ENVOL 2014

Dirk Hoffmann
Centre de Physique des Particules de Marseille

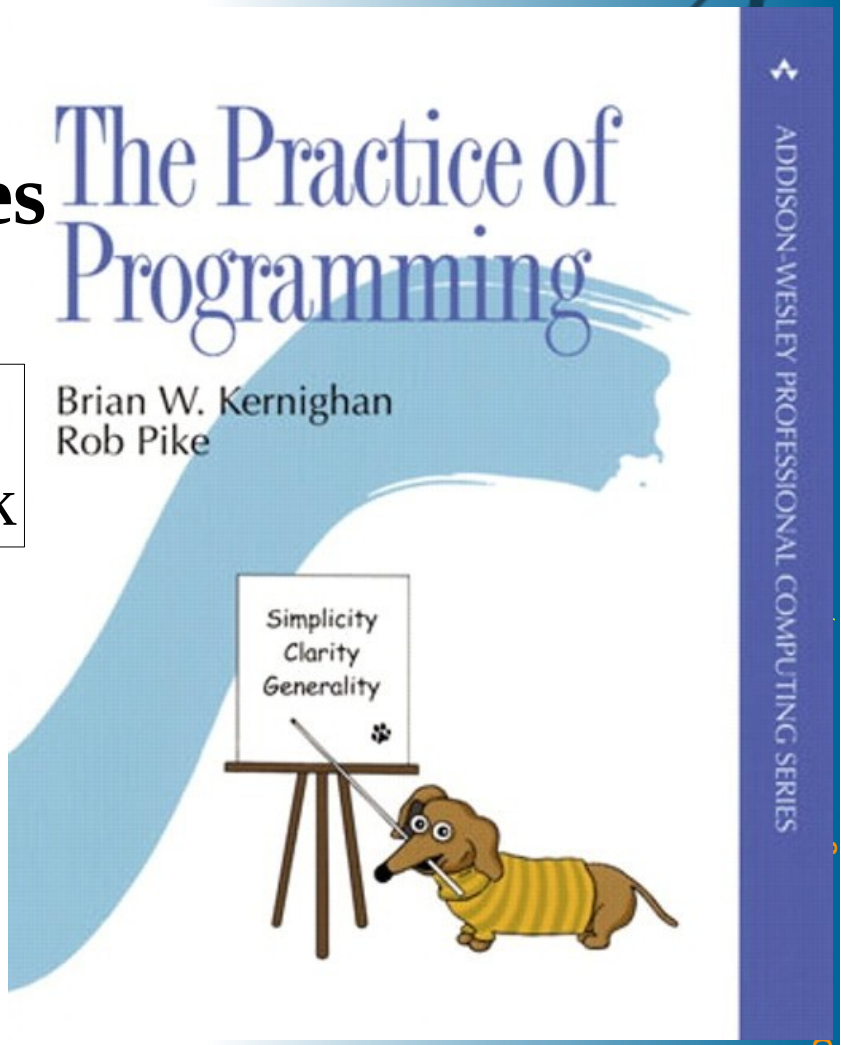
Programmation pratique de tests

The Practice of Programming

B. W. Kernighan, R. Pike

- **Style**
- **Algorithms and Data Structures**
- **Design and Implementation**
- **Interfaces**
- **Debugging**
- **Testing**
- **Performance**
- **Portability**
- **Notation**

Markov
codé en C, Java, awk



Généralités

- **Les tests peuvent indiquer la présence d'erreurs, mais ne peuvent pas prouver leur absence.** (E. Dijkstra)
- **Penser au problème en codant.**
- **Tests systématiques (du facile au complexe).**
- **Automatisation.**
- **Astuces.**
- **Génération de code. Notation.**

```
SUM(C1:C32)
```

Tester en codant (1)

- **Test après compilation – test en relecture de code**
- **Vérifier les conditions aux bords** – mentalement

```
int i;
char s[MAX];

for (i=0; (s[i] = getchar() != '\n' && i < MAX-1; ++i)
    ;

s[--i] = '\0';
```

```
for (i=0; i < MAX-1; i++)
    if ((s[i] = getchar() == '\n')
        break;
s[i] = '\0';
```

```
for (i=0; i < MAX-1; i++)
    if ((s[i] = getchar() =='\n' || s[i] == EOF)
        break;
s[i] = '\0';
```

Tester en codant (2)

- **L'autre limite (pre-/post-conditions)**
  Est-ce que les conditions avant et après exécution du code sont valables ?
- **Exemple :**
  `n=0`

```
double avg(double a[], int n) {
    int i;
    double sum;

    sum = 0.0;
    for (i=0; i<n; i++)
        sum += a[i];
    return sum / n;
}
```

- **Que devrait retourner `avg()` pour `n==0` ?**

Tester en codant (2)

- **L'autre limite (pre-/post-conditions)**
  Est-ce que les conditions avant et après exécution du code sont valables ?
- **Exemple :**
  `n=0`

```
double avg(double a[], int n) {
    int i;
    double sum;

    sum = 0.0;
    for (i=0; i<n; i++)
        sum += a[i];
    return sum / n;
}
```

- **Que devrait retourner `avg()` pour `n==0` ?**
- **Il n'y a pas de bonne réponse, mais une qui est fausse pour sûr : ignorer le cas !**

Tester en codant (3)

- **Utiliser assertions (en C/C++, Java, python, …)**
  - Permet de pointer efficacement sur la source d'erreur.
- **Programmation défensive**
  - Oubliez : « Aah, ça n'arrive jamais »
  - Un logiciel / une fonction programmée défensivement se protège contre les mauvaises utilisations
  - Pointeur NULL, index hors limites, division/zéro, …
  - Détecter, avertir, défléchir

```
if (points < 0 || points > 20)
    resultat = "?";
else if (points > 18)
    resultat = "A";
else if (points > 16)
    resultat = "ECA";
else
    ...
```

Tester en codant (4)

- **Vérifier les retours d'erreurs (*error code*)**
- **C'est souvent simple et trop souvent négligé**
  - *Disk full?*
  - *Permissions denied?*

```
fp = fopen(outfile, "w");

while (...)
    /* write output file */
    fprintf(fp, ...);

if (fclose(fp) == EOF) {
    /* Problem occured? */
    /* some output error occured */
}
```

- **Le problème peut avoir des conséquences sérieuses (ou en aura éventuellement dans l'avenir !).**

Tests systématiques

- **Tests incrémentaux**
- **Tester les parties simples d'abord**
- **Connaître le résultat attendu (!)**
- **Lois de conservation – trouver, utiliser**
  - Outils `wc`, `sum`, …
- **Comparer des implémentations indépendantes**
- **Mesurer la couverture des tests**
  - Outils professionnels, profilers, …

Automatisation de tests (1)

- **Tests de régression (TR) automatiques**
  surtout en modifiant un code existant (legacy)
- **Exemple : l'appli qui tue (*killer application*)**

```
for i in ka_data.*
do
    old_ka $i > out1
    new_ka $i > out2
    if ! cmp -s out1 out2
    then
        echo $i: BAD
    fi
done
```

- **Implicitement, on suppose que la version précédent produisait un résultat correct.**
  Garbage in = garbage out
- **On teste régulièrement le TR lui-même.**

Automatisation de tests (2)

- **Tests *self-contained* (auto-contenant)**
- **Exemple de** awk

  `$i++` **signifie** `($i)++`**, et non pas** `$(i++)`

```
echo 3 5 | newawk '{i=1; print $i++; print $1, i}' > out1

echo '3
4 1' > out2 # correct answer

if ! cmp -s out1 out2; then
    echo 'BAD: field increment test failed.'
fi
```

```
try {if ($1==1) print "yes"else print "no"}
1       yes
1.0     yes
1E0     yes
0.1E1   yes
10E-1   yes
01      yes
+1      yes
10E-2   no
10      no
```

Que faire, si je trouve une erreur ?

- **Si l'erreur n'était pas découverte par la suite de tests existants, alors il faut concevoir un nouveau test qui indique ce nouveau type de problème.**
- **Application à la version non-corrigée du code**
- **Nouvelle série de tests ?**
- **Ajout de défenses dans le programme ?**

- **Ne jamais jeter un test**
- **Garder un historique**

Echaffaudages de test (1)

Echaffaudages de test (2)

- **L'accès aux modules et éléments à tester est facilité par un programmation modulaire.**
- **Pour les fonctions mathématiques, manipulations de chaînes de caractères, tri, …, facile à concevoir**
- **Exemple : `memset()`**
  - Fonction C équivalente
  - Difficultés à anticiper avec l'accélération par ASM

```
void* memset(void* s, int c, size_t n) {
    size_t i;
    char *p;
    p = (char*)s;
    for (i=0; i<n; i++) p[i]=c;
    return s;
}
```

```
s0 = malloc(big); s1 = malloc(big);
foreach combination (n,c,offset):
    set s0, s1 to known pattern
    run slow memset(s0+offset, c, n)
    run fast memset(s0+offset, c, n)
    compare(s0, s1) byte by byte
```

```
offset = 10, 11, ..., 20
c = 0, 1, 0x7F, 0x80, 0xFF, 0x11223344
N = 0, 1, 2, 3, 4, 5, 7, 8, 9, 15, 16, 17,
  31, 32, 33, ..., 65535, 65536, 65537
```

Tests sous charge

- *“Stress tests”*
- **Tester les cas « impossibles »,**
  - **nom de variable de longueur 1594 caractères**
  - **paramètres d'entrée aléatoires**
  - **paramètres délibérément malveillants**
- **1998, NYT “Internet Worm”**
- **Danger, si x.y.z trop grand ?**

```
char *p;
p = (char*) malloc(x * y * z)
```

The security hole is caused by what is known as a “buffer overflow error.” Programmers are supposed to include code in their software to check that incoming data are of safe type and that the units are arriving at the right length. If a unit of data is too long, it can overrun the “buffer” – the chunk of memory set aside to hold it. In that case, the E-Mail program will crash, and a hostile programmer can trick the computer into running a malicious program in its place.

Astuces

- **Comment vérifier les cas limites pour de grands tableaux par exemple ?**
  - Modifier le code !

```
#ifdef TESTTING
#define ARRAY_LENGTH 20
#else
#define ARRAY_LENGTH 1E9
#endif
```

```
void* testmalloc(size_t n) {
    static int _count = 0;
    if (++count > 10) return NULL;
    else return malloc(n);
}
```

tout en pensant à les désactiver ensuite (!)

- **Initialiser avec 0xDEADBEEF au lieu de zéro (0xBEBECAFE, 0xAFFE) – faciles à reconnaître**
- **Ne jamais tester avec des bugs existants connus**
- **Messages supplémentaires lors des tests (*verbose*)**
- **Tester sur plusieurs machines, OS, compilateurs, …**

Qui fait les tests ?

- **Quelqu'un qui connaît le code (“*white box testing*”)**
- **Il n'y a pas de prestataire miracle « institut de certification de code ». (sic !)**
- **Donald E. Knuth (TeX)**

> I get into the meanest, nastiest frame of mind that I can manage, and I write the nastiest [testing] code I can think of; then I turn around and embed that in even nastier constructions that are almost obscene.

- **Trouver des erreurs, pas déclarer le logiciel « sans faute »**
- ***Black box testing* (utilisateur/programmeur externe) trouve d'autres types d'erreurs.**
- **Par exemple : revue extérieure**

Conclusion (« leçons »)

- **A lire : “case study” du logiciel Markov**

- **Mieux on commence son code,**
  moins on trouvera des bugs dans la suite.
- **Les tests des cas limites sont très efficaces.**
- **Automatiser le plus possible évite la fatigue,**
  l'inattention et d'autres faiblesses humaines.
- **Tests de régression pour comparer avant-après**
- **Ça doit rentrer dans les habitudes.**
  The most important rule of testing is to do it.