

# La quête du logiciel de qualité – (Libres conseils 22b/42)

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

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

## Le guichet de billetterie

La thèse de doctorat d'Alexander a servi de base à son livre *Notes on the Synthesis of Form* (NdT : Remarques sur la synthèse de la forme), paru en 1964. Il essayait de rationaliser le processus de conception en le définissant comme une progression depuis une série de conditions préalables jusqu'à un résultat final, grâce à une analyse des forces qui déterminent le design.

Permettez-moi de citer Richard Gabriel, dont je parlerai plus tard, quand il décrit l'époque où Alexander essayait de concevoir un guichet de billetterie en prenant appui sur ses idées mathématiques :

*Alexander dit (à propos de la qualité sans nom) :*

*Il s'agit d'un type subtil de liberté issu de contradictions internes. (Alexander, 1979)*

*Cette assertion fait écho aux origines de sa recherche sur la qualité. Elle commença en 1964 alors qu'il était en train de réaliser une étude pour le San Francisco Bay Area Rapid Transit District ([BART](#)) basée sur le travail rapporté dans *Notes on the Synthesis of Form* (Alexander 1964), lui-même*

*basé sur sa thèse de doctorat. L'une des idées-clés de ce livre était qu'avec une bonne conception, il doit y avoir une relation sous-jacente entre la structure du problème et la structure de la solution – les bonnes conceptions passent par la rédaction d'une analyse des besoins, l'analyse de leurs interactions sur les bases d'incompatibilités potentielles, produisant ainsi une décomposition hiérarchisée des différentes parties, et la reconstitution d'une structure dont*

*la hiérarchie structurelle est l'exact complémentaire de la hiérarchie fonctionnelle établie durant l'analyse du programme. (Alexander, 1964)*

*Alexander a analysé le système de forces mises en jeu dans la conception d'un guichet. Lui et son groupe avaient rédigé un cahier des charges en 390 points pour couvrir tous les cas de figure d'usage de l'édicule. Certaines spécifications concernaient des choses telles qu'être là pour obtenir des billets, pouvoir faire de la monnaie, pouvoir déplacer les personnes qui font la queue, réduire la durée d'attente pour obtenir des billets. Il a toutefois remarqué que certaines parties du système n'étaient pas concernées par ces spécifications et que le système lui-même pouvait s'enliser parce que ces autres forces – celles qui ne faisaient pas l'objet d'une spécification – agissaient pour arriver à leur propre équilibre au sein du système. Par exemple, si une personne s'arrêtait et qu'une autre s'arrêtait également pour parler avec la première, cela pouvait créer un embouteillage susceptible de mettre en échec les mécanismes mis au point pour maintenir une circulation fluide. Bien sûr, l'absence d'embouteillage faisait partie des spécifications. Mais il n'y avait rien que les concepteurs puissent faire pour l'empêcher par le biais d'un mécanisme adapté.*

En tant que programmeur, ça doit vous rappeler quelque chose ?

Vous pouvez élaborer une conception magnifique et parfaitement rigoureuse, mais qui s'effondrera quand vous la construirez effectivement parce que des éléments que vous n'aviez pas anticipés apparaîtront alors. Ce n'est pas un échec de votre conception, mais de quelque chose d'autre ! Richard Gabriel poursuit :

*Alexander disait ceci :*

*Il devint alors clair que le bon fonctionnement d'un système ne dépendait pas uniquement de la satisfaction d'une série de conditions préalables. Il s'agissait plutôt d'un système qui trouve sa cohérence en lui-même et en équilibre avec les forces internes générées par ledit système que de l'accord avec une série quelconque de conditions préalables que nous aurions arbitrairement définie. Cela m'intriguait beaucoup car l'idée qui prévalait en général à l'époque (en 1964) était que, pour l'essentiel, tout était fondé sur des objectifs. Toute mon analyse des conditions préalables tendait à converger avec le point de vue de la recherche opérationnelle qui pose qu'il faut établir des objectifs, etc. Ce qui m'ennuyait, c'est qu'une analyse correcte du guichet ne pouvait se baser uniquement sur des objectifs quelconques ; il y avait des réalités qui émergeaient du centre du système lui-même et qui, peu importe votre degré de réussite, avaient un rapport avec le fait que vous ayez créé une configuration stable au regard de ces réalités.*

Et c'est le cœur du problème : comment créer une configuration stable avec les réalités qui en émergent au fur et à mesure que vous la construisez ?

## **La nature de l'ordre**

Bien que Christopher Alexander ait eu conscience d'avoir produit quelque chose de précieux avec ses recherches et catalogues de modèles, il n'était pas complètement satisfait.

D'où venaient les modèles ? Pouvaient-on les créer à partir de rien ou devait-on se satisfaire de ce qu'avait produit jusque-là l'architecture traditionnelle ? D'ailleurs, les modèles étaient-ils réellement nécessaires ? Comment pouvait-on mieux définir et évaluer ou mesurer cette « Qualité sans nom » ?

Alexander passa les vingt années suivantes à tenter de répondre à ces questions. En étudiant le processus réel de création par lequel des environnements bien construits avaient vu le jour, il découvrit que certains processus sont indispensables pour créer des villes ou des édifices agréables – ou toute création humaine en fait. Il arriva aux conclusions suivantes :

- La nature crée des choses qui ont toutes une quinzaine de propriétés en commun (je vous expliquerai plus tard). Cela se produit uniquement par des processus naturels – physique et chimie de base – bien que nous ne sachions pas clairement pourquoi des procédés très différents produisent des résultats similaires ;
- On retrouve ces propriétés dans les architectures traditionnelles ou les villes qui ont simplement évolué au cours du temps. Tous les modèles décrits dans *A Pattern Language* peuvent être obtenus en suivant une méthode fondée sur ces propriétés ;
- Chaque propriété peut également décrire une transformation de l'espace existant ;
- La seule façon de réussir une bonne conception consiste à utiliser ces transformations, une à la fois.

Ceci a été publié en 2003 – 2004 en quatre tomes intitulés *The Nature of Order* (NdT : « La nature de l'ordre »).

# Les quinze propriétés

Le premier tome de La nature de l'ordre traite de quinze propriétés qui apparaissent dans tous les systèmes naturels. Je les résumerai très brièvement (voir les références pour des illustrations et de plus amples explications).

- **Des niveaux d'échelle.** La gamme de tailles est équilibrée, sans changement brutal dans la taille d'objets adjacents. Les éléments ont une échelle fractale ;
- **Des centres forts.** Les différentes parties de l'espace ou de la structure sont clairement identifiables ;
- **Des frontières solides.** Les lignes délimitent les choses. Dans les systèmes vivants, les bords sont les environnements les plus productifs (par exemple, toutes les créatures qui vivent au bord de l'eau) ;
- **Des répétitions alternées.** Haut/bas, épais/fin, forme A et forme B. Les objets oscillent et alternent afin de créer un bon équilibre ;
- **Un espace positif.** L'espace adopte une belle forme convexe et close. Ce n'est pas de l'espace excédentaire. Pensez à la manière dont les cellules d'un diagramme de Voronoï grandissent vers l'extérieur à partir d'un ensemble de points ou à la manière dont les grains d'un épi de maïs se développent à partir de petits points jusqu'à ce qu'ils touchent les grains adjacents ;
- **Une bonne forme.** Les voiles d'un bateau, la coquille d'un escargot, le bec d'un oiseau. Ils parviennent à la forme optimale qui sert leur fonction, ce qui est magnifique ;
- **Des symétries locales.** Le monde n'est pas symétrique dans son ensemble. Cependant, les petites choses tendent à être symétriques parce que, de cette manière, c'est plus facile. Votre maison n'est pas symétrique, mais chaque fenêtre l'est ;
- **Une profonde imbrication et de l'ambiguïté.** Les rues

sinueuses des vieilles villes. Les axones des neurones. Il est difficile de séparer la forme du fond, ou l'avant-plan de l'arrière-plan. Deux centres forts sont renforcés si un troisième est placé entre eux de manière à ce qu'il appartienne aux deux ;

- **Du contraste.** Vous pouvez distinguer où une chose se termine et où la suivante commence parce qu'elles ne se fondent pas l'une dans l'autre ;
  - **Des degrés.** Les choses se confondent les unes les autres là où c'est nécessaire. Les concentrations dans des solutions, les congères ou les talus, les câbles supportant un pont. La manière dont la bande passante décroît alors que vous vous éloignez de l'antenne ;
  - **Des aspérités.** Le monde n'est ni exempt de frottement, ni doux. Les irrégularités sont bénéfiques car elles permettent à chaque élément de s'adapter à son environnement, plutôt que d'être une copie conforme qui n'irait pas aussi bien ;
  - **Des échos.** Les choses se répètent et se font écho. Elles sont uniques dans la précision de leur forme mais leurs contours généraux se répètent à l'infini ;
  - **Du vide.** Parfois, vous avez un grand espace vide pour la tranquillité de la forme. Un lac, une cour, le cadre d'une image ;
  - **De la simplicité et du calme intrinsèque.** Les choses sont aussi simples que possible, sans être simplistes ;
  - **De l'interdépendance.** Chaque chose est dépendante de tout le reste. On ne peut pas séparer un poisson du bassin et des plantes aquatiques. On ne peut pas séparer une colonne de la base du bâtiment.
-

# La quête du logiciel de qualité (Libres conseils 22a/42)

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

Traduction Framalang : Sphinx, peupleLà, lerouge, goofy, alpha, Julius22, SaSha\_01, vvision, lamessen, Bob, lamessen, Garburst, okram

## Le logiciel comme Qualité sans Nom

**Federico Mena Quintero**

*Federico Mena Quintero est l'un des pères fondateurs du projet GNOME et fut auparavant le mainteneur de GIMP. Il travaillait aux Red Hat Advanced Development Labs durant les débuts de GNOME puis fut l'un des premiers à être recruté chez Ximian où il a principalement travaillé sur le calendrier d'Evolution. Il travaille toujours autour de GNOME pour Novell/Suse et vit au Mexique.*

Lorsque j'apprenais la programmation, j'ai remarqué que je rencontrais souvent le même problème, encore et encore. J'écrivais souvent un programme qui fonctionnait relativement bien et était même bien structuré. Mais après quelques modifications et améliorations, je ne pouvais plus l'améliorer davantage. Soit sa complexité me surpassait, soit il était écrit de telle manière qu'il ne permettait pas d'évolution, comme une maison que vous ne pouvez pas agrandir à cause d'un toit pentu ou que vous ne pouvez pas étendre sur les côtés à cause des murs qui l'entourent.

À mesure que je progressais, j'ai appris à gérer cette complexité. Nous apprenons tous à le faire avec différents outils et techniques : l'abstraction, l'encapsulation, l'orientation objet, les techniques fonctionnelles, etc. Nous apprenons comment diverses techniques nous permettent d'écrire des programmes plus complexes.

Cependant, le problème d'un programme trop optimisé ou bien trop intimement enchevêtré pour être modifié a persisté. Parfois, je pensais tenir une superbe conception. Mais la modifier d'une façon quelconque l'aurait « amochée » et je ne le souhaitais pas. D'autres fois, j'avais quelque chose avec tellement de parties imbriquées que je ne pouvais plus y connecter quoi que ce soit sans que tout s'effondre sous son propre poids.

Il y a quelques années, la manie de la réécriture a débuté sans que j'y prête trop attention. Je me disais que c'était une façon de nettoyer le code, mais après ? Je sais déjà comment prendre un morceau de code et le transformer en fonction ; je sais déjà comment prendre des morceaux de code similaires et les transformer en classes dérivées. Je sais déjà écrire du code presque entièrement propre. Quel est donc le problème ?

J'ai écarté la refactorisation(1) en la considérant comme destinée aux programmeurs les moins expérimentés ; comme de jolies recettes de nettoyage de code mais rien qui ne puisse être découvert par soi-même.

La même chose s'est produite avec les canevas de conception. Je pensais qu'ils donnaient simplement des noms pompeux tels que Singleton et Strategy à des types de structures de tous les jours qu'on utiliserait naturellement dans un programme. Peut-être mon ego de programmeur était-il trop démesuré pour considérer ces travaux avec sérieux. C'est alors que quelque chose s'est produit.



# Le travail de Christopher Alexander

Il y a quelques années, mon épouse et moi avons acheté une petite maison de plain-pied et nous souhaitions l'agrandir. Nous envisagions d'avoir un enfant et avions, par conséquent, besoin de plus d'espace. J'avais besoin d'un vrai bureau à domicile, pas une simple niche inoccupée dans laquelle mon bureau et mes étagères de livres tiendraient à peine. En tant que cuisiniers avides, nous avions tous les deux besoin d'une cuisine plus spacieuse et confortable que celle de notre maison. Mon épouse avait besoin d'une pièce bien à elle.

Nous ne voulions pas payer pour un architecte coûteux et aucun de nous deux n'avait la moindre connaissance en matière de construction. Comment allions-nous concevoir notre maison ?

Par moments, en naviguant sur le Web, je me rappelais que j'avais déjà vu le nom d'un auteur, le titre d'un livre ou quelque chose comme ça. Il se peut que je n'y aie pas vraiment porté attention par le passé mais, d'une manière ou d'une autre, plus je vois la même chose mentionnée, plus il est probable que je finirai par m'y intéresser suffisamment pour aller voir de quoi il s'agissait. « Oh, plusieurs personnes ont déjà mentionné ce nom ou ce livre ; je devrais peut-être y jeter un coup d'œil »

C'est exactement ce qui s'est passé avec le nom de Christopher Alexander. J'avais lu que c'était un architecte assez spécial (de vrais bâtiments, pas de logiciels), connecté en quelque sorte au monde du logiciel par le biais de techniques orientées objet. J'ai été passionné par son travail dès que j'ai commencé à le découvrir par mes lectures.

Dans les années 1970, Christopher Alexander était un mathématicien et professeur d'architecture à l'université de Californie, Berkeley. Lui et un groupe d'architectes de la même mouvance que lui ont parcouru le monde, essayant de comprendre pourquoi il existait des endroits construits par

des humains (cités, villes, parcs, immeubles, maisons) où il était très agréable de vivre, confortables, conviviaux et jolis, tandis que d'autres endroits ne l'étaient pas. Des lieux agréables étaient présents dans toutes les architectures traditionnelles du monde – européennes, africaines, asiatiques et américaines – amenant à l'idée que des facteurs communs étaient sans doute extractibles.

Alexander et son équipe ont réparti leurs découvertes au sein d'une liste de motifs architecturaux cohérents et publié trois livres : *The Timeless Way of Building* (NdT « L'art de la construction intemporelle » en français), où sont décrites la philosophie et la méthode nécessaires à une bonne construction ; *A Pattern Language* (NdT : « Un langage de schémas » en français), que je décris ci-après et *The Oregon Experiment* (NdT « L'expérience de L'Oregon » en français) où sont détaillés la conception et la construction d'un campus universitaire grâce à leur méthode.

## Une grammaire de modèles

Un modèle (ou schéma) est un problème récurrent lors de la conception et de la construction d'objets, en lien avec les contraintes qui modèlent le problème et avec une solution connectée, quasi-récurivement à d'autres modèles-pères ou modèles-fils. Par exemple, considérons le GRADIENT D'INTIMITÉ, un modèle important dans le livre (les modèles étant en capitales dans ce livre pour en faciliter l'identification, je ferai donc de même) :

### GRADIENT D'INTIMITÉ

**Modèles-pères, et préambule** : ... si vous savez à peu près où vous avez l'intention de placer les ailes du bâtiment, LES AILES DE LUMIÈRE, et combien d'étages elles auront, le NOMBRE D'ÉTAGES, et où L'ENTRÉE PRINCIPALE se trouve, alors il est temps de travailler sur la disposition approximative des zones

principales de chaque niveau. Dans tout bâtiment, la relation entre les zones publiques et les zones privées est de la plus haute importance.

**Énoncé du problème** : à moins que les volumes d'un édifice ne soient disposés dans un ordre correspondant à leur degré d'intimité, les visites des étrangers, amis, invités, clients, famille seront toujours un peu gênantes.

**Explication** : je ne vais pas tout citer. Prenez, par exemple, un appartement où la seule façon d'atteindre la salle de bain est de passer d'abord par la chambre à coucher. Les visites sont toujours gênantes car vous avez l'impression de devoir d'abord ranger votre chambre si vous voulez que vos visiteurs puissent utiliser les toilettes. Ou bien considérez des bureaux dans lesquels vous ne voulez pas qu'un espace de travail calme soit à côté de la réception, car alors celui-ci ne sera pas calme du tout – vous voulez qu'il soit plus privé, à l'arrière.

**Résumé de la solution** : disposez les espaces de l'édifice de façon à ce qu'ils créent une suite qui commence avec l'entrée et la partie du bâtiment ouverte au public, puis conduit aux zones un peu plus privées pour finir avec les domaines les plus intimes.

**Modèles-fils à consulter** : ZONES COMMUNES AU CENTRE. HALL D'ENTRÉE pour les maisons ; UNE PIÈCE PROPRE À CHACUN pour les particuliers. UNE RÉCEPTION VOUS SOUHAITE LA BIENVENUE dans les bureaux avec, BUREAU SEMI-PRIVÉ à l'arrière.

Les modèles deviennent plutôt précis, ils n'imposent cependant jamais un style ou une forme déterminée au résultat. Par exemple, il existe un modèle qui s'appelle « ÉTAGÈRES OUVERTES ». Des placards profonds vous incitent à mettre les choses les unes derrière les autres, ainsi, vous ne pouvez plus voir ni attraper les objets qui sont au fond. Ils prennent aussi beaucoup de place. Les étagères dont la

profondeur permet de ne mettre qu'un seul objet restent rangées et vous savez toujours, en un seul coup d'œil, où se trouve chaque chose. Les objets que vous utilisez fréquemment ne devraient pas être derrière une porte.

Vous pouvez ainsi entrevoir l'essence même des modèles de conceptions : de bonnes recettes éprouvées qui n'imposent pas de contraintes inutiles à votre implémentation

Les modèles ne demandent pas un style particulier ou des décorations superflues : le livre ne vous dit pas : « appliquez ces motifs floraux sur les rampes », mais plutôt : « les pièces d'une maison devraient être orientées de telle sorte que le soleil les éclaire harmonieusement, au moment de la journée où elles sont le plus utilisées – à l'est pour les chambres le matin, à l'ouest pour le salon l'après-midi ».

J'avais acquis un exemplaire de *A Pattern Language* peu avant de commencer à agrandir notre maison. Le livre fut une révélation : c'était le moyen d'aborder la conception de notre maison et nous pouvions alors la réaliser par nous-mêmes au lieu de payer très cher une solution inadaptée. Nous étions capables de faire un plan sommaire de notre maison et ensuite d'imaginer des détails plus fins au fur et à mesure de la construction. C'est le genre de livres qui, pendant que vous le lisez, parvient à confirmer les intuitions que vous éprouviez confusément – le genre de livres qui vous fait dire en permanence : « Bien sûr, c'est exactement à ça que je pensais »

*Design Patterns*, le fameux livre publié par Gamma et autres, puise directement son inspiration dans les schémas architecturaux d'Alexander. Ils voulaient faire la même chose : dresser une liste de problèmes apparaissant fréquemment lorsque l'on programme et présenter de bonnes solutions qui n'imposeront pas de contraintes inutiles à votre implémentation.

Une chose dont j'ai pris conscience en lisant *A Pattern Language*, c'est une chose importante que l'on retrouve à la fois dans les modèles architecturaux et logiciels) : ils nous fournissent un vocabulaire pour discuter de la façon dont les objets sont construits. Il est beaucoup plus pratique de dire : « Les propriétés de cet objet ont des observateurs » plutôt que : « il est possible de lui attribuer des fonctions de rappel qui sont appelées lorsque ses propriétés changent ». Ce que je pensais être uniquement des termes pompeux ne sont, en fait, qu'une manière d'exprimer la connaissance de manière compacte.

## La Qualité Sans Nom

La plus grande partie de l'explication d'Alexander sur les modèles de conception et leur philosophie se rapporte à quelque chose qu'il appelle « La qualité sans nom ». Vous connaissez des endroits qui ont cette qualité sans nom. Elle est présente dans le café où vous aimez aller lire car la lumière de l'après-midi entre avec exactement la bonne intensité. Et il y a là-bas des chaises et des tables, c'est toujours plus ou moins rempli de gens sans pour autant que vous vous sentiez oppressé. Elle est présente au coin d'un parc où un arbre ombrage un banc. Il y a peut-être un peu d'eau vive, et il importe peu qu'il pleuve ou bien que le temps soit ensoleillé, cela semble toujours être un plaisir d'y être. Pensez à une maison de *hobbit* où tout est à portée de main, où tout est confortable et où tout est fait élégamment.

Un objet ou un endroit possède la *qualité sans nom* s'il est convivial, s'il a évolué dans le temps selon sa logique propre, s'il ne recèle pas de contradiction interne, n'essaie pas d'attirer l'attention et semble réunir toutes les qualités archétypales – comme s'il avait suivi tout naturellement la voie optimale pour aboutir à sa conception. Plus important,

Alexander affirme que c'est une qualité objective et non subjective, et qu'elle peut être mesurée et comparée. Bien que cela semble être une définition floue, c'est l'état le plus abouti dans lequel Alexander a pu l'amener lors de la première phase de son travail. La vraie révélation n'apparaîtrait que plus tard.

En tant que développeurs, nous avons tous vu de beaux programmes à un moment donné. Ce sont peut-être les exemples de *Programming Pearls*, un beau livre que tout hacker devrait lire. Peut-être avez-vous vu un algorithme bien implémenté qui regorge de justesse. Vous vous souvenez peut-être d'un morceau de code très compact, très lisible, très fonctionnel et très correct. Ce logiciel possède la *qualité sans nom*. Il était devenu évident pour moi que je devais apprendre à écrire des logiciels qui atteignent le niveau de la qualité sans nom et que la méthode d'Alexander était le bon point de départ pour y parvenir.

## **(à suivre...)**

(1) Refactorisation : bien que critiqué, cet anglicisme (voir <http://fr.wikipedia.org/wiki/Refactorisation>) semble d'un emploi courant chez les développeurs. On pourrait parler plus simplement de remaniement