La quête du logiciel de qualité – suite et fin (Libres conseils 22c/42)

Chaque jeudi à 21h, rendez-vous sur le framapad de traduction, le travail collaboratif sera ensuite publié ici même.

Traduction Framalang : peupleLà, lerouge, goofy, alpha, Sky, Julius22, vvision, okram, lamessen

Les transformations qui préservent la structure

Le deuxième tome de *The Nature of Order* décrit comment chacune de ces propriétés définit également une transformation. En voici des exemples.

- **Des frontières solides**. Vous pouvez parfois transformer quelque chose positivement en lui adjoignant une frontière. Vous installez une palissade autour d'un jardin qui sert alors d'ornement, de coupe-vent afin que des vents forts ne viennent pas endommager le jardin, mais elle existe aussi comme structure en soi. Dans une interface graphique utilisateur, des boîtes à ascenseurs sans cadre sont difficiles à distinguer de l'arrière-plan de la fenêtre (pensez à toutes ces pages web blanches dont les formulaires d'entrée de texte n'ont pas de cadre). Vous placez une corniche sur le toit d'un immeuble afin que la transition entre l'immeuble et le ciel ne soit pas abrupte.
- **Des symétries locales**. Il est plus facile de construire de petites parties de manière symétrique : parce qu'elles sont fabriquées sur un tour, parce qu'on doit y accéder des deux côtés, parce qu'elles se plient comme

un livre. Faire des choses asymétriques, seulement pour l'intérêt de la chose, demande un travail supplémentaire et il est plus difficile d'obtenir quelque chose qui fonctionne bien.

- **Un espace positif**. Vous vous sentez trop exposé quand vous êtes au bureau ? Ajoutez une étagère à mi-hauteur à côté de vous pour délimiter votre espace mais ne vous enfermez pas complètement. Est-ce que votre interface utilisateur donne l'impression qu'il y a beaucoup d'espace restant une fois que vous avez mis les contrôles en place ? Faites plutôt en sorte que les contrôles entourent l'espace utilisable.

Chacun des points qui précèdent est une transformation qui préserve la structure. Vous effectuez un changement dans la structure existante non pas en la démolissant et en la refaisant, mais en ajustant une chose après l'autre selon ces propriétés comme transformations.

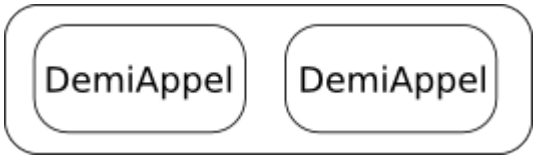
En termes de logiciel, il s'avère que c'est ce en quoi le « *Refactoring* » consiste surtout, quand vous traduisez les concepts en code. Réorganiser, c'est seulement appliquer des transformations qui préservent la structure ou, comme Martin Fowler – l'auteur de *Refactoring* – l'aurait présenté, des transformations qui préservent le comportement. Vous ne changez pas ce que fait le programme ; vous changez seulement la manière dont il est construit à l'intérieur, morceau par morceau.

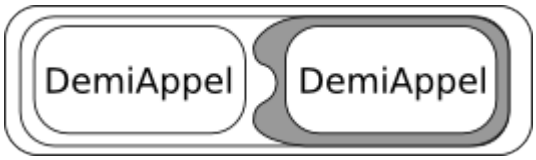
En extrayant un morceau de code et en l'insérant dans une fonction nommée, vous ajoutez essentiellement une frontière solide autour du code et créez un noyau robuste. En enlevant une variable globale et en ajoutant des variables de classe, vous permettez la robustesse, car chaque instance peut maintenant avoir une valeur différente dans cette variable, comme nécessaire. En ayant un producteur/consommateur, ou un émetteur/récepteur, vous avez des symétries locales, des imbrications fortes et ambiguës, et une bonne forme.

Richard Gabriel, l'une des principales personnalités du *Common Lisp* , a étudié comment appliquer les théories d'Alexander au logiciel (et aussi à la poésie ; le code n'est-il pas similaire à la poésie après tout ?). Il donne l'exemple suivant :

1. Imaginez que vous créez la classe AppelTelephonique. C'est un objet central implicite, qui pourrait être beaucoup plus puissant.

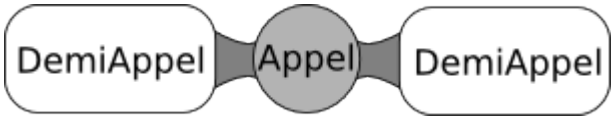
2. Gerard Meszaros dans le modèle *DemiObjet + Protocole* suggérait de séparer l'objet en deux demi-appels, liés par un protocole. On obtient ainsi une symétrie locale, un centre fort et un effet d'échelle.
3. Maintenant, dessinons cela sous forme de diagramme. On obtient alors de la symétrie locale, de l'effet



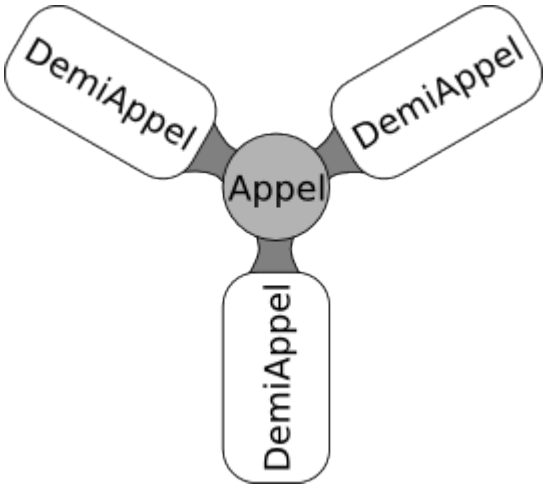


   d'échelle, des frontières, une imbrication forte et de l'ambiguïté. C'est ici que Meszaros a arrêté sa démarche.

4. Richard Gabriel suggère alors de renforcer les centres existants en appliquant d'autres transformations qui préservent la structure. Que faire de l'objet central implicite au milieu ? Vous lui ajoutez une frontière explicite (Appel) qui lie les demiAppels entre eux. Cela améliore les symétries locales, maintient l'imbrication forte et l'ambiguïté, et c'est composable.



5. Oui, c’est composable. Des appels multidirectionnels, des conférences téléphoniques, tout cela s’effectue



grâce à la mise en œuvre de transformations qui préservent la structure.

Chaque développeur garde probablement une image mentale du programme qu’il est en train de créer ou de modifier. La partie la plus difficile dans la modification d’un code que vous n’avez pas écrit est de commencer par visualiser cette image mentale. Quand vous travaillez pour que le code affiche une image plus jolie, il s’améliore – et Alexander nous propose une bonne façon de le faire.

Le processus fondamental

Alexander argumente longuement pour expliquer l’intérêt de suivre ce processus : appliquer des transformations qui préservent la structure est la seule manière de réussir une conception de qualité et fonctionnelle. Cela ne vaut pas seulement pour les immeubles, mais pour tout ce que nous construisons. Peu importe que vous partiez de l’existant – un programme, un bâtiment ou une ville – ou que vous partiez de zéro. Nous imitons la nature dans ses processus d’évolution et de régénération, mais nous allons plus vite.

1. Commencez avec ce que vous avez : un espace vide, un immeuble déjà construit ou bien un programme qui ne ressemble à rien et difficile à utiliser.
2. Identifiez les centres existant dans cet espace. Trouvez le centre le plus faible ou le moins cohérent.
3. Voyez comment appliquer l'une au moins des quinze transformations qui préservent la structure afin de renforcer ce centre faible. A-t-il besoin d'être délimité ? A-t-il besoin de se confondre avec son entourage ? A-t-il besoin de plus de détails ? A-t-il besoin d'être dégagé ?
4. Trouvez les nouveaux centres qui sont apparus quand vous avez appliqué les transformations à l'ancien centre. Cette nouvelle combinaison rend-elle les choses plus fortes ? Les rend-elle plus jolies ? Les rend-elle plus fonctionnelles ?
5. Assurez-vous que vous avez fait la chose la plus simple possible.
6. Retournez au début pour l'étape suivante.

Un résumé extrêmement simple pourrait être : trouvez les mauvaises parties, améliorez-les de la façon la plus simple possible, testez les résultats, réitérez.

Alexander ne tient pas à détruire les choses juste pour les reconstruire de façon différente. Il ne s'agit pas de pas démolir des quartiers d'une ville pour les reconstruire mais de les améliorer progressivement. Pour les logiciels, il est bien connu que vous n'allez pas réécrire quelque chose juste parce que vous ne le comprenez plus. Démolir quelque chose, c'est perdre toutes les connaissances qui avaient été incorporées à cette chose en train d'être détruite, même si elle semble étrange dans son état actuel.

De même, Alexander s'oppose à la création de modèles détaillés au préalable. Il donne un bon argument montrant pourquoi les modèles pré-établis ne peuvent pas fonctionner en fin de compte : parce qu'on ne peut pas prévoir de manière absolue

tout ce qui va se passer lors de la construction et de l'implémentation ; parce qu'on oubliera forcément une partie des détails de l'environnement au sein duquel notre création évoluera ; parce que la nature en elle-même n'est pas pré-ordonnée et croît plutôt de manière libre et pousse sans pitié à l'évolution jusqu'à ce que les éléments qui la constituent survivent d'eux-mêmes.

De cette façon, vous ne concevez pas l'interface utilisateur en entier ou la structure complète, pour un grand programme, en une seule étape. Vous allez du grand au petit ou du petit au grand (niveaux d'échelle) ; vous testez chaque partie individuellement jusqu'à ce que ce soit bon (des centres solides) ; vous vous assurez que les parties ne sont pas trop déconnectées les unes des autres (interdépendance). Vous déplacez quelques widgets là où ils sont plus accessibles ou plus proches des données auxquelles ils se réfèrent. Vous enlevez quelques cadres et séparateurs pour réduire le désordre. Par dessus tout, vous évaluez continuellement ce que vous avez créé avec de vrais utilisateurs et des cas d'usage réels pour confronter les choses à la réalité, et non à votre imagination.

Un nom pour la qualité

Tout au long de *The Nature of Order*, Alexander parvient à montrer que les environnements ou les structures construites selon cette méthode finissent toutes par avoir la Qualité sans Nom. Il appelle cela une structure vivante. Cela peut être mesuré et comparé. Ce n'est plus sans nom ; on peut parler d'environnements ou de programmes qui ont une structure plus ou moins vivante par rapport à d'autres – et nous tendons à développer et à obtenir toujours plus de cette propriété.

J'ai seulement intitulé cet article « Le logiciel comme Qualité sans Nom » parce que ça semblait ainsi plus mystérieux.

Je ne peux prétendre connaître la façon parfaite de concevoir et écrire des logiciels. Mais au moins, j'ai une bonne méthode basée sur ce qui produit de bonnes choses ailleurs. Cela a fonctionné pour ma maison et, jusqu'à présent, ça a très bien marché pour mes logiciels. J'espère que ça fonctionnera bien pour vous aussi !

Références

- Christopher Alexander, *A Pattern Language*. Version en ligne : http://bit.ly/8n6igg (NdÉ : lien invalide, le 1er février 2013).
- Christopher Alexander, *The Nature of Order*. Une page web très moche http://www.natureoforder.com (NdÉ : visité le 1er février 2013).
- Photos et dessins des 15 propriétés – http://bit.ly/b82Dxu (NdÉ : visité le 1er février 2013).
- Richard Gabriel, *Patterns of Software*. Un superbe libre qui traite d'un grand nombre d'aspects du développement des logiciels, en transposant les idées de Christopher Alexander pour atteindre les meilleures techniques possibles en développement de logiciel. Version en ligne : http://bit.ly/dqGUp4 (NdÉ : visité le 1er février 2013).
- Richard Gabriel, Christopher Alexander, *the search for beauty*. Une excellente présentation des idées de Christopher Alexander et une galerie de modèles dans le domaine du logiciel. http://bit.ly/ztE6cp (NdÉ : visité le 1er février 2013).
- Richard Gabriel, *The Nature of Order.* Le monde post-modèles. Une autre très bonne présentation, qui fait suite à la précédente, explique les 15 propriétés, le processus fondamental et comment cela peut s'appliquer au logiciel. http://dreamsongs.com/Files/NatureOfOrder.pdf (NdÉ : visité le 1er février 2013).
- Federico Mena Quintero, *Software that has the Quality*

Without A Name. Présentation Desktop Summit de Berlin en 2011. http://bit.ly/oYgJUf (NdÉ : visité le 1er février 2013).