Discours des méthodes

par Ivan Maffezzini

> *Il est vrai que nous ne voyons point qu'on mette par terre toutes les maisons d'une ville pour le seul dessin de les refaire d'autres façons et d'en rendre les rues plus belles, mais on voit bien que plusieurs font abattre les leurs pour les rebâtir et que même quelques fois ils y sont contraints quand elles sont en danger de tomber d'elles-mêmes.*
>
> Descartes, *Discours de la méthode*

Introduction

En génie logiciel (GL), les articles et les débats sur les méthodes sont innombrables, comme sont innombrables les méthodes appliquées dans les projets. Nous avons écrit « méthodes » mais nous aurions très bien pu écrire « méthodologies », car la signification originelle de ce dernier terme — « *étude des méthodes* » — est pratiquement disparue pour faire place à la signification de *« ensemble de règles et de démarches adoptées pour conduire une recherche*[1] », ce qui le rend synonyme de « méthode ».

Pour éclaircir la terminologie, nous allons donner quatre définitions très générales et ensuite présenter un schéma conceptuel en UML.

Méthode : Démarche rationnelle (et, donc, communicable) permettant d'obtenir un certain résultat, souvent à l'aide d'outils.

Méthode officielle : Méthode dont le contexte, les limites, les contraintes, les étapes, les notations et les outils sont décrits dans des documents reconnus par l'entreprise qui applique la méthode.

Méthodologie : Composition d'une ou plusieurs méthodes en un tout cohérent dans le but de livrer un produit. La méthodologie couvre tout le cycle de vie du logiciel.

Méthodologie officielle : Méthodologie dont toutes les méthodes sont officielles.

Dans un domaine riche en ambiguïtés comme celui des méthodes, pour ne pas emboîter le pas à la langue anglaise, où l'épithète « *formal* » peut signifier autant « fondé sur le mathématiques pour permettre des calculs » que « officiel », nous réservons le signifiant « formel » pour « fondé sur les mathématiques » et nous employons « officiel » pour signifier « ce qui émane de l'organisation et qui n'est pas *ad hoc* ». Dans ce sens, une méthode *officielle* peut être ou ne pas être *formelle*.

À moins d'avis contraire, dans la suite de l'article, nous ne considérerons que les méthodologies et les méthodes officielles que nous désignerons par MO.

La fonction du schéma de la figure suivante n'est pas de modéliser les méthodologies mais seulement de rendre un peu plus clairs les concepts de « méthodologie » et de « méthode » en représentant quelques associations avec d'autres concepts du domaine. Pour rendre le schéma moins encombré, toutes les associations, à moins d'indication contraire, sont de type m..n (ce qui n'est pas conforme à la notation UML).

[1] Trésor de la langue française.

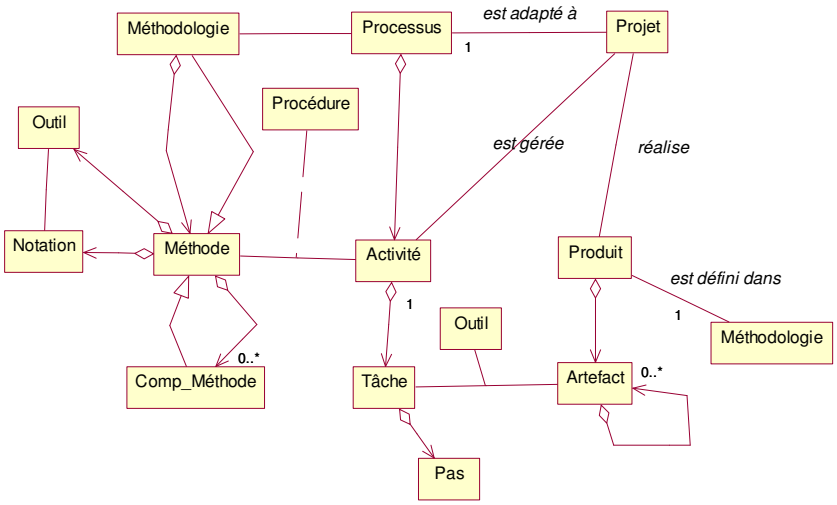

Figure 1 : Méthodologie et processus

Une méthodologie est constituée d'au moins une méthode qui, à son tour, peut être décomposée en composants (Comp_méthode), qui sont les plus petits éléments réutilisables d'une méthodologie. Les méthodologies et les composants des méthodes sont des spécialisations des méthodes et comme celles-ci sont constituées d'outils et de notations. Il est à noter qu'une méthode peut appartenir à plus d'une méthodologie tout comme un composant de méthode peut appartenir à plus d'une méthode, ce qui permet de créer des méthodologies par composition.

Les processus et les produits sont structurés comme les méthodologies. Il existe des outils pour faciliter l'emploi des méthodes, et il existe aussi des outils pour faciliter la génération des artefacts lorsque les humains, dans le cadre d'un projet, exécutent une tâche. Les artefacts peuvent être composés d'artefacts : une fiche qui décrit une exigence, par exemple, peut être un artefact ayant un cycle de vie propre et être dans un document de spécification des exigences du logiciel qui, lui aussi, est un artefact. Un projet s'adapte au processus et, dans le projet, on gère des activités et on réalise un produit. L'organisation du produit est définie dans la méthodologie. L'association entre méthodes et activités, qui sont des parties du processus, est caractérisée par des procédures. La structuration de processus en activités et de celles-ci en tâches est empruntée à ISO 12207 avec l'ajout des *pas* qui sont les composants élémentaires des tâches et sont les plus petites parties réutilisables d'un processus. [1]

Ce n'est pas un hasard si nous réservons pour la fin la description de l'association entre **méthodologie** et **processus,** car il s'agit d'une association qui, dans la brève histoire du GL, a eu — et continue d'avoir — une vie « difficile ». Pour certains auteurs, c'est le processus qui a une dimension méthodologique, pour d'autres, c'est la méthodologie qui a comme constituant un processus. [2]. (Il y a aussi ceux qui, comme J. Arlow, identifient méthodologie avec processus : « Unified Process *est une méthodologie* » [3]). Pour ne prendre parti ni pour une approche ni pour l'autre, nous avons choisi une association anonyme et bidirectionnelle, permettant ainsi au lecteur de donner une place centrale à la méthodologie ou au processus, selon sa vision du GL.

Mais terminons cette introduction avec une mise en garde : l'identification des MO avec les méthodes « lourdes et bureaucratiques » héritées des premiers pas du GL qui a été le cheval de bataille de *eXtreme Programming* et des développements agiles, appartient au passé. Actuellement *eXtreme Programming* et les méthodes de développement agile aussi sont considérés comme des MO. Un bon exemple de changement culturel à ce propos est donné

par le méta-modèle SPEM 2.0 (*Software & System Process Engineering Metamodel*) où l'on met en évidence que : « *L'objectif est de s'adapter à une vaste gamme de méthodes de développement, à des processus de styles différents, aux cultures, aux niveaux de formalisation, aux modèles de cycle de vie et aux communautés*. [Les concepts de SPEM] *sont idéaux pour la représentation des processus agiles et des approches par équipes autonomes*. » [4]

Honneur

Si les activités de réalisation du logiciel aspirent au titre de « génie », elles doivent pouvoir être maîtrisées pour qu'à la fin du projet un produit de qualité soit livré dans des temps et à des coûts acceptables. Mais, dans le domaine de la technique, une condition *sine qua non* de la « maîtrise », c'est l'emploi d'une MO qui, seule, rend la démarche reproductible et, idéalement, indépendante des individus.

Voici une liste partielle et non ordonnée d'éléments positifs qu'une MO peut apporter dans un projet :

- **Gestion de projet.**
    - *Préparation du plan de projet*. Une MO, avec son processus associé, facilite la création de plans de projets à l'aide d'un plan de projet abstrait qui sera instancié dans un contexte donné. Une entreprise pourrait, par exemple, avoir dans sa MO une procédure décrivant comment spécialiser le plan de projet abstrait de l'entreprise pour l'adapter au type de projet et ensuite l'instancier dans un projet donné.
    - *Suivi*. La documentation et les outils prévus par la MO, et l'enchaînement des activités du processus associé, en facilitant la visibilité de la progression du projet et de l'état des artefacts rendent le suivi plus aisé.
    - *Maîtrise*. La MO donne non seulement les points d'ancrage mais aussi les procédures qui facilitent les prises de décision, surtout quand il s'agit de mettre en exécution des plans de redressement suite à des échecs.
- **Humains**
    - *Intégration*. La MO crée un contexte de travail et un ordre qui facilite l'entrée du nouveau personnel dans l'entreprise et, surtout, facilite le passage d'un projet à un autre dans la même entreprise.
    - *Formation.* La structuration des méthodes et des procédures, l'existence d'une documentation et d'outils facilitant l'apprentissage guident le nouveau personnel en facilitant ainsi sa formation.
    - *Connaissance*. Les « canaux » officiels créés par la MO facilitent la diffusion des connaissances et la création d'une culture d'entreprise qui, à son tour, favorise la productivité.
- **Productivité**. Les éléments suivants contribuent à l'amélioration de la productivité :
    - *Réutilisation*.
        - *Des méthodes (et des composants des méthodes).* Si l'approche méthodologique n'est pas monolithique mais a une structure apparentée à celle de la figure 1, les méthodes et les composantes des méthodes peuvent être réutilisées pour créer une MO mieux adaptée au projet.

        - *Des artefacts*. La MO favorise la réutilisation en favorisant la normalisation. Voir le point suivant.
    - *Normalisation*. L'existence de normes (internationales, des constructeurs ou de l'entreprise) facilite :
        - La création d'artefacts à l'aide de gabarits partiellement remplis ;
        - La compréhension des documents, des modèles et des programmes rédigés dans des notations normalisées.
    - *Salaires.* L'organisation des tâches avec des artefacts bien définis favorise l'affectation des personnes selon leurs capacités et leur formation, ce qui module ainsi les salaires.
    - *Créativité.* La MO permet aux intervenants de se concentrer sur des activités plus proprement humaines (création, raisonnements analogiques) car « tout » ce qui est routine est pris en charge par des outils ou, à cause des « impositions » de la MO, est devenu un automatisme pour les individus.
- **Qualité** La qualité, l'un des concepts les plus fuyants du GL, peut être améliorée si la MO :
    - *Fixer l'approche à l'activité de mesurage*. En s'appuyant, surtout pour les entreprises « débutantes », sur des méthodes comme celles qui sont proposées dans la norme IEEE 1061. [5]
    - *Établir les règles permettant une amélioration continuelle du processus*. Amélioration qui dépend du niveau de définition des activités et des artefacts.

Au stade actuel de l'évolution technique, il n'existe pas de MO sans outils, sans procédures et sans notations. L'entrelacement entre les outils, les notations et les MO est si fort que l'on peut confondre la notation ou l'outil avec la MO. C'est pour cela que la mise en garde de Jim Arlow « *UML n'est pas une méthodologie* » [2] n'est, à notre avis, ni un indice de la crainte de l'ignorance des lecteurs ni un indice de l'ambiguïté du terme « méthodologie ». Il est évident que UML n'est pas une MO ! mais UML est si souvent associé à des méthodes de développement que bien des ingénieurs du logiciel ont besoin d'un rappel.

Pour nous libérer d'une critique facile, ajoutons que s'il vrai qu'il existe des projets qui n'ont pas besoin de MO — MO agiles ou lourdes, peu importe — il est aussi vrai que ces projets ne sont pas des projets en GL[2]. On ne demande pas à un gamin qui « bâtit » un monticule de sable sur une plage d'employer des MO, on le demande par contre à une firme d'ingénieurs qui doit concevoir et superviser la construction d'un gratte-ciel de 200 étages ! En raison de l'évolution des outils informatiques qui deviennent toujours plus puissants, bien des projets qui semblaient complexes il y a pas longtemps se transforment en création de « monticules »[3], mais cela ne doit pas nous entraîner à un refus enfantin des MO car de nouveaux problèmes « gratte-ciel » n'ont cesse de naître pour nous rappeler que la maîtrise du développement est loin d'être un jeu d'enfants.

Pour terminer cette défense, ajoutons que les MO situationnelles fondées sur des méta-modèles comme SPEM [4] ou Noesis [6] nous semblent être des réponses appropriées aux critiques classiques comme celles qui sont rapportées par Brian Fitzgerald [7].

---

[2] La non-existence de MO met automatiquement un projet hors du génie, même si cela ne le place ni hors de la science ni hors de la technique.

[3] Qu'il suffise de penser à l'évolution des outils pour construire et programmer les interfaces personne-machine.

Déshonneur

L'engouement pour les nouveautés qui est le propre des domaines nouveaux comme le GL fait souvent oublier que les mots ont besoin de prendre une certaine assiette pour pouvoir être employés efficacement dans les interventions sur la réalité — comme le GL est censé le faire. Le foisonnement de nouvelles disciplines, approches, méthodologies, processus, cadres, patrons, notations, normes… qui manquent souvent du minimum de rigueur nécessaire au travail technique aurait besoin, sinon d'un frein, au moins d'une approche plus minimaliste. Dans le cadre des MO, il faudrait sans doute s'arrêter à la position de ISO 12207 : « [En GL] *il y a une prolifération de normes, de procédures, de méthodes, d'outils et d'environnements pour développer et gérer le logiciel. Cette prolifération a rendu l'ingénierie et la gestion du logiciel difficiles, surtout lors de l'intégration de produits et services. Le GL*[4] *doit migrer de cette prolifération vers un cadre commun qui peut être employé par les praticiens pour "parler la même langue" pour créer et gérer le logiciel. Cette norme internationale fournit un tel cadre commun.* » Les ingénieurs du logiciel, dont les matériaux de base sont les mots et les dessins, devraient non seulement s'aligner sur le réalisme et la simplicité de la position ISO présentée dans la norme ISO 12207, mais aussi redouter toute MO qui se présente comme *La* solution à la prétendue crise du GL et insister sur le fait qu'il faut, avant tout, « *parler la même langue* ». Cette solution d'ISO laisse ouvertes toutes les portes pour générer des méthodologies *ad hoc* qui n'aspirent pas à devenir des MO généralisables. Et en reprenant la célèbre expression que Feyerabend appliquait à la physique, on peut dire que du point de vue des MO « *tout est bon* ». « Tout est bon », dans le sens que, lorsqu'on fait autre chose que répéter ce qui a déjà été fait dans le même contexte avec les mêmes intervenants, ce n'est qu'après coup que l'on sait ce qui est bon.

Parler la même langue ? Cela ne signifie certes pas assécher la langue comme le fait Ron Burback dans une tentative de formalisation des MO : « *Une méthodologie est un algorithme qui trouve une solution faisable dans un environnement donné d'un espace multi-strates constitué des plans d'analyse, conception, mise en œuvre et test, qui commence avec la racine représentée par l'énoncé du problème et s'achève avec l'objectif représenté par le test d'acceptation du système.* […] *Si une solution existe, la performance de la méthodologie de génie logiciel est définie comme le nombre de pas nécessaires pour trouver une solution faisable* ». [6] Il ne s'agit que d'une thèse de doctorat, il est vrai, mais c'est bien dans la naïveté des étudiants que l'on peut mieux voir les faiblesses des conceptualisations des experts. Cette formalisation qui naît du désir — compréhensible et précieux — de mettre de l'ordre dans le fatras terminologique existant est une solution pire que le mal car elle risque de nous faire jeter le bébé (de la rigueur) avec l'eau sale du bain.

Parler la même langue n'implique pas employer un terme dans tous les contextes en diluant ainsi sa signification et en le rendant pratiquement inutile. Quand on jette un coup d'œil sur toutes les dimensions de « méthodologie/méthode » on est en droit de se demander si le terme garde une signification quelconque au-delà de démarche — ce qui est, pour le moins, très générique et fort peu utile. Qu'est-ce que la partie commune à « méthodologie » dans les syntagmes suivants : MO objet, de conception, de mesure de la qualité, de raffinement, agile, avec l'accent mis sur l'architecture, orientée utilisateur, de test des *spreadsheets*, adaptée aux situations, pour les architectures orientées services, etc. Ces syntagmes font référence à différentes dimensions : étendue par rapport au cycle de vie (conception), notations (objet), gestion de projet (agile), point de vue (accent mis sur l'architecture)… mais les dimensions souvent se superposent, empêchant toute caractérisation systématique ou en ajoutant un si grand nombre de détails que l'on est en droit de se demander si on n'est pas en train de cacher

[4] D'une manière assez cavalière, j'ai traduit « software discipline » par « génie logiciel ».

qu'il s'agit de méthodes *ad hoc*. Mais des méthodes *ad hoc* qui ont tous les défauts majeurs des MO : les détails impliquent une bureaucratisation toujours plus poussée et d'énormes difficultés de coordination. Par exemple : à l'intérieur d'une MO orientée objet on peut avoir des MO de test objet et, à l'intérieur de celles-ci, des MO spécifiques comme celle qui est proposée par Greg Rothermel pour le test des *spreadsheets* [9]. Mais pourquoi s'arrêter aux *spreadsheets* ? Pourquoi pas une MO de test pour les *spreadsheets* à douze colonnes pour le Maghreb ? Si l'on ne veut pas inventer de nouveaux termes qui risqueraient de rester des hapax, il faudrait non seulement expliciter longuement le contexte mais renoncer à la méthodologie telle que représentée dans la figure 1. et ne considérer que les composants des méthodes. Ce qui nous amène à considérer des MO flexibles ou situationnelles fondées sur des méta-modèles.

Mais cette solution est loin d'être intéressante. La flexibilité, qui est censée régler bien des problèmes, en introduit malheureusement d'autres qui sont encore plus difficiles à régler. C'est pour régler ce deuxième type de problème que l'on a créé le domaine de l'ingénierie des méthodes, ingénierie qui est censée maîtriser la production et la gestion des MO. Voilà donc des méta-MO [4] [6] qui guident les ingénieurs des méthodes qui eux, cela va de soi, ont besoin de méta-méta-MO pour définir les méta-MO… et la chaîne peut continuer *ad libitum*. Mais, même si on s'arrête à un seul « méta », il suffit de considérer des normes comme SPEM 2.0 pour se rendre compte que si de telles normes permettent de bâtir des MO agiles, elles sont tout sauf agiles. Et si SPEM 1.1 n'a pas eu de succès comme on l'a écrit dans la version 2.0 de la norme (« *la sémantique était ambiguë et difficile à comprendre par ceux qui devaient adopter la norme et donc elle n'a pas été employée* »), nous croyons que ce n'est pas tellement l'ambiguïté qui était en jeu mais la complexité ; complexité qui est toujours présente dans la version 2.0. Et ce n'est pas parce que l'on applique aux MO les épithètes thaumaturgiques qui, aujourd'hui (2007), ouvrent toutes les portes du GL (« *agile, itératif, centré sur l'architecture, piloté par les risques et la qualité* » [4]) que l'on est absolument sûr que, derrière la porte, il y ait de quoi améliorer les méthodes de développement.

La physique, cette science qui est notre mère à tous, que nous appelons à l'aide quand nous voulons nous montrer « scientifiques », nous enseigne très peu de choses à propos des méthodologies au-delà du fait qu'il convient d'adopter une démarche rationnelle qui s'appuie sur les mathématiques. Einstein n'aurait jamais écrit son célèbre article en 1905 et le principe d'indétermination d'Heisenberg aurait été jeté dans la poubelle des sciences si les deux physiciens avaient suivi des MO. Les ingénieurs des méthodologies en GL nous font penser aux épistémologues qui, après coup, établissent les règles qui gouvernent la progression de la physique. Si les physiciens suivaient les indications des épistémologues — des Popper, des Lakatos, des Kuhn… peu importe — dans leur travail de « résolution de problèmes », ils passeraient leur temps à penser et à s'adapter aux méthodes au lieu de « faire avancer la science ». Les Popper des MO devraient se contenter de définir des MO *ad usum collegae,* sans essayer de les imposer aux ingénieurs du logiciel qui, si vraiment ils y sont obligés, à la fin du développement, modifierons la description de leur démarche pour l'adapter à la MO imposée.

Terminons cette « attaque » contre les MO avec une considération sur l'une des méthodologies les mieux vendues : UP (*Unified Process*). « *UP est une méthodologie* » [3], donc une **méthodologie est un processus** et puisque « *un processus du GL définit* qui, quoi, quand, comment *on développe le logiciel* » [3], une MO définit tout : les rôles des intervenants dans le projet (*qui*), les artefacts et les activités (*quoi*), l'enchaînement des activités et des tâches (*quand*), la manière de procéder et les outils (*comment*). Considérons l'une des forces de UP et de bien des processus modernes qui ont pris la relève de l'approche en cascade : l'itération. Le principe de l'itération est un principe très simple (pour les

ingénieurs du logiciel) car il s'agit du même principe qui sous-tend la modularité des programmes : en raison de la structure cognitive des humains, la solution d'un problème est plus facile si le problème est divisé en sous-problèmes qui, si la division est bien réalisée[5], sont moins complexes. Le principe est tellement bon qu'il n'y a pas d'être vivant (et pas seulement les humains !) qui ne l'applique pas quotidiennement. Mais le fait qu'il s'agisse d'un bon principe n'implique pas que l'on puisse l'appliquer dans la technique sans être guidés par d'autres principes moins facilement communicables. Aucun ingénieur du logiciel, par exemple, n'aurait jamais la tentation de réduire le problème en réduisant les modules à une seule instruction. Mais où s'arrêter dans la division ? Il faut des règles souvent difficilement formulables car elles dépendent de trop de paramètres (performances, facilité d'entretien, interopérabilité, fiabilité, etc.). Pour l'itération, c'est le même problème. Où s'arrêter dans la division ? Généralement, on parle d'itérations d'une semaine. D'où vient-elle cette semaine ? Sans doute du fait que, dans notre organisation du travail, la semaine est une unité de comptage très importante avec des séparations assez longues (deux jours) pour permettre de changer de contexte (ou travailler la fin semaine si le projet est en retard). Mais la semaine a l'air encore plus stupide que « une page = un module » proposé par IBM dans les années 1970. La longueur dépend du type de projet, des exigences, du domaine, de l'équipe, etc. c'est-à-dire que la durée de l'itération est tellement variable, qu'elle devrait être un paramètre de la MO. La durée des itérations, dans certains projets, pourrait être égale à la durée du projet, ce qui nous donnerait un projet avec une seule itération, donc… sans itérations ! Drôle de MO que cet UP fondé sur les itérations qui pourraient ne pas exister.

Au-delà

La MO *eXtreme Programming*[6] permet de souligner deux excès qui guettent les MO :

- L'excès de documentation et de planification à long terme qui met au centre les méthodes au lieu du produit. C'est l'excès que *eXtreme Programming* a placé sur la sellette et qui a favorisé sa diffusion.
- L'excès de confiance, de bonnes intentions et d'optimisme dans le dialogues qui est au fondement du *eXtreme Programming* : « *Sans doute que tout ce dont nous avions besoin* [par rapport à la lourdeur des vieilles MO] *était d'apprendre à se parler* [les clients avec les développeurs et les développeurs entre eux]. » [12]

Excès n'implique pas que dans certains projets il ne soit pas correct d'être dans l'excès, mais que la majorité des projets sont quelque part au milieu. De quels outils et de quelles connaissances doit être doué un chef de projet qui cherche à placer son projet entre ces deux extrêmes ? Il n'y a pas de réponse simple mais, dans la situation actuelle, il nous semble que les MO situationnelles[7] ou compositionnelles soient les plus prometteuses [6] [4]. Mais elles aussi ne sont pas sans danger. Ces MO, censées s'adapter aux besoins du projet, pourraient engendrer un refus de la part des praticiens si elles sont trop complexes à mettre en place à cause du trop grand nombre de paramètres en jeu. Ce risque est loin d'être sans importance, car la volonté d'optimiser les MO dans un cadre d'ingénierie des MO pourrait remettre les MO au premier plan en faisant passer le produit au second. S'il n'y a pas de partage des responsabilités entre tous les participants, les développeurs pourraient considérer les ingénieurs des MO et les qualiticiens, dans le meilleur des cas comme du bois mort et dans le pire comme des sbires.

---

[5] Et l'on sait si elle est « bien » réalisée seulement si les sous-problèmes sont moins complexes !

[6] Comme le souligne R. C. Martin *eXtreme programming* n'est pas du codage sauvage mais « une méthode de développement » où les humains sont à l'avant plan [12].

[7] Terme qui, même s'il est presque synonyme de *ad hoc*, a des implications que ce dernier n'a pas. Le fait que la situation influence la méthode ne veut pas dire que l'on définisse la méthode après coup.

Limiter la portée du GL en le rendant plus simple nous semble une manière de diminuer ce danger. Le limiter et lui « enlever » ce qu'il a absorbé à cause des vicissitudes historiques mais qui lui est étranger, et du point de vue méthodologique et du point de vue des connaissances. Les exigences qui ont déjà un domaine à elles (le génie des exigences) pourraient être la première partie à séparer. [10] Cette séparation n'implique bien sûr pas qu'il faille spécifier complètement les exigences avant de concevoir et de coder mais que les connaissances, les outils et les notations des exigences ne sont pas nécessairement ceux du GL[8]. Proposer une telle séparation indique clairement que nous sommes du côté de ceux qui considèrent « *l'idée de développement sans coutures* (seamless) [comme] *une indication que la description du problème et l'analyse ont reçu trop peu d'attention* » [11], et qui croient que des MO pour les exigences seraient un moyen d'« accorder plus d'attention au problème et à l'analyse ». Mais, bien que les MO du génie des exigences et celles du GL ne puissent qu'être très différentes, pour que les objectifs du projet soient satisfaits, il faut qu'elles aient en commun des frontières clairement définies sans être rigides pour faciliter l'échange d'artefacts et le dialogue entre les parties prenantes. Ce qui est loin d'être facile.

Une avenue possible serait de trouver les discriminateurs principaux permettant d'organiser les MO dans une hiérarchie de concepts pour faciliter la composition. La recherche devrait être faite sans crainte de mettre en doute certains discriminateurs « classiques » comme : approche objet, dimensions du projet, durée de vie du produit, criticité, connaissances du personnel, la culture de l'entreprise, l'agilité, etc. Mise en doute qui n'implique pas le refus mais qui peut permettre à d'autres dimensions éventuellement mises à l'écart dans la littérature de faire (ou refaire) surface.

Informatiser un domaine implique, avant tout, l'établissement d'une taxinomie et des règles que le logiciel doit respecter. L'étendue, la complexité et la précision de ces règles sont des facteurs qui influencent énormément la manière de développer le logiciel, les outils et les notations employés. Si le domaine possède des machines avec lesquelles le logiciel doit dialoguer, les interfaces de ces machines ont un grand impact sur les règles, surtout sur leur précision. À titre d'exemple : dans une application pour gérer un contrôleur de disque, toutes les règles sont définies dans les spécifications[9] du contrôleur et donc on ne court aucun risque (en principe et du moins du point de vue méthodologique) dans le développement. La présence de machines dans un domaine peut être vue comme l'opération de « fermeture » de la signification des termes du domaine. Pour dénoter cette fermeture, nous proposons le syntagme *cadenassage sémantique* : tout se passe comme si les machines posaient un cadenas sur la signification en empêchant les nuances d'entrer/sortir. Pour mieux préciser le concept de cadenassage sémantique on peut l'opposer à l'appauvrissement sémantique opéré par une description en langue naturelle des exigences du domaine. L'appauvrissement sémantique est une condition utile, mais ni nécessaire ni suffisante, pour le cadenassage sémantique. À notre avis, non seulement le cadenassage sémantique est une dimension qui a une grande influence sur les MO, mais il pourrait être le discriminateur principal. Un protocole HDCL qui s'appuie sur des services prédéfinis est un exemple de cadenassage complet et le développement pourrait (devrait) être réalisé avec des MO formelles.

Même si nous sommes conscients que nos propositions pour aller au-delà d'une opposition stérile entre MO « lourdes » et « agiles » sont loin d'être mûres, nous croyons que leur approfondissement pourrait apporter quelques éléments positifs à notre discipline qui n'a de

[8] Malgré les prétentions de UML et des MO comme UP.

[9] En première approximation, nous pouvons supposer que les spécifications sont non ambiguës. Du point de vue des MO, cette approximation est dangereuse, car la MO doit tenir compte de la moindre ambiguïté dans les spécifications (ou de la possible mauvaise compréhension des lecteurs).

cesse de parler de crise et d'essayer de régler la crise avec les moyens qui l'ont, depuis les débuts, favorisée.

Références

[1] ISO, *ISO/IEC 12207-1995 Information technology – Software Life Cycle Process*.

[2] Graham Ian et *alii*, *The Open Process Specification*, Addison-Wesley, 1997.

[3] Jim Arlow, Ila Neustadt, *UML 2 and the Unified Process*, Addison-Wesley, 2005.

[4] OMG, *SPEM 2.0 RFP ad/2004-11-04 : 4th Revised Submission*, Internal version 1.7, September 2006.

[5] IEEE, IEEE Std 1061 1998, *Standard for a Software Quality Metric Methodology.*

[6] E. Dominguez, M. A. Zapata, Noesis: Toward a Situational Method Engineering Technique, www.elsevier.com/locate/infosys (visité le 19 février 2007).

[7] Brian Fitzgerald, « Formalized Systems Development Methodologies: a Critical Perspective », *Information System Journal*, January 1996

[8] Ron Burback, *Software Engineering Methodology: the Watersluice*, Dissertation, Stanford University, 1998.

[9] Greg Rothermel *et alii*, « A Methodology for Testing Spreadsheets », *ACM Transaction on Software Engineering and Methodology*, Vol. 10, no. 1, January 2001

[10] Ivan Maffezzini *et alii*, « Prolégomènes à une critique du GL : partie 1 contextualisation », *Génie Logiciel*, septembre 2003, numéro 66.

[11] Michael Jackson, *Problem Frame*, Addison-Wesley, 2001.

[12] R. C. Martin « eXtreme Programming Development through Dialog », *IEEE Software*, July/August 2000.