Antipattern

En génie logiciel, les **anti-patrons** ou **antipattern** sont des erreurs courantes de conception des logiciels. Leur nom vient du fait que ces erreurs sont apparues dès les phases de conception du logiciel, notamment par l'absence ou la mauvaise utilisation de patrons de conception, appelés *design pattern* en anglais.

Les anti-patrons se caractérisent souvent par une lenteur excessive du logiciel, des coûts de réalisation ou de maintenance élevés, des comportements anormaux et la présence de bugs.

Il existe également les GreyPatterns (dont le bénéfice ou les inconvénients ne sont pas clairement établis).

Sommaire

Anti-patrons de développement

Abstraction inverse

L'abstraction inverse se produit lorsque l'on construit un objet logiciel avec une interface qui n'offre pas des fonctions nécessitées par les développeurs qui l'utilisent, alors qu'il pourrait les offrir. L'interface n'offre que des fonctions plus complexes. Le résultat est que l'utilisateur de l'objet doit se servir des fonctions complexes fournies par l'objet pour programmer un comportement simple.

Exemple : avoir un objet qui ne fait que des calculs en virgule flottante, et être obligé d'utiliser cet objet pour faire du calcul avec des entiers.

Action à distance

L'action à distance se caractérise par l'emploi immodéré de variables globales ou des interdépendances accrues entre objets.

Ancre de bateau

L'ancre de bateau est un composant inutilisé mais qui est gardé dans le logiciel pour des raisons politiques, en pensant que ce code servira plus tard.

Attente active

L'attente active désigne une boucle qui ne contient qu'une instruction : tester une condition, jusqu'à ce qu'elle soit enfin vérifiée, et que le morceau de code puisse poursuivre son déroulement. Cet anti-pattern est courant en programmation concurrente, car c'est un autre processus qui doit modifier des variables pour pouvoir « libérer » la boucle d'attente active. L'attente est *active* puisque le processus qui attend consomme du temps machine, ce qui constitue un gaspillage. On peut s'affranchir de cette mauvaise technique grâce à la programmation événementielle, ou bien par l'utilisation de signaux. Dans certains cas (Spinlock), cette technique est utilisée délibérément pour éviter un de-scheduling du thread qui vérifie la condition, car le programmeur attend que cette condition soit vérifiée d'ici peu.

Interblocages et famine

Ce sont des erreurs courantes dues à une mauvaise conception des parties concurrentes du logiciel (par exemple lors de l'utilisation de threads). Elles se manifestent lorsque plusieurs morceaux de code veulent utiliser une ou plusieurs ressources en même temps, et que la stratégie d'allocation des ressources est viciée ou inexistante. Cela se traduit par des performances altérées, voire des « plantages ».

> Article connexe : Dîner des philosophes.

Erreur de copier/coller

> Article détaillé : Duplication de code.

La duplication de code sans vérification entraîne des incohérences. La meilleure solution étant encore de factoriser les parties communes au lieu de les dupliquer.

Programmation spaghetti

> Article détaillé : Programmation spaghetti.

Ceci fait référence à l'image d'un plat de spaghetti, dans lequel il serait impossible de modifier une petite partie du logiciel sans altérer le fonctionnement de tous les autres composants.

Réinventer la roue (carrée)

> Articles détaillés : Réinventer la roue et Réinventer la roue carrée.

Par analogie à l'inutile réinvention de la roue, la roue carrée fait référence au fait de mal réinventer une solution existante ou bien de réinventer une mauvaise solution, non existante de ce fait.

Surcharge des interfaces

La surcharge des interfaces fait référence à des pratiques courantes en conception d'interfaces utilisateurs Web, où plusieurs boutons ont le même effet.

L'objet divin

> Article détaillé : God object.

L'objet divin est un composant du logiciel assurant trop de fonctions essentielles. C'est le contraire de la méthode diviser pour régner.

Vous n'en aurez pas besoin (YAGNI) (http://c2.com/cgi/wiki?YouArentGonnaNeedIt)

Ne pas respecter le concept YAGNI est un anti-pattern. le concept YAGNI dit qu'il ne faut pas implémenter maintenant quelque chose dont on pense qu'il sera utile plus tard.

Anti-patrons architecturaux

ArchitectureAsRequirements

consistant à spécifier une architecture par simple préférence ou parce qu'elle est nouvelle, alors qu'il n'y en a pas besoin et que le client n'en a pas exprimé le désir.

ArchitectureByImplication

consistant à ne pas documenter l'architecture utilisée par un projet et à ne pas la spécifier.

Coulée de lave

La coulée de lave se produit lorsqu'une partie de code encore immature est mise en production, forçant la lave à se solidifier en empêchant sa modification.

Deuxième Système

Problème de refactoring, lors de la réécriture d'un système avec trop de confiance il est possible d'aboutir a une architecture en sur-design qui se retrouve être lourde et inadaptée (c2.com (http://c2.com/cgi/wiki?SecondSystemEffect)).

Marteau doré

Avec un bon marteau, tous les problèmes ressemblent à des clous. Cette conception consiste à réutiliser une technologie familière de manière obsessionnelle appliquée à beaucoup de problèmes.