# Comment mesurer la complexité dans le test statique de logiciels

Les normes qui portent sur les logiciels critiques exigent de plus en plus souvent de s'intéresser à la complexité du code. La mesure de la complexité cyclomatique en est un exemple. Des logiciels intègrent désormais cette mesure statique comme l'outil Tessy de l'éditeur allemand Hitex. Mais encore faut-il savoir bien interpréter les résultats.

AUTEUR

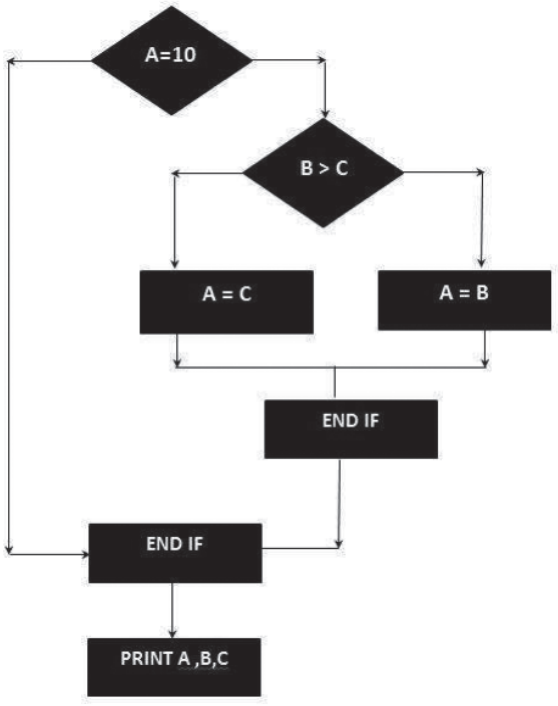
Frank Büchner, ingénieur principal qualité logicielle, Hitex GmbH.

Les normes relatives au développement de logiciels critiques pour la sécurité exigent souvent de gérer la complexité des logiciels. Par exemple, la norme CEI 61508, norme pour la sécurité fonctionnelle des systèmes électriques/électroniques programmables, demande de façon explicite dans son annexe B (informationnelle) de la partie 3 du tableau B.9 (Approche modulaire) une mesure de « contrôle de complexité logicielle ». Cette mesure est recommandée pour les niveaux d'intégrité du logiciel (SIL, Safety Integrity Level) 1 et 2. Et cette mesure est fortement recommandée pour les niveaux SIL 3 et 4.

De même, la norme ISO 26262:2011, norme pour les véhicules routiers, demande dans la partie 6, section 5, tableau 1 que la complexité soit traitée en tant que sujet propre via des règles de modélisation et de codage. Contrairement à la norme CEI 61508, dans la norme ISO 26262, ce sujet est fortement recommandé pour tous les niveaux d'intégrité de sécurité automobile (ASIL, Automotive Safety Integrity Level).
Dans la même veine, la norme CEI 62304, qui s'adresse aux développeurs de logiciels médicaux, donne la conformité du logiciel aux normes de codage comme exemple de critères d'acceptation pour les tests unitaires de logiciels à la section 5.5.3. Dans sa section B.5.5, cette norme indique notamment la prise en compte des exigences pour la gestion de la complexité dans les exemples de normes de codage citées.

**1 COMPLEXITÉ CYCLOMATIQUE**

Dans cet exemple d'analyse de code, il y a 7 nœuds et 8 liens, donc la complexité cyclomatique est de 3.




```
IF A = 10 THEN
IF B > C THEN
A = B
ELSE
A = C
ENDIF
ENDIF
PRINT A
PRINT B
PRINT C
```

## Qu'est-ce que la mesure de la complexité ?

Dans ce paysage normatif, comment alors mesurer la complexité d'un logiciel et établir en conséquence une gestion de la complexité logicielle ? Une approche possible est de s'intéresser à une mesure qui porte déjà la notion de complexité dans son nom : la complexité cyclomatique ou mesure de McCabe. Il s'agit d'un outil de métrologie logicielle (développé par Thomas McCabe en 1976) pour mesurer la complexité d'un programme informatique. Cette mesure reflète le nombre de décisions d'un algorithme en comptabilisant le nombre de « chemins » au travers d'un programme représenté sous la forme d'un graphe. Cette métrique mesure de fait la complexité du flux de contrôle du code source du logiciel, via un graphe de ce flux de contrôle qui se compose de nœuds et de chemins (ou liens).
La formule pour la mesure de la complexité cyclomatique V(G) est la suivante :

**$V(G) = c - n + 2$, où c est le nombre de chemins et n le nombre de nœuds.**

La complexité cyclomatique V(G) est ainsi définie comme le nombre de chemins (c) moins le nombre de nœuds (n) plus 2.
La complexité cyclomatique est équivalente en fait au nombre de chemins linéaires indépendants existant dans un code source et indique donc le nombre de cas de test nécessaires pour exécuter tous ces liens

dans le graphe de flux de contrôle. L'exécution de ces cas-types permet alors d'obtenir une couverture totale des branches (figure 1). On peut remarquer que la complexité cyclomatique augmente linéairement avec le nombre de décisions binaires dans le logiciel.

L'une des faiblesses de cette approche est néanmoins la non-prise en compte des calculs. En effet, théoriquement, un logiciel complexe avec de nombreux calculs, mais sans aucunes décisions (donc sans branches) aurait une complexité cyclomatique de 1, alors qu'un petit logiciel avec une seule décision aurait 2 comme valeur de la complexité cyclomatique. Ce qui est contre-intuitif.

Si l'on prend un exemple plus concret, on observe qu'il est nécessaire de bien analyser la signification de cette métrique. Dans le cas concret présenté figure 2, la fonction fswitch() sur le côté gauche a une complexité élevée, alors que la fonction sinus() sur le côté droit a une complexité faible. Ce qui semble aller à l'encontre de la perception humaine. Car la déclaration switch peut être saisie et comprise par un humain en un coup d'œil, alors que sa complexité cyclomatique est élevée (supérieure à 10). En fait, il faut comprendre que les calculs n'augmentent pas la complexité cyclomatique. Ainsi la fonction sinus() effectue des calculs détaillés, et il est difficile pour un observateur humain de voir ce que la fonction calcule, et encore moins de vérifier si elle le fait correctement. Là encore, il existe un écart important entre la valeur de la complexité cyclomatique, qui est de 2, et la fonctionnalité, qui est difficile à comprendre. De même, les affectations ou les entrées/sorties n'augmentent pas la complexité cyclomatique d'un logiciel bien qu'elles compliquent la compréhension du logiciel pour l'homme.

Malgré cela, la complexité cyclomatique est souvent utilisée pour mesurer la complexité des logiciels. McCabe, l'inventeur de cette mesure, prend 10 comme chiffre clé et valeur acceptable pour la complexité cyclomatique. Les logiciels comportant des numéros plus élevés devant alors être analysés en détail, voire révisés. Ce nombre est largement discuté dans la communauté des développeurs, mais il faut garder à l'esprit qu'à l'origine la métrique a été appliquée pour les programmes écrits en langage Fortran et non en C ou C++.

La version V4.1 de l'outil de test automatisé de modules ou d'unités de logiciels embarqués Tessy de la firme allemande Hitex (distribué en France par NeoMore) intègre pour la première fois une véritable fonction d'analyse statique, car l'outil est désormais capable de mesurer la complexité cyclomatique des objets à tester.

De la complexité cyclomatique, plusieurs autres mesures sont en outre dérivées, à savoir les complexités cyclomatiques totale, moyenne et maximale. Il est alors possible de définir des limites pour la complexité cyclomatique : si sa valeur est supérieure à la limite supérieure, la valeur est affichée en gras et en rouge (indiquant une erreur), si elle se situe entre la limite inférieure et la limite supérieure, la valeur est ombrée en jaune (indiquant un avertissement), tandis que les valeurs inférieures à la limite inférieure sont surlignées en vert.

**2 EXEMPLES DE CODES À ANALYSER**

Dans cet exemple d'analyse de code, la valeur de la complexité cyclomatique est à interpréter en fonction de la nature du code : beaucoup de calcul et peu de décision, ou l'inverse.

```
void f_switch(int month)
{
   switch (month)
   {
        case  1: days = 31; break;
        case  2: days = 28; break;
        case  3: days = 31; break;
        case  4: days = 30; break;
        case  5: days = 31; break;
        case  6: days = 30; break;
        case  7: days = 31; break;
        case  8: days = 31; break;
        case  9: days = 30; break;
        case 10: days = 31; break;
        case 11: days = 30; break;
        case 12: days = 31; break;
        default: days =  0; break;
   }
}
```

```
double sinus(double x_deg)
{
   int i;
   double temp, x_rad;
   int sign = -1;

   x_rad = x_deg / 180 * pi;
   temp = x_rad;

   for(i=3; i<=(MAX_FAC-2); i+=2)
   {
       temp += sign * pot(x_rad,i) / fac(i);
       sign *= -1;
   }

   return(temp);

}
```

## Un affichage de la complexité cyclomatique des objets sous test

Prenons l'exemple d'un programme auquel est appliqué un certain nombre de valeurs correspondant à des cas de test. L'analyse est réalisée avec l'outil Tessy.

L'objet de test « is_equilateral » a une complexité cyclomatique de 2, ce qui est acceptable et donc surligné en vert. Pour l'objet de test « is_isoceles », il est de 12, ce qui est supérieur à la limite spécifiée par McCabe que le logiciel Tessy utilise actuellement comme limite inférieure, et il est donc ombré en jaune. Pour l'objet de test « is_right », elle est de 22, ce qui est au-dessus de la limite supérieure actuellement utilisée de Tessy qui est de 20, elle est donc marquée en rouge.

Pour le module «Triangle», la complexité totale est de 38 (2+12+12+22+2), la complexité moyenne est de 9,50, la complexité la plus élevée est de 22 (pour « is_right ») et le rapport du nombre de cas de test sur la complexité cyclomatique (TC/C) est 0,23, ce qui est beaucoup trop faible. Ceci s'explique par le fait que l'on a utilisé 5 cas de test sur une complexité cyclomatique de 22. Il faudrait idéalement produire 22 cas pour obtenir un rapport de 1 qui est acceptable. ■

**RÉFÉRENCES BIBLIOGRAPHIQUES**


- DIN EN 61508 (VDE 0803): 2011-02, DIN Deutsches £Institut für Normung e.V. und VDE Verband der Elektrotechnik, Elektronik, Informationstechnik e.V. (IEC 61508:2010)
- ISO 26262, International Standard, Road Vehicles – Functional Safety, First Edition, 2011
- IEC 62304, Edition 1.1, 2015-06, VDE Verlag GmbH Stephen H. Kan, Metrics and Models in Software Quality Engineering, 1995, Addison-Wesley, Reading, Mass., USA
- www.hitex.de/tessy