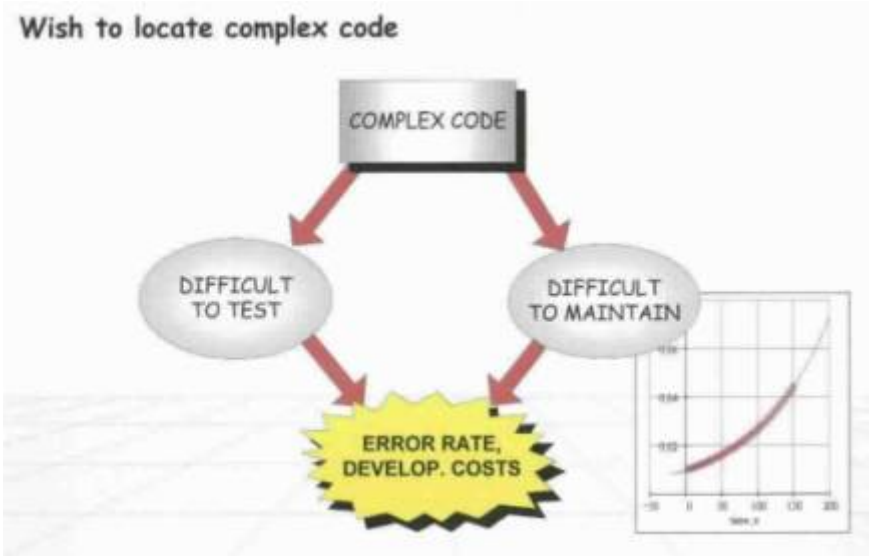
Les métriques (McCabe, Halstead), l'index de maintenabilité et leur influence sur la qualité

par Klaus LAMBERTZ, Verifysoft Technology

La complexité de code a une influence directe sur la qualité et le coût d'un logiciel. Elle impacte sur la durée de vie et l'exploitation d'un logiciel, et plus particulièrement sur son taux de défauts, sa testabilité et sa maintenabilité. Une bonne compréhension et maîtrise de la complexité d'un code permet de développer un logiciel de meilleure qualité.



La logique veut que plus un document 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 mesuré 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.

Les métriques peuvent être classées en trois catégories :

- 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.

Cet article décrit les métriques traditionnelles (les métriques des lignes de code, le nombre cyclomatique de McCabe et les métriques de Halstead) ainsi l'index de maintenabilité.

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 « lines of code ») puisqu'elles sont simples, faciles à comprendre et à compter. Cependant ces mesures ne prennent pas en compte le contenu d'intelligence et la disposition du 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 (en anglais number of blank lines).

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, 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.

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%.

Le nombre cyclomatique de Mc Cabe : v(G)

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 donnés.

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 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 celles avec une complexité de McCabe plus bas. 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. A ceci on ajoute le nombre de nouvelles branches.

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

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

Les Métriques de Halstead

Les métriques de complexité de Halstead qui procurent une mesure quantitative de complexité ont été introduit par l'américain Maurice Halstead. Ils 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 :

- nombre total des opérateurs uniques (n1)
- nombre total des opérateurs (N1)
- nombre total des opérandes uniques (n2)
- 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 * log2(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.

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

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

Le *Niveau de programme* (L) est l'inverse du Niveau de difficulté. Un programme de bas niveau est plus enclin aux erreurs qu'un programme de haut niveau.

$$L = 1 / D$$

L'Effort à l'implémentation (E) est proportionnel au volume (V) et au niveau de difficulté (D). Cette métrique est obtenu par la formule suivante :

$$E = V * D$$

Halstead a découvert que diviser l'effort par 18 donne une approximation pour le *Temps pour implémenter* (T) un programme en secondes.

$$T = E / 18$$

Il est même possible d'obtenir le « nombre de bugs fournis » (B) qui une estimation du nombre d'erreurs dans le programme. Cette valeur donne une indication pour le nombre d'erreurs qui devrait être trouver lors du test de logiciel. Le « nombre de bugs fournis » est calculé selon la formule suivante:

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

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

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.

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 :

* la maintenabilité calculée sans les commentaires (MIwoc, Maintainability Index without comments): MIwoc = 171 - 5.2 * ln(aveV) -0.23 * aveG -16.2 * ln(aveLOC)

Sachant que:

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.

Si la valeur est 85 ou plus la maintenabilité est bonne. Une valeur entre 65 et 85 indique une maintenabilité modérée. Si l´index de maintenabilité est inférieur à 65, la maintenance est difficile. Il est donc préférable de re-écrire des parties « mauvaises » du code.

Conclusions

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 difficile 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.

Mesure de complexité avec Testwell CMT++ et Testwell CMTJava

Les outils Testwell CMT++ et Testwell /CMTJava permettent de mesurer la complexité du code. Grace à l'analyse des fichiers non-préprocessés les outils Testwell sont extrêmement rapide. Même des grands projets peuvent être analysés en quelques minutes.

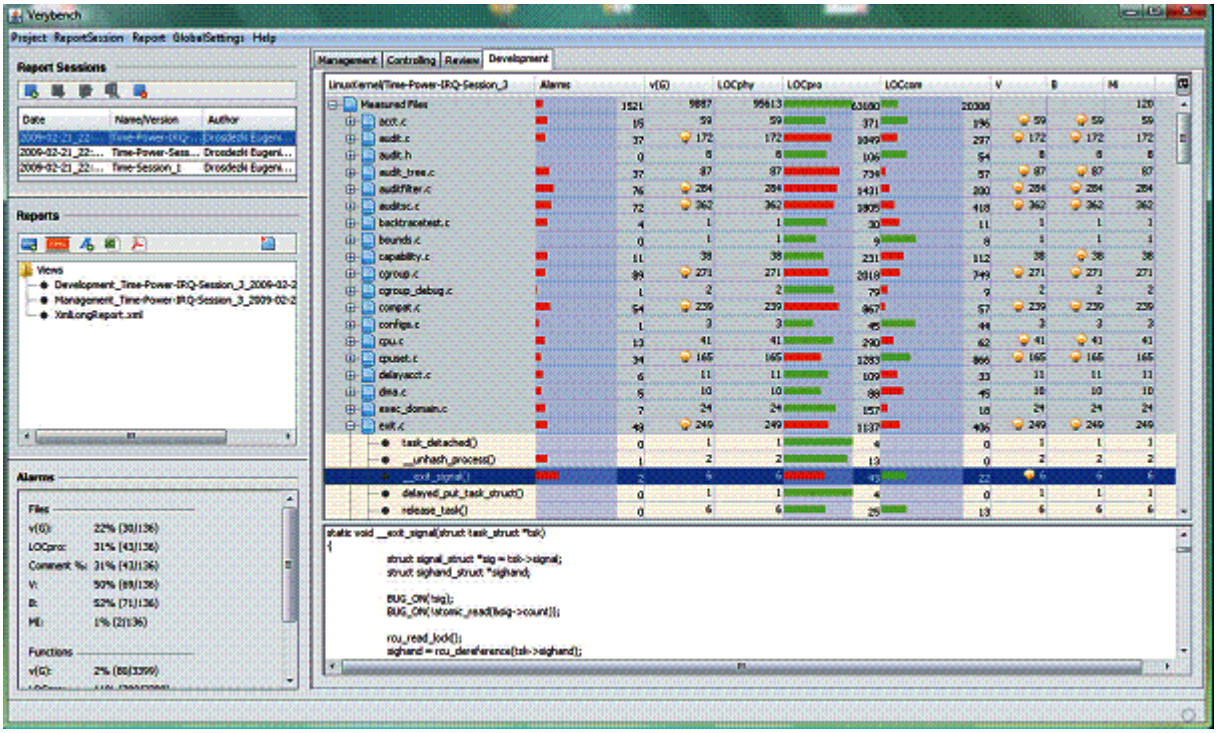
L'interface « Verybench » de Testwell CMT++/CMTJava permet d'obtenir différentes présentations de métriques pour développeurs, réviseurs, testeurs et le management.

En plus des sorties 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.

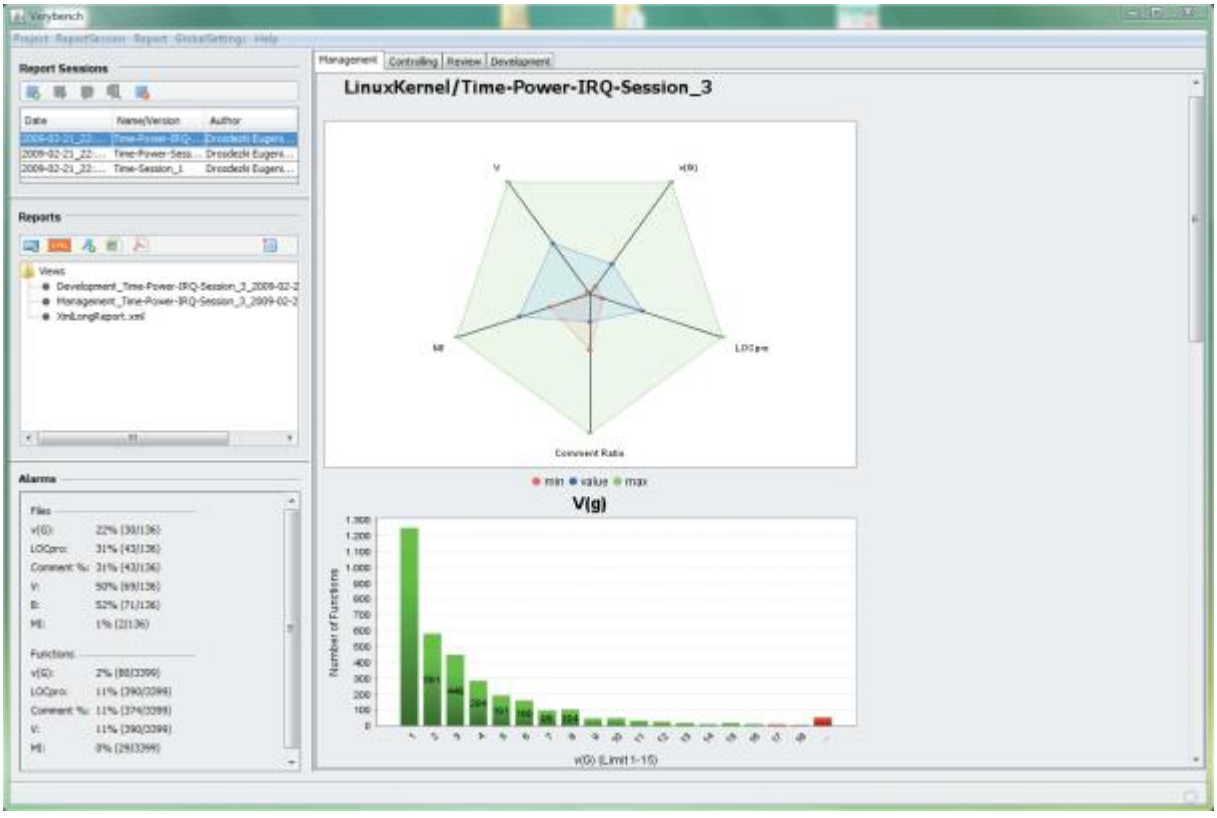
Cette vue montre seulement les fichiers qui se trouvent « à l’extérieur » des limites d ´alertes.

Vue « management »

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

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

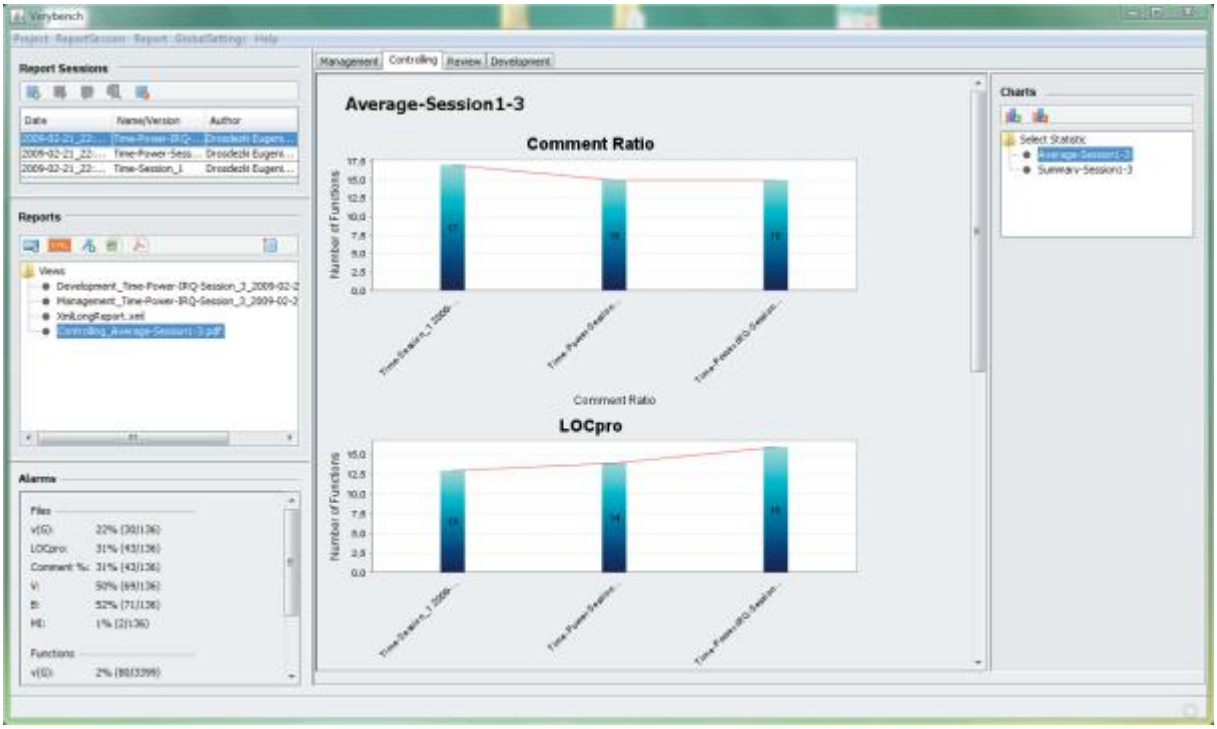
De plus des diagrammes-barres sont établis. Ce diagramme permet de voir, pour la métrique choisie, le nombre de fonctions comprises pour chaque valeur.



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

KLAUS LAMBERTZ

Klaus Lambertz est co-fondateur de Verifysoft Technology http://www.verifysoft.com une société qui a pour objectif de fournir des solutions et des conseils dans le domaine des tests et d'analyse des logiciels. Avant de créer Verifysoft Technology en 2003 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

VERIFYSOFT TECHNOLOGY

Verifysoft Technology http://www.verifysoft.com est spécialisée dans le domaine de la qualité logiciel. La société propose des outils de test tels que Conformiq Qtronic (générateur automatique de test), Testwell CTC++ (couverture de code) et Testwell CMT++/CMTJava (mesure de complexité) ainsi que des séminaires.

Verifysoft Technology se situe à Offenbourg près de Strasbourg.

De nombreuses enseignes de renomée telles que Areva, Alcatel-Lucent, Sopra,

Thalès, Sagem ou encore Siemens et Volkswagen ainsi que de nombreuses PME font déjà confiance aux technologies de Verifysoft.