

Architecture des ordinateurs

A. Une motivation : performances	4
B. Mini historique	12
C. Des transistors aux unités programmables	32
D. Mémoire	37
E. Accro aux tests ? Une micro introduction au test logiciel	45
Pourquoi le test logiciel ?	
Les risques en informatique	
Il faut aller plus loin ! ♣	
F. Conclusion : qu'avons nous vu ce semestre ?	65

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les concepts suivants :

- Bases de la programmation impérative
- Génie logiciel : programmation modulaire, tests
- Complexité

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les concepts suivants :

- Bases de la programmation impérative
- Génie logiciel : programmation modulaire, tests
- Complexité

Pourquoi aller plus loin ?

Résumé des épisodes précédents ...

Pour le moment nous avons vu les concepts suivants :

- Bases de la programmation impérative
- Génie logiciel : programmation modulaire, tests
- Complexité

Pourquoi aller plus loin ?

Passage à l'échelle :

Résumé des épisodes précédents ...

Pour le moment nous avons vu les concepts suivants :

- Bases de la programmation impérative
- Génie logiciel : programmation modulaire, tests
- Complexité

Pourquoi aller plus loin ?

Passage à l'échelle :

Performances !

A. Une motivation : performances

performances.cpp

```
for ( int i = 0; i < n; i++ )  
    for ( int j = 0; j < n; j++ )  
        t[i][j] = i+j;
```

A. Une motivation : performances

performances.cpp

```
for ( int i = 0; i < n; i++ )
    for ( int j = 0; j < n; j++ )
        t[i][j] = i+j;
```

Complexité ?

A. Une motivation : performances

performances.cpp

```
for ( int i = 0; i < n; i++ )  
    for ( int j = 0; j < n; j++ )  
        t[i][j] = i+j;
```

Complexité? $O(n^2)$

A. Une motivation : performances

performances.cpp

```
for ( int i = 0; i < n; i++ )  
    for ( int j = 0; j < n; j++ )  
        t[i][j] = i+j;
```

Complexité? $O(n^2)$

Voyons cela en pratique :

```
> g++ -std=c++11 -O9 performances.cpp -o performances
```

```
> time performances 5000  
performances 5000  0,06s user 0,06s system 98% cpu 0,130 total  
  
> time performances 10000  
performances 10000  0,35s user 0,26s system 99% cpu 0,610 total  
  
> time performances 20000  
performances 20000  1,46s user 1,09s system 99% cpu 2,560 total
```

Analyse du banc d'essai

Jusqu'ici, tout va bien :

– La complexité pratique est de $O(n^2)$:

taille $n \times 2 \implies$ temps $\times 4$

Analyse du banc d'essai

Jusqu'ici, tout va bien :

– La complexité pratique est de $O(n^2)$:

$$\text{taille } n \times 2 \implies \text{temps } \times 4$$

– Environ 10^9 opérations par secondes

```
> time performances 10000
performances 10000 0,35s user 0,26s system 99% cpu 0,610 total
```

Analyse du banc d'essai

Jusqu'ici, tout va bien :

- La complexité pratique est de $O(n^2)$:
taille $n \times 2 \implies$ temps $\times 4$
- Environ 10^9 opérations par secondes

```
> time performances 10000
performances 10000 0,35s user 0,26s system 99% cpu 0,610 total
```

Une bizarrerie!?!

- Avec $t[i][j] \rightarrow t[j][i]$: 10 fois plus lent!

```
> time performances 10000
performances 10000 3,47s user 0,18s system 99% cpu 3,650 total
```

Bilan

- Notre **modèle d'exécution** ne rend pas compte de ce phénomène
- Pour le raffiner il faut mieux connaître
l'architecture des ordinateurs

La préhistoire des ordinateurs

- 1614, John Neper invente les logarithmes et la **règle à calcul** :



La préhistoire des ordinateurs

- 1614, John Neper invente les logarithmes et la **règle à calcul** :



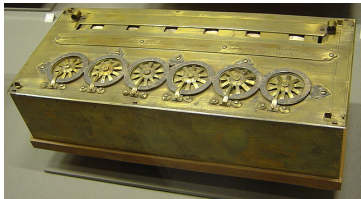
- 1623, Wilhelm Schickard construit une machine mécanique en appliquant les idées de Neper

La préhistoire des ordinateurs

- 1614, John Neper invente les logarithmes et la **règle à calcul** :



- 1623, Wilhelm Schickard construit une machine mécanique en appliquant les idées de Neper
- 1642, Pascal présente une machine qui additionne et soustrait les nombres de 6 chiffres en base 10 : **la Pascaline**
Exemplaire du Musée des Arts et Métiers :



La préhistoire des ordinateurs

- 1614, John Neper invente les logarithmes et la **règle à calcul** :



- 1623, Wilhelm Schickard construit une machine mécanique en appliquant les idées de Neper
- 1642, Pascal présente une machine qui additionne et soustrait les nombres de 6 chiffres en base 10 : **la Pascaline**
Exemplaire du Musée des Arts et Métiers :



- 1672, amélioration par Leibnitz :
chariot mobile pour multiplications et divisions

Le XIX^e siècle

- 1805, Joseph Jacquard (d'après des idée de Falcon en 1728) : cartes perforées pour métiers à tisser : c'est le premier **programme**.

Le XIX^e siècle

- 1805, Joseph Jacquard (d'après des idées de Falcon en 1728) : cartes perforées pour métiers à tisser : c'est le premier **programme**.
- 1822, Charles Babbage essaye de faire une **Machine Analytique**
 - machine pilotée par cartes perforées décrivant la séquence d'instructions (programme extérieur) ;
 - notion de processeur, de mémoire (magasin), d'unité centrale (moulin) ;
 - entrée et sortie par cartes perforées.
 - programmes d'**Ada Augusta Lovelace**
 - ne fonctionnera jamais

Le XIX^e siècle

- 1805, Joseph Jacquard (d'après des idées de Falcon en 1728) : cartes perforées pour métiers à tisser : c'est le premier **programme**.
- 1822, Charles Babbage essaye de faire une **Machine Analytique**
 - machine pilotée par cartes perforées décrivant la séquence d'instructions (programme extérieur) ;
 - notion de processeur, de mémoire (magasin), d'unité centrale (moulin) ;
 - entrée et sortie par cartes perforées.
 - programmes d'**Ada Augusta Lovelace**
 - ne fonctionnera jamais
- 1854, **Machine à Différences** (Babbage et Scheutz).

Le XIX^e siècle (suite)

- 1854, Georges Bool, **Une étude des lois de la pensée**
Calcul des propositions, logique élémentaire
- Calculateur statistique d'Hermann Hollerith pour les recensements
- Fondation en 1890 de **Tabulating Machine Company** qui devient en 1908 **International Business Machine**

Le XX^e siècle : les idées fondatrices

- 1930, Vannevar Bush : analyseur différentiel analogique, utilisé jusque dans les années 1960
- 1936, Alan Turing et les **Machines de Turing**
Ce que l'on peut calculer et ce que l'on ne peut pas

Le XX^e siècle : les idées fondatrices

- 1930, Vannevar Bush : analyseur différentiel analogique, utilisé jusque dans les années 1960
- 1936, Alan Turing et les **Machines de Turing**
Ce que l'on peut calculer et ce que l'on ne peut pas
- 1938, Claude Shannon invente la **Théorie de l'information** :
tout peut être représenté par des 0 et des 1
c'est le début de la numérisation

Le XX^e siècle : balbutiements

- 1938, Konrad Zuse, John Atanasoff, Georges Stibitz :
Machine à calculer électromécanique
(Zuse : tubes à vide, jugée irréalisable par le gouvernement allemand)
- 1939 La machine électromécanique Enigma (cryptographie)
- 1939-1944, Howard Aiken, machine électromécanique :
 - Multiplication de nombres de 23 chiffres en 6 secondes
 - Addition en 3 dixièmes de seconde

Naissance de l'ordinateur : 1945

- 1945, John Eckert et John Mauchly construisent l'**ENIAC**
(Electronic Numerical Integrator And Calculator) :
18000 tubes, 30 tonnes
Multiplication de nombres de 10 chiffres en 3ms
- 1945, Von Neumann propose l'**EDVAC**
(Electronic Discrete Variable Automatic Computer) :
Problèmes de retards et de brevets
- 1949, Maurice Wilkes construit l'**EDSAC**
(Electronic Delay Storage Automatic Calculator)

Ordinateurs de 1^{re} génération, 1945–1953

- Composants : relais, tubes à vides, résistances
- Logiciels : langage machine seulement
- 1951, Eckert et Mauchly, construisent l'**UNIVAC**
- 1953, IBM 701 (40 exemplaires) puis IBM 650 (1500 exemplaires)
- **Problème de fiabilité des tubes à vides...**

Industrie de l'informatique, 2^e génération, 1953–1963

- 1948, John Bardeen, Walter Brattain et William Shockley découvrent **le transistor**
- Composants :
 - transistors, mémoire à tores de ferrite
 - imprimantes, bandes magnétiques
- Logiciels :
 - apparition des systèmes d'exploitation
 - langages évolués FORmula TRANslator (1957)
 - COmmon Business Oriented Language (1959)
- Apparition de l'industrie, IBM, DEC, HP, etc.

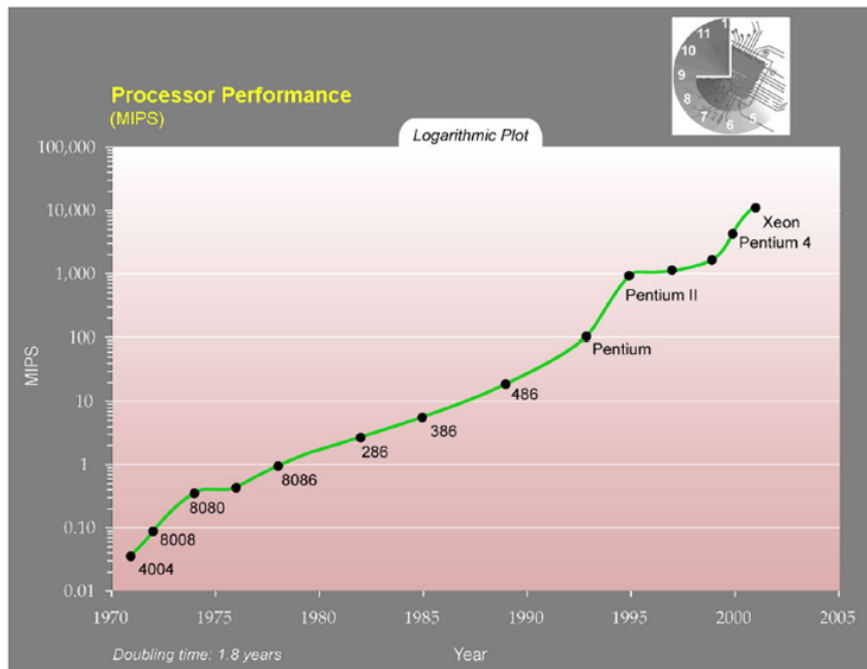
Industrie de l'informatique, 3^e génération, 1963–1975

- Composants : circuits intégrés
- Machine :
 - faible consommation énergétique
 - fiable, encombrement réduit
- Évolutions :
 - Multiprocesseur, temps partagé, accès interactif
 - Apparition des réseaux
 - Premiers problèmes de compatibilité entre machines

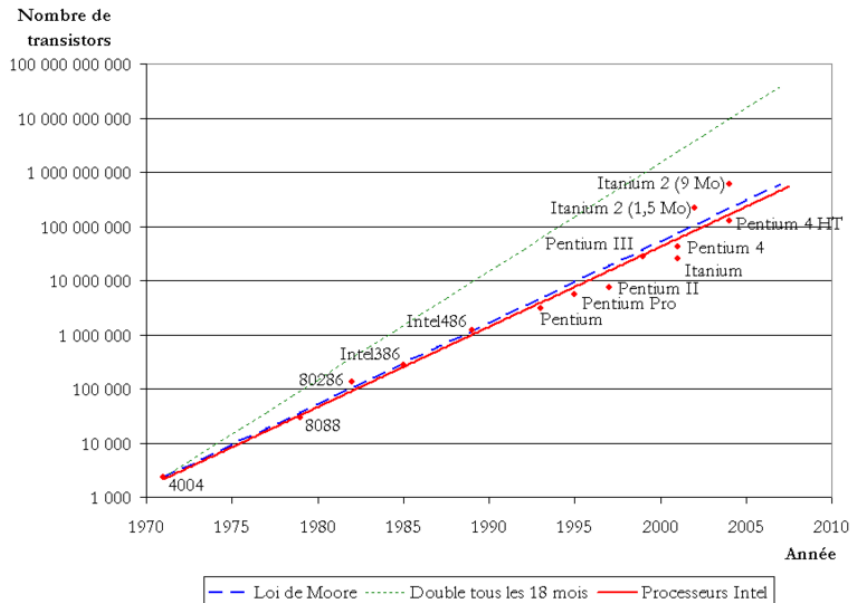
Industrie de l'informatique, 4^{ème} génération, 1975–?

- Composants :
 - Very Large Scale Integration :
Centaines de milliers puis millions de transistors sur une puce !
 - Premier microprocesseur INTEL 4004 (1971)
- Logiciel
 - Traitement distribué, machine virtuelle
 - Réseau, base de données
 - Accessibilité au grand public
- Évolution :
 - Parallélisme d'exécution (pipe-line, vectorisation)
 - Ordinateurs personnels
 - Augmentation en puissance

Évolution : performance des processeurs



Évolution : nombre de transistors par processeur



Loi de Moore : doublement tous les 18 mois

Jusqu'à quand ?

On atteint les limites physiques

- Transistors à l'échelle de quelques centaines d'atomes
- Prix des usines qui double à chaque itération

Jusqu'à quand ?

On atteint les limites physiques

- Transistors à l'échelle de quelques centaines d'atomes
- Prix des usines qui double à chaque itération

Évolution récente : parallélisme d'exécution

- Multicoeur / multiprocesseur
- Pipe-line
- Fermes d'ordinateurs (par ex. dans le « nuage »)

Mais : **Principes de Von Neumann toujours valables**

C. Des transistors aux unités programmables

Idée clef

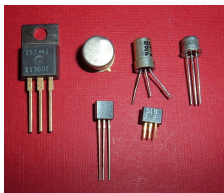
Un bit va être représenté par une tension en un point d'un circuit électronique

Par exemple :

- 1 : +5V
- 0 : +0V

Le transistor : un interrupteur commandé

Composant électronique :



Miniaturisation :

Une puce électronique : des milliards de transistors

Principe de fonctionnement

Des transistors aux portes logiques

Exemple : portes « et »

Exemple : portes « ou »

Exemple : portes « non »

Des portes logiques aux unités de calcul

Exemple : un additionneur un bit

Exemple : un additionneur deux bits

Unités programmables

Jeu d'instructions d'un processeur

Ensemble des instructions reconnues par l'unité de contrôle

Par exemple (ici sous forme d'**assembleur**) :

- `mov a, b` : copier le contenu de la case a dans la case b
- `imul a, b` : multiplier le contenu de a par celui de b et mettre le résultat dans b
- ...

Architecture du processeur

- En principe, un programme binaire fonctionnera sur tout processeur supportant le même jeu d'instruction
- Pour cela, on appelle souvent le jeu d'instruction l'**architecture du processeur**
- Exemples : x86, x86-64, ARM, sparc, ...

D. Mémoire

Niveau physique : mémoriser un bit ?

D. Mémoire

Niveau physique : mémoriser un bit ?

Exemple : un condensateur

- Principe : charger électriquement
- Problème : il y a des pertes
 - ⇒ Rafraîchissement régulier nécessaire !
- Application : mémoire vive

D. Mémoire

Niveau physique : mémoriser un bit ?

Exemple : un condensateur

- Principe : charger électriquement
- Problème : il y a des pertes
⇒ Rafraîchissement régulier nécessaire !
- Application : mémoire vive

Exemple :

- Principe : magnétiser un composant
- Problème : plus lent
- Application : mémoire permanente :
disque durs, mémoire flash : clefs USB, disques SDRAM

D. Mémoire

Niveau physique : mémoriser un bit ?

Exemple : un condensateur

- Principe : charger électriquement
- Problème : il y a des pertes
⇒ Rafraîchissement régulier nécessaire !
- Application : mémoire vive

Exemple :

- Principe : magnétiser un composant
- Problème : plus lent
- Application : mémoire permanente :
disque durs, mémoire flash : clefs USB, disques SDRAM

Problème

Changer d'état consomme de l'énergie

⇒ *principale cause de consommation d'un ordinateur*

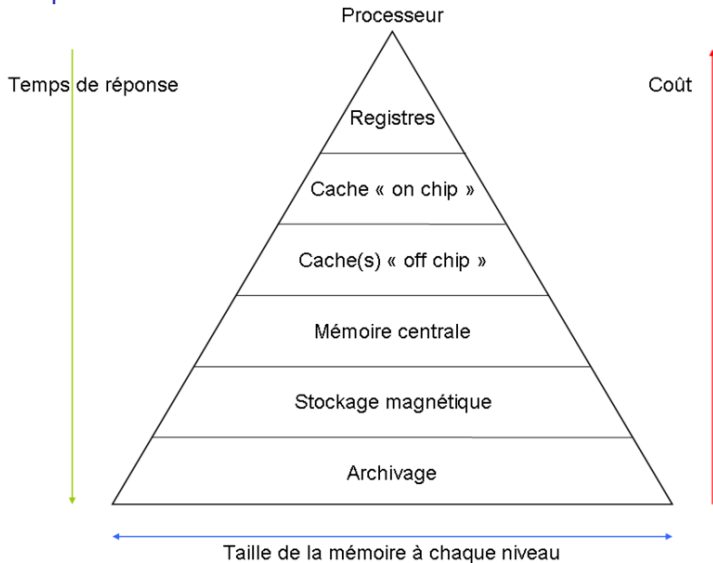
Les niveaux de mémoire

Problème

- La mémoire très rapide est très chère
- La lumière ne va pas vite!!!
à 300000km/s et 1GHz : 30 cm par cycle d'horloge
- Ondes électromagnétiques dans des circuits :
nettement plus lent

Les niveaux de mémoire

Solution : plusieurs niveaux de mémoire



Explication de la bizarrerie

[performances.cpp](#)

```
for ( int i = 0; i < n; i++ )
    for ( int j = 0; j < n; j++ )
        t[i][j] = i+j;
```

Analogie : parcourir une encyclopédie en 10 tomes

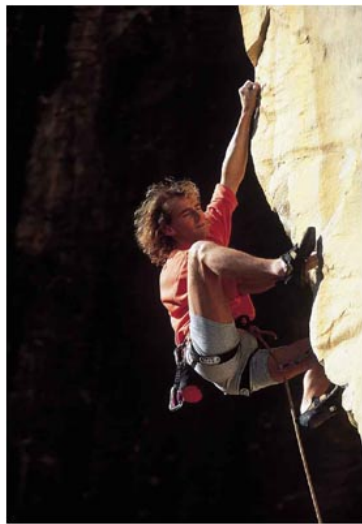
Fautes de caches

Architecture système ?

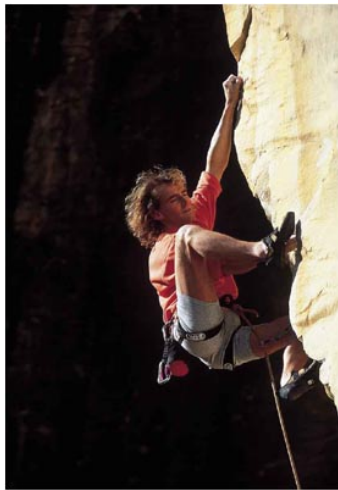
Trois niveaux possibles

1. Vue générale
 - Agencement des divers composants (processeur, mémoire, ...)
 - relations, communications entre eux
2. Niveau macroscopique
 - Structure interne des composants
 - Jeu d'instructions
3. Niveau microscopique
 - composants électriques / électroniques
 - présentés sous leur aspect logique

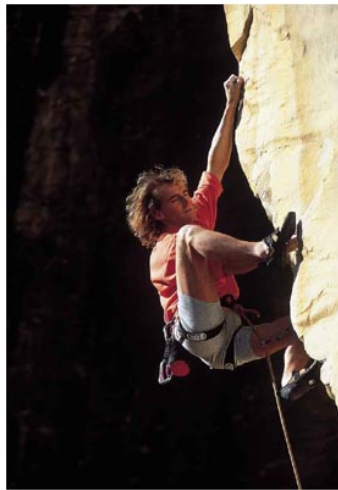
E. Accro aux tests ? Une micro introduction au test logiciel



La métaphore du grimpeur

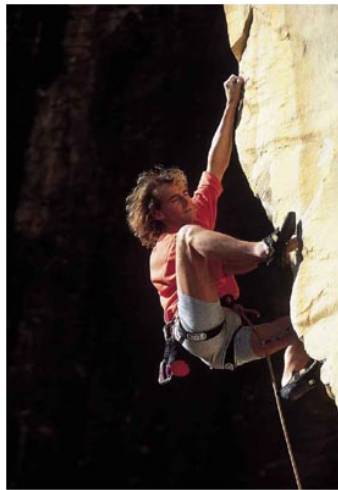


La métaphore du grimpeur



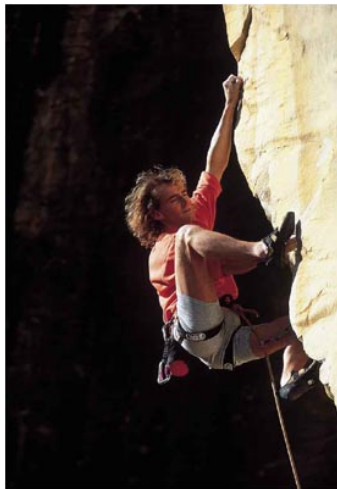
- Qui peut tenter les voies les plus difficiles ?

La métaphore du grimpeur



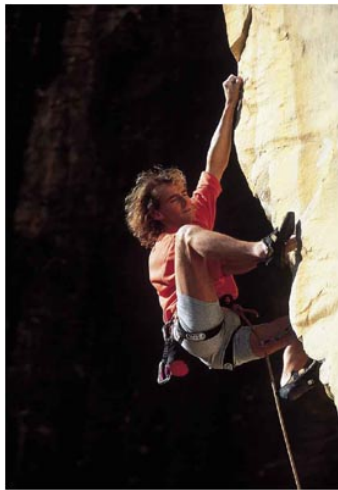
- Qui peut tenter les voies les plus difficiles ?
- Qui peut tenter des mouvements audacieux ?

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 ?

E.2. Les risques en informatique

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

E.2. Les risques en informatique

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

Niveau de sureté

Exemple : normes pour l'aviation :

<http://fr.wikipedia.org/wiki/D0-178>

Les risques en informatique, pour tous

- Je soutiens mon projet dans 5 minutes. Ma démo va-t'elle marcher ?

Les risques en informatique, pour tous

- Je soutiens mon projet dans 5 minutes. Ma démo va-t'elle marcher ?

Stress

Les risques en informatique, pour tous

- Je soutiens mon projet dans 5 minutes. Ma démo va-t'elle marcher ?
Stress
- Je suis tenté de changer xxx ; vais-je tout casser ?

Les risques en informatique, pour tous

- Je soutiens mon projet dans 5 minutes. Ma démo va-t'elle marcher ?

Stress

- Je suis tenté de changer xxx ; vais-je tout casser ?

Stress

Les risques en informatique, pour tous

- Je soutiens mon projet dans 5 minutes. Ma démo va-t'elle marcher ?

Stress

- Je suis tenté de changer xxx ; vais-je tout casser ?

Stress

Asphyxie de la créativité

Vos filets de sécurité

Les tests automatiques (ASSERT)

- Mesure de la robustesse de votre code

Vos filets de sécurité

Les tests automatiques (ASSERT)

- Mesure de la robustesse de votre code

La gestion de version (git, mercurial, ...)

- Principe : sauvegarder les version intermédiaires

Vos filets de sécurité

Les tests automatiques (ASSERT)

- Mesure de la robustesse de votre code

La gestion de version (git, mercurial, ...)

- Principe : sauvegarder les version intermédiaires
- Tout devient réversible
- Retour à la dernière version qui marche ; ouf !

Vos filets de sécurité

Les tests automatiques (ASSERT)

- Mesure de la robustesse de votre code

La gestion de version (git, mercurial, ...)

- Principe : sauvegarder les version intermédiaires
- Tout devient réversible
- Retour à la dernière version qui marche ; ouf !
- Travail collaboratif
- Apprenez à l'utiliser !

Vos filets de sécurité

Les tests automatiques (ASSERT)

- Mesure de la robustesse de votre code

La gestion de version (git, mercurial, ...)

- Principe : sauvegarder les version intermédiaires
- Tout devient réversible
- Retour à la dernière version qui marche ; ouf !
- Travail collaboratif
- Apprenez à l'utiliser !

Programmation incrémentale

- Un petit changement à la fois

Vos filets de sécurité

Les tests automatiques (ASSERT)

- Mesure de la robustesse de votre code

La gestion de version (git, mercurial, ...)

- Principe : sauvegarder les version intermédiaires
- Tout devient réversible
- Retour à la dernière version qui marche ; ouf !
- Travail collaboratif
- Apprenez à l'utiliser !

Programmation incrémentale

- Un petit changement à la fois

Bases des **Méthodes agiles**

- Remettre le développeur au coeur du processus
- http://fr.wikipedia.org/wiki/Extreme_programming

E.3. Il faut aller plus loin ! ♣

- Article « infecté par les tests »

[http:](http://junit.sourceforge.net/doc/testinfected/testing.htm)

[//junit.sourceforge.net/doc/testinfected/testing.htm](http://junit.sourceforge.net/doc/testinfected/testing.htm)

- [http:](http://fr.wikibooks.org/wiki/Introduction_au_test_logiciel)

[//fr.wikibooks.org/wiki/Introduction_au_test_logiciel](http://fr.wikibooks.org/wiki/Introduction_au_test_logiciel)

- Tests unitaires en Java :

<http://junit.org/>

- Tests unitaires en C++ :

[http:](http://cppunit.sourceforge.net/doc/cvs/cppunit_cookbook.html)

[//cppunit.sourceforge.net/doc/cvs/cppunit_cookbook.html](http://cppunit.sourceforge.net/doc/cvs/cppunit_cookbook.html)

F. Conclusion : qu'avons nous vu ce semestre ?

F. Conclusion : qu'avons nous vu ce semestre ?

Les bases de la programmation impérative

- Instructions de contrôle de flot (conditionnelles, itératives)
- Types de données
- Fonctions, modules
- Entrées-sorties

F. Conclusion : qu'avons nous vu ce semestre ?

Les bases de la programmation impérative

- Instructions de contrôle de flot (conditionnelles, itératives)
- Types de données
- Fonctions, modules
- Entrées-sorties

Un peu de méthodologie de développement

- Documentation, tests
- Programmation incrémentale

F. Conclusion : qu'avons nous vu ce semestre ?

Les bases de la programmation impérative

- Instructions de contrôle de flot (conditionnelles, itératives)
- Types de données
- Fonctions, modules
- Entrées-sorties

Un peu de méthodologie de développement

- Documentation, tests
- Programmation incrémentale

Une esquisse de quelques outils

- Algorithmique, structures de données, complexité, architecture, ...

F. Conclusion : qu'avons nous vu ce semestre ?

Les bases de la programmation impérative

- Instructions de contrôle de flot (conditionnelles, itératives)
- Types de données
- Fonctions, modules
- Entrées-sorties

Un peu de méthodologie de développement

- Documentation, tests
- Programmation incrémentale

Une esquisse de quelques outils

- Algorithmique, structures de données, complexité, architecture, ...

Ce n'est que le début de l'aventure !