Autour des arbres qui cachent la forêt

Pour bien savoir les choses, il en faut savoir le détail, et comme il est presque infini, nos connaissances sont toujours superficielles et imparfaites. (La Rochefoucauld).

Le diable est dans le détail. (Expression consacrée)

Le bon Dieu est dans le détail (Gustave Flaubert)

Par Ivan Maffezzini

1 Introduction

On peut « se perdre dans les détails[1] », comme on peut « oublier un détail important » ; on peut « admirer les détails d'une œuvre » et s'ennuyer en écoutant des « histoires dans le menu détail » ; il arrive que l'on se débarrasse de quelque chose de non important ou de quelque chose de gênant avec un simple « ce n'est qu'un détail ». « Détail » peut varier, souvent sans que l'on sache pourquoi : d'« important » à « menu » ou d'« essentiel » à « infime ». Les détails glissent entre les mains ou collent aux doigts, selon. Ils attirent les oppositions. Ils couvrent un très grand spectre de valeurs et leur position dépend des circonstances, des objectifs, de la stratégie, des connaissances, des paradigmes cachés… quand on n'entre pas dans les détails, on peut dire n'importe quoi sur les détails.

Dans le domaine du génie logiciel (GL) aussi, le détail occupe une place particulière. Il peut être oublié comme sans importance dans une abstraction pour pouvoir saisir le tout ou être analysé en profondeur pour bien comprendre le système. Il peut être l'élément qui empêche de comprendre ou celui qui ouvre des voies insoupçonnées. Un court texte plein de bon sens comme [1] est un exemple paradigmatique de l'emploi du terme « détail » : « *Tout le temps passé à insérer des détails inutiles dans une exigence est normalement enlevé aux détails utiles d'une autre exigence* ». Ce qui se réduit à une lapalissade du genre « ce qui est utile est plus utile que ce qui est inutile ».

Mais, qu'est-ce qu'un détail en GL ? Impossible de trouver une définition, sinon indirectement et souvent en lien avec « abstraction ». Voici, à titre d'exemple, la définition d'abstraction tirée du guide d'UML 2.0 [2] : « *L'acte d'identifier les caractéristiques essentielles d'une chose qui la différencie des autres genres de choses et d'**omettre les détails** qui ne sont **pas importants** d'un certain point de vue.* » (C'est nous qui soulignons). Ce qui nous permet de proposer une définition qui pourrait sans doute faire l'unanimité : « Un détail est tout ce qui n'est pas important, à un certain moment, selon un certain point de vue ». Définition qui, hélas ! soulève au moins autant de problèmes qu'elle n'en résout.

[1] Un merci particulier à Louis Martin professeur au département d'informatique de l'UQAM qui m'a aidé à ne pas me noyer dans les détails.

2 Honneur

Peu importent les méthodes, indépendamment des outils, quel que soit le domaine à automatiser, la progression vers le produit final peut être mesurée par la quantité de détails intégrés aux artefacts. Exemple : une lettre qui fait état de nouveaux besoins est détaillée dans le document *Principe d'opération* qui, à son tour, est détaillé dans la *Spécification des exigences du système* détaillée dans la *Spécification des exigences logicielles* qui sera détaillée dans… et cela, jusqu'au code. Voir le passage de la lettre au code — de quelques dizaines de mots à des dizaines de milliers d'instructions — comme un ajout de détails est probablement le point de vue le mieux partagé en GL. Que, lors du passage des exigences à la conception, l'on dévie des détails du problème vers ceux de la solution ne fait que confirmer la progression : les détails de conception, en tant qu'interprétations des détails du problème, aident à mieux définir ce dernier. Il n'est sans doute pas faux de dire que la maîtrise de l'intégration des détails est ce qui caractérise tout cycle de développement d'un logiciel, toutes méthodes confondues.

Mais pourquoi ?

Sans doute à cause du phénomène psychologique qui contraint notre manière de saisir le monde : l'attention portée aux détails empêche de saisir l'ensemble[2], et donc d'établir une direction unitaire et de contrôler les déviations éventuelles. Cela n'est pas vrai uniquement pour la technique logicielle : toute tentative visant à maîtriser le processus de création d'un objet technique repose sur l'hypothèse que « les détails viennent après ». Il est même possible de généraliser ces considérations à toute approche opératoire guidée par des objectifs. Les armées napoléoniennes qui, au début du XIX^e^ siècle, progressaient dans les plaines ukrainiennes n'étaient certainement pas guidées par un homme qui se perdait dans les détails, et le détail qui le perdit — et surtout perdit 400 000 hommes de son « projet » —, le célèbre « général hiver », était tout sauf un détail !

Il est vrai qu'un détail peut nous perdre, mais s'il peut nous perdre, c'est parce que d'autres détails ont pris le dessus et nous ont empêché de l'intégrer au bon moment.

Mais, retournons au GL, où les détails sont l'antidote à l'ambiguïté. Même si, dans un document comme celui de IEEE qui sert de guide pour l'écriture des exigences logicielles [3] on affirme qu'une bonne spécification devrait être non ambiguë, cette « qualité » est, sinon impossible à obtenir, certainement très souvent trop coûteuse et, surtout, elle augmente les risques de ne pas livrer le produit. On aura éventuellement une spécification des exigences « parfaite » du point de vue de l'ambiguïté mais difficile à valider, à modifier, à communiquer… un frein au projet au lieu d'être un tremplin. Cette approche « centrée sur les exigences » éventuellement valide pour des systèmes très stables et « sémantiquement pauvres » est loin d'être la norme. Et même ceux qui, comme nous, ne croient pas que les approches agiles soient une panacée, doivent admettre que le fait d'insister sur l'importance d'avancer dans les détails de la conception et éventuellement du codage sans que tous les détails des exigences soient définis est un grand pas en avant dans notre domaine.

---

[2] Veniamin, le patient étudié pendant de longues années par Luria et que l'incapacité d'oublier les détails rendait incapable de comprendre le sens des métaphores les plus simples, était une représentation vivante de ce que signifie « se perdre dans les détails ».

Des ambiguïtés continueront à exister dans le document[3] et elles ne seront éliminées qu'à force de détails dans les artefacts suivants. Peu importe le rythme d'élimination des ambiguïtés, ce qui est certain, c'est que le point de départ (là où naît le problème) est un point de départ plein d'imprécisions et d'ambiguïtés. Étant donné que :

1. les détails ne doivent pas obscurcir la compréhension du problème ;
2. ce sont les détails qui permettent d'éliminer les ambiguïtés en rendant les artefacts toujours plus précis ;
3. dans la machine finale, les ambiguïtés doivent être éliminées si on ne veut pas qu'elles se transforment en erreurs,

une bonne pratique en GL est de faire entrer les détails « peu à peu » dans le projet afin qu'ils trouvent leur place dans l'ensemble, sans trop bouleverser ce qui est déjà bien établi. Même les « extrémistes » qui pensent que seule compte la construction du code en interaction avec les autres programmeurs ne nieraient pas qu'une certaine vision de ce que doit faire une classe précède les détails de codage.

3 Déshonneur

Dans la technique, toute approche qui fait passer du général au particulier est efficace seulement si, au préalable, il y a eu un travail ascendant qui, en partant du particulier, a permis de construire le général. L'approche descendante est fort utile pédagogiquement, et peut être très importante en philosophie, mais la science comme la technique ont progressé surtout à cause de l'importance qu'elles ont donné « aux détails du réel ». Peu importe votre position épistémologique[4], à la fin, un détail du réel sera en même temps le « diable » qui fait écrouler votre théorie et le « bon Dieu » qui ouvre de nouvelles voies[5].
L'approche descendante en GL à depuis fort longtemps un champ d'application assez restreint : la conception et la construction de « petits » programmes aux entrées et aux sortie parfaitement définies.

Pour les problèmes plus complexes et, surtout, pour les problèmes dont on ne connaît pas les contours, cette approche n'est pas seulement intenable mais, quand elle est vraiment[6] suivie, elle porte à des solutions « belles sur papier » mais fonctionnellement inacceptables. Ne pas connaître « les contours » du problème ce n'est ni un accident ni un manque d'approfondissement : c'est la condition de travail normale lorsqu'on se trouve

---

[3] À moins, bien sûr, de formaliser l'écriture des exigences. Mais qu'est-ce que « formaliser », sinon une manière de détailler guidée par la logique et les mathématiques ?

[4] Même des rationalistes qui suivent une épistémologie comme celle de Imre Lakatos doivent admettre que, à un certain moment, la ceinture protectrice ne peut plus défendre le noyau dur contre les attaques des détails du réel qui sévit hors de la théorie.

[5] La forêt n'existe pas sans les arbres, mais les arbres existent indépendamment de la forêt. On peut frapper de la tête contre un arbre ou cueillir les fruits qu'il nous offre. La forêt n'est qu'une tache que l'on observe de loin. Et ceux qui disent marcher dans la forêt veulent simplement dire qu'ils marchent dans les espaces que les arbres n'occupent pas.

[6] Souvent, surtout quand l'approche descendante était plus à la mode, on montrait sur papier une progression du haut vers le bas en oubliant tout le va-et-vient entre détail et généralisation qui avait permis de réaliser le programme. On enlevait les taches pour rendre l'œuvre acceptable auprès d'une communauté de chercheurs qui n'osait pas admettre qu'il fallait se salir les mains avec tout autre chose que la déduction.

devant un problème qui n'est pas un simple problème informatique — ce qui ne veut pas dire qu'il s'agit d'un problème informatique simple !

Mais pourquoi les détails sont-ils si importants et doivent-ils être mis de l'avant ? Pour quelque chose d'extraordinairement simple : ce sont les détails qui permettent de comprendre. De comprendre et donc de « saisir » le réel lui-même[7] ; de nous donner l'assurance que l'on sera capable de construire une machine qui, sans rien comprendre, mettra en relation les phénomènes du monde réel pour résoudre le problème qui obscurcissait le réel. C'est parce que les détails imposent un point d'arrêt conceptuel que l'on peut bâtir un réseau de concepts dans les mailles duquel on peut espérer d'emprisonner les phénomènes qui s'agitent derrière les mots. Et c'est parce que la compréhension a mis des limites au réel que l'on peut « jeter les détails » dans la machine et la laisser les manipule à notre façon.

Jusqu'à ce que l'on touche au détail et que l'on sache le manipuler, on comprend sans comprendre : on est seulement capable de répéter — comme nous tous le savons très bien dès les premières années de l'école.

Pour comprendre le problème et le domaine à automatiser, il faut les détails du problème, du domaine, des exigences, de l'environnement… Pour comprendre le programme et le système il faut les détails de la conception, du code, des interfaces… D'un côté l'environnement auquel la machine « contrainte par le programme » imposera des contraintes, de l'autre côté l'environnement dans lequel la réflexion « contrainte par les détails » puisera les détails pour construire le programme. Simple ? Pas nécessairement. Ça dépend des détails. (Voir à propos de la « compréhension » l'encadré)

4 Au-delà

Peu importe que ce soit le diable ou le bon dieu qui se cache dans le détail, ce qui est certain c'est que ce sont les détails qui causent le succès ou la faillite d'un projet. Mais une fois que l'on a dit cela on n'a rien dit d'utile à cause, d'une part, du côté extrêmement subjectif des détails et, d'autre part, de l'impossibilité d'en connaître la portée avant la fin du projet — quand, hélas ! il est trop tard.

Mais, si on ne connaît l'importance d'un détail qu'à la fin d'un projet, devons nous pour autant abandonner le concept de détail comme vide et inutilisable ? Certes, on le peut. Mais cela nous semble être la voie facile qui nous porte à jeter le bébé avec l'eau du bain. Pour ne pas jeter le bébé, il faut donc trouver un moyen de savoir si un détail est un détail très tôt dans le projet. Tâche toute autre que facile mais, à notre avis, loin d'être impossible si la méthode de travail facilite la venue des détails sans que ceux-ci obscurcissent la vue d'ensemble.

Considérons un projet dans lequel il s'agit de résoudre un problème pour lequel on n'a pas de vue d'ensemble (la forêt) préexistante. Un VRAI problème dans le domaine à informatiser et donc un problème pas nécessairement bien défini.

---

[7] Qui malheureusement ne se réduit pas aux mots, comme l'informatisation ne cesse de le montrer, même si le voile des mots est indéchirable.

Pour créer des réseaux de significations qui permettent à ceux qui ont le besoin d'automatiser et à ceux qui devront automatiser d'avoir des points de référence communs solides, il est nécessaire de creuser les exigences du domaine. « Creuser » les exigences implique aller jusqu'aux détails jugés les plus infimes pour être sûr d'avoir bien compris. Et, pour être sûr, il ne faut pas craindre de s'appuyer sur des détails de programmation qui nous aideront à éclaircir le problème « par analogie ». Mais, dans un domaine qui n'est pas un simple domaine « jouet », les exigences et les concepts sous-jacents sont trop nombreux pour être abordés en même temps. Il faut donc trouver une manière de les catégoriser et de leur donner un poids qui puisse guider dans la démarche de réalisation du produit. Parmi les différentes catégorisations que l'on retrouve en GL (priorité, nécessité, criticité, type[8], etc.), nous croyons que la stabilité est celle qui peut le mieux aider à bâtir une approche à l'automatisation qui prenne en charge les détails dès le début. Voici les grandes lignes d'une démarche possible :

1. Les utilisateurs et les experts du domaine assignent un niveau de stabilité aux exigences, aux objets, aux objectifs, etc. du domaine.
2. Les éléments les plus stables sont isolés et considérés comme le noyau pour cerner le problème et cela indépendamment de la priorité, de la criticité, etc.
3. Création d'une coupe transversale dans le domaine qui va des exigences au code. Cette coupe devient un projet en soi qui doit conduire à un premier incrément livrable. Projet où exigences, conception et codage sont réalisés en parallèle pour créer la synergie nécessaire à une vraie compréhension du problème. Si le projet est trop grand à cause du fait qu'un nombre trop important d'exigences est stable, d'autres critères seront employés pour réduire la taille des incréments.

On pourrait nous objecter qu'il est rare que l'on ait des cas « difficiles » comme le précédent, qu'à cause de la progression de l'automatisation les « problèmes aux contours flous » sont toujours moins nombreux. Objection accueillie : s'il n'y a pas d'élément à éclaircir dans le domaine mais que seul existe le problème informatique, alors ce sont les détails techniques qui comptent. Comme quoi on ne se libère pas des détails à moins de se libérer de la réalité.

5 Bibliographie

[1] Carl Chang, Mark Christiansen, "Blueprint for the ideal Requirement Engineer", *IEEE Software*, March 1996.

[2] James Rumbaugh, I. Jacobson, G. Booch, *The Unified Modeling Language Reference Manua*l, Second Edition, Addison Wesley 2005.

[3] IEEE, IEEE std 830-1998, Recommended Practice for Software Requirements Specification

[8] Dans le *Guide pour développer des spécifications du système* d'IEEE (IEEE Std. 1233 1998), par exemple, sont proposé vingt-quatre (24) types différents qui vont de la *fiabilité*, à la *facilité de transport*, de l'*ergonomie* aux *performances*…

ENCADRÉ

Un ingénieur du logiciel sourd et les mouvements du *Requiem* de Mozart.

SCEPTIQUE. Pour construire un programme qui permette à un ordinateur d'interagir avec l'environnement, faut-il comprendre l'environnement ?

INGÉNIEUR DU LOGICIEL . Certes. La condition préalable à toute informatisation est la compréhension des concepts du domaine.

SCEP. Je vous crois, mais… donnez-moi un peu plus de détails afin que je puisse comprendre.

ING. Quel genre de détails ?

SCEP. À vrai dire, je ne sais pas très bien quel genre de détail, mais je pourrais me venir en aide avec un exemple très simple. Comme vous le savez, je collectionne les enregistrements du *Requiem* de Mozart et j'aimerais avoir, sur écran ou sur papier, les mouvements des différentes interprétations ordonnés selon la durée.

ING. Très simple. Il s'agit-là d'un cas particulier du besoin d'ordonner les mouvements des interprétations de n'importe quelle œuvre, qui est un cas particulier du besoin d'ordonner des objets. Il suffit que l'utilisateur saisisse le titre de l'œuvre et si les informations sur la durée sont dans la base de données…

SCEP. Je ne suis intéressé que par les interprétations du *Requiem* de Mozart, mais si en compliquant un peu le choses vous vous simplifiez la vie… c'est parfait. Donc, pour écrire ce programme, vous devez connaître la signification de mouvement, d'interprétation et d'œuvre, j'imagine.

ING. Bien sûr. Mais pas seulement. D'écran, d'imprimer et d'ordonner aussi.

SCEP. Je n'en doute pas. Mais il s'agit là de concepts de votre domaine, de l'informatique, ou de la vie quotidienne. Ce qui m'intéresse, ce sont les concepts du domaine : mouvement, œuvre et interprétation. Arrêtons-nous à *mouvement*.

ING. Très bien. Pour afficher la liste ordonnée, il suffit que la base de données contienne, pour chaque interprétation, les mouvements avec leur durée.

SCEP. Et pour faire cela vous devez comprendre ce qu'est un mouvement.

ING. Bien sûr.

SCEP. Est-ce que vous savez ce qu'est un mouvement ?

ING. Plus ou moins un mouvement est… est, disons une subdivision d'une interprétation.

S. Pas tout à fait. Si vous voulez parler de « subdivision », il faudrait alors dire qu'il s'agit d'une subdivision de l'œuvre et non de l'interprétation.

ING. Oui… donc une œuvre contient des mouvements… mais… mais vous avez dit que vous voulez connaître la durée des mouvements dans les interprétations. Vous m'avez embrouillé les idées.

SCEP. Fort probablement parce qu'elles l'étaient déjà un peu. Le mouvement est une caractéristique de l'œuvre, mais sa durée dépend de l'interprétation.

ING. Ok. Donc, chaque œuvre a un titre et plusieurs mouvements. L'application sera très simple : il s'agit de rechercher toutes les interprétations d'une œuvre donnée, de lire la

durée de chaque mouvement selon l'ordre d'apparition dans l'œuvre, de les ordonner et de les afficher.

SCEP. Parfait. Je vois que notre échange vous a déjà permis de détailler des éléments qui, pour moi, étaient implicites, comme le fait qu'il est préférable d'afficher les mouvements dans l'ordre d'apparition dans l'œuvre. Pour « mes » *Requiem* j'aurais alors le *Requiem-kyrie* suivi du *Dies irae*, etc.

ING. *Dies irae* ? Est-ce que les mouvements ont un nom ?

SCEP. Pas dans toutes les œuvres. Dans le *Requiem* de Mozart, par exemple, oui.

ING. Très simple, on peut donc présenter les informations sous forme de tableau : dans la première colonne le numéro séquentiel du mouvement, dans la deuxième le titre, quand il existe, dans la troisième la durée.

SCEP. Maintenant que vous avez eu un détail de plus sur les mouvements, serez-vous capable d'expliquer à l'un de vos collègues ce qu'est un mouvement de façon à ce qu'il puisse construire le programme.

ING. Certes. Je lui dirai à peu près ceci : *œuvre* est un concept ayant un *nom* et un ou plusieurs *auteurs* et elle est constituée de *mouvements* qui ont eux aussi un *nom* optionnel. Une *œuvre* peut avoir plusieurs *interprétations* et chaque *mouvement* a une *durée* qui dépend de l'*interprétation*.

SCEP. Est-ce qu'un ingénieur du logiciel sourd dès la naissance pourrait écrire cette application ?

ING. Oui.

SCEP. Comment une personne sourde peut-elle comprendre ce qu'est un mouvement si elle ne sait même pas ce qu'est la musique ?

ING. Beethoven était sourd, n'est-ce pas ?

SCEP. Mais il y a eu une période de sa vie où il n'était pas sourd ! Si je comprends bien, votre collègue a besoin seulement de savoir qu'un œuvre musicale est quelque chose qui est constitué d'une suite ordonnée de quelque chose d'autre qui a un nom et une durée et que, quand on l'affiche à l'écran, on écrit « Durée du mouvement x min et y sec. » Il n'a aucun besoin de savoir ce qu'est un mouvement.

ING. Il n'a pas besoin de le savoir si quelqu'un lui a tout décrit.

SCEP. Donc si moi, mélomane, je lui détaille tout ce dont j'ai besoin…

ING. Il n'a pas besoin de comprendre. Vous avez raison. Il n'a pas besoin de comprendre, comme l'ordinateur ne comprend pas.

SCEP. Maintenant vous allez trop loin ! Votre collègue, pour préparer ses tables, doit comprendre mon langage ordinaire — écrit, bien sûr ! — et c'est cette compréhension qui lui permet d'informatiser sans rien comprendre au domaine qu'il informatise.