LA CONCEPTION DE LOGICIELS À L'AIDE DE DESIGN PATTERNS

par Dr Kiss Cool
[Attention au deuxième effet...]

Une bonne pratique de développement logiciel est d'utiliser des design patterns ou patrons de conception. Beaucoup de gens se plaignent que cette pratique ne soit pas plus suivie... mais tout le monde utilise les design patterns et je vais le montrer dans cet article !

Un design pattern, s'il est besoin de le rappeler, est une recette permettant de résoudre un problème connu de programmation. Nous faisons tous cela tous les jours, inutile de lire de fastidieux ouvrages comme « Elements of Reusable Object-Oriented Software » publié en 1995 par ceux que l'on nomme le « Gang of Four » : Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides. Il faut pratiquement 400 pages à ces individus pour expliquer aux plus crédules d'entre nous comment écrire du code ! On croit rêver !

Je vais décrire dans cet article ces problèmes que nous résolvons tous les jours et surtout donner un nom à ces design patterns que nous employons sans pour autant avoir perdu des heures à comprendre leur formalisme abscons.

1 Golden Hammer

Vous maîtrisez parfaitement une technologie ou une technique et vous la réemployez pour résoudre n'importe quel problème. Cette bonne pratique permet de gagner du temps en évitant des recherches sur la façon de traiter tel ou tel problème : on utilise toujours le même outil. Par analogie, si vous avez un marteau à la main pourquoi perdre du temps à aller chercher un tournevis pour visser une vis ? Tapez dessus, elle tiendra également !

Ne cherchez donc pas si un langage est plus adapté qu'un autre à une problématique donnée, si certaines méthodes sont plus apropriées... faites simplement ce que vous savez faire.

2 Not Invented Here

Ce design pattern est très proche du précédent : l'objectif est de ne pas perdre de temps. Des éléments dont vous avez besoin existent déjà mais plutôt que de rechercher dans la documentation du langage pour éventuellement, en fin de compte, vous apercevoir qu'ils n'existent pas, vous préférez vous lancer bille en tête pour les réinventer. Vous avez besoin d'une liste ? Créez donc un nouvel objet `List` auquel vous ajouterez des fonctionnalités au fur et à mesure que vous découvrirez que vous en avez besoin ! Et si vous changez de projet et que vous avez à nouveau besoin d'une liste, vous aurez acquis une certaine expertise qui vous permettra de réécrire le code encore plus rapidement.

Les mauvaises langues diront que vous réinventez la roue carrée, que les éléments que vous avez créé sont forcément moins bons que ceux qui ont été testés et éprouvés par des milliers de développeurs. Vous pourrez leurs répondre que vous au moins vous êtes capable de les réécrire et qu'en plus vous avez économisé un temps considérable en ne lisant pas la documentation : ce sont vos objets et vous savez comment ils fonctionnent !

3 Copy and Paste Programming

Certaines structures mettent en place des outils permettant de détecter la duplication de code pour inciter les développeurs à factoriser leur code, trouver une solution générique... Mais résoudre un problème par copier/coller avec une petite modification de code ne prend que quelques secondes ! Pourquoi aller s'embêter à réfléchir à une implémentation plus générale ? Il y a de la place sur les disques dur, ce n'est quand même pas pour gagner quelques lignes de code en moins qu'il va falloir se creuser la cervelle pendant des heures. Un peu de bon sens, que diable !

4 Lava Flow

Tester de nombreuses fois le code avant de le mettre en production pour limiter les risques d'erreur est vraiment une tâche fastidieuse ! Alors pourquoi le faire ? Dès que le code est prêt, passez le en production et vous obtiendrez au moins deux avantages :

- les différences entre la branche de production et la branche de développement seront minimes,
- le code injecté se comportera comme une coulée de lave : il se solidifiera, empêchant des modifications et permettra de conserver ainsi une sorte d'historique du code.

Quel ne sera pas votre bonheur lorsqu'après des années de développement vous pourrez jouer à Indiana Jones et gratter les différentes couches de lave pour parvenir à l'Arche Perdue...

5 Spaghetti Code

Tout le monde connaît la programmation spaghetti qui est souvent assimilée, à tort, à une mauvaise pratique. Dans un plat de spaghetti de nombreux composants, les spaghetti, sont liés et constituent un tout. Il s'agit donc d'une très bonne pratique permettant d'obtenir un ensemble cohérent : tirez un spaghetti et vous modifiez toute la structure.

L'instruction phare permettant de mettre en place ce design pattern est le `goto`. Malheureusement certains langages bannissent cette instruction, ce qui est fort regrettable. Notez qu'elle n'est pas essentielle mais elle apporte une certaine aisance dans le développement. On peut citer deux exemples de langages avec d'un côté PHP qui permettra d'appliquer cette technique pratiquement sans s'en apercevoir et d'un autre côté Python qui au contraire va disposer de nombreux pièges pour empêcher le développeur consciencieux de faire son travail.

6 Action at a distance

Ce design pattern est à associer avec le précédent pour une meilleure efficacité. Il consiste à utiliser au maximum les variables globales, ce qui permet de s'affranchir du passage de bon nombre de paramètres et rend ainsi le code beaucoup plus concis.

Pour une plus grande efficacité cette bonne pratique peut être associé au design pattern « Hard Code ».

7 Boat Anchor

Lorsque l'on développe du code, on aime qu'il serve. Si l'on s'aperçoit au cours du développement qu'il est devenu inutile, quelle tristesse que de le voir détruit, de voir des heures de travail anéanties et reniées... Alors conservez ce code dans le programme ! C'est vous qui l'avez écrit, il est forcément excellent et si les gens ne se rendent pas compte maintenant à quel point il peut être utile, ils le découvriront plus tard ! C'est simplement qu'ils ne sont pas encore prêts, vous êtes sans doute en avance sur votre temps. Dans l'histoire les grands précurseurs ont toujours été des incompris.

Si ce design pattern porte le nom d'ancre de bateau ce n'est pas pour rien : il vous permet de stabiliser le code en limitant les modifications. Ce qui ne sert plus est conservé pour une hypothétique utilisation future.

8 Poltergeist

Pour que deux objets communiquent entre eux, il est toujours intéressant d'en créer un troisième qui aura pour seule fonction de transmettre des informations entre les deux objets. C'est un objet qui sera créé pour cette tâche puis disparaîtra... pour ré-apparaître lors d'une nouvelle communication, d'où l'analogie avec un fantôme.

L'intérêt de ce design pattern réside dans le fait que l'on pourra obtenir de très nombreuses classes et que les diagrammes illustrant ces objets seront plus fournis et du plus bel effet lors d'une présentation.

9 God Object

Plutôt que d'avoir à rechercher les fonctionnalités essentielles du logiciel dans plusieurs objets, concentrez-les dans un seul objet, un objet divin qui aura le contrôle de tout. « Diviser pour régner » ? Foutaises !

10 Hard Code

Combien de développeurs s'encombrent avec des fichiers de configuration lourds et inutiles ? La solution est pourtant toute simple : coder tous les paramètres en dur !

Si vous utilisez un langage compilé ce design pattern sera encore plus intéressant car il nécessitera une recompilation et du coup peu de béotiens oseront poser leurs sales pattes sur votre joli code.

11 Error hiding

Il est toujours désagréable d'être confronté à un message d'erreur quel qu'il soit. Pourtant, il arrive que des bogues provoquent des affichages non désirés. La solution consiste alors à intercepter l'exception et à la traiter... par le mépris ! L'utilisateur n'a pas besoin de savoir qu'il y a un problème minime, il n'a pas à être dérangé par ce genre de futilités qui risqueraient de l'émouvoir et éventuellement feraient qu'il aille jusqu'à demander un correctif. Couvrons donc pudiquement ces erreurs d'une absence de traitement, tout le monde sera satisfait !

Conclusion

Vous avez pu le voir, il ne faut pas 400 pages pour énoncer les principaux design patterns. Un peu de bon sens et une pratique régulière du développement suffisent largement ! Vous pourrez désormais mettre un nom sur les bonnes pratiques que vous employez et qui font peut-être rire autour de vous... mais sachez que vous suivez la bonne voie et que tôt ou tard tout ces développeurs se rendront compte de leurs erreurs. À ce moment là, vous pourrez les regarder de haut, leurs renvoyer leur remarques acerbes et rire à gorge déployée. Et je rirai avec vous... mouahahaha !

Toutefois n'oubliez pas :

« il n'y a pas de mauvais langage... il n'y a que de mauvais développeurs ». ■