Cours Test et Qualité
Chapitre 4 :Les métriques

INSTITUT SUPERIEUR INFORMATIQUE ISI المعهد العالي للاعلامية

Responsable du cours :
Héla Hachicha

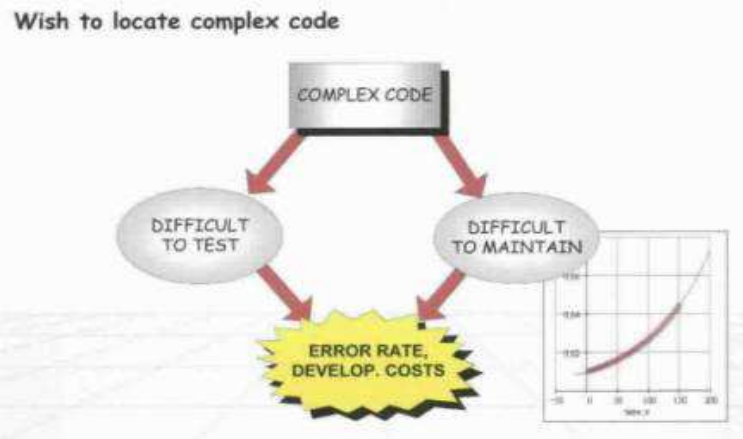
Année Universitaire : 2016 - 2017

Introduction

- La complexité de code
  - a une influence directe sur la qualité et le coût d'un logiciel.
  - a un impacte sur
    - la durée de vie
    - l'exploitation d'un logiciel
    - son taux de défauts
    - sa testabilité
    - maintenabilité.

3

Introduction



4

Introduction

- Une bonne compréhension et maîtrise de la complexité d'un code
  → permet de développer un logiciel de meilleure qualité.
- Plus un document est complexe

  → plus il sera difficile à comprendre et à analyser
  → donc à corriger.

5

Introduction

- En développement de logiciel, la complexité d'une application a un impact direct sur
  - le pourcentage d'erreurs
  - la robustesse du code,

puisque la complexité du logiciel se reflète sur sa difficulté à être testé.

6

Introduction

- **Mesure de la complexité d'un logiciel** dès le début du codage.

Un logiciel de qualité
+
Des coûts de test et de maintenance faibles

- Pour quantifier la complexité d'un logiciel :

Métriques

7

Les métriques

- Les métriques peuvent être classées en trois catégories :
  - métriques mesurant le processus de développement;
  - métriques mesurant des ressources ;
  - métriques de l'évaluation du produit logiciel.
    - Les métriques de produits mesurent les qualités du logiciel.
    - Exemple:
      - les métriques orientées objet
      - les métriques traditionnelles

8

Métriques orientées objet

- Les métriques orientées objet prennent en considération les relations entre éléments de programme (classes, méthodes).

9

Métriques traditionnelles

- Ils se divisent en deux groupes :
  - les métriques mesurant la taille et la complexité:
    - Les métriques de ligne de code
    - les métriques de Halstead.
  - Les métriques mesurant la structure du logiciel
    - Ils se basent sur des organigrammes de traitement ou des structures de classe.
    - Exemple:
    - la complexité cyclomatiquede de McCabe

10

Les métriques des Lignes de code

- Pour quantifier la complexité d'un logiciel, les mesures les plus utilisées sont

les lignes de code
(LOC acronyme de « linesof code).

11

Les métriques des Lignes de code

- On peut distinguer les types de métriques de lignes de code suivants :
  - *LOCphy : nombre de lignes physiques (total des lignes des fichiers source) ;*
  - *LOCpro : nombre de lignes de programme (déclarations, définitions, directives, et code ;*
  - *LOCcom : nombre de lignes de commentaire ;*
  - *LOCbl : nombre de lignes vides (number of blank lines).*

12

Les métriques des Lignes de code

- *Quelles sont les limites acceptables ?*
  - La longueur des fonctions devrait être de 4 à 40 lignes de programme.
  - Une définition de fonction contient au moins un prototype, une ligne de code, et une paire d'accolades → 4 lignes.

13

Les métriques des Lignes de code

- Règle 1:

Une fonction plus grande que 40 lignes de programme doit pouvoir s'écrire en plusieurs fonctions plus simples.

14

Les métriques des Lignes de code

- Règle 2:

La longueur d'un fichier devrait contenir entre 4 et 400 lignes de programme

- Un fichier de 4 lignes de programme correspond à une seule fonction de 4 lignes
- Un fichier de plus de 400 lignes de programme est généralement trop long pour être compris en totalité.

15

Les métriques des Lignes de code

- Règle 3:

Pour aider à sa compréhension, on estime qu'au minimum 30 % et 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 75% du fichier est commenté, le fichier n'est plus un programme, mais un document.

16

Les Métriques de Halstead

- Les Métriques de Halstead
  - sont introduit par l'américain Maurice Halstead.
  - procurent une mesure quantitative de complexité
  - sont basées sur l'interprétation du code comme une séquence de marqueurs, classifiés comme un opérateur ou une opérande.

17

Les Métriques de Halstead

- Notation:
- nombre total des opérateurs uniques (n1)
- nombre total des opérateurs (N1)
- nombre total des opérandes uniques (n2) (termes, constantes , variables)
- nombre total des opérandes (N2)

- Exemple :
- a := a + 1;
  - 3 opérateurs ➔ + := ;
  - 2 opérandes ➔ a 1

18

Les Métriques de Halstead

- La *Longueur du programme (N) :*

  $N = N1 + N2$

- La *Taille du vocabulaire (n) :*

  $n = n1 + n2$

19

Les Métriques de Halstead

- le *Volume du Programme (V) :*

$$V = N * \log_2(n)$$

- Le volume d'une fonction devrait être compris entre 20 et 1000.
- Le volume d'un fichier devrait être au minimum à 100 et au maximum à 8000.

20

Les Métriques de Halstead

- *Le Niveau de difficulté (D) ou propension d'erreurs du programme:*

$$D = (n_1 / 2) * (N_2 / n_2)$$

- Si les mêmes opérandes sont utilisés plusieurs fois dans le programme, il est plus enclin aux erreurs.

21

Les Métriques de Halstead

- Le *Niveau de programme (L)*
  L = 1 / D
- *L'Effort à l'implémentation (E) :*
  E = V * D
- Halstead a découvert que diviser l'effort par 18 donne une approximation pour le *Temps pour implémenter (T) un programme en secondes.*
  T = E / 18

22

Les Métriques de Halstead

- *« nombre de bugs fournis »*

  B = (E (2/3)) / 3000

- Cette valeur donne une indication pour le nombre d'erreurs qui devrait être trouver lors du test de logiciel.

Les Métriques de Halstead : Exemple 1

```
z = 0;
while x > 0
    z = z + y;
    x = x - 1;
end-while
print (z);
```

Questions :

1. Déterminer le nombre d'opérateurs uniques
2. Déterminer le nombre d'opérandes uniques

- Opérateurs : `= ; while/end-while > + - print ()` $\eta_1 = 8$
- Opérandes : `= z 0 x y 1` $\eta_2 = 5$

Les Métriques de Halstead : Exemple 1

Questions :

3. Déterminer le nombre d'opérateurs
4. Déterminer le nombre d'opérandes
5. Estimation de la longueur
6. Estimation du Volume

```
z = 0;
while x > 0
    z = z + y;
    x = x - 1;
end-while
print (z);
```

Longueur d'un programme

- N_1 : Nombre total d'opérateurs
- N_2 : Nombre total d'opérandes
- $N = N_1 + N_2$: Nombre total de jetons

Opérandes		Opérateurs	
=	3	z	4
;	5	0	2
w/ew	1	x	3
>	1	y	2
+	1	1	1
-	1		
print	1		
()	1		

$N_1 = 14$, $N_2 = 12$ donc $N = 12$

Estimation de la longueur

- Estimation de N à partir de η_1 et de η_2

$$N^{est} = \eta_1 + \log_2(\eta_1) + \eta_2 \times \log_2(\eta_2)$$

- Dans notre exemple : $N^{est} = 8 \times \log_2(8) + 5 \times \log_2(5) = 8 \times 3 + 5 \times 2.32 = 35.6$
  - Ici, $N^{est} >> N$
  - En pratique, *Si la différence entre N et N^{est} est supérieure à 30%, il vaut mieux renoncer à utiliser les autres mesures de Halstead*

25

Les Métriques de Halstead : Exemple 1

Volume

- Estimation du nombre de bits nécessaires pour coder le programme mesuré

$$V = N \times \log_2(\eta_1 + \eta_2)$$

- Dans notre exemple : $V = 26 \times \log_2(13) = 26 \times 3.7 = 96,2$

26

Les Métriques de Halstead : Exemple 2

- Calculez les mesures de Halstead pour le pseudo-code suivant :

```
read x , y , z;
type=”scalène”;
i f ( x==y or x==z or y==z ) type=”isocèle”;
i f ( x==y and x==z ) type=”équilatéral”;
i f ( x>=y+z or y>x+z or z>=x+z ) type=”pas un
  triangle”;
i f ( x<=0 or y<=0 or z<=0) type=”données erronées”;
print type;
```

27

Le nombre cyclomatiquede Mc Cabe

- La complexité Cyclomatique :
  - introduite par Thomas McCabeen 1976,
  - est le calcul le plus largement répandu des métriques statiques.
  - Conçue dans le but d'être indépendante du langage
  - indique le nombre de chemins linéaires indépendants dans un module de programme
  - représente finalement la complexité des flux de donnés.
  - correspond au nombre de branches conditionnelles dans l'organigramme d'un programme.

28

Le nombre cyclomatiquede Mc Cabe

- Mac Cabe étudier le logiciel en analysant le **graphe de contrôle** du programme et calcule la **complexité** structurelle ou **nombre cyclomatique** de ce graphe
- Soit
  - n = Nombre de noeuds (blocs d'instructions séquentielles)
  - e = Nombre d'arcs (branches suivies par le programme)
  - v = nombre cyclomatique

29

Le nombre cyclomatiquede Mc Cabe

- Le nombre cyclomatique
  - évalue le nombre de chemins d'exécution dans la fonction
  - donne une indication sur l'effort nécessaire pour les tests du logiciel.

30

Le nombre cyclomatiquede Mc Cabe

- **Calcul du nombre cyclomatique:**
  - Cas n° 1: 1 point d'entrée; 1 point de sortie
    - v= e-n +2
  - Cas n° 2 i points d'entrée; s points de sortie
    - v = e - n + i + s
- Rappel
  - v = nombre cyclomatique
  - n = Nombre de noeuds
  - e = Nombre d'arcs

Le nombre cyclomatiquede Mc Cabe

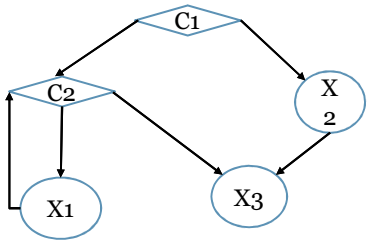
Cas 1:1 point d entrée 1 point de sortie

Cas 2: I point d entrée S point de sortie **V(G)=e-n+2**

V(G)=e-n+i+s

Exemple :

```
If C1 then
   while(C2) loop
     X1
   End loop
Else X2
   X3
```



n=5 , e=6 V(G)=3

NB:i=1 e s=1

Le nombre cyclomatiquede Mc Cabe

- 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.

33

Le nombre cyclomatiquede Mc Cabe

- Principe :

chaque fonction devrait avoir un nombre de cas de tests au moins égal au nombre cyclomatique, pour que tous les chemins soient couverts au moins une fois.

34

Le nombre cyclomatiquede Mc Cabe

- *Quelles sont les limites acceptables pour le nombre cyclomatique?*
  - Une fonction devrait avoir un nombre cyclomatique inférieur à 15.
  - un fichier devrait avoir nombre cyclomatique ne devrait pas dépasser 100.

Le nombre cyclomatiquede Mc Cabe : Exercice

Soit le programme « recherche dichotomique » en langage C:

```
void recherche_dico (elem cle, elem t[], int taille, boolean &trouv, int &A)
{ int d, g, m;
    g=0; d=taille -1;
    A (d+g) /2;
    if (t[A]= =cle) trouv=true;
    else trouv=false;
    while (g <=d && !trouv)
          { m= (d+g) /2;
                  if (t[m]= =cle)
                  {       trouv=true;
                          A=m;
                  }
                  else if (t[m]> cle) g=m+1;
                          else d=m-1;
          }
}
```

Calculer le nombre cyclomatique de cette procédure.

L'Index de Maintenabilité(MI)

- *L'index de maintenabilité permet d'évaluer lorsque le coût de la correction du logiciel est plus élevé que sa réécriture.*
  - Il est conseillé de réécrire des parties du logiciel avec une mauvaise maintenabilité afin d'économiser du temps et donc de l'argent lors de la maintenance.
  - Il est calculé à partir des résultats de mesures de lignes de code, des métriques de Mc Cabe et des métriques de Halstead.

37

L'Index de Maintenabilité(MI)

- Il y a trois variantes de l'index de maintenabilité:
  - *1.la maintenabilité calculée sans les commentaires (MIwoc, MaintainabilityIndex withoutcomments):*

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

- Sachant que:
  - *aveV = valeur moyenne du volume d'Halstead par module*
  - aveG = valeur moyenne de la complexité cyclomatique par module
  - aveLOC = nombre moyen de lignes de code par module

38

L'Index de Maintenabilité(MI)

- Il y a trois variantes de l'index de maintenabilité:
  - *2.la maintenabilité concernant des commentaires (MIcw, MaintainabilityIndex comment weight) :*

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

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

MI = MIwoc+ MIcw

39

L'Index de Maintenabilité(MI)

- La valeur du MI indique la difficulté de maintenir une application
- Si la valeur est 85 ou plus:
  - la maintenabilité est bonne.
- Une valeur entre 65 et 85:
  - la maintenabilité modérée.
- Si la valeur est inférieur à 65:
  - La maintenance est difficile.

→ Il est donc préférable de re-écriredes parties « mauvaises » du code.

40

Conclusion

- Les métriques de ligne de code, le nombre cyclomatique de McCabe, les métriques de Halstead et l'index de maintenabilité sont des moyens efficaces pour mesurer la complexité, la qualité et la maintenabilité d'un logiciel.
- Ces métriques servent également à localiser les modules difficiles à tester et à maintenir.
- 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.