Example concret du test d'un système embarqué

Dans le test de système embarqués, il est fortement recommandé de combiner une analyse statique et des tests à l'exécution du logiciel, combiné avec une mesure de la couverture de test.
Le projet d'un fabricant d'appareils électroménagers peut servir d'exemple de pratique qui démontre de manière impressionnante cette affirmation.
Ce fabricant a été développé un logiciel pour l'unité de commande d'une gamme de lave-linge. Ceci a été implémenté dans le langage de programmation C dans le but de fonctionner sur un micro-contrôleur avec la technologie ARM.
Le logiciel pour l'unité de commande était identique pour toute la gamme des machines. Selon le modèle de lave-linge certaines fonctions devant être activées ou désactivées.
Les machines de la gamme de prix le plus élevée devraient par exemple être équipé d'un capteur pour mesurer le degré de saleté du linge afin de permettre au programme d'ajuster automatiquement le temps nécessaire du lavage selon les besoins. En revanche, les temps de lavage des machines plus simples sans capteur doivent être constants.

Pour garantir une qualité logicielle adéquate, des cas de test ont été créés pour atteindre 100% de couverture de condition/décision modifiée (MC/DC).
Avec ceci le fabricant pensait avoir suffisamment testé et avait initialement renoncé à une analyse statique.
Un regard sur le code source est nécessaire afin de décrire les problèmes qui se poseront plus tard.
Pour des raisons de confidentialité et pour une meilleur compréhension le code à été simplifié et réduit au nécessaire pour la description du problème.

Figure 1 *- Fonction de calcul du temps de lavage*

```
size_t durationMainWashCycle(size_t prog, size_t load, size_t staining) {
	if (((prog == 3) || (prog == 5) || (prog == 7)) && (load < 5)) {
		return staining * 5;
	}
	else if (((prog == 4) || (prog == 6)) && (load < 5)) {
		return staining * 8;
	}
	else if (((prog == 4) || (prog == 6)) && (load < 3)) {
		return staining * 7;
	}
	else {
		return staining * 9;
	}
}
```

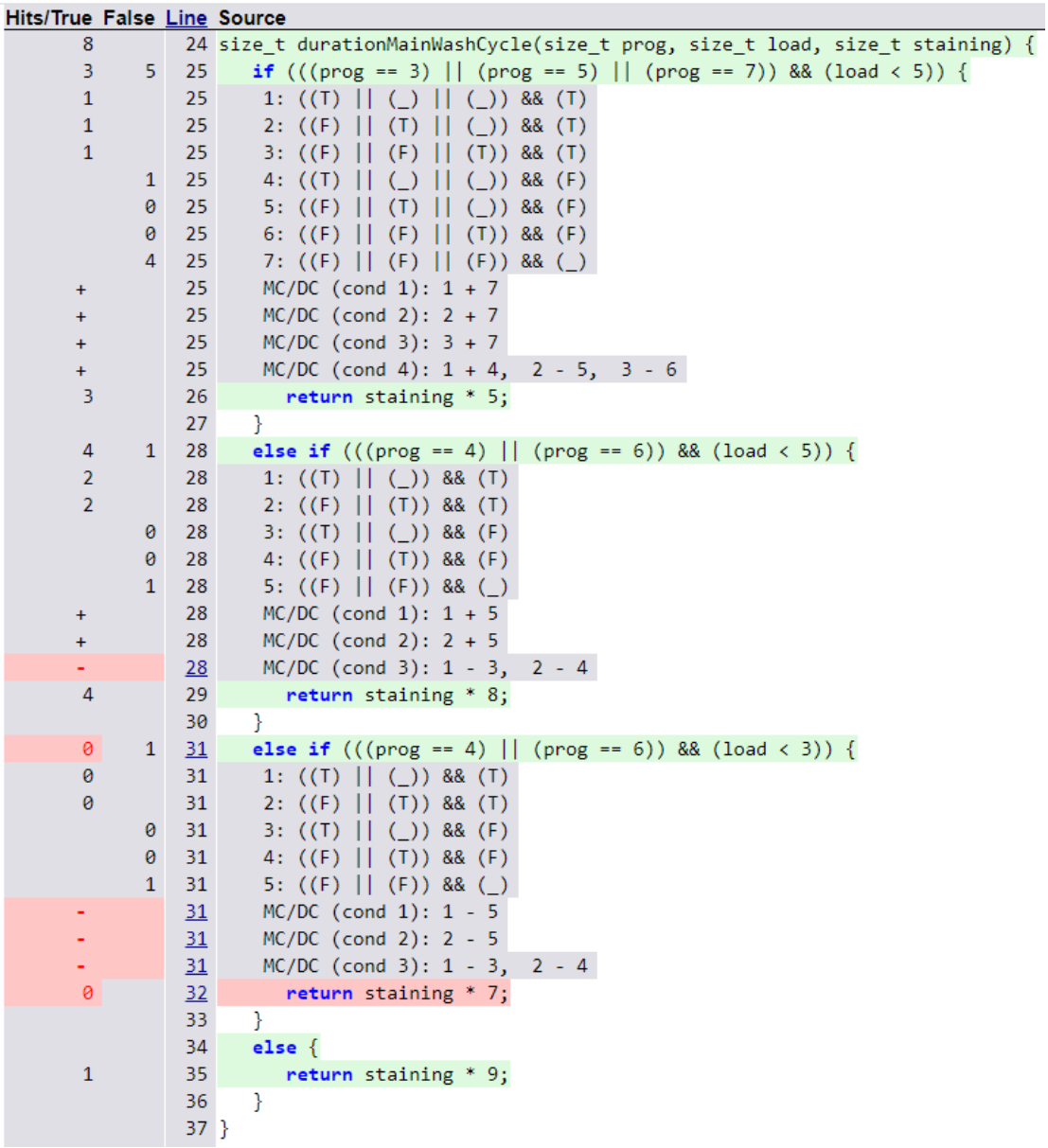
La fonction représentée calcule le temps de lavage en fonction du programme de lavage sélectionné, de la charge et du degré de saleté du linge.
Les cas de test énumérés dans le tableau ci-dessous (tableau 1) ont été exécutés pendant les tests du module, le degré de salissure "staining" (tachant) étant inclus comme facteur dans le résultat du test.
La mesure dans laquelle la version du produit «product_version» influence le résultat sera expliquée plus loin.

Tableau 1 *- Cas de test réalisés en lien avec la fonction „durationMainWashCycle()“*

No. de cas de test	product_version	prog	load	staining	résultat	Résultat attendu
1	11	3	4	3	15	15
2	11	5	4	3	15	15
3	11	7	4	3	15	15
4	11	3	6	3	27	27
5	12	4	2	1	**8**	**7**
6	12	6	2	1	**8**	**7**
7	13	4	4	1	8	8
8	13	6	4	1	8	8

Figure 2 montre la couverture de tests atteint avec ces cas de test.

Figure 2 *- Couverture de test (niveau MC/DC) analysé avec Testwell CTC++ Test Coverage Analyser*



Hits/True	False	Line	Source
8		24	size_t durationMainWashCycle(size_t prog, size_t load, size_t staining) {
3	5	25	if (((prog == 3) || (prog == 5) || (prog == 7)) && (load < 5)) {
1		25	1: ((T) || (_) || (_)) && (T)
1		25	2: ((F) || (T) || (_)) && (T)
1		25	3: ((F) || (F) || (T)) && (T)
	1	25	4: ((T) || (_) || (_)) && (F)
	0	25	5: ((F) || (T) || (_)) && (F)
	0	25	6: ((F) || (F) || (T)) && (F)
	4	25	7: ((F) || (F) || (F)) && (_)
+		25	MC/DC (cond 1): 1 + 7
+		25	MC/DC (cond 2): 2 + 7
+		25	MC/DC (cond 3): 3 + 7
+		25	MC/DC (cond 4): 1 + 4, 2 - 5, 3 - 6
3		26	return staining * 5;
		27	}
4	1	28	else if (((prog == 4) || (prog == 6)) && (load < 5)) {
2		28	1: ((T) || (_)) && (T)
2		28	2: ((F) || (T)) && (T)
	0	28	3: ((T) || (_)) && (F)
	0	28	4: ((F) || (T)) && (F)
	1	28	5: ((F) || (F)) && (_)
+		28	MC/DC (cond 1): 1 + 5
+		28	MC/DC (cond 2): 2 + 5
-		28	MC/DC (cond 3): 1 - 3, 2 - 4
4		29	return staining * 8;
		30	}
0	1	31	else if (((prog == 4) || (prog == 6)) && (load < 3)) {
0		31	1: ((T) || (_)) && (T)
0		31	2: ((F) || (T)) && (T)
	0	31	3: ((T) || (_)) && (F)
	0	31	4: ((F) || (T)) && (F)
	1	31	5: ((F) || (F)) && (_)
-		31	MC/DC (cond 1): 1 - 5
-		31	MC/DC (cond 2): 2 - 5
-		31	MC/DC (cond 3): 1 - 3, 2 - 4
0		32	return staining * 7;
		33	}
		34	else {
1		35	return staining * 9;
		36	}
		37	}

Cependant, la couverture de test de la fonction considérée n'était que de 71%.

À partir du rapport de couverture du test en relation avec les résultats du test un problème était immédiatement visible : le chemin avec la condition « if » (début en ligne 31) n'a pas été exécuté (donc pas testé), bien que les cas de test correspondants (no. 5 et 6) ont été effectués :

```
else if (((prog == 4) || (prog == 6)) && (load < 3)) {
                return staining * 7;
```

En plus le résultat de cas de test sont diffèrent du résultat attendu. Le rapport de couverture montré que la condition « if » a partir de la ligne 28 a été exécuté a la place.
L'échange des deux conditions if-else dans le code a corrigé cette erreur.

Un nouveau test a maintenant montré une couverture de test de 95% et les résultats des tests sont conforme aux attentes.

Le rapport de couverture de test a montré qu'un cas de test supplémentaire était nécessaire pour une couverture de test de 100% :

No. de cas de test	product_version	prog	load	staining	résultat	Résultat attendu
9	14	6	6	1	9	9

Un nouveau test, en tenant compte du cas de test manquant, a ensuite abouti à la couverture de test requise de 100%.
Après avoir fait les tests d'intégration et résolu quelques autres problèmes mineurs, les machines à laver sont finalement entrées en production et ont été expédiées.
Quelque temps après des clients insatisfaits se sont plaints. Avec certaines machines, le résultat du lavage n'était pas satisfaisant car le cycle de lavage principal était terminé trop tôt. Le fabricant de machines ne trouvait pas la source du problème. Dans un premier temps, un défaut du capteur de pollution a été suspecté et un échange a été effectué dans le cadre de la garantie. Cependant, il s'est avéré assez rapidement que cela ne résolvait pas le problème. Après avoir examiné de plus près les plaintes, il est devenu évident qu'elles étaient limitées à un certain type de machine de la gamme de produits.
La possibilité d'une erreur logicielle a alors été prise en compte et un prestataire de services a été chargé d'une analyse statique du code source.
Par l'analyse statique une erreur grave a été trouvé très rapidement.
La description de l'erreur par rapport au code illustré sur la figure 3 est illustrée sur la figure 4.

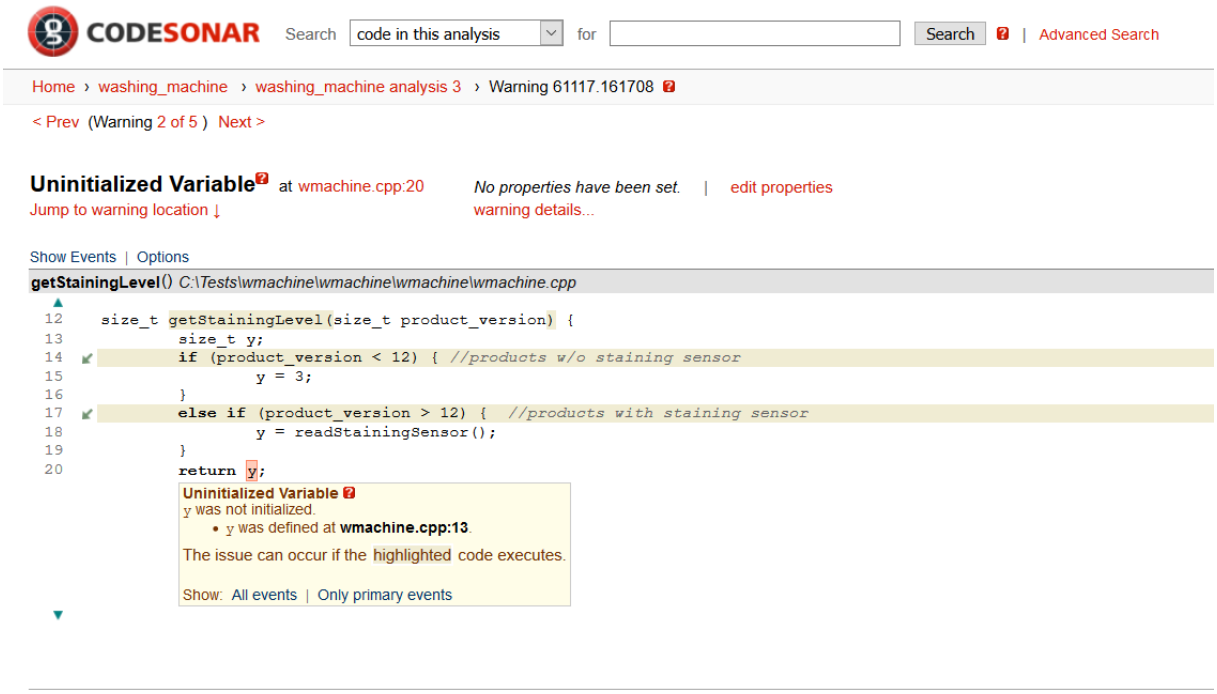
Figure 3 *- Détermination du degré de contamination en fonction de la version du produit*

```
size_t getStainingLevel(size_t product_version) {
	size_t y;
	if (product_version < 12) { //products w/o staining sensor
		y = 3;
	}
	else if (product_version > 12) {  //products with staining sensor
		y = readStainingSensor();
	}
	return y;
}
```

L'exigence selon laquelle tous les modèles à partir du modèle no. 12 interrogent le capteur de contamination et tous les modèles ci-dessous qui fonctionnent avec une valeur de contamination constante n'a pas été correctement implémenté.

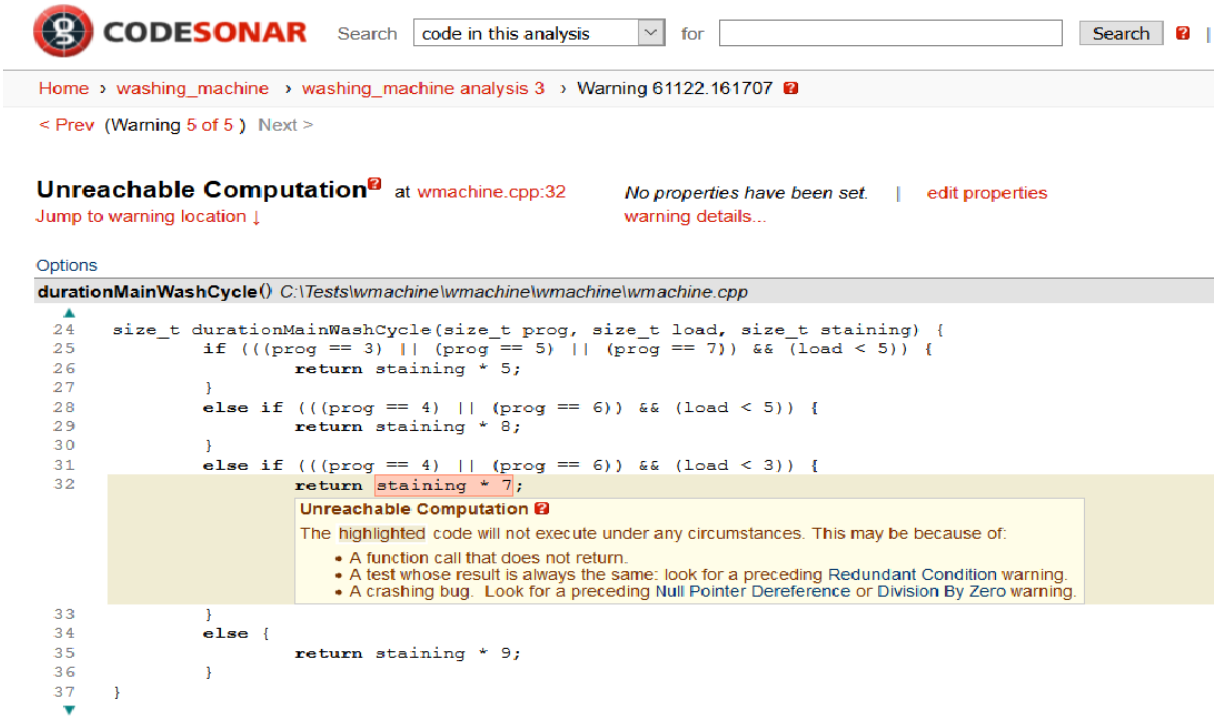
Étant donné que la valeur n'était pas perceptible lors des tests unitaires, cette erreur grave n'a pas été détectée.

Figure 4 *– Une variable non initialisée provoque un comportement indéfini (rapport de l'outil d'analyse statique GrammaTech CodeSonar)*



Le résultat de l'analyse statique (simplifié ici à des fins d'illustration) montre également que l'erreur aurait pu être trouvé déjà plus tôt pendant la phase de codage (cf. figure 5).

Figure 5 *– Code inaccessible (rapport de l'outil d'analyse statique GrammaTech CodeSonar)*



Après une correction d'erreur coûteuse après la livraison et les dommages associés à son image, le fabricant d'appareils électriques utilise désormais à la fois l'analyse statique et les tests dynamiques pour l'ensemble de ses logiciels. Depuis, il n'a plus rencontré de sérieux problèmes logiciels.

Cet exemple montre que l'analyse statique et les tests pendant l'exécution du logiciel (comme les test unitaires) sont complémentaires et indispensables en particulier dans le test des systèmes embarqués. Les tests doivent être associés à une analyse de couverture de tests afin de faire tous les tests nécessaires. Ceci est obligatoire pour les logiciels critique, mais également très utiles pour tous les autres secteurs.

Source :

Jean-François Pradat-Peyre et Jacques Printz :
PRATIQUE DES TESTS LOGICIELS, 4ème édition
Dunod, www.dunod.com, février 2021, ISBN 978-2-10-081995-9

https://www.dunod.com/sciences-techniques/pratique-tests-logiciels-ameliorer-qualite-par-tests-gerer-une-campagne-tests

Le livre « **Pratique des Tests Logiciels** » s'adresse aux développeurs, concepteurs et intégrateurs de logiciels ainsi qu'aux chefs de projets et aux architectes.

Avec la montée en charge du *big data*, et du *cloud computing*, la fiabilité des logiciels est plus importante que jamais. Concevoir du premier coup et sans aucune erreur un logiciel qui comporte plusieurs millions de lignes de code et plusieurs centaines de composants est évidemment impossible.

La **nécessité de faire des tests** au cours des différentes phases de conception paraît évidente et pourtant, dans la pratique, les tests sont souvent négligés et relégués au second plan. L'objectif de cet ouvrage est triple :

— donner les bases et les **bonnes pratiques** pour concevoir et mener à bien des tests ;

— fournir un **référentiel** en termes de méthodes et de vocabulaire ;

— préparer la **certification ISTQB** du métier de testeur.

Cette quatrième édition rend compte des évolutions dans la pratique des tests logiciels au cours des trois dernières années.