Expression des algorithmes

un bon niveau d'abstraction

Groupe algorithmique de l'IREM de Grenoble
{Anne.Rasse,**Jean-Marc.Vincent**,Benjamin.Wack}@imag.fr
{Maryline.Althuser,Herve.Barbe}@ac-grenoble.fr

2017

EXPRESSION DES ALGORITHMES

ALGORITHME

Un algorithme c'est ...

1. un moyen de communiquer à propos d'un problème/solution ;
2. une manière de résoudre un problème donné ;
3. une "formalisation" d'une méthode (qui sera prouvée et évaluée)
4. le premier pas (obligatoire) vers une implantation dans un langage de programmation

ALGORITHME

Un algorithme c'est ...

1. un moyen de communiquer à propos d'un problème/solution ;
2. une manière de résoudre un problème donné ;
3. une "formalisation" d'une méthode (qui sera prouvée et évaluée)
4. le premier pas (obligatoire) vers une implantation dans un langage de programmation

Quelques règles

1. Il y a de nombreuses manières d'écrire un algorithme
   Trouver son propre style ! ... Mais rester cohérent
2. un algorithme prends des entrées (input) et produit des sorties (output)
   Celles ci doivent être définies précisément
3. un algorithme peut utiliser d'autres algorithmes
   Approche "top-down" ... mais ces algorithmes doivent également être présentés

inspiré de Louis-Noël Pouchet

EXPRESSION D'UN ALGORITHME

Le langage algorithmique est une **convention** qui permet d'exprimer à un **lecteur**

1. l'idée de l'algorithme (principe, déroulement,...)
2. lui permettre de faire la preuve de celui-ci et de pouvoir analyser sa complexité
3. de pouvoir le traduire facilement dans un langage de programmation

Le langage algorithmique est donc plus ou moins proche d'un langage de programmation.
Exemple : tri bulle

EXPRESSION DES ALGORITHMES

TRI À BULLES : TD D'ALGORITHMIQUE

On souhaite trier un tableau d'éléments comparables. Le tableau est de taille n et les cases sont indicées de 1 à n.

Une itération ($i = 2$ à n) à chaque pas de laquelle si l'élément d'indice i est plus petit que l'élément d'indice $i - 1$ on échange l'élément d'indice i avec l'élément d'indice $i - 1$.

on répète l'itération précédente jusqu'à ce qu'il n'y ait plus d'échange

Finalement : les n éléments sont triés.

TRI À BULLES : OUVRAGE DE CORMEN ET AL.

TRI-BULLES(A)

1 **pour** $i \leftarrow 1$ **à** $longueur[A]$
2 **faire pour** $j \leftarrow length[A]$ **decr jusqu'à** $i + 1$
3 **faire si** $A[j] < A[j-1]$
4 **alors** permuter $A[j] \leftrightarrow A[j-1]$

seule l'idée est donnée,

il n'y a pas de déclaration de variables,

il est implicite que A est un tableau,

on compare directement les éléments du tableau

TRI À BULLES : OUVRAGE DE WIRTH

```
procedure bubblesort;
    var i,j: index;   x: item;
begin for i := 2 to n do
        begin for j := n downto i do
                if a[j−1] .key > a[j] .key then
                begin x := a[j−1]; a[j−1] := a[j]; a[j] := x
                end
        end
end {bubblesort}
```

Program 2.4 Bubblesort

parti pris de décrire les algorithmes dans un langage de programmation ici Pascal

pas d'abstraction des opérateurs, variables définies implicitement

syntaxe et propriétés du langage de programmation.

TRI À BULLES : OUVRAGE DE BEAUQUIER ET AL.

```
procédure TRI-BULLE(t, i, j);
   pour k de i à j − 1 faire
      pour p de j − 1 à k pas −1 faire
         si c(p + 1) < c(p) alors ÉCHANGER(t, p + 1, p)
         {c(k) est la clé de l'élément t(k)}
      finpour
   finpour.
```

travaille sur les indices

procedure : paramètres d'appels

notion de clé

TRI À BULLES : OUVRAGE DE HAREL ET AL.

(1) do the following $N - 1$ times:
  (1.1) point to the first element;
  (1.2) do the following $N - 1$ times:
    (1.2.1) compare the element pointed to with the next element;
    (1.2.2) if the compared elements are in the wrong order, exchange them;
    (1.2.3) point to the next element.

met en avant la notion de pointeur

une seule variable exprimée : la taille du tableau

pas d'indice d'itération

TRI À BULLES : WIKIPEDIA FR - EN

En français

```
tri_à_bulles(Tableau T)
   pour i allant de taille de T - 1 à 1
       pour j allant de 0 à i - 1
           si T[j+1] < T[j]
               échanger(T[j+1], T[j])
```

En anglais

```
procedure bubbleSort( A : list of sortable items )
   n = length(A)
   repeat
     swapped = false
     for i = 1 to n-1 inclusive do
       /* if this pair is out of order */
       if A[i-1] > A[i] then
         /* swap them and remember something changed */
         swap( A[i-1], A[i] )
         swapped = true
       end if
     end for
   until not swapped
end procedure
```

Autres sites

https://openclassrooms.com/courses/le-tri-a-bulles
http://rosettacode.org/wiki/Sorting_algorithms/Bubble_sort
http://www.sorting-algorithms.com/bubble-sort

TRI À BULLES : OUVRAGE DE KNUTH

Algorithm B (*Bubble sort*). Records $R_1, \ldots, R_N$ are rearranged in place; after sorting is complete their keys will be in order, $K_1 \leq \cdots \leq K_N$.

B1. [Initialize BOUND.] Set BOUND $\leftarrow N$. (BOUND is the highest index for which the record is not known to be in its final position; thus we are indicating that nothing is known at this point.)

B2. [Loop on j.] Set $t \leftarrow 0$. Perform step B3 for $j = 1, 2, \ldots,$ BOUND $- 1$, and then go to step B4. (If BOUND $= 1$, this means go directly to B4.)

B3. [Compare/exchange $R_j : R_{j+1}$.] If $K_j > K_{j+1}$, interchange $R_j \leftrightarrow R_{j+1}$ and set $t \leftarrow j$.

B4. [Any exchanges?] If $t = 0$, terminate the algorithm. Otherwise set BOUND $\leftarrow t$ and return to step B2. ∎

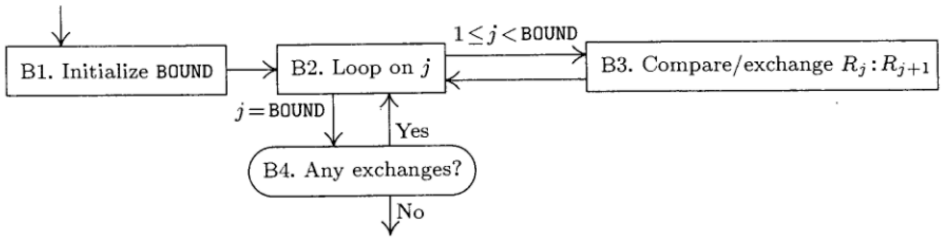


Fig. 15. Flow chart for bubble sorting.

représentation graphique

schéma itératif (steps), non modulaire

TRI BULLE : OUVRAGE DE LUC DAMAS

EXPRESSION D'UN ALGORITHME : SYNTHÈSE

Conclusion ?

EXPRESSION DES ALGORITHMES

LE CRÊPIER

```
Crêpier-Itératif (T)
    for i =Taille (T) to 2 do
        k = 1 ;
        for j = 2 to i
            if T[j] > T[k]
                k = j
        Retourne-préfixe (T,k)
        Retourne-préfixe (T,n)
```

LE CRÊPIER

Crêpier-Itératif (T)

```
for i =Taille (T) to 2 do
    k = 1 ;
    for j = 2 to i
        if T[j] > T[k]
            k = j
    Retourne-préfixe (T,k)
    Retourne-préfixe (T,n)
```

Retourne-préfixe (T,i)

```
j = 1 while j < i − j + 1
    Échange (T,j,i − j + 1)
    j = j + 1
```

Échange (T,i,j)

$$x = T[i]$$
$$T[i] = T[j]$$
$$T[j] = x$$

LE CRÊPIER

Crêpier-Itératif (T)

Données: Un tableau T d'éléments comparables

Résultats: Les éléments rangés dans le même tableau par ordre croissant

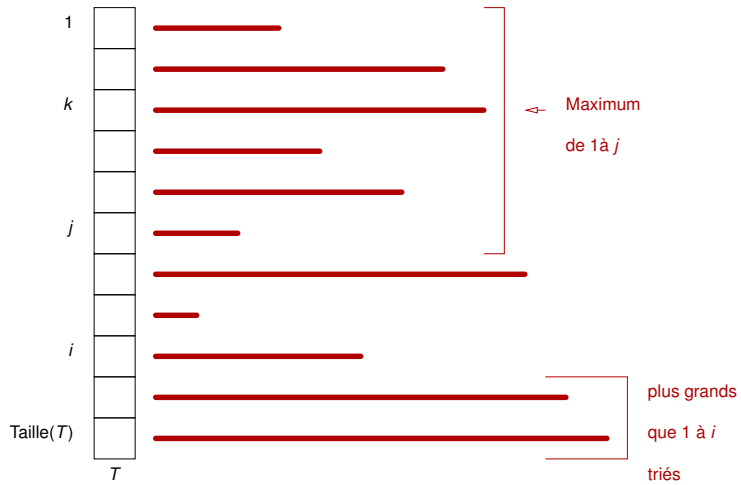
```
for i =Taille (T) to 2 do
    // Placer la bonne crêpe, la plus grande des i
       premières en position i
    k = 1 ; // indice de la plus grande crêpe
    for j = 2 to i
        // recherche de la plus grande crêpe
        if T[j] > T[k]
            k = j
    Retourne-préfixe (T,k) // positionnement de la plus
       grande crêpe au sommet de la pile
    Retourne-préfixe (T,i) // positionnement de la plus
       grande crêpe à sa position i
```

PREUVE DE L'ALGORITHME

Décrire l'état des variables de l'algorithme

Crêpier-Itératif (T)
Données: Un tableau T d'éléments comparables
Résultats: Les éléments rangés dans le même tableau par ordre croissant

```
for i = Taille(T) to 2 do
    // Placer la bonne crêpe, la plus grande
       des i premières en position i
    k = 1 ; // indice de la plus grande crêpe
    for j = 2 to i
        // recherche de la plus grande crêpe
        if T[j] > T[k]
            k = j
        ****************************************
    Retourne-préfixe (T,k)
    // positionnement de la plus grande
       crêpe au sommet de la pile
    Retourne-préfixe (T,i)
    // positionnement de la plus grande
       crêpe à sa position i
```



LE CRÊPIER (PREUVE)

Crêpier-Itératif (T)
Données: Un tableau T d'éléments comparables
Résultats: Les éléments rangés dans le même tableau par ordre croissant

```
for i =Taille (T) to 2 do
    // {Les éléments de T d'indice plus grand que i sont triés et plus grands que les
       éléments d'indice de 1 à i}
    // Placer la bonne crêpe, la plus grande des i premières en position i
    k = 1 ; // indice de la plus grande crêpe
    for j = 2 to i
        // {L'élément T[k] est plus grand que les éléments d'indice 1 à j − 1.}
        // recherche de la plus grande crêpe
        if T[j] > T[k]
            k = j
        // {L'élément T[k] est plus grand que les éléments d'indice 1 à j.}
    Retourne-préfixe (T,k) // positionnement de la plus grande crêpe au sommet de la pile
    Retourne-préfixe (T,i) // positionnement de la plus grande crêpe à sa position i
    // {Les éléments de T d'indice plus grand que i − 1 sont triés et plus grands que les
       éléments d'indice de 1 à i − 1}
```

- Les éléments **{ ..... }** sont appelées des **assertions** qui portent sur les valeurs des variables et ont une valeur logique vrai/faux.
- Dans notre exemple on montre que si l'assertion est vraie en début d'itération, elle sera vérifiée en fin d'itération (invariant d'itération).
- Il reste à montrer que l'assertion est vérifiée au début de l'itération.
- En fin d'itération, l'assertion vraie prouve la correction du programme.

LE CRÊPIER (APPROCHE RÉCURSIVE)

Crêpier-Récursif (T,n)
Données: Un tableau T d'éléments comparables de taille n
Résultats: Les éléments rangés dans le même tableau par ordre croissant

if $n \neq 1$
  $k =$**Indice-du-max** (T,n)
  `// recherche de l'indice de la valeur maximale du tableau de 1 à n`
  Retourne-préfixe (T,k)
  `// positionnement de la plus grande crêpe au sommet de la pile`
  Retourne-préfixe (T,n)
  `// positionnement de la plus grande crêpe à la position n`
  Crêpier-Récursif (T,$n-1$)
  `// appel récursif pour trier le tableau de 1 à n-1`

Expression plus élégante, plus facile à prouver par induction (par récurrence)

LE CRÊPIER : PETITE HISTOIRE

Quel est le nombre minimal $f(n)$ de retournements pour trier un tas arbitraire de n crêpes ?

Références

- Dweighter, Harry ; Garey, Michael R. ; Johnson, David S. ; Lin, Shen (1977), *Solutions of Elementary Problem E2569*, Amer. Math. Monthly 84 : 296
  Pseudonyme de Jacob E. Goodman
- Gates W.H. ; Papadimitriou, C.H. *Bounds for sorting by prefix reversal*. Discrete Math. 27 (1979), 47–57.
- Bulteau, L. ; Fertin, G. ; Rusu, I. *Pancake Flipping is Hard*. 37th International Symposium on Mathematical Foundations of Computer Science, Aug 2012, Bratislava, Slovakia. 7467, pp.247-258, 2012, Lecture Notes in Computer Science, Springer Verlag. Hal version

Valeurs de $f(n)$

- $f(n)$ est connu pour $n \leqslant 19$.
- Meilleur encadrement
$$\frac{15}{14}n \leqslant f(n) \leqslant \frac{18}{11}n + \mathcal{O}(1).$$
- le problème *MIN-SBPR* est $\mathcal{NP}$-dur
- Online Encyclopedia of Integer Sequences A058986

EXPRESSION DES ALGORITHMES

PRINCIPES POUR LA CONCEPTION D'ALGORITHMES

1. Déterminer les entrées et les sorties (spécification)
2. Trouver la structure de donnée adaptée pour ce problème
   - Ne pas hésiter à pré-traiter les entrées pour les mettre sous une forme adéquate
3. Essayer de réduire le problème à un problème connu
   - Tri, recherche, chemin dans un graphe,...
   - Vérifier si une solution existe déjà (livres, internet,...)
4. Décider de la manière d'approcher le problème : itératif/récursif/mix
   - Dépend de la manière de penser, de la facilité à trouver des invariants, de la manière de décomposer le problème en sous problèmes,...
5. **Écrire** l'algorithme
6. Faire tourner l'algorithme sur des exemples simples et des exemples "limite".
7. Donner les invariants de l'algorithme et faire la preuve
8. Évaluer la complexité de l'algorithme

PRINCIPES POUR L'EXPRESSION DES ALGORITHMES

Ma position personnelle (que certains de mes collègues ne partagent pas) :

- Un algorithme est toujours accompagné de schémas et de texte
  représentation de l'état des variables,
- utiliser les conventions
  (i,j sont des indices, x un réel, n un entier par exemple une taille de tableau,. . .).
- utiliser des identificateurs explicites, une variable un usage
  noms de variables, de fonctions,
- la forme est importante
  l'indentation doit permettre de comprendre la structure de l'algorithme
- mettre des commentaires dans le code et accompagner l'algorithme par une explication en français
- ne mettre que l'information importante : minimiser l'encre
- être cohérent
  utiliser le même formalisme pour toutes les présentations d'algorithmes

de manière plus générale

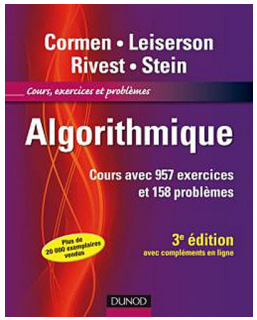
favoriser tout ce qui aide à la compréhension et éliminer ce qui peut perturber la lecture

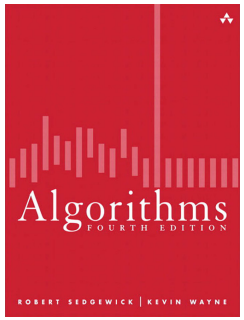
EXPRESSION DES ALGORITHMES

BIBLIOGRAPHIE : OUVRAGES DE RÉFÉRENCE

- **Algorithmique** *Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein.*. Dunod, 2010.
  Ouvrage de référence internationale en algorithmique. Très pédagogique il peut être utilisé en autoformation, lorsque les bases sont acquises. Couvre l'ensemble du domaine.
- **Algorithms** *Robert Sedgewick and Kevin Wayne*. Addison Wesley, 2011.
  Une approche thématique permettant de reprendre les différents et paradigmes de l'algorithmique. La présentation est soignée, les détails des implémentations en Java sont très utiles.

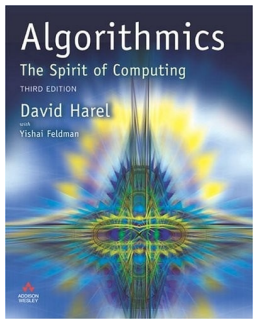
  Des versions précédentes en français : *Robert Sedgewick* Algorithmes en C ou Algorithmes en Java chez Dunod

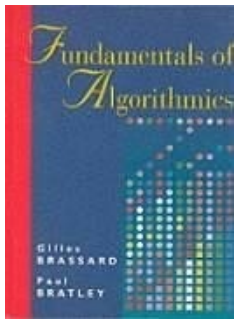


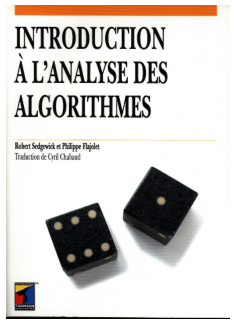


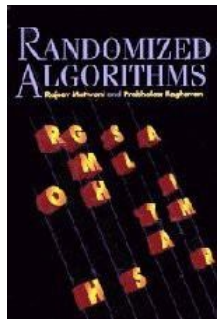
BIBLIOGRAPHIE : OUVRAGES PLUS AVANCÉS

- **The Design and Analysis of Algorithms** *Dexter C. Kozen* Springer, 1991.
  Excellent ouvrage pour de l'algorithmique avancée. Présenté sous forme de séquence de lectures "indépendantes" il va directement à l'essentiel. Les principes algorithmiques sont ainsi mis en valeur.
- **Algorithmics : The Spirit of Computing** *David Harel and Yishai Feldman* Addison Wesley, 2004.
  Orienté méthodologie, cet ouvrage propose une vue transversale en abordant successivement, méthode et analyse, limitations et robustesse, extensibilité... intéressant pour le recul pris.
- **Introduction à l'analyse des algorithmes** *Robert Sedgewick and Philippe Flajolet* Addison Wesley 1995
  Ouvrage théorique sur l'analyse de la complexité des algorithmes
- **Fundamental of Algorithms** *Gilles Brassard and Paul Bratley* Prentice Hall 1996
- **Randomized Algorithms**, *R. Motwani and P. Raghavan*, Cambridge University Press, 1995.



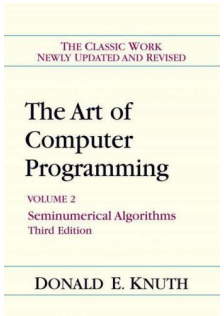


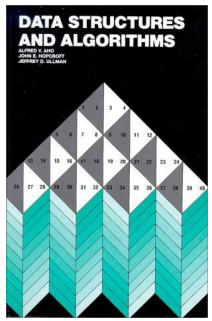


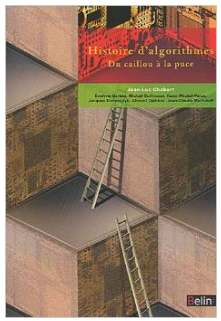


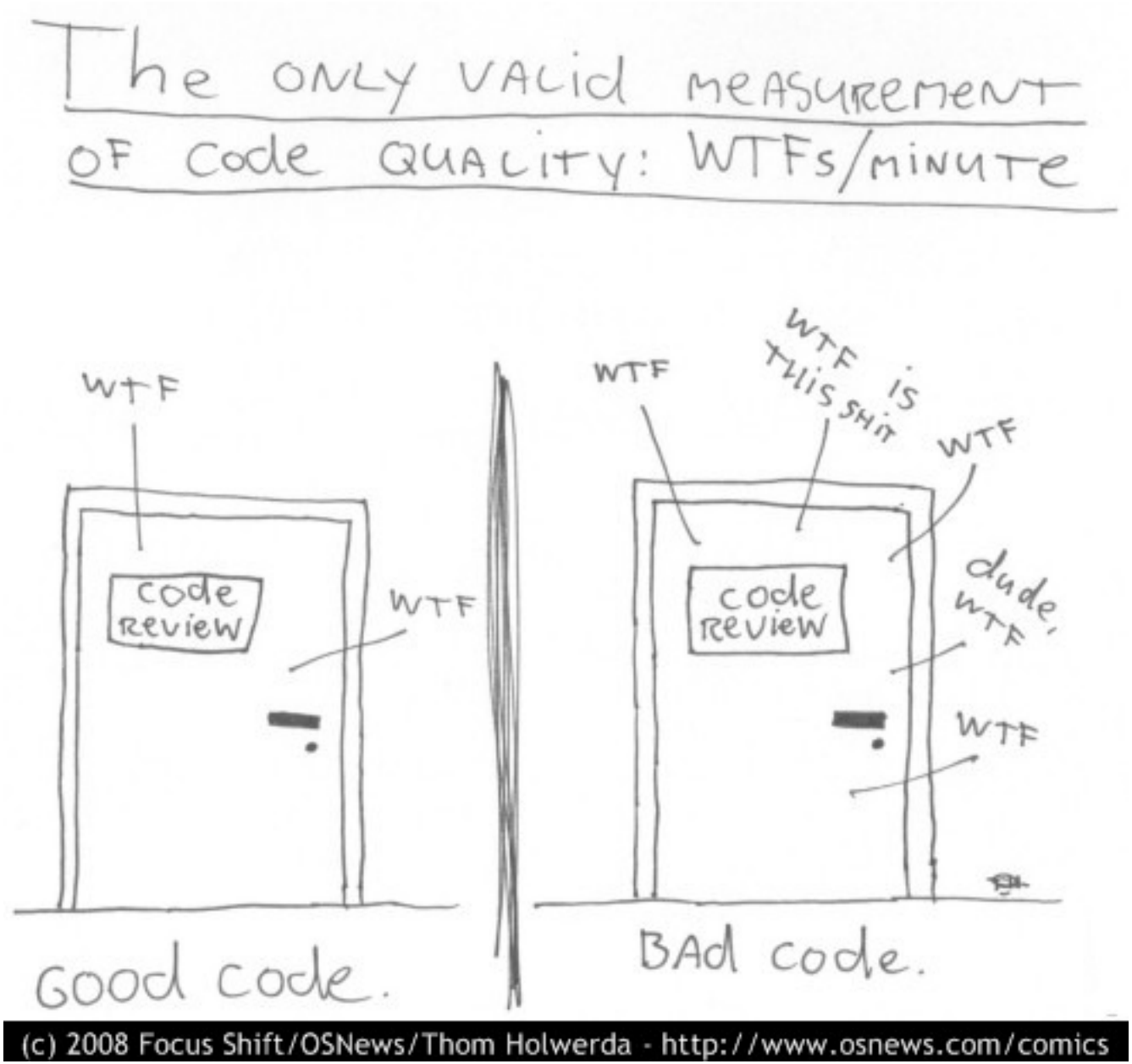
BIBLIOGRAPHIE : OUVRAGES HISTORIQUES DE RÉFÉRENCE

- **The Art of Computer Programming, Vol 1-4** *Donald E. Knuth*, Addison-Wesley, 1998.
  Ouvrage historique et encore d'actualité pour la conception et l'analyse d'algorithmes
- **Data Structures and Algorithms** *Alfred V. Aho, J.E. Hopcroft, et Jeffrey D. Ullman* Addison Wesley 1983
- *Jean-Luc Chabert et al.* **Histoires d'algorithmes** Belin 2010
  Une histoire des algorithmes avec un point de vue calcul et calcul numérique








The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE
WTF
code review
WTF
Good code.
WTF
WTF IS THIS SHIT
WTF
code review
dude.
WTF
WTF
BAd code.
(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics