

Anti-patterns de tests unitaires

par Bruno Orsier ([Site Web de Bruno Orsier](#))

Date de publication :

Dernière mise à jour : 05/10/2007

En complément de mon **tutoriel** sur le développement dirigé par les tests, cette page propose une traduction des anti-patterns de tests unitaires recensés par **James Carr** sur son **blog**. Traduction faite avec la permission de James Carr.

I - Introduction**II - Liste de James Carr**


- Le menteur
- L'overdose d'initialisations
- Le géant
- La mascarade
- L'inspecteur
- Les restes copieux
- Le héros local
- Le pinailleur
- L'attrapeur caché
- L'escroc
- La grande gueule
- Le glouton
- Le séquenceur
- La dépendance cachée
- L'énumérateur
- L'étranger
- L'évangéliste du système d'exploitation
- Le succès envers et contre tout
- La course gratuite
- L'unique
- Le voyeur
- L'escargot


III - Commentaires sur le blog de James Carr


- Le coucou
- La mère poule
- L'oie sauvage
- Le dodo
- Le prévisible
- Le chasseur de trésors
- Le dégueulasse

IV - Remerciements

I - Introduction

Nous avons vu dans le tutoriel l'intérêt de travailler avec des tests unitaires. Cette manière de travailler comporte toutefois certains risques de dérapages. Heureusement les professionnels pratiquant ces tests ont identifié un certain nombre de choses à éviter et les ont cataloguées sous forme de liste d' **anti-patrons** (anti-pattern en anglais).

Tout comme les "bons" patrons de conception, un anti-patron doit avoir été identifié de manière indépendante par plusieurs personnes et doit être décrit de manière assez formalisée, avec ses avantages, ses inconvénients etc. Ici nous nous contentons toutefois d'une liste assez informelle, qui a tout le même le mérite d'avoir été discutée en détail sur le forum  **testDrivenDevelopment**. James Carr a ensuite recensé ces anti-patrons dans son blog (section II ci-dessous), puis divers commentateurs en ont ajouté de nouveaux (section III).

Les lecteurs recherchant plutôt des anti-patrons bien formalisés pourront consulter le site  <http://xunitpatterns.com/> ainsi que le livre correspondant.

II - Liste de James Carr

Le menteur

Une classe de test unitaire dont toutes les méthodes passent et qui donc apparaît valide, mais après une inspection détaillée on découvre qu'elle ne teste pas du tout ce que l'on pensait.


L'overdose d'initialisations

Un test qui demande beaucoup d'initialisations avant de commencer à tester. Quelquefois plusieurs centaines de lignes de code sont nécessaires pour initialiser l'environnement pour un seul test, avec plusieurs objets impliqués, ce qui rend difficile de savoir ce qui est réellement testé.

Le géant

Une classe de test unitaire qui comporte des tests valides, mais comprend des milliers de lignes de code et contient beaucoup de cas de tests. Ceci peut être une indication que le système testé est un "objet dieu", à savoir un objet qui fait tout, qui voit tout, donc dont les responsabilités sont trop importantes.

La mascarade

Doubler (mock en anglais) des objets est parfois très utile. Mais il arrive que des développeurs s'égarent dans leur effort de doubler ce qui n'est pas à tester. Dans ce cas un test unitaire contient tellement de doublures de tests (mocks, stubs, fakes :  voir définitions) que le système testé n'est en fait pas testé du tout : ce sont les données fournies par les doublures qui sont testées.

L'inspecteur

Un test unitaire qui viole l'encapsulation afin d'atteindre 100% de couverture de code, et qui connaît tellement bien l'objet testé que toute tentative de remaniement cassera le test existant, si bien que toute modification de code entraînera une modification de test.

Les restes copieux

Un test unitaire crée des données qui persistent quelque part (sur le disque dur par exemple). Ces données sont ensuite utilisées à tort par un autre test, qui échouera ou réussira à cause d'un effet de bord.

Le héros local

Un cas de test qui dépend de l'environnement de développement. Par conséquent il passe sur les machines de développement, mais échoue ailleurs.

Le pinailleur

Un test qui examine une sortie complète alors qu'en réalité il est intéressé par seulement de petites parties. Par conséquent ce test doit continuellement être mis à jour sur des points de détails pas réellement importants. Endémique dans le test d'applications Web.

L'attrapeur caché

Un test qui à première vue semble ne faire aucune vérification car les assertions sont absentes. En fait le test est basé sur des exceptions lancées par le code testé et il considère que le framework de test va capturer l'exception et la reporter à l'utilisateur comme un échec du test.

L'escroc

Une classe de test unitaire qui comporte beaucoup de tests sur des effets de bord mineurs (et faciles à tester) mais qui ne teste jamais le comportement central désiré. Cela peut arriver dans les tests d'accès à une base de données, où une méthode est appelée, puis le test sélectionne des informations de la base et exécute des assertions sur le résultat.

La grande gueule

Une classe de test unitaire qui remplit la console avec des messages de diagnostic, d'informations, etc. même quand les tests passent. Ces messages sont quelquefois nécessaires durant la création des tests, mais ils risquent d'être laissés là, bien qu'inutiles.

Le glouton

Un test qui attrape les exceptions et masque la pile d'appel ; il peut même les remplacer par des messages d'échecs contenant moins d'informations, voire les écrire dans un journal et laisser le test passer.

Le séquenceur

Un test unitaire qui dépend du fait que des éléments d'une liste non ordonnée vont toujours apparaître dans le même ordre durant les assertions.

La dépendance cachée

Un proche cousin du héros local, un test unitaire qui exige que des données aient été correctement remplies avant que le test ne s'exécute. Si ces données ne sont pas fournies, le test échoue mais donne très peu d'indications sur ce qui manque, ce qui force les développeurs à explorer des quantités de code pour trouver ce qui ne va pas.

L'énumérateur

Une classe de test unitaire dans laquelle le nom de chaque méthode de test est simplement une énumération, i.e. test1, test2, test3. Par conséquent l'intention de chaque cas de test est obscure et le seul moyen de l'éclaircir est de lire le code du test.

L'étranger

Une méthode de test qui n'appartient pas réellement à la classe de test unitaire dont elle fait partie. Elle teste un objet B séparé, probablement un objet utilisé par l'objet A actuellement testé. Mais le cas de test s'est mis à tester cet objet B directement, au lieu de s'appuyer sur les sorties de l'objet A (qui utilise B pour son propre comportement). Également connu sous le nom du cousin éloigné.

L'évangéliste du système d'exploitation

Un test unitaire qui dépend d'une fonction ou caractéristique particulière du système d'exploitation. Par exemple un cas de test qui dépendrait du caractère de fin de ligne de Windows et qui échouerait sous Linux.

Le succès envers et contre tout

Un test écrit pour réussir d'abord au lieu d'échouer d'abord. Avec pour effet de bord malheureux que le test passe tout le temps, même quand il devrait échouer.

La course gratuite

Au lieu d'ajouter une nouvelle méthode pour un nouveau cas de test, une assertion est ajoutée dans une méthode existante.

L'unique

Une combinaison de plusieurs anti-patterns, en particulier la course gratuite et le géant : une classe de test unitaire qui contient une seule méthode de test qui teste toutes les fonctionnalités offertes par un objet. Un indicateur courant est que le nom de la méthode de test est souvent le même que celui du test unitaire, et qu'elle contient de nombreuses lignes d'initialisation et d'assertions.

Le voyeur

Un test qui, à cause de ressources partagées, peut voir les données résultant d'un autre test, ce qui peut faire échouer le test même si le système testé est parfaitement valide. Également connu sous le nom des invités indésirables.

L'escargot

Un test incroyablement lent. Les développeurs ont le temps d'aller aux toilettes, fumer une cigarette, ou pire, sont obligés de démarrer le test en partant à la fin de la journée.

III - Commentaires sur le blog de James Carr

Cette section propose une traduction d'un sous-ensemble des anti-patterns suggérés par des commentateurs sur le blog de James Carr.

Le coucou

Dans une classe de tests unitaires, une des méthodes de tests utilise les mêmes (potentiellement longues) initialisations que les autres méthodes, puis elle élimine tout ou partie de ces initialisations pour faire les siennes.

La mère poule

Une initialisation commune qui fait beaucoup plus que ce dont les méthodes de test ont réellement besoin. Par exemple elle crée des quantités de structures de données complexes qu'elle remplit avec des valeurs uniques apparemment importantes, alors que les tests vérifient uniquement la présence ou l'absence de quelque chose. Cela peut indiquer que l'initialisation a été écrite avant les tests eux-mêmes, ce qui est un cas subtil de violation des principes du développement dirigé par les tests.

L'oie sauvage

Un test unitaire qui, bien qu'initialement simple, demande la création et l'initialisation d'une partie de l'application toujours plus importante au fil du temps. Il peut alors consommer un temps de développement disproportionné et fait perdre le bénéfice de la rapidité du cycle du TDD. Dans le pire des cas, le TDD est complètement abandonné. Ce cas est courant chez les novices en TDD qui ne seraient pas encore à l'aise avec le principe de simuler des réponses.

Le dodo

Un test unitaire qui teste un comportement qui n'est plus nécessaire dans l'application. A la fois le test et le comportement en question devraient probablement être supprimés.

Le prévisible

Un test unitaire qui teste des conditions multi-threadées avec le même ordonnancement des threads à chaque fois, ou des données aléatoires qui sont en fait toujours exactement les mêmes (ce qui arrive si on utilise toujours la même initialisation du générateur de nombres aléatoires).

Le chasseur de trésors

Des tests qui assurent l'obtention du pourcentage désiré de couverture de code, en exerçant le maximum de chemins dans le code, mais qui ne testent rien de réellement utile.

Le dégueulasse

Un test qui crée des ressources persistantes, mais ne les nettoie pas après lui.

IV - Remerciements

Merci à **Dia** pour sa relecture.

