Qualité des logiciels :

Analyse statique de code et tests dynamiques - Procédures complémentaires pour l'assurance qualité

Par Klaus Lambertz (Verifysoft Technology GmbH)

Pour garantir la qualité des logiciels, deux procédures complémentaires sont utilisées au cours de leur développement : L'analyse statique de code et les tests dynamiques associés à la mesure de la couverture de code. Cet article montre les avantages et les limites de ces deux méthodes et explique pourquoi les tests statiques et dynamiques sont complémentaires et tous deux nécessaires pour garantir un logiciel de haute qualité.

L'analyse statique est une vérification de la structure interne d'une application. Il ne s'agit pas d'une évaluation de la fonctionnalité d'un logiciel.
L'analyse statique permet d'accéder au code sans l'exécuter. Cette analyse peut être effectuée manuellement dans le cadre d'un " Walkthrough " ou une inspection. Bien qu'utile, la qualité de ces évaluations manuelles dépend des employés et de leur état de veille au moment de leur analyse manuelle.
C'est pourquoi les " Walkthrough " et les inspections doivent être complétées par des vérifications automatiques à l'aide d'outils d'analyse statique de code.
Avec les outils d'analyse statique avancés, il est possible d'obtenir une analyse une analyse au-delà des limites du module et une analyse des grandes bases de code avec des millions de lignes de code.
En gros, l'analyse statique consiste à vérifier la présence d'erreurs et le respect des règles de programmation. Pour ce faire, il n'est pas nécessaire d'utiliser des cas de test - l'analyse est effectuée automatiquement par l'outil.

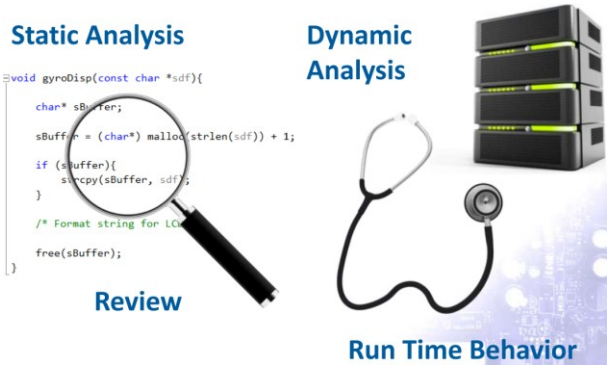

Fig. 1: Analyse statique et tests dynamiques

Les tests dynamiques, quant à eux, adoptent l'approche inverse. Ils sont exécutés pendant qu'un programme est en cours d'exécution. Pendant l'exécution, on vérifie si l'application fonctionne comme prévu. Les cas de test sont utilisés pour vérifier si le logiciel fonctionne conformément aux exigences. Cela signifie que l'on détermine si le logiciel fait ce qu'il doit faire.

En ce qui concerne le coût des tests de logiciels, il est intéressant d'examiner les recherches de Barry Boehm. Bien que les chiffres exacts varient en fonction du produit et du projet, le coût de l'élimination des erreurs augmente toujours de manière exponentielle pendant la durée du processus de développement du logiciel. Lorsqu'un problème est détecté au début de la phase de développement, il peut être corrigé à un coût relativement faible. Si un bogue est détecté pendant la phase de développement, il peut être corrigé même par le développeur lui-même.

Dans le pire des cas, un problème est détecté après la livraison du logiciel. Cela peut entraîner des rappels coûteux, des arrêts de production, des pertes financières et des dommages corporels.

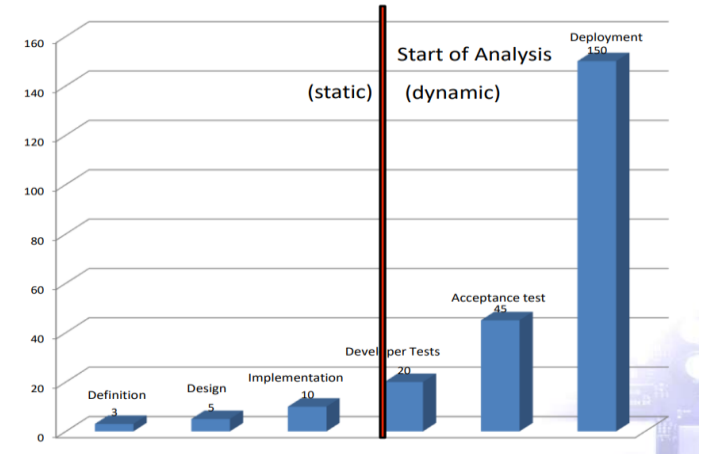

Fig. 2: Coût de la gestion des erreurs (selon Barry Boehm)

Pour les tests, cela signifie que plus les tests sont effectués tôt, plus les erreurs sont détectées rapidement. Elles peuvent alors être corrigées plus tôt, à un moment où la correction est encore relativement peu coûteuse.
En ce qui concerne les tests dynamiques, le logiciel doit être exécuté. Ces tests ne peuvent donc être effectués qu'après le code est disponible.

L'analyse statique, quant à elle, peut être réalisée plus tôt en parallèle de l'écriture de code et déjà avec des petites parties de code.
L'analyse statique du code permet de détecter les erreurs avant que les tests dynamiques ne soient effectués. C'est ce que l'on appelle le "shift left", une pratique qui consiste à déplacer les tests, la qualité et l'évaluation des performances au début du processus de développement.
Une fois qu'un outil d'analyse statique de code est mis en place, il fonctionne "tout seul" et sans effort. Une bonne approche consiste à mettre l'outil à la disposition de chaque développeur, de sorte que le code de chaque développeur ne soit intégré avec le code développé par autres développeurs qu'après avoir été analysé.

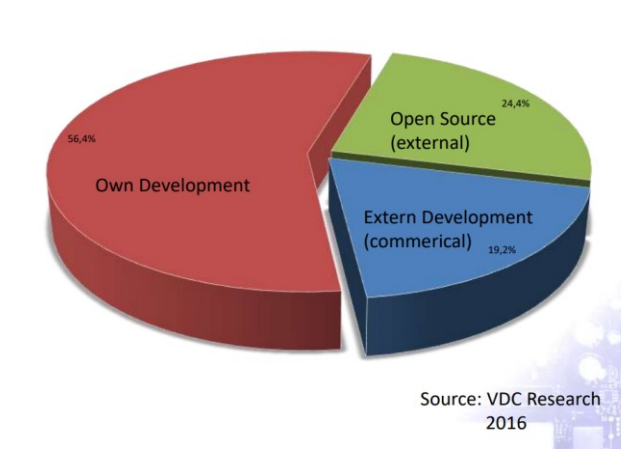

Fig. 3: Code développé en interne et en externe

De nos jours, seule une partie des logiciels est écrite au sein de l'entreprise. Selon une étude de VDC Research réalisée en 2016, seulement 56 % du code est développé en interne. Le reste est constitué de logiciels open-source et de logiciels tiers développés en externe, pour lesquels seuls les binaires sont généralement disponibles, et non le code source. Dans ce contexte, il est bon de savoir que de bons outils d'analyse statique peuvent analyser non seulement le code source mais aussi le code binaire (même si la qualité de l'analyse binaire ne correspond pas toujours à celle de l'analyse du code source).

Aujourd'hui, les outils d'analyse statique vérifient les éléments suivants :

- La syntaxe
  La qualité et la profondeur des vérifications syntaxiques sont bien meilleures que celles des compilateurs.

- La sémantique
  Par exemple, la phrase "Il fait plus froid la nuit qu'à l'extérieur" est syntaxiquement correcte, mais la sémantique ne l'est pas, car cette phrase est absurde. Cependant, l'analyse statique du code ne vérifie la sémantique que dans une mesure limitée, car l'outil doit comprendre ce que signifie le code.

- Les problèmes de flux de contrôle
  Par exemple, un code inaccessible, des sauts dans ou hors des boucles.

- Les problèmes de flux de données
  Comme des variables non initialisées.

- Les problèmes de simultanéité (concurrency problems)
  Il s'agit notamment des « race conditions » (par exemple, les « data races »), des synchronisations qui ont été oubliées ou effectuées de manière incorrecte, et des interblocages (deadlocks).
  Ces problèmes sont une discipline clé de l'analyse statique – ils ne peuvent très souvent pas être détectés lors de l'analyse dynamique au moment de l'exécution.

- Les règles de programmation et de codage
  Les plus connues sont les règles MISRA, mais il en existe beaucoup d'autres. De nombreuses normes de sécurité exigent ces contrôles. De nombreux outils offrent également la possibilité de mettre en œuvre les règles propres à l'entreprise.

- La maintenabilité / les métriques
  Il s'agit par exemple des mesures de Halstead et McCabe. Il existe beaucoup d'autres mesures, même des mesures telles que le "nombre estimé de bogues dans un logiciel" et le "temps de compréhension d'un code".
  Même si le code n'est pas nécessairement mauvais si certains seuils métriques sont dépassés, il existe une corrélation entre la complexité du code et la propension aux erreurs, la testabilité et la maintenabilité.

- L´architecture
  Quelle est la structure du logiciel ? Cette question est particulièrement intéressante pour les codes hérités que personne ne connaît.

- La sécurité
  En raison de la connectivité des logiciels modernes, les contrôles des failles de sécurité deviennent de plus en plus importants. L'analyse statique du code permet de déterminer si le code contient des structures permettant des attaques extérieures.

L'analyse statique de code est obligatoire pour les logiciels critiques liées à la sécurité.

Les normes relatives au développement de logiciels critiques liés à la sécurité, telles que la norme ISO 26262 dans le secteur automobile, exigent une analyse statique de code. L'analyse du flux de contrôle et l'analyse du flux de données sont "fortement recommandées" pour le niveau ASIL C et supérieures. L'analyse statique est obligatoire pour ASIL B et supérieures et recommandée pour ASIL A. Les recommandations des autres normes sont similaires.

Methoden	ASIL			
	A	B	C	D
Walk-through	++	+	o	o
Inspection	+	++	++	++
Semi-formal verification	+	+	++	++
Formal verification	o	o	+	+
Control flow analysis	+	+	++	++
Data flow analysis	+	+	++	++
Static code analysis	+	++	++	++
Semantic code analysis	+	+	+	+

\+ recommended ++ highly recommended

Fig. 4: Méthodes d'analyse statique selon l'ISO 26262

Les outils d'analyse statique du code fournissent de nombreuses vérifications

Les outils d'analyse de code statique disposent de nombreux vérificateurs de sécurité fonctionnelle, notamment pour les dépassements de mémoire, le déréférencement de pointeur NULL, la division par zéro, la libération de la mémoire des variables non hébergées, les fuites de mémoire, les instructions de retour manquantes, et bien d'autres encore.

Les contrôles de sécurité contre des attaques portent notamment sur le dépassement de la mémoire tampon, les chaînes C non terminées, les dépassements de la taille d'allocation par addition, multiplication et soustraction, les noms DNS codés en dur, le stockage en clair du mot de passe, l'utilisation de la fonction system(), et bien d'autres encore.

En résumé, voici les principaux avantages de l'analyse statique du code :
- Peut être effectuée au début du processus de développement
- Fournit un grand nombre de vérificateurs différents
- L'analyse est effectuée automatiquement et généralement sans aucun effort.

La couverture de code est obligatoire pour s'assurer que l'ensemble du code est testé.

L'analyse statique du code est complétée par des tests en cours d'exécution : les tests dynamiques.
Au cours des tests dynamiques, l'application est alimentée par des cas de tests à partir du niveau des tests unitaires. Les résultats des tests sont comparés aux résultats attendus, ce qui permet de vérifier si l'application fonctionne comme prévu.

Pour cela, la qualité des tests est importante. Les tests doivent être basés sur les exigences. Il convient en particulier de tester les valeurs limites et les conditions aux frontières. Il est également important de mesurer la couverture du code, pour voir si l'ensemble de l'application a été testé.

La couverture de code indique si le logiciel a été entièrement testé. S'il reste des parties non testées dans le code, celles-ci peuvent contenir des erreurs dangereuses cachées dans l'application, ce qu'il convient d'éviter.

Cependant, la couverture de code ne donne pas d'informations sur la qualité des tests. Faire "n'importe quel" test pour augmenter le taux de couverture n'est pas une bonne approche.

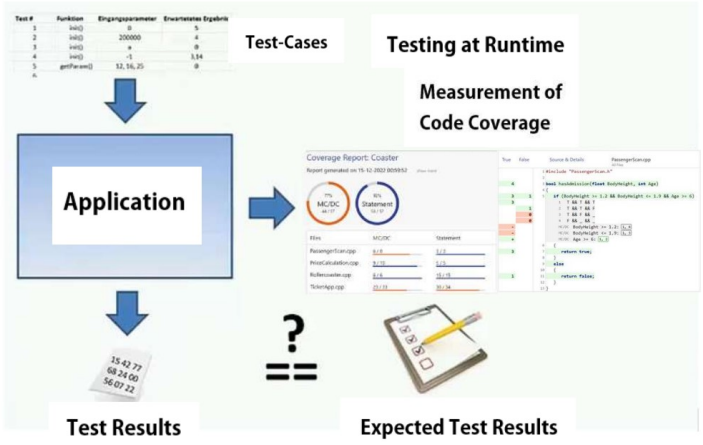

Fig. 5: Test dynamique à l'exécution

La question est toujours de savoir ce que signifie "tests complets" ou "couverture à 100 %".

Cela dépend du niveau de couverture choisi ou requis. Pour commencer, on peut supposer que chaque déclaration a été testée une fois. Il s'agit de Statement Coverage, un niveau de couverture relativement faible.

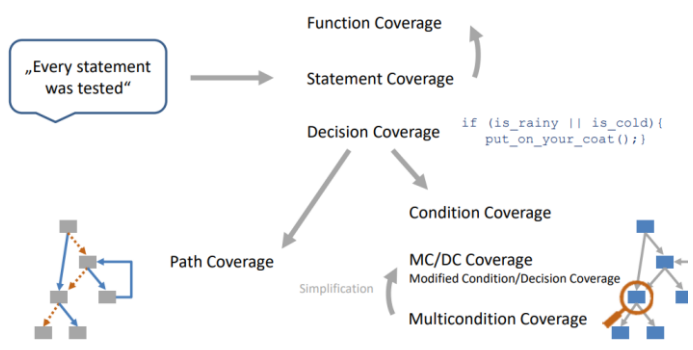

Fig. 6: Les niveaux de Code Coverage

Function Coverage est encore plus faible. Decision Coverage et Condition Coverage sont déjà plus fortes, et Multicondition Coverage est très stricte. Ici, chaque combinaison possible de toutes les conditions atomiques doit être évaluée comme "vraie" et comme "fausse".

Pour un logiciel comportant un grand nombre de conditions atomiques, cela se traduira par un nombre élevé de cas de test. C'est pourquoi il existe une simplification de la couverture Multicondition : la couverture Condition/Decision modifiée (MC/DC), qui produit les mêmes résultats de test avec moins de cas de test. Cette couverture MC/DC est le niveau le plus élevé exigé par les normes de sécurité.

Toutes les normes de sécurité ont un point commun : plus le niveau de sécurité est élevé, plus le niveau de couverture requis est élevé. La norme ISO 26262 exige pour ASIL D, le niveau de sécurité automobile le plus élevé, une couverture MC/DC. Il en va de même pour les normes applicables aux autres secteurs, par exemple l'aviation et l'industrie ferroviaire, ainsi que pour la norme "générale" CEI 61508. Le niveau requis le plus élevé est toujours MC/DC-Coverage.

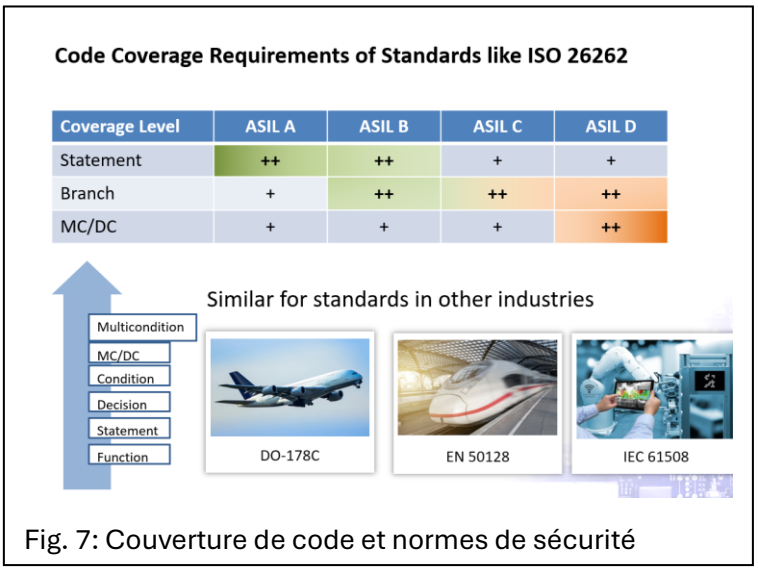

Fig. 7: Couverture de code et normes de sécurité

Analyse statique du code et tests dynamiques associés à la couverture du code

Fig. 8: Analyse statique et tests dynamiques

Les points suivants peuvent être résumés :
La principale application de l'analyse statique du code est la détection des erreurs qui conduisent à un comportement non défini. La détection des problèmes de simultanéité constitue un point fort particulier.

Lorsque nous parlons d'analyse statique du code, nous parlons principalement d'erreurs non déterministes. En d'autres termes, il s'agit d'erreurs qui ne se produisent pas toujours. C'est la raison pour laquelle elles sont difficiles à détecter avec des tests en cours d'exécution. Parfois, ces erreurs non déterministes n'apparaissent qu'après des années, parfois jamais, mais elles peuvent survenir à tout moment.

Bien que l'analyse statique du code révèle des problèmes graves dans le code source, elle ne peut fournir aucune information sur la fonctionnalité correcte du logiciel. Cette information peut être livré par les tests dynamiques dès qu'un programme (ou une partie de celui-ci) peut être exécuté.
L'analyse statique et l'analyse dynamique sont toutes deux essentielles et doivent être utilisées de manière complémentaire. Ce n'est pas l'un ou l'autre, c'est l'un et l'autre !

Critères de sélection des outils

Dans ce qui suit, je décrirai quelques critères de sélection pour le choix des outils. Tout d'abord, comme dans le processus de développement, examinons les outils d'analyse statique de code. Nous avons vu que ces outils présentent de grands avantages : déploiement dès le début du processus de développement, vérification des directives de programmation et détection des erreurs sans avoir à écrire de cas de test.

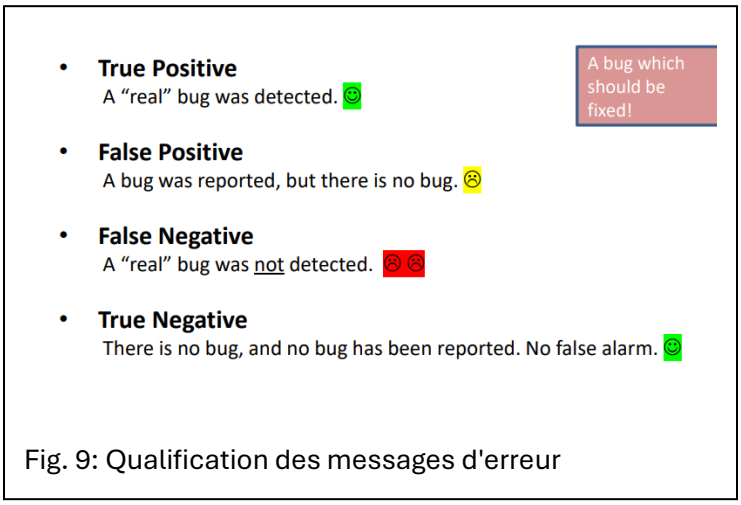

Fig. 9: Qualification des messages d'erreur

Il y a cependant une chose qui rend la découverte des erreurs un peu plus difficile : l'analyse statique du code ne détecte pas toujours les erreurs avec une précision de 100 %.
Lorsque l'outil d'analyse statique signale des "True Positives" et des "True Negatives", tout va bien. Les "True Positives" sont des erreurs qui sont affichées par l'outil et qui sont de vraies erreurs que vous devez corriger. "True Negatives" signifie que l'outil ne signale pas d'erreur et qu'il n'y a pas non plus d'erreur dans le code.

Les choses sont plus difficiles avec les "False Negatives" et les "False Positives". Les "False Negatives" sont des erreurs réelles qui ne sont pas détectées par l'outil. Les "False Positives sont des "erreurs" affichées par l'outil, mais qui ne sont pas des erreurs : ce sont de fausses alertes.
Vous corrigerez bien sûr les True Positives, mais vous examinerez également les False Positives, car vous ne savez pas qu'il s'agit de faux positifs. Cela signifie que vous devez examiner tous les messages d'erreur, qu'il s'agisse d'erreurs réelles ou de fausses alertes.
Le problème des False Positives est qu'ils vous font perdre du temps et qu'après cinq False Positives, vous risquez de ne plus être attentif au sixième, qui pourrait être une véritable erreur...

Il est donc important de disposer d'un outil d'analyse statique du code qui trouve autant d'erreurs réelles que possible (True Positives), qui ne manque aucune (ou très peu) d'erreurs réelles (False Negatives) et qui ne présente pas (ou très peu) de False Positives.
Lors du choix d'un outil, il est également important d'évaluer les « checkers », les règles de programmation et les métriques qui sont pris en charge par l'outil. La possibilité de mettre en œuvre ses propres règles est un avantage.
Il est important que certaines vérifications et règles puissent être désactivées afin de ne pas obtenir tous les messages d'erreur lors de la première exécution de l'outil. Ainsi,

le testeur a la possibilité de se concentrer d'abord sur les erreurs les plus importantes. Pour la même raison, la possibilité d'exclure des parties du projet au cours de l'analyse est avantageuse.

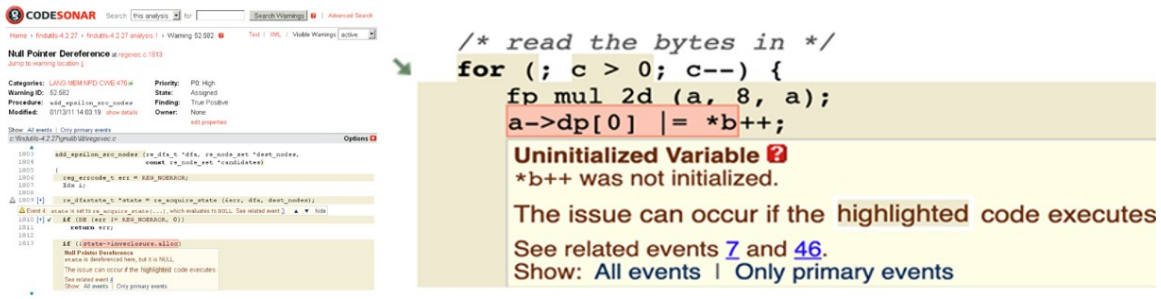

Fig. 10: Message d'erreur d'un outil d'analyse statique (CodeSonar par CodeSecure)

Une bonne présentation des messages d'erreur est importante pour la compréhension. Elle permet ainsi de gagner du temps. Il doit être possible de classer les messages et de disposer de l'aide nécessaire pour qualifier les avertissements. Il est important que les messages puissent être attribués à des membres individuels de l'équipe pour qu'ils les corrigent. Il doit également y avoir des règles d'accès aux résultats.

La performance de l'outil d'analyse statique du code est particulièrement importante pour les grandes bases de code. Lors du choix d'un outil, il convient de vérifier si des bases de code plus importantes, comportant plusieurs millions de lignes de code, peuvent être analysées dans un délai raisonnable. Pour atteindre cet objectif, il serait utile que l'analyse puisse être répartie sur plusieurs ordinateurs. Une analyse incrémentale doit être possible. Le matériel et l'espace mémoire nécessaire doivent être évalués.

En ce qui concerne la sélection d'un analyseur de couverture de code, il est évident que l'outil doit prendre en charge les niveaux de couverture de code requis. Pour un projet ASIL D dans l'industrie automobile, la couverture MC/DC est obligatoire. Même si vous disposez actuellement d'un niveau de sécurité inférieur, vous devriez envisager un outil capable d'analyser des niveaux de couverture plus élevés, car vous pourriez avoir plus tard des projets avec des exigences plus strictes. Des décisions prospectives, vraiment tournées vers l´avenir, lors de l'acquisition d'un outil permettent d'éviter d'avoir à acheter un outil deux fois : un "bon marché" d'abord et un "bon", plus performant, plus tard.

L'outil doit bien sûr fonctionner avec le compilateur que vous utilisez. Mais ici aussi, pour les raisons mentionnées ci-dessus, il est judicieux de choisir un outil qui fonctionne avec plusieurs compilateurs ou, mieux, avec tous les compilateurs.
L'analyseur de couverture doit pouvoir fonctionner avec vos cibles embarqués. Pour cette raison, il est important que l'outil ait un faible coût d'instrumentation. L'instrumentation consiste à ajouter des compteurs à votre code (ou à une copie de votre code) afin de mesurer la couverture.

Comme pour l'analyse statique du code, l'aspect des rapports est également important pour l'outil de couverture du code. Les rapports doivent être clairs et faciles à comprendre, et montrer à la fois une vue d'ensemble de la couverture de votre projet et le détail de la couverture de votre code source.

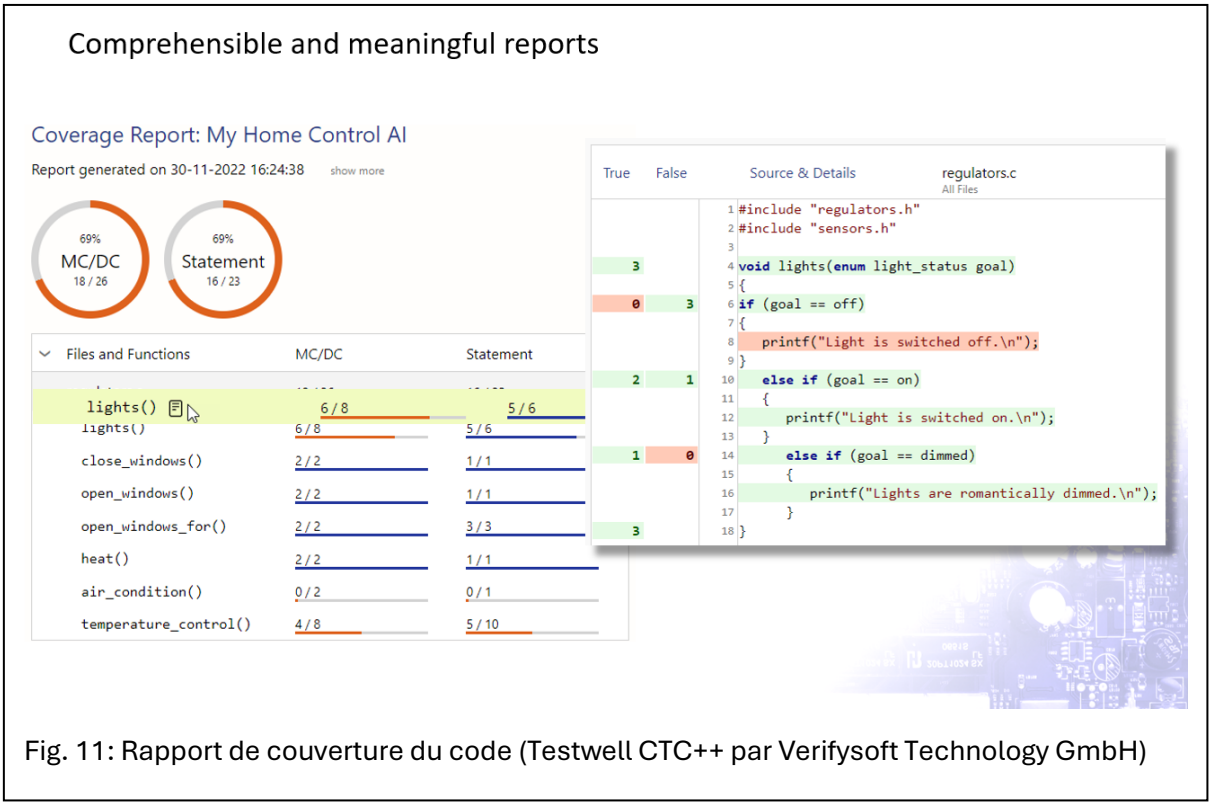

Fig. 11: Rapport de couverture du code (Testwell CTC++ par Verifysoft Technology GmbH)

Parfois, il n'est pas possible de tester toutes les parties du code, par exemple lorsque vous devez appliquer un style de programmation défensif. Dans ce cas, il est utile que l'outil offre la possibilité d'ajouter des "justifications" pour expliquer pourquoi certaines parties du code n'ont pas été couvertes par des tests.

Lorsque vous utilisez un outil de couverture de code pour des projets critiques en matière de sécurité, vous devez prouver que l'outil utilisé est qualifié. Lorsque l'outil dispose d'un certificat (par exemple un certificat TÜV), il peut être utilisé pour de nombreuses normes de sécurité sans autres mesures de qualification. Pour les projets DO178-C dans l'aéronautique, la situation est un peu différente. L'outil doit être qualifié, mais ici aussi le certificat de l'outil sera utile comme point de départ.

Cette qualification/certification est également nécessaire pour les outils d'analyse statique du code.

Les critères de sélection suivants concernent les outils d'analyse statique et dynamique :
Le coût et le modèle de licence d'un logiciel de test et d'analyse sont toujours importants. Lorsque l'on parle de coût, il ne s'agit pas seulement du prix de l'outil. Il s'agit également du coût de la mise en place de l'outil, de la formation des utilisateurs et du temps nécessaire pour comprendre les rapports. Ici, un outil "bon marché" peut devenir coûteux... Un outil "bon marché", qui cause beaucoup de problèmes, n'est finalement pas bon marché...

Il est également important de vérifier l'étendue de l'utilisation d'un outil : un seul utilisateur, l'ensemble du projet, un site, ou même une utilisation à l'échelle du pays ?

Lorsque vous choisissez un outil, vous devez également vous intéresser à la qualité du support technique. Lors de l'évaluation d'un outil, vous aurez déjà une idée du temps de réponse et de la coopération entre l'utilisateur et le fournisseur de l'outil.

En résumé, nous avons vu que l'analyse statique du code et les tests dynamiques associés à la couverture de code sont nécessaires et doivent être utilisés de manière complémentaire. Il existe sur le marché de très bons outils qui vous aident à appliquer ces méthodes. Avant d'acquérir des licences, les outils doivent être évalués afin de vérifier s'ils répondent aux exigences de votre projet actuel et des projets que vous pourriez avoir à l'avenir.

À propos de l'auteur

Klaus Lambertz est fondateur et PDG de Verifysoft Technology GmbH www.verifysoft.com , une société qui fournit des outils de test et d'analyse de logiciels. Verifysoft Technology a été fondée en 2003 à Offenburg (Allemagne) et compte actuellement plus de 750 clients satisfaits dans plus de 40 pays. Outre l'analyseur de couverture de code Testwell CTC++, la société fournit des outils complémentaires pour l'analyse statique du code. Verifysoft propose également des séminaires sur les tests de logiciels. Certaines parties de ce document sont basées sur des publications de Royd Lüdtke (Director Static Code Analysis, Verifysoft) et du Dr. Sabine Poehler (Product Manager Testwell CTC++, Verifysoft).