

Testwell CMT++ Code Complexity Measures Tool for C and C++

Le test et la maintenance sont des sources majeures de coût dans les projets de logiciels ; il faut ajouter à cela les coûts engendrés par des programmes de mauvaise qualité ou erronés, qui peuvent être très élevés, et parfois même fatals pour une entreprise. Une grande partie de ces coûts peuvent être attribués à un code inutilement complexe.

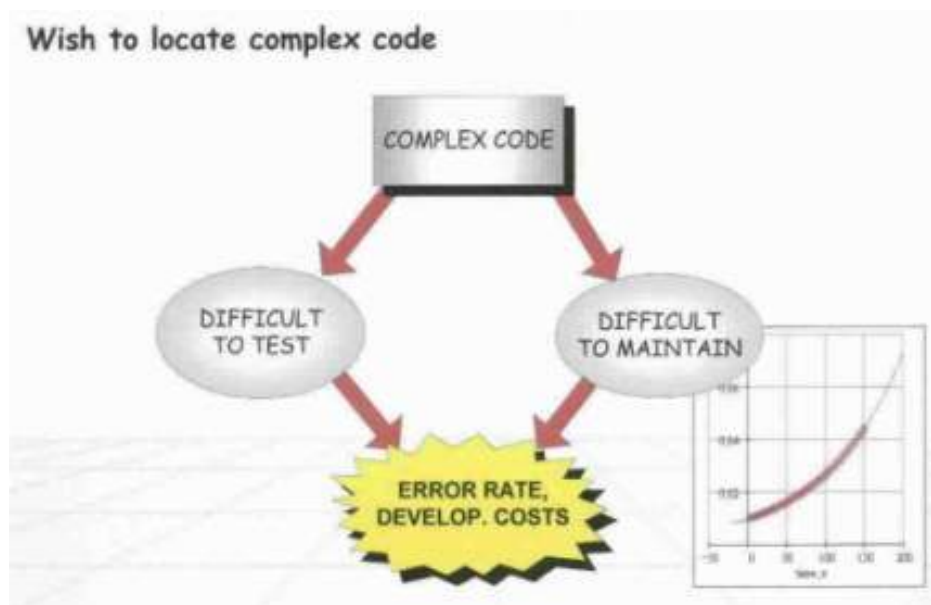
La logique veut que plus un programme est complexe, plus il sera difficile à comprendre et à analyser, et donc à corriger. En terme de développement de logiciel, la complexité d'une application a un impact direct sur le pourcentage d'erreurs et la robustesse du code, puisque la complexité du logiciel se reflète sur sa difficulté à être testé.

Pour garantir un logiciel de qualité tout en assurant des coûts de test et de maintenance faibles, la complexité d'un logiciel devrait être mesurée dès le début du codage. Ainsi lorsque les valeurs recommandées sont dépassées, le développeur peut intervenir rapidement.

Pour quantifier la complexité d'un logiciel, on se sert de métriques.

Testwell CMT++ et CMTJava fournissent des estimations sur les probabilités d'erreurs dans l'application, sur la durée requise pour la compréhension du code, sur le volume logique du code, etc ...

Comme l'équipe travaillant sur un projet n'a généralement pas le temps de vérifier tout le code, CMT++ et CMTJava vous permettent de localiser les modules risquant de poser des problèmes.
















Voici une analyse (summary view) de Testwell CMT++ :


```
*****
*           CMT++, Complexity Measures Tool for C/C++, Version 4.2           *
*                                                                           *
*                   COMPLEXITY MEASURES REPORT                             *
*                                                                           *
*                   Copyright (c) 1993-2007 Testwell Oy                    *
*****
```

License notice: This is a limited period evaluation copy license.

The input CMT++ report was produced at Fri Jul 23 16:07:44 2010
 cmt options: -o H:\cmt\report.tmp -f H:\cmt\files.txt
 Html'ized by cmt2html v2.2 at Fri Jul 23 16:07:47 2010
 cmt2html options: -i H:\cmt\report.tmp -nsb

This is SUMMARY view. Go to [DETAILED](#) view. See [instructions](#).

| Alarms-% | Measured object | v(G) | LOCphy | LOCpro | c% | V | B | MI |
|---|-----------------|------|--------|-------------|----|---------------|--------------|------------|
|  | sid_adm.c | 13 | 385 | 284 | - | 7766 | 1.41 | 84 |
|  | sid_config.c | 1 | 828 | 789- | - | 22026- | 1.07 | 24- |
|  | sid_config.h | 2 | 131 | 73 | | 2270 | 0.22 | 98 |
|  | sid_gamem.c | 1 | 154 | 116 | - | 940 | 0.32 | 102 |
|  | sid_sa.c | 35 | 714 | 479- | - | 15045- | 3.96- | 101 |
|  | adm.h | 4 | 68 | 12 | | 161 | 0.03 | 124 |
|  | basicDefines.h | 3 | 55 | 11 | | 221 | 0.04 | 125 |
|  | ConfigData.h | 4 | 457 | 241 | | 6908 | 0.85 | 70 |
|  | crc32.h | 4 | 93 | 14 | | 165 | 0.04 | 119 |
|  | ga.h | 4 | 64 | 16 | | 455 | 0.10 | 117 |
|  | hmi410_common.h | 2 | 63 | 7 | | 123 | 0.02 | 127 |
|  | ipt_data.h | 4 | 82 | 26 | | 425 | 0.08 | 115 |
|  | lastRitesBSP.h | 4 | 34 | 14 | | 195 | 0.04 | 126 |

 OVERALL (22 %)

OVERALL SUMMARY:

| Measure | 47 Files | | | 108 Functions | | |
|----------------------------|-----------|-----------|----------|---------------|-----------|---------|
| | Alarmed | % | Limits | Alarmed | % | Limits |
| Cyclomatic number v(G) | 0 | 0 | 1-100 | 5 | 4 | 1-15 |
| Program lines LOCpro | 7 | 14 | 4-400 | 27 | 25 | 4-40 |
| Comment % | 19 | 40 | 30-75 | 58 | 53 | 30-75 |
| Volume V | 9 | 19 | 100-8000 | 41 | 37 | 20-1000 |
| Estimated number of bugs B | 6 | 12 | 0-2 | 0 | 0 | n/a |
| Maintainability index MI | 1 | 2 | 65- | 6 | 5 | 65- |
| Total | 42 | 14 | | 137 | 25 | |

Files: 47 LOCphy: 9387 LOChl: 1081 LOCpro: 5559 LOCcom: 3150 '': 2760
 v(G) : 273 MI without comments : 74 MI comment weight : 28 MI: 102

Comment analyser le rapport de complexité de votre code ?

Les fichiers indiqués en rouge contiennent des métriques situées hors de valeurs conseillées.

Le rapport a sept colonnes avec des métriques suivantes :
v(g), LOCphy, LOCpro, c%,V, B et MI

v(g)

v(G) est le nombre cyclomatique de Mc Cabe. La complexité cyclomatique (complexité de McCabe), introduite par Thomas McCabe en 1976, est le calcul le plus largement répandu des métriques statiques. Conçue dans le but d'être indépendante du 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 données. Il correspond au nombre de branches conditionnelles dans l'organigramme d'un programme.

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

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. Du fait que le nombre cyclomatique décrit 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 ceux avec une complexité de McCabe plus basse. Le principe de base est que 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.

Les constructions de langage suivants incrémentent le nombre cyclomatique : *if (...), for (...), while (...), case ...:, catch (...), &&, ||, ?, #if, #ifdef, #ifndef, #elif*. Le calcul commence

toujours avec la valeur 1. À ceci on ajoute le nombre de nouvelles branches.

Quelles sont les limites acceptables pour le nombre cyclomatique v(G) ?

Une fonction devrait avoir un nombre cyclomatique inférieur à 15. Si une fonction a plus de 15 chemins d'exécution il est difficile à comprendre et à tester. Pour un fichier le nombre cyclomatique ne devrait pas dépasser 100.

LOCphy

LOCphy indique le nombre de lignes physiques (total des lignes des fichiers sources). LOCphy est une des métriques de ligne de code (une des mesures les plus utilisées). LOC est l'acronyme de « lines of code ». Les métriques de ligne de code sont simples, faciles à comprendre et à compter.

LOCpro

LOCpro indique le nombre de lignes de programme (déclarations, définitions, directives, et code).

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, qui font 4 lignes.

En règle générale, une fonction plus grande que 40 lignes de programme doit pouvoir s'écrire en plusieurs fonctions plus simples. A toutes règles son exception, les fonctions contenant un état de sélection avec beaucoup de branches ne peuvent pas être décomposées en fonctions plus petites sans réduire la lisibilité.

La longueur d'un fichier devrait contenir entre 4 et 400 lignes de programme, ce qui équivaut déjà à un fichier possédant entre 10 et 40 fonctions. Un fichier de 4 lignes de programme correspond à une seule fonction de 4 lignes, c'est la plus petite entité qui peut raisonnablement occuper un fichier source complet.

Un fichier de plus de 400 lignes de programme est généralement trop long pour être compris en totalité.

c%

c% indique si le pourcentage de lignes de commentaires est « correct ». 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.

Seul un fichier « header » déroge à cette règle car lorsqu'il est correctement commenté, le pourcentage de commentaires peut parfois dépasser 75%.

Des fonctions et des fichiers avec un tiret rouge contiennent un nombre trop élevé ou insuffisant de commentaires.

V

V indique le volume, une des métriques de Halstead. Les métriques de complexité de Halstead qui procurent une mesure quantitative de complexité ont été introduites par l'américain Maurice Halstead. Elles sont basées sur l'interprétation du code comme une séquence de marqueurs, classifiés comme un opérateur ou une opérande.

Toutes les métriques de Halstead sont dérivées du nombre d'opérateurs et d'opérandes. Testwell CMT++ et CMTJava « comptent » le nombre total des opérateurs uniques (n1), le nombre total des opérateurs (N1), le nombre total des opérandes uniques (n2) et le nombre total des opérandes (N2).

Sur la base de ces chiffres on calcule :

- La *Longueur du programme* (N) : $N = N1 + N2$.

- La *Taille du vocabulaire* (n) : $n = n1 + n2$

A partir de là, on obtient le *Volume du Programme* (V) en multipliant la taille du vocabulaire par le logarithme 2 : $V = N * \log_2(n)$

Le volume d'une fonction devrait être compris entre 20 et 1000. Le volume d'une fonction, d'une ligne et sans paramètre, qui n'est pas vide est d'environ 20. Une fonction avec un volume supérieur à 1000 comporte probablement trop de choses. Le volume d'un fichier devrait être au minimum à 100 et au maximum à 8000.

Dans notre exemple (copie d'écran ci-dessus) le volume de deux fichiers sont supérieurs au valeur conseillé de 8000, donc marqués en rouge.

B

B indique le « nombre de bugs fournis » (number of delivered bugs), une estimation du nombre d'erreurs dans le programme. Cette valeur donne une indication sur le nombre d'erreurs qui devrait être trouvé lors du test de logiciel.

Des expériences ont montré qu'un fichier source écrit en C ou C++ contient malheureusement très souvent plus d'erreurs que B ne le suggère.

Voici (pour information) les calculs fait par CMT++/CMTJava afin d'obtenir le « nombre de bugs fournis » :

Il faut passer par le *Niveau de difficulté* (D) ou propension d'erreurs du programme

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

Puis on calcule l'*Effort à l'implémentation* (E), une valeur proportionnelle au volume (V) et au niveau de difficulté (D). Cette métrique est obtenu par la formule suivante :

$$E = V * D$$

L'*Effort à l'implémentation* (E) sert à obtenir le « nombre de bugs fournis » (B) avec la formule suivante:

$$B = (E^{(2/3)}) / 3000$$

MI

MI (la dernière colonne de notre copie d'écran) indique l'index de maintenabilité (en anglais maintainability index, 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.

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.

Les formules exactes pour obtenir MI sont disponibles sur notre site web : http://www.verifysoft.com/fr_maintainability.html

Un index de maintenabilité de 85 et plus indique une bonne maintenabilité.

Si la valeur est entre 65 et 85 le maintenabilité est modérée.

Un index de maintenabilité inférieur à 65 indique que la fonction ou le fichier est difficile à maintenir. Dans l'exemple c'est le cas pour le fichier sid_config.c (marqué en rouge).

Comment comprendre le rapport « résumé » de votre analyse de complexité?

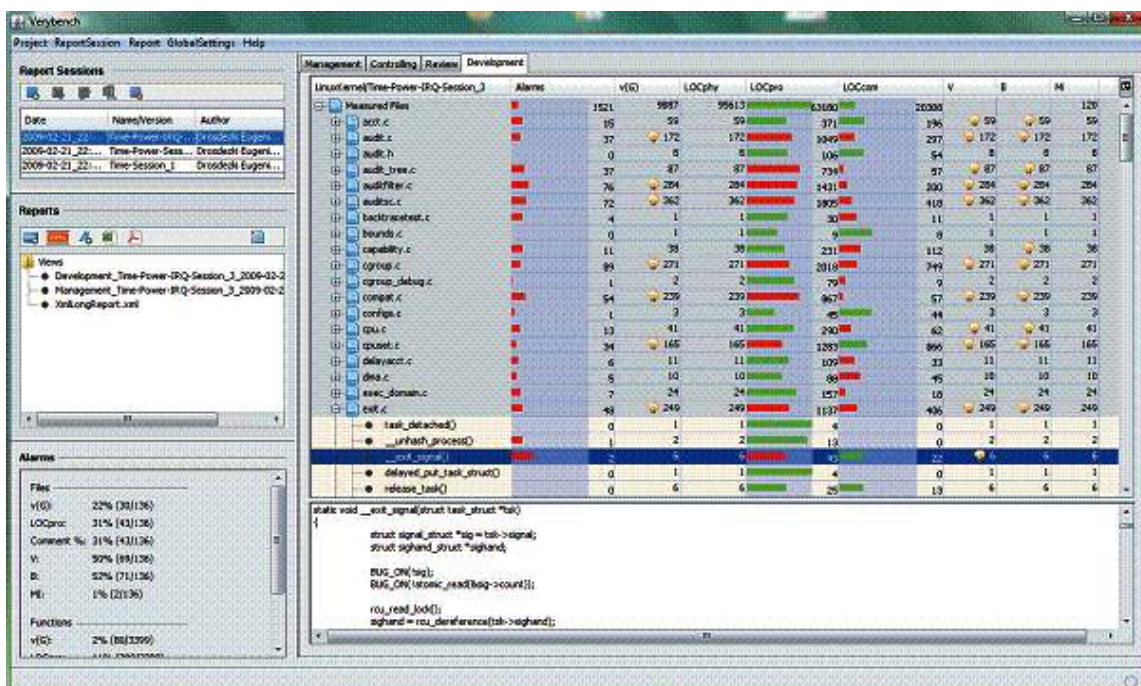
La partie basse de la copie d'écran montre les valeurs conseillées pour chaque métrique. Elle indique le nombre de fichiers « hors limites » et le pourcentage de « mauvais » fichiers et fonctions par rapport au nombre total de fichiers et fonctions.

Verybench for CMT++ et Verybench for CMTJava : nouvelles interfaces graphiques pour Testwell CMT++/CMTJava

L'interface « Verybench » de Testwell CMT++/CMTJava permet d'obtenir différentes présentations de métriques pour développeurs, réviseurs, testeurs et le management. En plus des sorties « classiques » de Testwell CMT++/CMTJava textes, HTML, XML et CSV il est possible d'obtenir des rapports en PDF avec une présentation très claire qui permet de voir rapidement les défaillances de qualité.

Vue « développement »

Les développeurs ont une connaissance détaillée de leurs codes sources et veulent mesurer la qualité de leur travail pour y ajouter des améliorations au niveau des codes. Ainsi, la vue de développement dispose d'une représentation en arbre qui montre tous les fichiers analysés et les fonctions d'une session de report ainsi que les résultats adéquats des mesures.



Dans la représentation en arbre de cette vue, toutes les valeurs mesurées qui ne se trouvent pas à l'intérieur des limites prétextées, sont caractérisées par des petites ampoules lumineuses. La colonne d'alerte et les métriques LOCpro et LOCcom se visualisent par des barres d'avancement.

Lors de l'ouverture de la vue développement, la représentation en arbre s'affiche pour le niveau « fichier ». En cliquant sur un fichier précis les fonctions de ce fichier sont présentées plus en détails. Au choix d'un fichier ou fonction, le code source adéquat est également affiché.

Vue « revue »

La vue „revue“ aide à identifier les parties «critiques » ayant besoin d'un contrôle et d'une attention particulière.

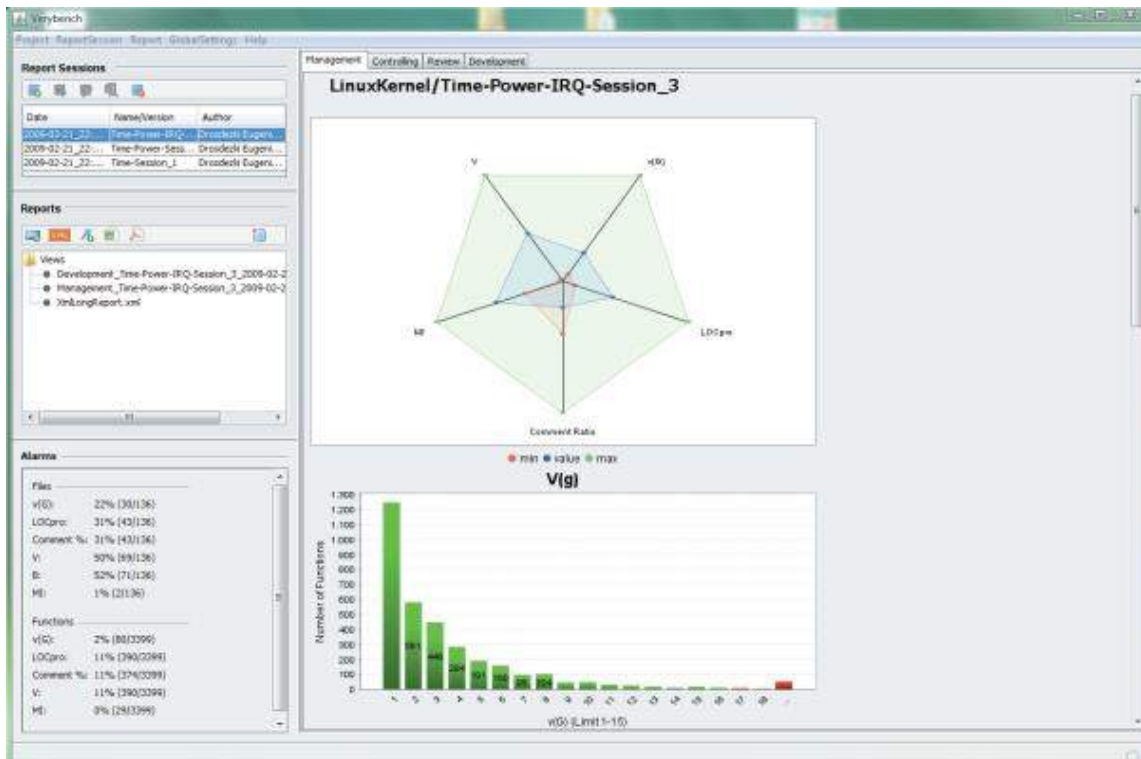
Cette vue montre seulement les fichiers qui se trouvent « à l'extérieur » des limites d'alerte.

Vue « management »

La vue management donne un aperçu rapide indiquant le stade de développement et la qualité du projet.

Les moyennes des métriques de tous les fichiers sources d'une session sont présentées dans un diagramme de Kiviati.

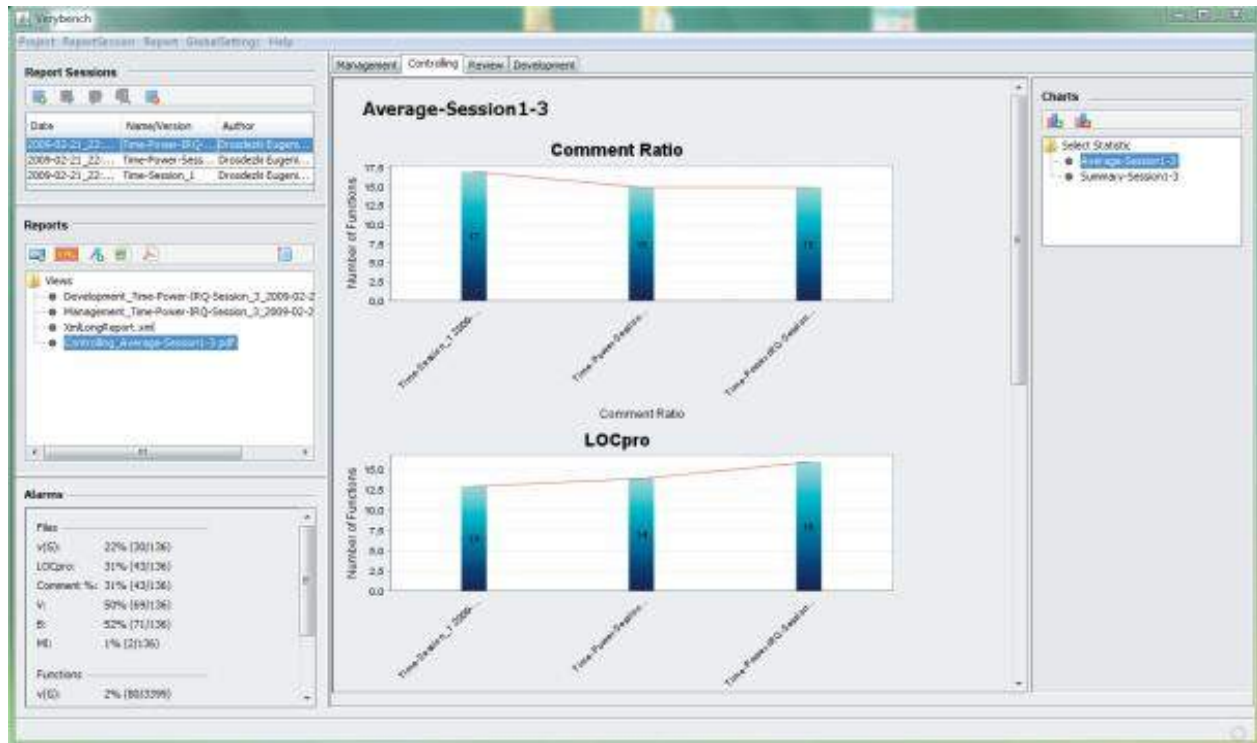
De plus des histogrammes sont établis. Ce diagramme permet de voir, pour la métrique choisie, le nombre de fonctions comprises pour chaque valeur.



Vue « audit »

La vue audit permet de produire des statistiques sur plusieurs sessions et de poursuivre le développement temporel de la complexité d'un projet.

On peut obtenir des statistiques pour des valeurs sommaires et moyennes d'une session ainsi que des valeurs d'une fonction déterminée.



En savoir plus :

Testwell CMT++ et CMTJava : http://www.verifysoft.com/fr_cmtx.html

Métrique de Halstead : http://www.verifysoft.com/fr_halstead_metrics.html

Nombre cyclomatique McCabe : http://www.verifysoft.com/fr_mccabe_metrics.html

Index de maintenabilité : http://www.verifysoft.com/fr_maintainability.html

Interface graphique Verybench : http://www.verifysoft.com/fr_cmtx_verybench.html