De la nécessité des modèles

> *We should reconcile ourselves with the fact that we are confronted, not with one concept, but with several different concepts which are denoted by one word; we should try to make these concepts as clear as possible to avoid further confusions, we should agree to use different terms for different concepts; and then we may proceed to a quiet and systematic study of all the concepts involved*. (Alfred Tarski. Cité en http://www.arts.mcgill.ca/philo/speaks/415/Tarski.html#x1-190004)

1 Introduction

Le jour où l'on planta le mot « modèle » dans le champ, passablement inculte, du génie logiciel (GL) il eût fallu avoir une imagination fort débridée pour deviner que, quelques saisons plus tard, ce mot eût fleuri jusque dans les parcelles les plus rocailleuses du domaine. Qui aurait pu soupçonner qu'en l'espace de quelques années, il eût supplanté le mot « module » dans la majorité des articles, livres et conférences proposant des remèdes aux maux de notre discipline[1] ? Très peu de gens. Certes, l'importance accrue des phases initiales du développement demande l'abandon de « module », terme trop lié à la programmation structurée, en faveur de termes plus généraux. Mais, ne sommes-nous pas tombés dans l'excès opposé ? Ce qui est certain, c'est que « modèle » a toutes les qualités requise pour s'imposer comme mot magique et donner ainsi naissance au lieu commun qui veut qu'il n'y a pas de développement sérieux sans modélisation.

Il est au moins quatre raisons qui favorisent la diffusion d'un mot ou d'une expression :

1. Le mot est très générique et il est employé à la place d'autres termes plus précis. Le verbe « faire » et le substantif « chose » sont deux exemples extrêmes empruntés au français courant.
2. Le mot est doté d'une grande polysémie (des significations différentes mais assez précises) qui lui permet d'être employé dans des contextes très différents. L'entrée « Modèle », pour s'en tenir à notre propos, dans le grand Robert, a neuf acceptions différentes dont trois seulement dans des domaines spécifiques.
3. Le mot répond à un besoin ponctuel très fort. En GL, par exemple, l'expression « contrôle de la configuration » répond ou a répondu à un besoin fort et répandu.
4. Le mot est très publicisé par ceux qui ont le pouvoir d'influencer les utilisateurs. On arrive à des cas extrêmes où le nom d'une marque devient un nom commun comme le très connu « Frigidaire ».

Au-delà des qualités qui favorisaient son adoption dans les nouvelles approches du développement en GL, nous ne sachions pas que le succès de « modèle » est attribuable aussi aux quatre causes précédentes. Nous ne trouverons pas d'opposition si nous réglons les deux dernières cavalièrement en disant que le succès commercial de UML y est pour quelque chose et que le besoin de décrire les systèmes et les domaines au moyen de langues spéciales était présent dès les débuts du GL.

---

[1] SWEBOK, en tant que « *guide pour le corpus des connaissance en génie logiciel* » est un bon terrain pour vérifier les succès des mots dans la discipline. Dans les 202 pages de la version 2004, « modèle » et ses dérivés apparaissent 295 fois, tandis que « module » et ses dérivés n'apparaissent que 19 fois.

Mais ce sont la généricité et la polysémie qui, dans un contexte d'analyse des lieux communs, nous semblent les causes les plus intéressantes,.

Souvent, les termes du langage commun, quand ils sont transportés dans un domaine technique, perdent une partie de leur polysémie et de leur généricité et assument une signification très spécifique. Cette restriction du champ sémantique est ce qui permet de plus ou moins les désambiguïser — ce qui les rend souvent incompréhensibles aux non spécialistes. Dans le cas de « modèle », le passage du langage commun au langage technique du GL n'a pas eu cet effet. Il y a eu pratiquement l'effet inverse : aux significations du langage commun (*abstraction, élément de référence, prototype, représentant-type d'une catégorie, représentation schématisées,* etc.) on a ajouté celle de la logique mathématiques et de la linguistique (*structure dans laquelle tout énoncé d'une théorie est vrai, construction abstraite axiomatisée*). Cet élargissement de la signification rend très difficile l'emploi du terme dans un domaine technique comme le GL où l'exigence de « maîtrise » de développement est loin d'être satisfaite.

Que faire ? Jeter « modèle » avec tous ses signifiés ? Sans doute pas. Son énorme succès ne justifie pas que, par une prise de position puriste ou dogmatique, on le rejette complètement. Le proverbe « on ne jette pas le bébé avec l'eau du bain », nous semble s'appliquer parfaitement à notre « modèle » qui, une fois bien lavé, mérite sans doute d'être conservé.

2 Honneur

La manière la plus efficace — selon certains, la seule, et je fais partie de ceux-là — d'appréhender la réalité pour la contrôler et éventuellement la modifier, c'est de bâtir des modèles. Les modèles, en faisant abstraction des détails du réel qui sont nuisibles à l'intellection, favorisent la compréhension des concepts et de leurs associations et donc[2] la compréhension de la réalité. Mais, qu'est-ce qu'un modèle ? Étant donné le succès de la « langue[3] de modélisation » UML, nous allons prendre la définition donnée dans son Manuel d'utilisation [1] : « *Une simplification de la réalité, créée pour mieux comprendre le système à créer ; une abstraction sémantiquement close d'un système.* » Même si les deux définitions sont séparées par un simple « ; », elles sont tellement différentes qu'il vaut la peine de séparer le concept de modèle en deux. Nous appellerons ModèleSR (SR pour **S**implification de la Réalité) le modèle qui satisfait la première définition du Manuel de UML et ModèleAS (AS pour Abstraction du Système) celui qui satisfait la seconde.

2.1 ModèleSR

Reprenons la définition et analysons-la en détails :

1. il y a un système à réaliser (c'est le but) ;
2. il faut essayer de comprendre le mieux possible le système que nous voulons créer (c'est la condition pour réaliser le système) ;

[2] Ce « donc » qui aimerait sans doute passer inaperçu est trop chargé d'histoire et lourd de conséquences pour que nous n'ajoutions pas au moins cette note.

[3] Nous avons assez d'intérêt pour la langue française pour ne pas appeler UML un langage et sans doute pas assez de courage pour laisser tomber langue aussi et parler de « notation UML ».

3. pour mieux comprendre le système, il faut simplifier la réalité (c'est le moyen).

Il est pratiquement impossible de ne pas être d'accord avec l'esprit de la définition, mais nous croyons qu'il faut la changer légèrement pour mieux saisir l'importance de ModèleSR. Nous dirons donc que :

1. *Le but est de construire une machine*. Le passage de « système » à « machine » permet d'éviter certaines ambiguïtés liées au terme « système ». En tant qu'ingénieurs, nous construisons des machines qui s'intègrent dans un système pouvant contenir autres choses que des machines. Un programme est, bien sûr, lui aussi une machine.
2. *La condition est, avant tout, de comprendre la réalité*. Pour réaliser le système, il faut comprendre la réalité dans laquelle la machine s'installe et les modifications qu'elle est censée y apporter. La modification que nous proposons — changer « système » par « réalité »[4] — a un certain nombre d'impacts sur l'approche au développement.
3. *Le moyen est de simplifier la réalité*[5]. Ici, nous ne faisons aucune modification à la formulation originale. Mais il n'est sans doute pas inutile d'ajouter que la réalité comprend aussi les besoins et les exigences qui poussent à la modifier et qu'il est très important de ne pas confondre les besoins de changements avec les contraintes que la réalité (le domaine) impose sur ces changements mêmes.

Au terme de cette brève analyse, nous proposons la définition suivante : « *Une simplification de la réalité, créée pour mieux la comprendre et pouvoir ainsi y insérer des machines.* »

Selon cette définition, ModèlesSR est donc un instrument permettant aux humains de voir et de communiquer ce qui est important dans la partie de la réalité que l'on veut automatiser. Mais, ce qui est important n'étant pas nécessairement donné à priori, ModèlesSR doit aussi aider à comprendre ce qui est important et cela ne peut être réalisé qu'en explicitant certains concepts qui sont normalement implicites[6]. L'opération d'explicitation est réalisée dans un premier temps en décrivant en une langue naturelle et, dans un deuxième temps, en une langue de « modélisation ». Le simple fait de traduire d'une langue à l'autre oblige les intervenants à éclaircir les concepts, souvent en les multipliant. Cette multiplication, et donc cette complexification du ModèlesSR, est la condition, nécessaire mais non suffisante, à un éclaircissement et donc une simplification de la réalité.

ModèleSR est un artefact inséré dans la réalité du projet et il a une vie indépendante de la réalité qu'il décrit même si, en théorie, il devrait toujours la refléter. Par exemple : si, dans la réalité où l'on veut insérer la machine, il n'y a plus de modem

[4]Dans un article du numéro 73 de cette revue, par exemple, on écrit : « *Dans le domaine de la modélisation, le terme système est utilisé pour faire référence à la réalité* ». Ce qui est un constat d'une « réalité » (sic !) factuelle, mais aussi signe d'une grande confusion.

[5] Nous ne sommes pas satisfaits du terme « réalité » non plus, mais la justification du changement de ce terme pour un autre déborderait du thème de cet article.

[6] Ce qui ne veut pas dire que tout puisse être explicité. Surtout, que veut dire « tout », dans une approche d'automatisation de la réalité avec les lois qui structurent les domaines et les humains qui aimeraient plier le domaine à leurs exigences ?

parce que tout est numérisé, le modèleSR qui décrivait, que sais-je ? *détection de la porteuse* doit être modifié pour refléter la nouvelle réalité. Doit être : c'est-à-dire l'humain doit intervenir sur le modèle s'il ne veut pas que le modèle ne soit pas inutile ou même nuisible.

NOTE : « Simplification de la réalité » ne veut pas dire, comme on l'entend souvent, « ignorer les détails », mais pouvoir établir quels éléments considérés comme des détails ne le sont pas et quels éléments jugés fondamentaux sont en effet des détails. Cela parce que les détails ne sont pas donnés à priori mais sont « créés » dans le processus même de modélisation SR en fonctions des buts visés. Une approche erronée en GL, qui a souvent des conséquences catastrophiques, accepte les détails qui ne sont des détails que parce que la réalité n'a pas été analysée assez attentivement, et vice versa. FIN DE LA NOTE

2.2 ModèleAS

La définition du Manuel de UML de ModèleAS est, d'une part, moins problématique que celle de ModèleSR, car elle ne contient ni buts ni moyens et, au lieu de faire référence à réalité et à système, elle ne fait référence qu'à système. D'autre part, elle est plus problématique car l'expression « sémantiquement close » est loin d'être claire. Pour restreindre le champ sémantique de ModèlesAS que le mot « système » contribue à élargir, nous introduisons le terme « machine ». Nous avons ainsi : « Un ModèleAS est une représentation abstraite d'une machine ». Avoir coupé « sémantiquement close » selon nous n'est pas important. Voir dans l'encadré *abstraction sémantiquement close* la justification de cette coupure.

La nécessité d'avoir un modèle de la machine ne nous semble pas poser de problème, une difficulté éventuelle réside dans le fait de définir combien de modèlesAS il faut avoir pour faciliter sa construction. Sans oublier que, parmi ces modèles, il y a le programme lui-même qui « *est prévu pour modéliser le comportement de quelque chose* » [2].

3 Déshonneur

Puisqu'en GL tout est modèle, tout le monde modélise.

Voici une liste — loin d'être exhaustive — d'expressions en GL contenant le terme modèle : *modèle* logique, développement dirigé par les *modèles*, *modèle* objet, *modèle* conceptuel, *modèle* de données, *modèle* physique, *modèle* mathématique, *modèle* du système, *modèle* d'ingénierie, *modèle* environnemental, *modèle* abstrait, méta *modèle* ontologique, *modèle* des processus, méta *modèle* linguistique, *modèle* de qualité, *modèle* sémantique, *modèle* utilisateur, *modèle* du programmeur, *modèle* formel, *modèle* dynamique, *modèle* statique, *modèle* du domaine, *modèle* de développement, *modèle* du cycle de vie, *modèles* visuels.

Nous nous sommes déjà demandé : « *Est-ce que le terme* modèle *garde une signification stable en changeant de syntagme ou est-ce que le terme qui l'accompagne en bouleverse la signification* ? » [3]. Et sur la réponse nous n'avons pas de doutes : dans plusieurs cas, la signification change et c'est à cause de tous ces changements que « modéliser » est devenu un mot fourre-tout qui nous dupe tous — ou presque.

3.1 Développement leste (*agile development*)

La critique de la modélisation à partir de la généricité et de la polysémie de « modèle » n'est pas la seule possible. Bien des gens considèrent que les modèles — et une grande partie de la documentation — ne sont pas utiles dans le développement du logiciel. À titre d'exemple, considérerons ce qu'écrit A. Cockburn : « *Caractériser le développement logiciel* […] *comme construction de modèles conduit directement à des mauvaises décisions de projet.* […] *Les produits du travail d'une équipe doivent être mesurés par rapport à leur* capacité à communiquer avec le groupe ciblé. *Il n'est pas important que les modèles soient incomplets, représentés avec une syntaxe incorrecte, et en fait non comme le monde réel* s'*ils communiquent suffisamment avec les destinataires*. » [4].

Ce qui est surtout important pour les tenants du développement leste comme A. Cockburn et pour ceux de la programmation extrême, c'est la communication dans l'équipe, les échanges informels entre les programmeurs, en partant du code qu'ils ont — ou qu'ils pourraient avoir — devant leurs yeux. Cette position, qui semble aller à l'encontre de toute bonne pratique de GL, va chercher des assises théoriques [5] qu'il est difficile de réfuter par des arguments trop simples. Nous pourrions résumer, et simplifier, le raisonnement qui se trouve à la base de ce type d'anti-modélisation inspiré par [5] comme suit : « La documentation, sans les gens qui ont participé au développement du produit, n'est pratiquement pas utile parce qu'il est impossible de décrire les particularités qui permettent de faire évoluer un produit. Les ingénieurs du logiciel ont dans la tête des descriptions, des perceptions, même des *impressions*, qui ne pourront jamais être mises sur papier mais qui sont fondamentales pour pouvoir intervenir efficacement sur le produit. Lors de l'échange entre les membres de l'équipe, les modèles se forment dans les têtes et ce sont ces modèles qui permettent d'entretenir et de faire évoluer le produit et non les modèles sur papier. Quand il y a des interventions à faire sur un produit dont l'équipe de développement s'est dissoute, même s'il existe des modèles sur papier, il est préférable de refaire complètement le programme ».

3.2 Descriptions et modèles

La critique selon nous la plus valable et la plus intéressante pour améliorer les pratiques dans le GL est celle de Michael Jackson. M. Jackson ne conteste pas l'utilité des modèles en soi, mais il pense qu'à cause des emplois courants en GL, « modéliser » est devenu un terme creux et donc : « *Nous ne parlerons jamais d'activité de modélisation dans le développement, ni de modélisation du problème, ni d'une partie du monde, ni de la machine. C'est simplement trop déroutant*. » [6] Trop déroutant parce que l'on confond deux genres de modèles : les modèles analytiques et les modèles analogiques.

Dans la figure suivante, nous avons abordé le premier niveau de classification des modèles selon UML et selon M. Jackson. Dans le cas de UML, le discriminateur est la machine (le système dans la terminologie UML) ; pour M. Jackson, le discriminateur, c'est la manière de représenter : analytique ou analogique. Un modèle analytique est une description du domaine, de la machine ou de leur interaction. Comme exemple de modèle analytique emprunté à la physique, on peut considérer la description probabiliste de la position d'un électron avec l'équation de Schrödinger. Un modèle analogique « *n'est pas une description mais une autre réalité avec certaines propriétés semblables* » [6]. Toujours en nous inspirant de la physique, on peut penser au modèle atomique de Bohr.

La différence entre les deux modèles est si grande que… que la confusion règne. Un modèle analytique est un modèle plus précis mais moins expressif qu'un modèle analogique. La plus grande expressivité du modèle analogique dérive du fait que la compréhension (telle qu'on la considère dans le langage courant) est fondée sur l'analogie ou la transduction[7].

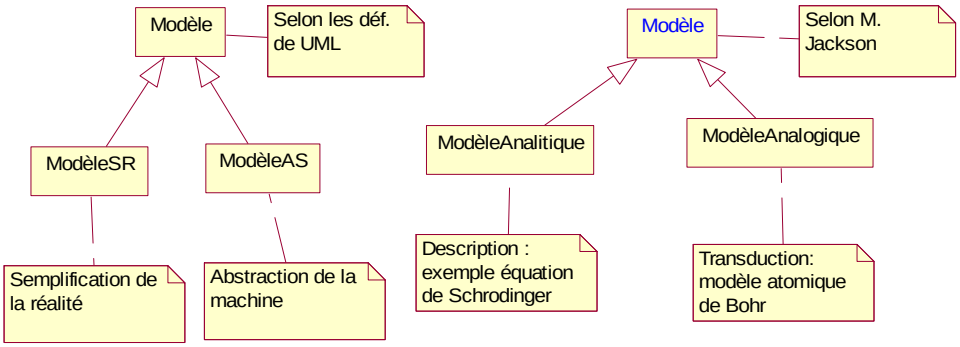


Fig. 1 : catégorisation des modèles selon UML et selon M. Jackson

Il nous semble que cette simple différence de classification (de modélisation ?) a des implications théoriques et pratiques énormes. Vaut-il la peine d'insister auprès des ingénieurs du logiciel (tenants des modèles ou de la programmation extrême, peu importe) sur le fait que le choix du premier discriminateur dans une classification a une influence énorme sur la manière de voir un concept ? Certainement pas.

4 Au-delà

Le première conclusion que l'on peut tirer de nos considérations est contenue dans la phrase de A. Tarski citée en exergue : « *We should agree to use different terms for different concepts* ». Il faudrait introduire plusieurs signifiants à la place de « modèle » et on pourrait commencer par deux signifiants comme le fait M. Jackson.

Le modèle analytique pourrait être le modèle qui est à l'origine d'un développement dirigé par les modèles, tandis que le modèle analogique serait un modèle de documentation dont le but principal serait de faciliter la compréhension. Cette première division est utile surtout pour la formation des nouveaux ingénieurs du logiciel qui n'ont surtout pas besoin d'une terminologie confuse au début de leur carrière. Voir à ce propos l'encadré *Marc est un homme*.

Est-ce que les tenants du développement leste pourraient refuser les deux types de modèles ? Certainement pas. Même les plus extrémistes ne pourraient pas refuser le « modèle décrit dans une langue de programmation ».

Voici quelques questions pour préparer le terrain pour aller au-delà.

Et si en GL on ne pouvait pas établir de règles générales dans les méthodologies de développement ? Et si le développement leste avait sa place et le développement « lourd » du GL traditionnel la sienne ? Et si l'influence du domaine est telle que la

[7] Que les petits des humains acquièrent le raisonnement inductif et déductif longtemps après le raisonnement transductif indique que ce dernier est plus « naturel ».

méthodologie, au moins dans la première partie du cycle de vie, devrait être dictée par le domaine où la machine doit être installée plutôt que par les exigences propres à la discipline du GL ? Et si la description du problème, du domaine et des exigences devait être faite hors du GL ? Et si, au risque de tomber en contradiction avec nous-même, il ne fallait pas parler d'ingénierie des exigences, ni d'ingénierie des modèles, ni d'ingénierie de la facilité d'utilisation ?

5 Références

[1] Booch G., Rumbaugh J., Jacobson I., *The Unified Modeling Language User Guide*, Addison-Wesley 199.

[2] Stoy J. E., *Denotational Semantics*, MIT Press, 1985.

[3] Maffezzini I., Kerhervé B. *Modèles légués*, RIFV' 05

[4] Cockburn A., *Agile Software Development*, Addison-Wesley, 2002

[5] Naur P. "Programming as Theory Building", pp. 37-48, *Computing: A human activity*, ACM Press 1992.

[6] Jackson M., *Problem Frames*, Addison-Wesley, 2001.

Encadré 1

Abstraction sémantiquement close

« *Sémantiquement close* », dans la deuxième définition de « modèle » de UML, nous semble créer plus de problèmes qu'elle n'en résout. Le fait que l'on retrouve cette définition dans des dizaines de sites web mais que nous n'avons trouvé nulle part une explication de « sémantiquement close » semble indiquer que :

1. la signification est tellement précise et connue qu'il ne vaut pas la peine d'y réfléchir ;
2. la signification n'est pas claire mais, dans le GL, on ne ressent pas le besoin de l'expliquer étant donné que, indépendamment de la signification du mot « modèle », les modèles UML « fonctionnent ».

Signification précise

Pour que la signification soit précise et connue il faut au moins qu'il existe une définition « standard » que la majorité des ingénieurs du logiciel accepte. La seule définition de « sémantiquement clos » que nous avons trouvée est celle de Tarski dans *The semantic Conception of Truth and the Foundation of Semantics* : « We have implicitly assumed that the language in which the antinomy is constructed contains, in addition to its expressions, also the names of these expressions, as well as semantic terms such as the term "true" referring to sentences of this language; we have also assumed that all sentences which determine the adequate usage of this term can be asserted in the language. A language with these properties will be called "*semantically closed*." » (http://www.ditext.com/tarski/tarski-c.html).

Toujours dans le même article, Tarski écrit que : « *The languages (either the formalized languages or — what is more frequently the case — the portions of everyday language) which are used in scientific discourse do not have to be semantically closed.* » Cette deuxième considération nous semble indiquer que les auteurs de UML ne pouvaient pas penser à la définition de Tarski, car la clôture sémantique semble être un défaut plutôt qu'une qualité. Mais il y quelque chose d'autre qui nous fait penser que les auteurs de UML ne pensaient pas à la définition de Tarski : Tarski parle d'une langue et donc sa définition s'applique, éventuellement, à UML et non à l'abstraction écrite en UML. On peut donc comprendre la phrase « UML est sémantiquement clos » mais il est difficile de comprendre « l'abstraction du système est sémantiquement close ». En prenant la signification de « sémantiquement clos » de Tarski, pour que la définition de modèle tienne, il faudrait la transformer ainsi : « Une langue sémantiquement close du système ». Mais cette définition ne fait qu'embrouiller les idées sur ce qu'est un modèle, ce qui apporte de l'eau au moulin du déshonneur.

Signification non claire

Parmi toutes les définitions possibles, nous proposons la suivante : une abstraction sémantiquement close d'un système est une description qui contient tous les éléments dont on a besoin pour créer la machine qui s'intégrera au système pour satisfaire les exigences établies.

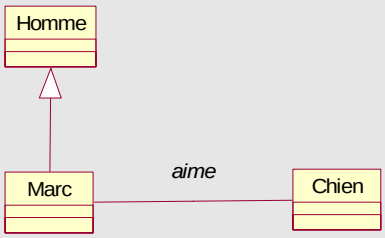
NOTE : Nous sommes loin d'être sûrs de la valeur de notre incursion dans la logique, par contre, nous sommes conscients que *ignorantia non excusat*.

Encadré 2

« Marc est un homme »

Voici un test que nous avons soumis à des étudiants en génie logiciel qui venaient de suivre — et de réussir — un cours d'analyse où ils avaient employé la notation UML.

`La figure suivante est une traduction en UML de la phrase « Marc est un homme qui aime les chiens ». Êtes-vous d'accord ? Pourquoi`



Quatre (4) étudiants seulement sur trente-trois (33) ont répondu que le diagramme n'était pas correct parce que *Marc* est une instance d'*Homme* et non une sous-classe. Les erreurs ne sont pas dues à un manque de connaissance de UML, car la majorité a écrit que *Marc* est une sous-classe d'*Homme* et que donc l'association entre *Homme* et *Marc* est correcte.

Il serait sans doute abusif de tirer des considérations théoriques de cette simple expérience, mais le résultat est tellement décevant et inattendu que nous nous permettons quelques réflexions.

Avant tout, il est clair que les étudiants savent conceptualiser même si, dans ce cas très simple, ils semblent ne pas avoir vu la différence entre un individu et son type (entre classe et occurrence). Dans la vie de tous les jours, il est bien rare qu'ils confondent Sylvie en os (et surtout en chair) avec le concept de femme !

Il nous semble plutôt que le fait que, dans la langue française, on emploie « est un » pour représenter la spécialisation entre concepts (un homme **est un** mammifère) comme pour représenter le lien entre l'individu et son type (Marc **est un** homme) est un piège psychologique. Si le triangle en UML signifie « est un » alors le triangle doit pointer de *Marc* à *Homme* ! S'il s'agit là d'une bonne interprétation, alors le fait de modéliser rend la description moins ambiguë et donc elle est utile.

Un « développeur leste » pourrait bien sûr rétorquer que, pour cela, on n'a pas besoin d'un diagramme UML : une fois qu'il aurait créé la classe *Homme*, l'étudiant l'aurait instanciée avec *Marc* comme valeur de l'attribut *nom* et donc il aurait fait son travail d'analyste-programmeur correctement. Dans un cas aussi simple, sans doute. Mais un « développeur lourd » pourrait à son tour dire que le fait qu'il n'ait pas modélisé hors langue de programmation ne lui clarifiera jamais les idées sur la différence entre type et individu, ce qui est fondamental pour un programmeur qui ne veut pas être un simple codeur.

Il y a sans doute aussi le fait que, quand les étudiants « modélisent » avec des graphiques, ils oublient leur « sens commun » et leurs connaissances de base tellement ils sont obnubilés par les objets graphiques. À ce propos, il faut ajouter que, dans ce même test, nous avons demandé d'expliquer la signification de modèle et, dans bien des cas, la réponse a été : « une représentation graphique ».