Complexité et qualité

Comment mesurer la complexité d'un logiciel

La complexité de code est un facteur ayant un impact direct sur la durée de vie et l'exploitation d'un logiciel, notamment sur son taux de défauts, sa testabilité et sa maintenabilité. Par conséquent, la complexité a une influence directe sur la qualité d'un logiciel et sur son coût.

Cet article explique :

- Comment mesurer la complexité d'un logiciel ;
- L'impact de la complexité sur la qualité ;
- Les métriques traditionnelles.

Ce qu'il faut savoir :

- Comprendre les termes utilisés en programmation ;
- Avoir des bonnes notions en programmation, notamment en C.

Niveau de difficulté

Le pourcentage d'erreurs et la robustesse du code dépendent de sa complexité. Une application très complexe est difficile à tester. La part des coûts d'un projet attribuée aux tests et aux corrections est plus importante pour un code ayant une complexité élevée. Même après les tests, un code complexe détient probablement plus d'erreurs qu'un code écrit en suivant les règles de la simplicité.

Pour fournir un logiciel de qualité et garantir le coût de test et de maintenance relativement faible, il est logique de mesurer déjà la complexité d'un logiciel pendant le développement. Ainsi lors de dépassement des valeurs recommandées, le développeur peut intervenir rapidement. Pour quantifier les qualités d'un logiciel, on se sert de métriques mesurant la qualité de celui-ci. Le standard IEEE 1061 de 1992 définit les métriques mesurant la qualité d'un logiciel comme une fonction, qui illustre une unité de logiciel dans une valeur numérique. Cette valeur calculée est interprétable comme le degré de satisfaction de la qualité de l'unité du logiciel. On classe généralement les métriques par :

- celles mesurant le processus de développement ;
- celles mesurant des ressources ;
- et celles de l'évaluation du produit logiciel.

Les métriques de produits mesurent les qualités du logiciel. Parmi ces métriques, on distingue les métriques traditionnelles et les métriques orientées objet.

Les métriques orientées objet prennent en considération les relations entre éléments de programme (classes, méthodes). Les métriques traditionnelles se divisent en deux groupes : les métriques mesurant la taille et la complexité, et les métriques mesurant la structure du logiciel. Les métriques mesurant taille et complexité les plus connus sont les métriques de ligne de code ainsi que les métriques de Halstead. Les métriques mesurant la structure d'un logiciel comme la complexité cyclomatique de McCabe se basent sur des organigrammes de traitement ou des structures de classe. Ci-dessous vous trouverez des explications sur les métriques traditionnelles (les métriques des lignes de code, le nombre cyclomatique de McCabe et les métriques de Halstead) ainsi que sur l'index de maintenabilité et leur utilisation pour l'amélioration de la qualité de logiciel. Pour illustrer les différentes métriques et les limites acceptables, appliquons les sur le programme C présent sur le Listing 1. Il s'agit d'un interpréteur évaluant une expression mathématique de la

forme *valeur1 opérateur valeur2*.

Les métriques des Lignes de code

Les mesures des lignes de code (LOC, *Lines of Code*) sont les mesures les plus traditionnellement utilisées pour quantifier la complexité d'un logiciel. Elles sont simples, faciles à compter, et très faciles à comprendre. Elles ne prennent cependant pas en compte le contenu d'intelligence et la disposition du code. On peut distinguer les types de lignes suivants :

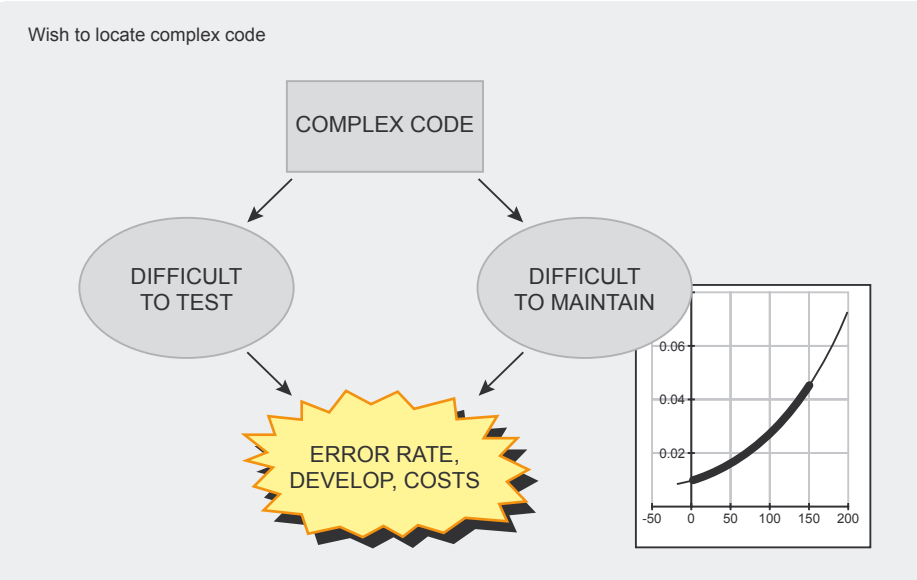

Figure 1. Complexité, testabilité et maintenabilité

- LOCphy: nombre de lignes physiques (total des lignes des fichiers source, Number of physical lines) ;
- LOCpro: nombre de lignes de programme (déclarations, définitions, directives, et code) (Number of program lines) ;
- LOCcom: nombre de lignes de commentaire (Number of commented lines) ;
- LOCbl: nombre de lignes vides (Number of blank lines).

Nous constatons que pour la fonction eval1, la mesure LOCphy est de 72. Elle contient 57 lignes de programme (LOCpro), 7 lignes vides (LOCbl) et 11 lignes de commentaires (LOCcom). Le total de LOCpro, LOCbl et LOCcom excède la valeur de LOCpro car il y a plusieurs lignes qui contiennent du code programme et des commentaires.

Quelles sont les limites acceptables ?

La longueur des fonctions devrait être de 4 à 40 lignes de programme. Une définition de

Listing 1 : Programme d´exemple

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void extract(char* char1, char* char2, int debut, int nb);
int eval1(char * ch)
{
int i;
int valeur1, valeur2;
int lgval2;
char *val1, *val2;
char operation;
int resultat;
/* Recherche d´un opérateur et de sa position */
for( i=0 ; *(ch+i) !='+' && *(ch+i) !='-' && *(ch+i) !='*' &&
                    *(ch+i) !='/' && *(ch+i)
!='\0'; i++)
{
}
/* Traitement des erreurs */
if(i==0) /* Le premier opèrande manque */
{
printf("erreur : pas de <valeur1>");
exit(0);
}
else if(i==strlen(ch)-1) /* Le deuxième opérande manque */
{
printf("erreur : pas de <valeur2>");
 exit(0);
}
else if(i==strlen(ch)) /* Il n´a a pad d´opérateur */
{
printf("erreur : pas de <operator>");
exit(0);
}
/* char Extraction de la chaîne de caractère correspondant au
                    premier opérande */
val1=(char*) malloc((i+1)*sizeof(char));
extract(ch,val1,0,i);
/* Transformation de la chaîne de carctère en entier */
sscanf(val1,"%d",& valeur1);
/* Récuperation de l´opérateur */
operation=*(ch+i);
/* Extraction de la chaîne de caractère correspondant au
                    deuxième opérande */
lgval2=strlen(ch)-(i+1);
val2=(char*) malloc((lgval2+1)*sizeof(char));
extract(ch,val2,i+1,lgval2);
/* Transformation de la chaîne de caractère en entier */
sscanf(val2,"%d",&valeur2);
/* Traitement de l´opération */
switch(operation)
{
case '+':
resultat=valeur1+valeur2;
break;
 case '-':
  resultat=valeur1-valeur2;
  break;
 case '*':
  resultat=valeur1*valeur2;
  break;
 case '/':
  if(valeur2 != 0)
  resultat=valeur1/valeur2;
  else
  {
  resultat=0;
  printf("Erreur : impossible de diviser par 0");
 exit(0);
}
}
return resultat;
}
/* Fonction qui extrait une sous-chaîne de chaîne1 dans chaîne2,
                    de nb caractères à
partir du caractère début */
void extract(char* char1, char* char2, int debut, int nb)
{
int i;
char1= char1+debut;
i=0;
while(i<nb)
{
* char2=* char1;
char1++;
char2++;
i++;
}
* char2='\n';
}

int main(int argc, char** argv)
{
int res;
if(argc!=2)
{
 printf("Erreur, utilisation du programme : eval1
                    <expression>");
}
else
{
 res=eval1(argv[1]);
 printf("Le résultat de l´opération est : %d",res);
}
}
```

fonction contient au moins un prototype, une ligne de code, et une paire d'accolades, qui font 4 lignes.

Une fonction plus grande que 40 lignes de programme implémente probablement beaucoup de fonctions. Les fonctions contenant un état de sélection avec beaucoup de branches sont une exception à cette règle. Les décomposer en des fonctions plus petites réduit souvent la lisibilité.

La fonction eval1 de notre exemple avec un LOCphy de 72 est donc trop longue. Elle pourrait être décomposée en plusieurs fonctions plus petites. Les autres fonctions ont une mesure LOCphy inférieure à 40, donc ont une taille correcte. *La longueur du fichier* devrait être de 4 à 400 lignes de programme. La plus petite entité qui peut raisonnablement occuper un fichier source complet est une fonction, et la longueur minimum d'une fonction est de 4 lignes. Les fichiers plus longs que 400 lignes de programme (10..40 fonctions) sont habituellement trop longs pour être compris en totalité. Au minimum 30 % et au maximum 75 % d'un fichier devrait être commenté. Si moins d'un tiers du fichier est commenté, le fichier est soit très trivial, soit pauvrement expliqué. Si plus de trois quarts du fichier est commenté, le fichier n'est plus un programme, mais un document. Dans un fichier header correctement commenté, le pourcentage de commentaires peut parfois dépasser 75%.

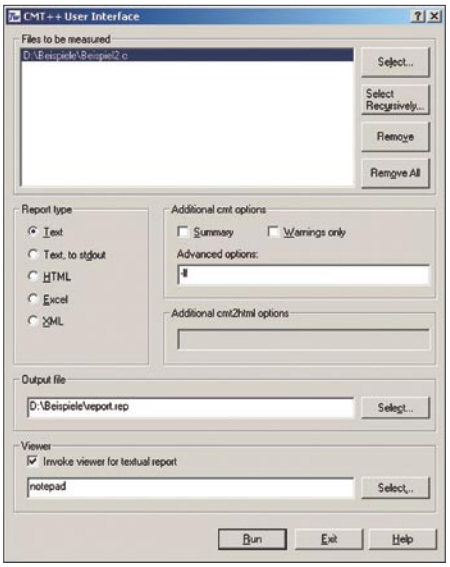

Figure 2. IHM d'un logiciel de mésure de complexité

La mesure LOCphy de notre fonction exemple est de 72, donc la taille du fichier ne gène pas sa compréhension. Par contre, nous ne trouvons que 11 lignes de commentaires, soit 15% du fichier, ce qui montre que le fichier est trop peu commenté, même s'il est relativement trivial.

Le nombre cyclomatique de Mc Cabe

La complexité C*yclomatique*, également référée comme complexité de programme ou complexité de McCabe) a été introduite par Thomas McCabe en 1976. Elle est le calcul le plus largement répandu des métriques statiques. La métrique est conçue pour être indépendante du langage et du format de langage. La métrique de McCabe indique le nombre de chemins linéaires indépendants dans un module de programme et représente finalement la complexité des flux de donnés. Il correspond au nombre de branches conditionnelles dans l'organigramme d'un programme.

Pour un programme qui consiste en seulement des états séquentiels, la valeur pour v(G) est 1.

Plus le nombre cyclomatique est grand, plus il y aura de chemins d'exécution dans la fonction, et plus elle sera difficile à comprendre et à tester. Le nombre cyclomatique v(G) est l'une des plus importantes mesures de complexité afin d'estimer l'effort nécessaire pour les tests du logiciel.

Du fait que le nombre cyclomatique décrive la complexité du flux de contrôle, il est évident que les modules et les fonctions ayant un nombre cyclomatique élevé auront besoin de plus de cas de tests que les modules ayant un nombre cyclomatique plus bas. Comme principe de base, chaque fonction devrait avoir un nombre de cas de tests au moins égal au nombre cyclomatique.

Comment est calculée la métrique de Mc Cabe ?

Le nombre cyclomatique McCabe v(G) est calculé à partir des définitions de fonctions (ou membres) et des déclarations de classes/structures. Voici les branches entrant en compte dans le calcul de v(G) :

- chaque *if-statement* introduit une nouvelle branche au programme et, par conséquent, incrémente v(G) ;
- chaque itération (une boucle for ou while) introduit des branches et par conséquent augmente le facteur v(G) ;
- chaque *switch-statement* du commutateur-rapport, incrémente v(G) une branche de cas n'augmente pas la valeur de v(G), car il ne doit pas augmenter le nombre de branches dans le flux de commande. S'il y a deux ou plus de cas... : qui n'ont pas de code entre eux, la mesure McCabe est augmentée uniquement une fois pour l'ensemble des cas.
- chaque morceau (...) attrapé dans un bloc d'essai (*try-block*) incrémente v(G) ;
- les constructions exprl ? expr2 : expr3 incrémente v(G).

Il faut noter que v(G) est insensible aux branches inconditionnelles comme goto-, return-, et les *break-statements* bien qu'ils augmentent sûrement la complexité. En résumé, les cons-

IDENTIFIER	Toutes les marques qui ne sont pas des mots réservés.
TYPESPEC	(spécificateurs de type) Mots réservés qui spécifient le type : ***bool***, ***char***, ***double***, ***float***, ***int***, ***long***, ***short***, ***signed***, ***unsigned***, ***void***. Cette classe inclut aussi quelques mots clefs non standards spécifiques de compilateur.
CONSTANT	caractère, nombre, constante ou chaîne de caractères.

Tableau 1. Les métriques de Halstead considérés comme opérandes

SCSPEC	(spécificateurs de classes de stockage) mots réservés qui spécifient la classe de stockage : auto, extern, inlin, register, static, typedef, virtual, mtuable.
TYPE_QUAL	(qualifiants de type) mots réservés qui qualifient le type : const, friend, volatile.
RESERVED	Autres mots réservés de C++ : asm, break, case, class, continue, default, delete, do, else, enum, for, goto, if, new, operator, private, protected, public, return, sizeof, struct, switch, this, union, while, namespace, using, try, catch, throw, const_cast, static_cast, dynamic_cast, reinterpret_cast, typeid, template, explicit, true, false, typename. Cette classe inclut aussi quelques mots clefs non standards spécifiques de compilateur.
OPERATOR	! != % %= & && || &= ( ) * *= + ++ += , - -- -= -> . ... / /= : :: < << <<= <= = == > >= >> >>= ? [ ] ^ ^= { } | |= ~ on compte généralement le semicolon ; parmi les opérateurs (tous les tokens sont soit opérateur ou opérande).

Tableau 2. Les métriques Halstead considérés comme opérateurs

Listing 2. Résultat de mesure de complexité pour toutes les fonctions

```
*****************************************************************
*    CMT++, Complexity Measures Tool for C/C++, Version 4.1     *
*                                                               *
*                  COMPLEXITY MEASURES REPORT                   *
*                                                               *
*               Copyright (c) 1993-2006 Testwell Oy             *
*****************************************************************
License notice: This is a limited period evaluation copy license.
This report was produced at Mon Oct 16 18:44:32 2006
Options: -o D:\Beispiele\report.txt -lf -f C:\\cmt\files.txt
Metrics for eval1()
Operator                                        Frequency
--------                                        ---------
!=                                                      6
&                                                       2
&&                                                      4
()                                                     31
*                                                      14
+                                                      11
++                                                      1
,                                                      12
-                                                       3
/                                                       1
;                                                      33
=                                                      10
==                                                      3
break                                                   3
case ...:                                               4
else                                                    3
for()                                                   1
if()                                                    4
return                                                  1
sizeof                                                  2
switch()                                                1
{}                                                      7
Operand                                         Frequency
-------                                         ---------
"%d"                                                    2
"Erreur : impossible de diviser par 0"                  1
"erreur : pas de <operator>"                            1
"erreur : pas de <valeur1>"                             1
"erreur : pas de <valeur2>"                             1
'*'                                                     2
'+'                                                     2
'-'                                                     2
'/'                                                     2
'\0'                                                    1
0                                                       9
1                                                       5
ch                                                     12
char                                                    7
eval1                                                   1
exit                                                    4
extract                                                 2
i                                                      16
int                                                     5
lgval2                                                  4
malloc                                                  2
operation                                               3
printf                                                  4
resultat                                                7
sscanf                                                  2
strlen                                                  3
val1                                                    4
val2                                                    4
valeur1                                                 6
valeur2                                                 7
Metrics                                         Value
-------                                         -----
Cyclomatic number          (v(G))                  14
Number of physical lines   (LOCphy)                72
Number of program lines    (LOCpro)                57
Number of blank lines      (LOCbl)                  7
Number of commented lines  (LOCcom)                11
Program length             (N)                    279
Number of operators        (N1)                   157
Number of operands         (N2)                   122
Vocabulary size            (n)                     52
Number of unique operators (n1)                    22
Number of unique operands  (n2)                    30
Program volume             (V)                1590.423
Number of delivered bugs   (B)                   0.572
Difficulty level           (D)                  44.733
Effort to implement        (E)               71144.908
Program level              (L)                   0.022
Implementation time        (T)                3952.495
Max nesting depth          (MaxND)                 3
Maintainability Index w.o.c(MIwoc)                60.165
MI comment weight          (MIcw)                 28.460
MI with comments           (MI)                   88.625
-----------------------------------------------------------
Metrics for extract()
Operator                                        Frequency
--------                                        ---------
()                                                      1
*                                                       5
+                                                       1
++                                                      3
,                                                       3
;                                                       8
<                                                       1
=                                                       4
while()                                                 1
{}                                                      2
Operand                                         Frequency
-------                                         ---------
'\n'                                                    1
0                                                       1
char                                                    2
char1                                                   5
char2                                                   4
debut                                                   2
extract                                                 1
i                                                       4
int                                                     3
nb                                                      2
void                                                    1
Metrics                                         Value
-------                                         -----
Cyclomatic number          (v(G))                   2
Number of physical lines   (LOCphy)                17
Number of program lines    (LOCpro)                14
```

Listing 2. Résultat de mesure de complexité pour toutes les fonctions

```
Number of blank lines       (LOCbl)              2
Number of commented lines   (LOCcom)             1
Program length              (N)                 55
Number of operators         (N1)                29
Number of operands          (N2)                26
Vocabulary size             (n)                 21
Number of unique operators  (n1)                10
Number of unique operands   (n2)                11
Program volume              (V)                241.577
Number of delivered bugs    (B)                  0.067
Difficulty level            (D)                 11.818
Effort to implement         (E)                2855.006
Program level               (L)                  0.085
Implementation time         (T)                158.611
Max nesting depth           (MaxND)              2
Maintainability Index w.o.c (MIwoc)             96.109
MI comment weight           (MIcw)              18.348
MI with comments            (MI)               114.456
-------------------------------------------------------
Metrics for main()
Operator                                     Frequency
--------                                     ---------
!=                                                   1
()                                                   4
*                                                    2
,                                                    2
;                                                    4
=                                                    1
[]                                                   1
else                                                 1
if()                                                 1
{}                                                   3
Operand                                      Frequency
-------                                      ---------
"Erreur, utilisation du programme : eval1 <expression>"
                                                     1
"Le résultat de l´opération est : %d"                1
1                                                    1
2                                                    1
argc                                                 2
argv                                                 2
char                                                 1
eval1                                                1
int                                                  3
main                                                 1
printf                                               2
res                                                  3
Metrics                                      Value
-------                                      -----
Cyclomatic number           (v(G))               2
Number of physical lines    (LOCphy)            14
Number of program lines     (LOCpro)            13
Number of blank lines       (LOCbl)              1
Number of commented lines   (LOCcom)             0
Program length              (N)                 39
Number of operators         (N1)                20
Number of operands          (N2)                19
Vocabulary size             (n)                 22
Number of unique operators  (n1)                10
Number of unique operands   (n2)                12
Program volume              (V)                173.918
Number of delivered bugs    (B)                  0.041
Difficulty level            (D)                  7.917
Effort to implement         (E)               1376.850
Program level               (L)                  0.126
Implementation time         (T)                 76.492
Max nesting depth           (MaxND)              2
Maintainability Index w.o.c (MIwoc)             100.963
MI comment weight           (MIcw)               0.000
MI with comments            (MI)                100.963
-------------------------------------------------------
exemple.c
Operator                                     Frequency
--------                                     ---------
!=                                                   7
#                                                    3
&                                                    2
&&                                                   4
()                                                  37
*                                                   23
+                                                   12
++                                                   4
,                                                   20
-                                                    3
/                                                    1
;                                                   46
<                                                    1
=                                                   15
==                                                   3
[]                                                   1
break                                                3
case ...:                                            4
else                                                 4
for()                                                1
if()                                                 5
return                                               1
sizeof                                               2
switch()                                             1
while()                                              1
{}                                                  12
Operand                                      Frequency
-------                                      ---------
"%d"                                                 2
"Erreur : impossible de diviser par 0"               1
"Erreur, utilisation du programme : eval1 <expression>"
                                                     1
"Le résultat de l´opération est : %d"                1
"erreur : pas de <operator>"                         1
"erreur : pas de <valeur1>"                          1
"erreur : pas de <valeur2>"                          1
'*'                                                  2
'+'                                                  2
'-'                                                  2
'/'                                                  2
'\0'                                                 1
'\n'                                                 1
0                                                   10
1                                                    6
2                                                    1
argc                                                 2
argv                                                 2
ch                                                  12
```

Listing 2. Résultat de mesure de complexité pour toutes les fonctions

char	12
char1	6
char2	5
debut	3
eval1	2
exit	4
extract	4
i	20
include	3
int	13
lgval2	4
main	1
malloc	2
nb	3
operation	3
printf	6
res	3
resultat	7
sscanf	2
strlen	3
val1	4
val2	4
valeur1	6
valeur2	7
void	2

Metrics		Value
Cyclomatic number	(v(G))	16
Max cyclomatic number		14
Average cyclomatic number		6
Number of physical lines	(LOCphy)	111
Number of program lines	(LOCpro)	88
Number of blank lines	(LOCbl)	14
Number of commented lines	(LOCcom)	12
Program length	(N)	396
Number of operators	(N1)	216
Number of operands	(N2)	180
Vocabulary size	(n)	70
Number of unique operators	(n1)	26
Number of unique operands	(n2)	44
Program volume	(V)	2427.196
Number of delivered bugs	(B)	0.851
Difficulty level	(D)	53.182
Effort to implement	(E)	129082.700
Program level	(L)	0.019
Implementation time	(T)	7171.261
Max nesting depth	(MaxND)	3
Maintainability Index w.o.c	(MIwoc)	76.458
MI comment weight	(MIcw)	24.381
MI with comments	(MI)	100.839

tructions de langage suivants incrémentent le nombre cyclomatique : *if (...), for (...), while (...), case ...:, catch (...), &&, ||, ?, #if, #ifdef, #ifndef, #elif.* Le calcul commence toujours avec la valeur 1. A ceci on ajoute le nombre de nouvelles branches. Dans le cas de notre code de test :

- Pour la fonction *eval1*, `v(G)=14 (for : 1 ; && : 4 ; if : 4 ; case : 4)`
- Pour la fonction *extract*, `v(G)=2 (while : 1)`

Le calcul du v(G) du fichier entier commence également avec la valeur 1. Pour chaque fonction on ajoute la valeur v(G) de chaque fonction –1. Notre programme d´exemple a donc la complexité suivante :

```
v(G) = 1 (début) + 13 (fonction eval1) + 1
                      (fonction extract)
                      + 1 (fonction main)
                      = 16
```

Quelles sont les limites acceptables pour v(G) ?

Le nombre cyclomatique d'une fonction devrait être inférieur à 15. Si une fonction a un nombre cyclomatique de 15, il y a au moins 15 (mais probablement plus) chemins d'exécution dans son contenu. Plus de 15 chemins sont difficiles à identifier et tester. La limite raisonnable maximum du nombre Cyclomatique d'un fichier est de 100. Le nombre cyclomatique de chaque fonction et du fichier est donc ici inférieur à la limite.

Les Métriques de Halstead

Les métriques de complexité de Halstead ont été développées par l'américain Maurice Halstead et procurent une mesure quantitative de complexité. Les métriques d'Halstead sont basées sur l'interprétation du code comme une séquence de marqueurs, classifiés comme un opérateur ou une opérande. Halstead ira jusqu'à corréler ses formules avec le calcul du nombre d'hommes-mois nécessaires au développement d'un programme. Du fait qu'ils soient appliqués au code, ils sont plus souvent utilisés comme une métrique de maintenance. Il est évident que les mesures Halstead sont aussi utiles durant le développement, pour accéder à la qualité du code dans les applications denses. Puisque l'entretien devrait être un souci pendant le développement, les mesures de Halstead devrait devraient être utilisées durant le développement du code, afin de suivre les tendances de complexité. Voici les Métriques de Halstead :

- N : Longueur du programme (Program length) ;
- n : Taille du vocabulaire (Vocabulary size) ;
- V : Volume du programme (Program volume) ;
- D : Niveau de difficulté (Difficulty level) ;
- L : Niveau du programme (Program level) ;
- E : Effort d'implémentation (Effort to implement) ;
- T : Temps pour implémenter (Implementation time) ;
- B : Nombre de bugs estimés dans un module ou une fonction (Number of delivered bugs).

Comment sont calculées ces métriques ?

Toutes les métriques de Halstead sont dérivées du nombre d'opérateurs et d'opérandes.

Reprenons la fonction eval1 de notre précédent exemple pour détailler le calcul de cette métrique - le resultat est visible sur le Listing 3.

Le nombre total des opérateurs uniques (n1) est de 22 et le nombre total des opérateurs (N1) est de 157. Le nombre total des opérandes uniques (n2) est de 30 et le nombre total des opérandes (N2) est de 122. Sur la base de ces chiffres on calcule la *Longueur du programme* (N). N est la somme du nombre total d'opérateurs (N1) et d'opérandes (N2) du programme `N = N1 + N2`. Pour notre exemple N est donc `157 + 122 = 279`. La somme du nombre d'opérateurs uniques (n1) et d'opérandes uniques (n2) donne la *Taille duvocabulaire* (n) `n = n1 + n2` .La taille du vocabulaire de notre programme exemple est *22 + 30 = 52*. Avec la longueur du programme multipliée par le logarithme 2 de la taille du vocabulaire (n) on obtient le *Volume du programme* (V). Le volume de Halstead (V) décrit la taille de l'implémentation d'un algorithme. est basé sur le nombre d'opérations effectuées et d'opérandes gérées dans l'algorithme. Par conséquent V est moins sensible à la disposition du code que les mesures de lignes de code.Voici la formule :

```
V = N * log2(n)
```

Pour notre exemple cela donne `V = 279 * log2(52) = 1590,423`. Le volume d'une fonction devrait être au minimum à 20 et au maximum à 1000. Le volume d'une fonction,

d'une ligne et sans paramètre, qui n'est pas vide est d'environ 20. Un volume plus grand que 1000 indique que la fonction comporte probablement trop de choses. Le volume d'un fichier devrait être au minimum à 100 et au maximum à 8000. Ces limites sont basées sur les volumes mesurés pour les fichiers pour lesquels le nombre cyclomatique de Mc Cabe v(G) est près de sa limite recommandée. La métrique de Mc Cabe peut d'ailleurs être utilisée pour une double vérification.

Le Niveau de difficulté (D) ou propension d'erreurs du programme est proportionnel au nombre d'opérateurs unique (n1) dans le programme. La métrique D est aussi proportionnelle au ratio entre le nombre total d'opérandes (N2) et le nombre d'opérandes uniques (n2). Si les mêmes opérandes sont utilisés plusieurs fois dans le programme, il est plus enclin aux erreurs.

```
D = ( n1 / 2 ) * ( N2 / n2 )
```

Pour notre exemple D à la valeur suivante : `( 22 / 2 ) * ( 122 / 30 ) = 11 * 4,0666 = 44,733`

L'inverse du Niveau de difficulté donne *le Niveau de programme* (L) ou de la propension d'erreurs du programme. Un programme de bas niveau est plus enclin aux erreurs qu'un programme de haut niveau.

```
L = 1 / D
```

Pour notre exemple `L = 1 / 44,733 = 0,022`

L'Effort à l'implémentation (E) ou à compréhension d'un programme est proportionnel au volume (V) et au niveau de difficulté (D) d'un programme. Il est obtenu par la formule suivante :

```
E = V * D
```

L´effort à l´implémentation de notre exemple est donc `E = 1590,423 * 44,733 = 71144,908`

Le Temps pour implémenter (T) ou pour comprendre qu'un programme est proportionnel à l'Effort (E). Des expérimentations empiriques peuvent être utilisées pour le calibrage de cette quantité. Halstead a découvert que diviser l'effort par 18 donne une approximation pour le temps en secondes.

```
T = E / 18
```

Le T de notre programme est `T = 71144,908 / 18 = 3952,466`

Le temps nécessaire pour écrire notre fonction devrait donc être un peu plus d´une heure.

L'Effort à l'implémentation (E) sert pour calculer le Nombre de bugs fournis (B) selon la formule suivante:

```
B = ( E ** (2/3) ) / 3000   ** correspond "
                            à l'exposant"
```

Cette valeur donne une estimation du nombre d'erreurs dans l'implémentation. Le calcul B = (71144,908 ** (2/3)) / 3000 nous montre qu´un logiciel de la complexité de notre exemple a probablement 0,572 erreurs.

La métrique B peut être utilisée comme indication pour le nombre d'erreurs qui devraient être trouvées pendant le test du module. Des expériences ont montré que, quand on programme en C ou C++, un fichier source contient presque toujours plus d'erreurs que B ne suggère. Le nombre de défaut tend à grandir plus rapidement que B.

L'Index de Maintenabilité (MI)

L'Index de Maintenabilité a été défini durant les dix dernières années aux Etats-Unis. Il aide à réduire ou à renverser la tendance d'un système vers *l'entropie de code* ou l'intégrité dégradée, et pour indiquer quand il devient moins cher et moins risqué de réécrire le code plutôt que de le corriger. L'Index de Maintenabilité est calculé à partir des résultats de mesures de lignes de code, des métriques de Mc Cabe et des métriques de Halstead. Il y a trois variantes de l'Index de Maintenabilité :

Listing 3. Le calcul de la fonction eval1

```
/*Voici le nombre des opérateurs :*/
Operator                                Frequency
--------                                ---------
!=                                              6
&                                               2
&&                                              4
()                                             31
*                                              14
+                                              11
++                                              1
,                                              12
-                                               3
/                                               1
;                                              33
=                                              10
==                                              3
break                                           3
case ...:                                       4
else                                            3
for()                                           1
if()                                            4
return                                          1
sizeof                                          2
switch()                                        1
{}
/*ainsi que le nombre des opérandes :*/
Operand                                 Frequency
-------                                 ---------
"%d"                                            2
"Erreur : impossible de diviser par 0"          1
"erreur : pas de <operator>"                    1
"erreur : pas de <valeur1>"                     1
"erreur : pas de <valeur2>"                     1
'*'                                             2
'+'                                             2
'-'                                             2
'/'                                             2
'\0'                                            1
0                                               9
1                                               5
ch                                             12
char                                            7
eval1                                           1
exit                                            4
extract                                         2
i                                              16
int                                             5
lgval2                                          4
malloc                                          2
operation                                       3
printf                                          4
resultat                                        7
sscanf                                          2
strlen                                          3
val1                                            4
val2                                            4
valeur1                                         6
valeur2                                         7
```

- la maintenabilité calculée sans les commentaires (MIwoc, Maintainability Index without comments) :

MIwoc = 171 - 5.2 * ln(aveV) -0.23 * aveG - 16.2 * ln(aveLOC)

```
aveV = valeur moyenne du volume d'Halstead
                    Volume (V) par module
aveG = valeur moyenne de la complexité
                    cyclomatique v(G) par
                    module
aveLOC = nombre moyen de lignes de code
                    (LOCphy) par module
```

- la maintenabilité concernant des commentaires (MIcw, Maintainability Index comment weight) :

MIcw = 50 * sin(sqrt(2.4 * perCM))

- l'Index de maintenabilité (MI, Maintainability Index) est la somme de deux précédents :

MI = MIwoc + MIcw

La valeur du MI indique la difficulté (ou facilité) de maintenir une application entière. Calculons MI sur notre fonction eval1, d´abord la maintenabilité calculée sans les commentaires :

```
MIwoc = 171 - 5,2 * ln(aveV) -0,23 * aveG
                    -16,2 * ln(aveLOC)
aveV = valeur moyenne du volume d'Halstead
                    Volume (V) 1590,423
aveG = valeur moyenne de la complexité
                    cyclomatique v(G) 14
aveLOC = nombre moyen de lignes de code
                    (LOCphy) 72
```

La valeur moyenne est utilisée quand on veut avoir l´index de maintenabilité pour plusieurs fonctions ou pour le programme entier.

```
MIwoc = 171 - 5.2 * ln(1590,423) - 0.23 *
                    14 - 16.2 * ln(72)
MIwoc = 171 - 5.2 * 7,371755 - 0.23 * 14 -
                    16.2 * 4,276666
MIwoc = 171 - 38,333126        - 3,22
                    - 69,281989
MIwoc = 60,165
```

Ensuite la maintenabilité concernant les commentaires :

```
MIcw = 50 * sin(sqrt(2.4 * perCM))
```

Plus haut nous avons trouvé que la part des commentaires dans notre fonction est de 11/72, donc 15,2778%.

```
MIcw = 50 * sin(sqrt(2.4 * perCM))    ***was
                    ist perCM***
MIcw = 28,460
```

Enfin Index de maintenabilité:

```
MI = MIwoc + MIcw
MI = 60,165 + 28,460
MI = 88,625
```

Une valeur de 85 ou plus indique une bonne maintenabilité, une valeur entre 65 et 85 une maintenabilité modérée. Des modules avec un MI inférieur à 65 sont difficile à maintenir : il est donc préférable de re-écrire ces parties du code. Notre fonction eval1 a donc une maintenabilité correcte.

Le calcul des métriques pour d'autres fonctions de notre exemple

Si vous voulez vous exercer à trouver les métriques pour les autres fonctions, à vos calculettes. Le listing 2 vous montre toutes les métriques de notre programme exemple. Vous allez peut-être remarquer, que déterminer la complexité du code en calculant les différentes métriques est loin d'être une tâche aisée. C'est pourquoi il est judicieux d´utiliser un outil qui se charge de calculer pour vous les métriques et donner ainsi la complexité de vos fichiers en quelques secondes.

Conclusions

Les métriques de ligne de code, les métriques Halstead, le nombre cyclomatique de McCabe et l'index de maintenabilité sont des moyens efficaces pour mesurer la complexité, la qualité et la maintenabilité d'un logiciel. Ces métriques peuvent très bien servir pour mesurer la qualité d'un code. Vous pouvez ainsi localiser les modules en proie à une difficulté particulière de test et de maintenance. Des actions correctives peuvent alors être enclenchées pour corriger une complexité trop élevée plutôt que de garder des modules susceptibles d'être cher en maintenance.

KLAUS LAMBERTZ

Klaus Lambertz est co-fondateur de Verifysoft Technology ***http://www.verifysoft.com*** *basée à Offenbourg en Allemagne, près de la frontière française. Cette société a pour objectif de fournir des solutions et des conseils dans le domaine des tests et d'analyse des logiciels. Né en 1962 à Cologne/Allemagne, Klaus vit depuis 1993 en France. Avant de créer Verifysoft Technology il a occupé les postes d'ingénieur commercial* ***grands comptes*** *et de manager d'équipe des ventes pour différentes sociétés dans le domaine des tests de logiciels en France et en Allemagne.*

Contact : ***lambertz@verifysoft.com***

Xavier-Noël Cullmann

Xavier-Noël Cullmann, né 1984 à Mulhouse, a étudié à l'IUT Informatique d'Illkirch près de Strasbourg. Il a rejoint l'équipe de Verifysoft Technology début 2006. Il est le président de l'association Zilat ***http://www.zilat.net*** *organisant des Lanparties dans le Haut-Rhin.*

Contact : ***cullmann@verifysoft.com***