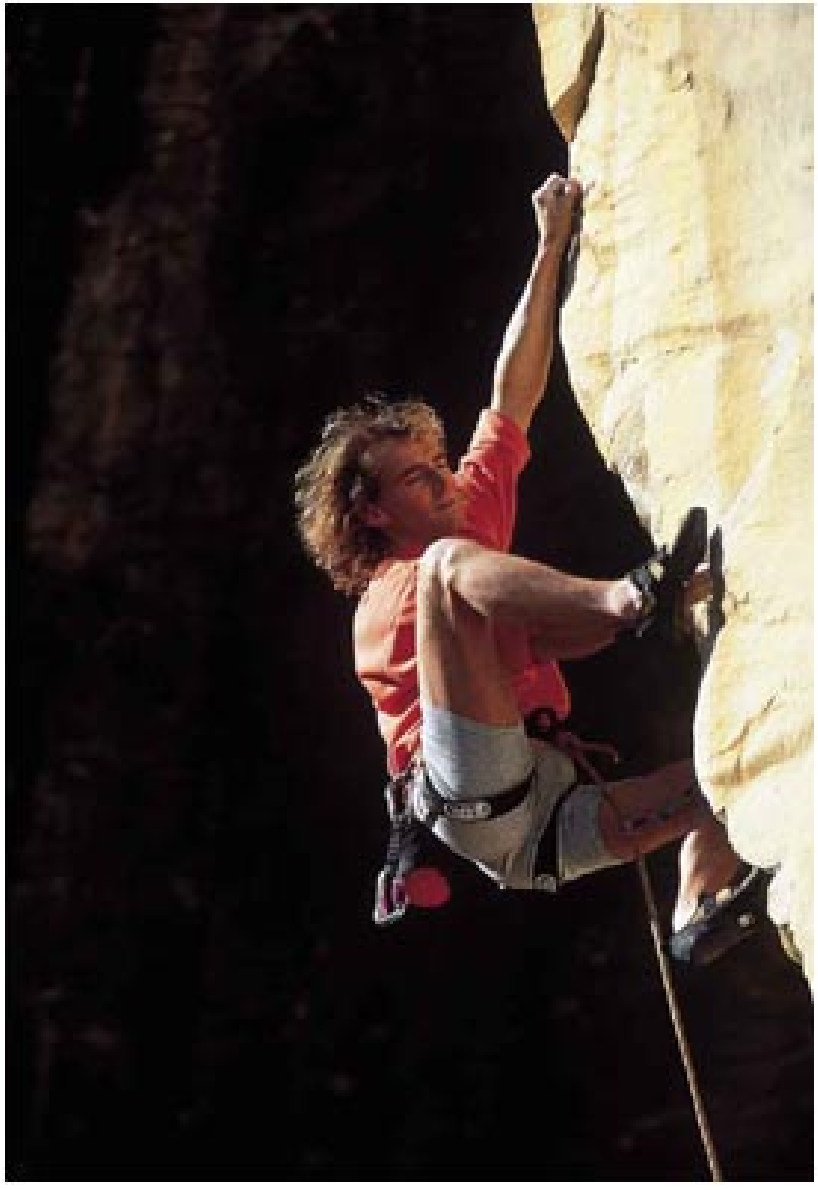
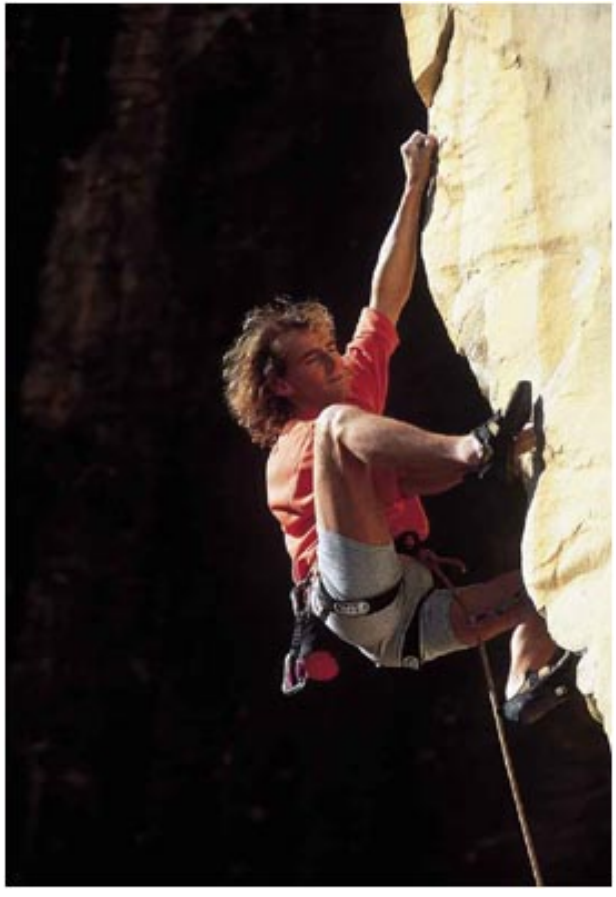
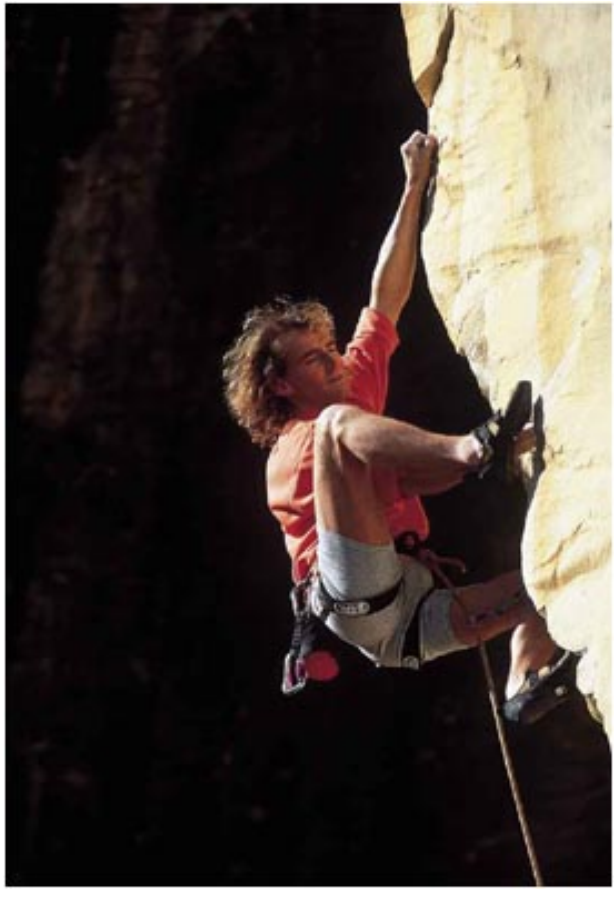
Accros aux tests ? Une introduction au test logiciel



La métaphore du grimpeur

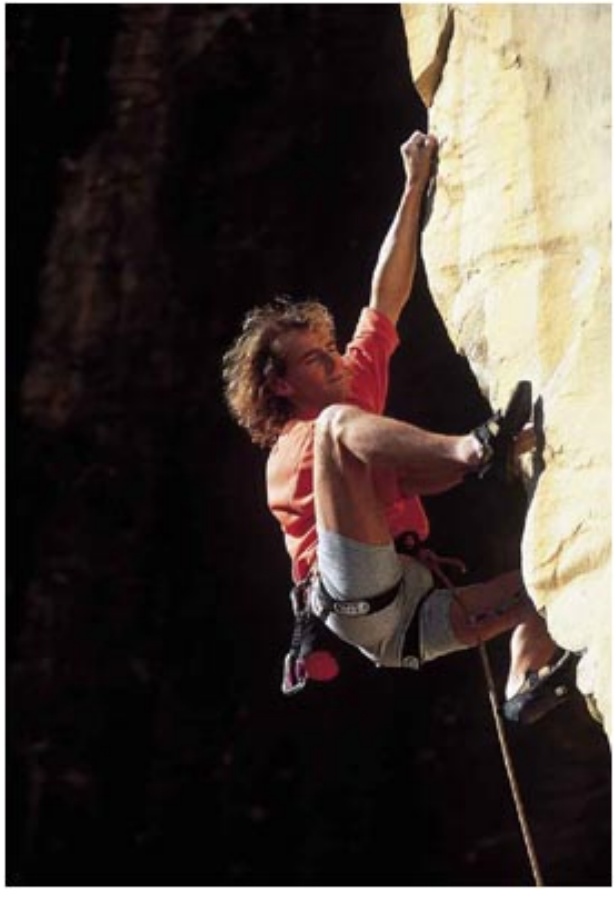


La métaphore du grimpeur



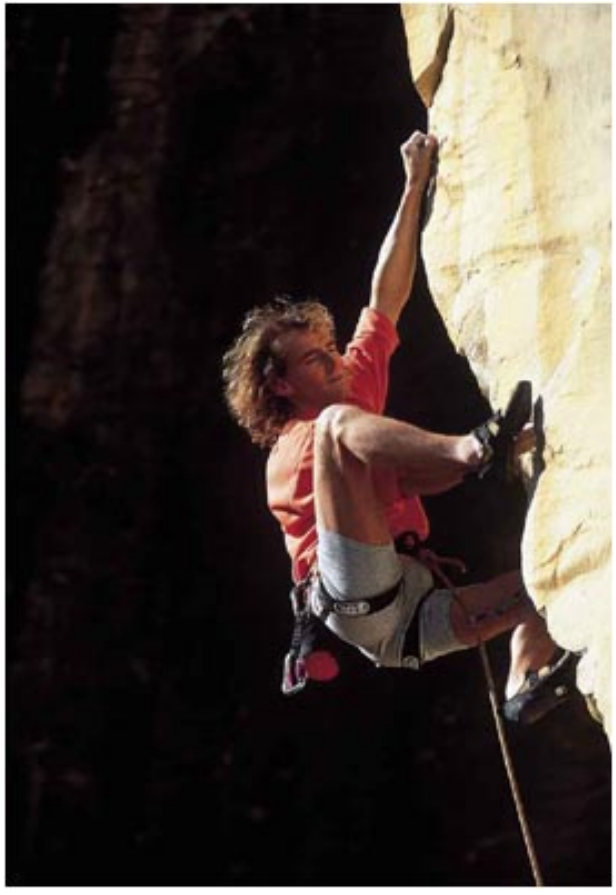
- Qui peut tenter les voies les plus difficiles ?

La métaphore du grimpeur



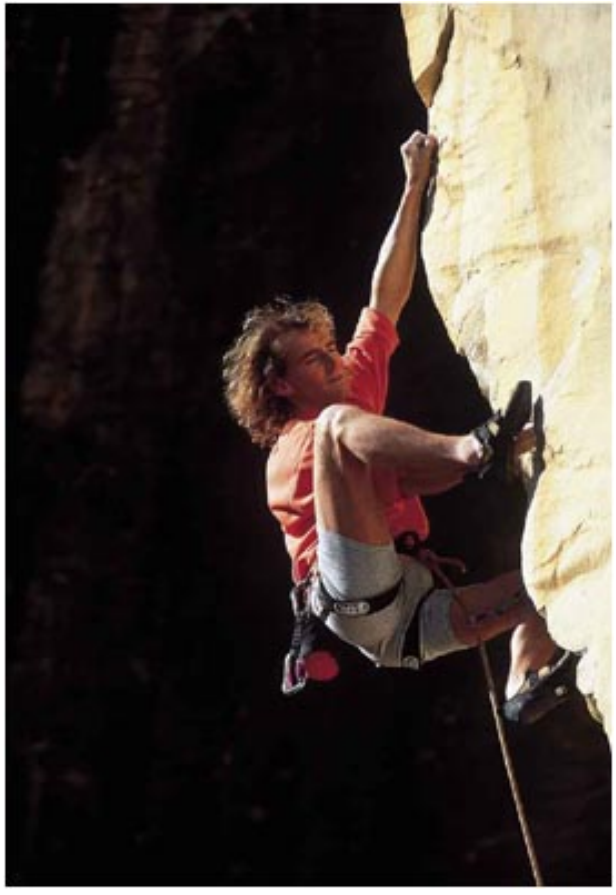
- Qui peut tenter les voies les plus difficiles ?
- Qui peut tenter des mouvements audacieux ?

La métaphore du grimpeur



- Qui peut tenter les voies les plus difficiles ?
- Qui peut tenter des mouvements audacieux ?
- Qui peut expérimenter et innover ?

La métaphore du grimpeur



- Qui peut tenter les voies les plus difficiles ?
- Qui peut tenter des mouvements audacieux ?
- Qui peut expérimenter et innover ?
- Qui se fait plaisir ?

Gestion du risque par les grimpeurs

- Un risque mortel

Gestion du risque par les grimpeurs

- Un risque mortel
- Un système de sécurité pas trop contraignant

Gestion du risque par les grimpeurs

- Un risque mortel
- Un système de sécurité pas trop contraignant
- Tout le monde l'utilise

Gestion du risque par les grimpeurs

- Un risque mortel
- Un système de sécurité pas trop contraignant
- Tout le monde l'utilise
  sauf quelques grimpeurs, par choix délibéré

Gestion du risque par les grimpeurs

- Un risque mortel
- Un système de sécurité pas trop contraignant
- Tout le monde l'utilise
  sauf quelques grimpeurs, par choix délibéré

Autres analogies :

- Ceinture de sécurité
- Casque

Gestion du risque par les grimpeurs

- Un risque mortel
- Un système de sécurité pas trop contraignant
- Tout le monde l'utilise
  sauf quelques grimpeurs, par choix délibéré

Autres analogies :

- Ceinture de sécurité
- Casque

Un bon système de sécurité doit être utilisé systématiquement

Gestion du risque par les grimpeurs

- Un risque mortel
- Un système de sécurité pas trop contraignant
- Tout le monde l'utilise
  sauf quelques grimpeurs, par choix délibéré

Autres analogies :

- Ceinture de sécurité
- Casque

Un bon système de sécurité doit être utilisé systématiquement
Tout système de sécurité est vite abandonné, sauf s'il est :

- Fiable
- Facile à utiliser, voire transparent
- Peu coûteux
- Pas trop obstructif

Les risques pour les logiciels ?

http://fr.wikibooks.org/wiki/Introduction_au_test_logiciel/Introduction

Niveau de sureté

http://fr.wikipedia.org/wiki/DO-178

La mort lente d'un logiciel

Quel est le comble de l'horreur pour un informaticien ?

La mort lente d'un logiciel

Quel est le comble de l'horreur pour un informaticien ?

L'usine à gaz qui explose dès qu'on la touche

La mort lente d'un logiciel

Quel est le comble de l'horreur pour un informaticien ?

L'usine à gaz qui explose dès qu'on la touche

Pour le projet :

- Tout l'énergie part dans la chasse au bogue
- Délais imprévisibles
- Mauvaise qualité externe
- Paralysie de toute évolution

La mort lente d'un logiciel

Quel est le comble de l'horreur pour un informaticien ?

L'usine à gaz qui explose dès qu'on la touche

Pour le projet :

- Tout l'énergie part dans la chasse au bogue
- Délais imprévisibles
- Mauvaise qualité externe
- Paralysie de toute évolution

Pour le développeur :

- Stress
- Asphyxie de la créativité

Approche traditionnelle

- Équipes très hiérarchisées :
  - Analystes programmeurs
  - Équipe de test
  - Développeur

Approche traditionnelle

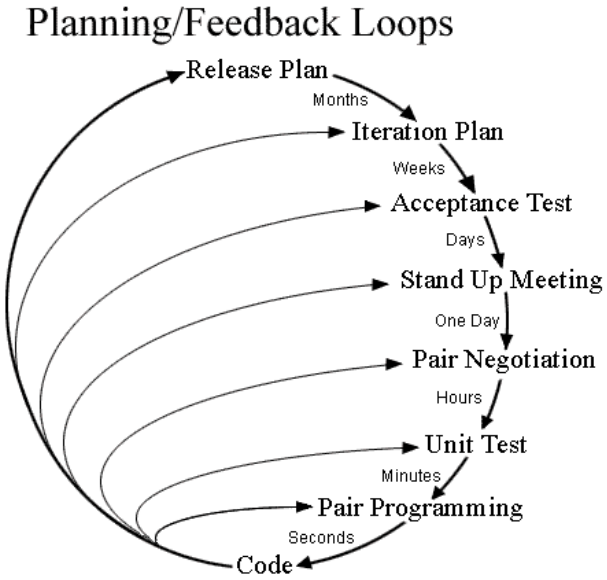
- Équipes très hiérarchisées :
  - Analystes programmeurs
  - Équipe de test
  - Développeur
- Procédures strictes (voire lourdes)

Approche traditionnelle

- Équipes très hiérarchisées :
  - Analystes programmeurs
  - Équipe de test
  - Développeur
- Procédures strictes (voire lourdes)
- Évolution lente, planifié longuement à l'avance

Méthodes agiles

`fr.wikipedia.org/wiki/Extreme_programming`

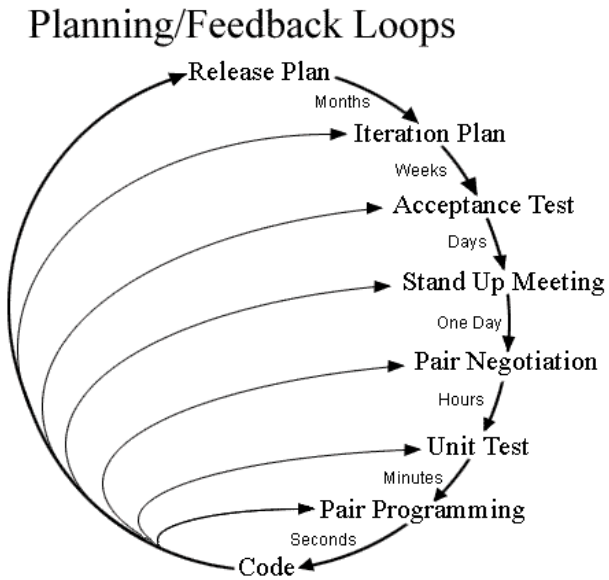


Objectif : renforcer l'agilité du développeur :

- Créativité
- Responsabilité
- Auto assurance : Tests !

Méthodes agiles

`fr.wikipedia.org/wiki/Extreme_programming`



Objectif : renforcer l'agilité du développeur :

- Créativité
- Responsabilité
- Auto assurance : Tests !

Les tests sont votre outil de libération !

Les objectifs et types de tests

- Mesurer la qualité du code

Les objectifs et types de tests

- Mesurer la qualité du code
- Mesurer les progrès

Les objectifs et types de tests

- Mesurer la qualité du code
- Mesurer les progrès
- Formaliser (et anticiper !) les demandes du client
  Tests fonctionnels
  Cahier de recette
  `http://fr.wikipedia.org/wiki/Recette_%28informatique%29`

Les objectifs et types de tests

- Mesurer la qualité du code
- Mesurer les progrès
- Formaliser (et anticiper !) les demandes du client
  Tests fonctionnels
  Cahier de recette
  `http://fr.wikipedia.org/wiki/Recette_%28informatique%29`
- Garantir la robustesse d'une brique avant d'empiler dessus
  Tests unitaires

Les objectifs et types de tests

- Mesurer la qualité du code
- Mesurer les progrès
- Formaliser (et anticiper !) les demandes du client
  Tests fonctionnels
  Cahier de recette
  `http://fr.wikipedia.org/wiki/Recette_%28informatique%29`
- Garantir la robustesse d'une brique avant d'empiler dessus
  Tests unitaires
- Anticiper la mise en commun de plusieurs briques
  Tests d'intégration

Les objectifs et types de tests

- Mesurer la qualité du code
- Mesurer les progrès
- Formaliser (et anticiper !) les demandes du client
  Tests fonctionnels
  Cahier de recette
  `http://fr.wikipedia.org/wiki/Recette_%28informatique%29`
- Garantir la robustesse d'une brique avant d'empiler dessus
  Tests unitaires
- Anticiper la mise en commun de plusieurs briques
  Tests d'intégration
- Anticiper la mise en production
  Tests de charge
- Alerte immédiate si l'on introduit un bogue
  Tests de non régression

Tests unitaires, pour le développeur

« Les tests, c'est bien, mais je n'ai pas le temps »

Tests unitaires, pour le développeur

« Les tests, c'est bien, mais je n'ai pas le temps »

- Tester une fois, tester toujours

Tests unitaires, pour le développeur

« Les tests, c'est bien, mais je n'ai pas le temps »

- Tester une fois, tester toujours
  Automatisation

Tests unitaires, pour le développeur

« Les tests, c'est bien, mais je n'ai pas le temps »

- Tester une fois, tester toujours
  Automatisation
- Infrastructure légère

Tests unitaires, pour le développeur

« Les tests, c'est bien, mais je n'ai pas le temps »

- Tester une fois, tester toujours
  Automatisation
- Infrastructure légère
  Exemple : `JUnit`

Exemple : la classe « Argent » (spécifications)

- Une instance de la classe `Argent` représente une somme d'argent dans une monnaie déterminée.
- Deux sommes d'argent sont identiques si elles représentent le même montant dans la même monnaie.
- La classe `Argent` contient :
  - un constructeur
  - deux accesseurs
  - une méthode `ajoute` qui permet de créer la somme d'argent résultant de l'ajout à la somme d'argent courante une autre somme de la même monnaie.

  Les prototypes sont les suivants :

```
public Argent(double montant, String monnaie
public double montant() ;
public String monnaie() ;
public Argent ajoute(Argent a) ;
```

La classe de test d'« Argent »

```
package monnaie ;
import junit.framework.TestCase ;

public class ArgentTest extends TestCase {

        Argent huitEuros, deuxEuros ;

        protected void setUp() throws Exception {
                // Initialisation
                super.setUp() ;
                huitEuros= new Argent(8,"euros") ;
                deuxEuros= new Argent(2, "euros") ;
        }

        /*
         * Test method for 'monnaie.Argent.ajoute(Argent)'
         */
        public void testAjoute() {
                // Invocation de la méthode à tester
                Argent resultat= huitEuros.ajoute(deuxEuros) ;
                // Analyse du résultat
                Argent attendu=new Argent(10, "euros") ;
                assertTrue(attendu.equals(resultat)) ;
        }
}
```

Écriture de la classe `Argent`

Démonstration sous éclipse

Principe du test `JUnit`

- La méthode `setUp` est appelée
- La première méthode de test est appelée
- La méthode `tearDown` est appelée
- La méthode `setUp` est appelée
- La seconde méthode de test est appelée
- La méthode `tearDown` est appelée
- Etc.

Développement piloté par les tests

`http://fr.wikipedia.org/wiki/Test_Driven_Development`

Durant le développement

Pour ajouter une nouvelle fonctionnalité :

- Écrire les spécifications (typiquement sous forme de javadoc !)
- Écrire le test correspondant
- Attention aux cas particuliers !
- Le développement est terminé lorsque les tests passent

Durant le débogage

Pour corriger un bogue signalé :

- Écrire un test qui met en évidence le bogue
- Le débogage est terminé quand les tests passent

Réflexions sur le test

Un bon test est un test qui échoue alors qu'on le croyait facile à passer (et vice versa)

Réflexions sur le test

Un bon test est un test qui échoue alors qu'on le croyait facile à passer (et vice versa)

Réflexions sur le test

Un bon test est un test qui échoue alors qu'on le croyait facile à passer (et vice versa)

Réflexions sur le test

Un bon test est un test qui échoue alors qu'on le croyait facile à passer (et vice versa)

Si tous les tests passent ce n'est pas que mon programme fonctionne mais que les tests sont mal choisis