L'enchaînement des normes

par Ivan Maffezzini

Le Comité d'étude des termes techniques français propose de remplacer *standard* au sens général par *norme*, mais de conserver *standard* dans le domaine des télécommunications. De même il recommande d'éviter *standardisation* et *standardiser* et de leur préférer *normalisation* et *normaliser. (TLF)*

La norme est là pour être violée. (adaptation d'une phrase de Georges Bataille)

Introduction

Dans cet article on considère le terme « norme » dans son acception la plus générale, comme désignant un ensemble de règles à suivre dans la réalisation d'un produit. On considère aussi que ces règles ne sont pas nécessairement édictées par un organisme international.

L'emploi d'une acception très générale permet de considérer comme des normes :

- Les documents que IEEE appelle « Pratiques recommandées » ou ce que ISO appelle « Rapports techniques »[1] ;
- Les normes *de facto* d'entreprises leaders en génie logiciel ;
- Les spécifications et les modèles de référence comme ceux du *Software Engineering Institute* (SEI) ou du *Object Management Group* (OMG).

Nous allons commencer par classer les normes en fonction des types « objets » dont elles règlent les échanges : humains ou machines.

Échanges entre machines. Pour que des machines (logicielles ou matérielles) communiquent, il faut que la structure et les fonctions de leurs jonctions soient définies dans les moindres détails. Comme exemples de ce type de normes on peut considérer les normes pour les bus des ordinateurs, pour la structuration des données sur un disque, pour les formats de sortie des traitements de texte etc. La prolifération de ces normes est un indice de l'avancement de la technique et de l'augmentation du nombre de nouveaux produits. Ces normes aident à maîtriser la construction des machines mais il serait injustifié de les considérer comme des normes de GL, c'est-à-dire comme des normes qui aident à maîtriser la réalisation d'applications informatiques. Les normes pour les protocoles de communications (les standards comme le *comité d'étude des termes techniques français* nous autorise à les appeler) sont un exemple significatif de normes qui ne concernent pas le GL même si elles sont, avec les normes pour les langues de programmation, parmi les plus importante en informatique. La norme décrivant TCP (un

[1] IEEE Std 830-1998, *IEEE Recommended Parctice for Software Requirements Specification* et ISO/IEC TR 9126-3 *Software Engineering – Product Quality Part 3 Internal Metrics,* sont deux exemples très connus de documents qui ne sont pas des normes au sens strict mais que nous considérons comme des normes à part entière. Le fait que IEEE Std 830-1998 fasse partie du recueil des standards en génie logiciel d'IEEE va dans le même sens.

protocole de transport), par exemple, n'est pas une norme de GL, même si les ingénieurs des logiciels doivent la connaître pour concevoir et mettre en œuvre le protocole. Une norme qui décrit une langue servant à définir des protocoles non plus n'est pas une norme de GL, même si, à partir de cette langue, on peut éventuellement générer le code exécutable. Et, même si on avait une méta-langue servant à définir des langues pour les protocoles, on serait encore fort loin du GL. Le fait de ne pas considérer les normes qui règlent les jonctions entre machines comme des normes du GL limite l'étendue du GL. Cette limitation va dans le même sens que le choix de IEEE pour le recueil [1].

Échange entre machines et êtres humains. Parmi ces échanges nous considérons seulement ceux qui ont lieu entre les ingénieurs du logiciel et les environnements d'analyse et de programmation. Les autres concernent l'ergonomie et le domaine à informatiser. En suivant en cela IEEE et pour des motifs historiques, nous ne considérons pas comme faisant partie du GL les normes qui décrivent les langues de programmation, même si ces langues qui fixent les interfaces entre le programmeur et le compilateurs représentent celles avec lesquelles l'ingénieur du logiciel, à l'état actuel de la technique, est le plus en contact. Nous considérons les normes pour les langues de programmation comme faisant partie de l'informatique. À noter que lorsqu'une norme est formellement définie elle peut régler en même temps l'échange entre machines et entre êtres humains et machine.

Échanges entre personnes et personnes (qui participent à la définition, la conception et la mise en œuvre des programmes). C'est ici la place privilégiée des normes de GL. C'est ici aussi que les normes ressemblent le plus aux normes (les lois) qui règlent la conduite des individus dans un État. C'est ici que la norme a un sens comme limitation de la liberté des humains pour enlever le plus d'ambiguïté possible dans un projet, fixer les éléments obligatoires et les frontières au-delà desquelles la qualité et/ou la productivité risquent d'être inacceptables. Les normes de programmation sont un bon exemple de normes qui limitent la liberté du programmeur pour augmenter la lisibilité des programmes et donc la productivité. À noter que même si les normes de programmation sont souvent les premières normes d'une entreprise, ne font pas partie du recueil de IEEE [1].

1. Honneur

Prétendre que sans normalisation il n'y a ni industrialisation, ni technique moderne, est un truisme. Non seulement parce que sans règles fixant la structure et les fonctions des jonctions les machines ne peuvent rien échanger, mais aussi parce que les normes, puisqu'elles limitent l'étendue des significations des termes employés par les « techniciens », permettent à ces derniers de communiquer de façon moins ambiguë : ce qui est une condition nécessaire pour améliorer la productivité et, de façon plus générale, la qualité.

Les normes en tant que matérialisation des expériences passées fixent des points de repère permettant d'une part d'encadrer la démarche de productions d'artefacts (processus) et de l'autre de se concentrer sur les exigences ponctuelles de chaque artefact (produit). Et même si, parfois, cette matérialisation des expériences passées devient un frein à l'innovation, cela n'est pas nécessairement négatif, surtout dans des domaines où le foisonnement des « nouveautés » cache parfois une extrême fragilité des fondations.

Ce qui est vrai pour la technique en général, l'est particulièrement pour le génie logiciel (GL), une technique encore assez immature et dont l'artefact final est impalpable[2]. Même si l'artefact final est impalpable, les artefacts intermédiaires — les chaînons qui attachent l'artefact final aux idées et aux besoins initiaux — ne le sont pas. Puisque ce qui compte c'est la chaîne entière qui va des besoins à l'exécutable, les normes pour les artefacts doivent elles aussi être « enchaînées ». À ce propos, ce paragraphe tiré d'un document de ECMA est assez significatif : « ECMA croit fermement qu'afin que les normes pour les ordinateurs soient vraiment utiles, il faut qu'il existe un vaste ensemble de telles normes […] Il s'ensuit que s'il y a des doutes par rapport à la conformité à une norme quelconque, cela peut remettre en question la valeur de la conformité à plusieurs autres normes à propos desquelles de tels doutes ne devraient pas exister[3]. » [2].

Pour ne pas passer trop de temps à enfoncer des portes ouvertes on fera quelques considérations sur les normes IEEE et sur une spécification de OMG.

1.1 IEEE

Les normes IEEE pour le GL sont un bon exemple d'un ensemble de 37 normes qui satisfont les croyances de ECMA et qui couvrent assez bien les processus, les produits et les techniques de base du GL. Cet ensemble a aussi la caractéristique d'être ouvert aux normes d'autres organisations et en particulier à ISO dont la norme ISO 12207 a été intégrée à l'ensemble avec une simple « normalisation » du nom et l'ajout d'un suffixe (IEEE/EIA std 12207.0). Suffixe assez important, car il a permis au comité de normalisation de créer un sous-ensemble cohérent concernant les processus du cycle de vie :

- IEEE/EIA std 12207.1 pour le cycle de vie des données ;
- IEEE/EIA std 12207.2 pour des considérations de mise en pratique.

À ce sous-ensemble, il faut aussi ajouter la norme IEEE/EIA std 1074, norme servant à développer des processus pour le cycle de vie du logiciel.

L'exemple de IEEE montre donc non seulement qu'il y a un ensemble de normes pour le GL comme le demande ECMA mais que, dans cet ensemble, on peut avoir des sous-ensembles dotés d'une cohésion forte qui permettent de mieux normaliser certaines aires spécifiques du GL.

1.2 OMG

Les spécifications concernant la notation de modélisation unifié (UML) de OMG sont un bon exemple de normalisation ayant un impact bénéfique sur les activités en GL. Inutile d'insister sur l'importance de la normalisation de la notation UML, sur l'utilité de sa langue formelle (OCL) et de l'intégration dans une approche d'architecture pilotée par les modèles. On se limitera à écrire quelques mots sur la spécification SysML [3] et, en particulier, sur le chapitre 16 qui traite des exigences.

SysML est « une nouvelle notation de modélisation unifiée » qui « réutilise un sous-ensemble de UML 2.1 et fournit des extensions » surtout pour spécifier les « exigences,

[2] La séquence de bits n'est « palpable » que par l'unité de contrôle et de traitement.
[3] Traduction de l'auteur.

la structure, le comportement, les assignations et les contraintes sur les propriétés du système ». SysML est donc utile pour diminuer le clivage entre la description des exigences en langue naturelle et la modélisation « classique » en UML. SysML aborde en particulier le besoin de réutilisation des exigences, besoin surtout, mais pas seulement, ressenti dans les entreprises qui développent des familles de produits. Même si cette spécification ne peut espérer atteindre tous ses objectifs que lorsque plusieurs outils capables d'inter-opérer seront disponibles, une simple analyse du chapitre 16 permettrait déjà à une entreprise de vérifier son approche par rapport aux exigences. Les éléments qui suivent, même s'ils n'ont rien de bien original, sont des pas en avant dans la création d'un ensemble cohérent de normes comme envisagé par ECMA :

- Introduction des concepts d'exigences « maîtresse » et « esclaves ». « Une exigence esclave est une exigence dont les propriétés du texte sont une copie à lecture seule des propriétés d'une exigence maîtresse ».
- Association entre une exigence originelle et des exigences dérivées.
- Association entre une exigence et les éléments de conception (pour la traçabilité).
- Association entre une exigence et la description des cas à tester (pour la vérification).
- Association entre une exigence et d'autres éléments de modélisation (comme les cas d'utilisation, par exemple).
- La possibilité d'attacher les raisons (*rationale*) qui ont incité à établir les associations.
- Une clarification du fait que les exigences, même si elles représentent des stéréotypes des classes, ne peuvent pas être spécialisées.

Sys_ML est un bon exemple d'une norme qui « arrive » au bon moment, quand les temps sont mûrs pour figer les connaissances que l'avancement des pratiques a rendues solides.

Déshonneur

Dans les domaines techniques, être contre les normes relève de l'absurde, de l'enfantillage ou de la provocation. Être contre la prolifération des normes en GL, par contre, relève de ce minimum de bon sens qui ne devrait jamais quitter un technicien. Sans arriver à la vision minimaliste de Naur [4], qui réduit le GL à la gestion de listes de vérification ou aux positions les plus extrêmes de la programmation extrême, le nombre de normes qui ont été crées en une quarantaine d'années et l'accélération du rythme depuis une décennie donnent le tournis même aux ingénieurs du logiciel les plus solides. Il suffit de penser que les normes de ISO pour la technologie de l'information sont plus que 500 et que les normes pour l'ingénierie des systèmes et du logiciel sont près de 100 (96, en date du 4 décembre 2006) pour se demander si un certain GL n'a pas plutôt besoin d'individus à la tournure d'avocat, de juge ou de bureaucrate que de concepteurs. Pour ceux qui croient que la qualité d'un logiciel (quand il ne s'agit pas d'un simple paramétrage ou d'applications « jouets ») requiert que la portion de créativité des individus ne soit pas trop mince, tout n'est pas une affaire de nombres. C'est surtout une question d'approche.

L'objection voulant qu'il n'y ait pas de créativité à l'état pur et que toute créativité est créativité par rapport à des normes n'est pas recevable car, dans cette affirmation-ci, on emploie le terme « norme » dans un tout autre sens que celui de la normalisation en GL. Les normes qui favorisent la créativité sont souvent des normes tacites, personnelles ou de groupe, qu'une explicitation, dans la mesure où elle serait possible, les rendrait lourdes et pratiquement impossible à appliquer. Vouloir donner la même signification au terme norme quand celui-ci s'applique à la définition de la langue de programmation C++ ou quand il indique des limites tacites imposées aux actions des ingénieurs du logiciel vide le terme de toute signification utile dans un contexte d'ingénierie[4].

Pour montrer le peu d'utilité et le danger des normes en GL, nous nous limiterons donc à quelques considérations sur les normes IEEE étiquetées comme normes de GL et à la spécification de SysML de OMG.

2.1 IEEE

Dans l'affirmation de ECMA qu'« il faut un vaste ensemble » [2] pour que les normes soient utiles, il est sous-entendu que cet ensemble doit garder une certaine cohérence — ce qui en fait quelque chose de plus riche qu'un simple ensemble ! Mais, si on se fie à ce que IEEE écrit à la page XV du premier volume de sa collection de normes, on est loin du compte : « Les différentes normes ont été écrites séparément en ayant comme objectif secondaire celui d'une cohérence d'ensemble. » Ce n'est pas un hasard si IEEE ne respecte pas les contraintes de ECMA : à l'époque où ECMA écrivait la considération citée, parmi les normes ECMA, il n'existait aucune norme de GL.

Considérons à titre d'exemple la norme IEEE std 829, qui décrit la documentation pour les essais [6] où l'on propose des tables des matières pour huit (8) types de documents. Comme dans toutes les nombreuses normes où l'on propose une table des matières, non seulement le contenu des chapitres est flou mais dans la section *Portée du document*, on écrit que : « le contenu de chaque section doit être adapté à l'application et à la phase des essais » ; que des sections peuvent être ajoutées comme on peut ajouter des documents ; que de nouvelles conventions peuvent être spécifiées… Si à cela on ajoute que la partie la plus utile de la norme, c'est la partie qui n'est pas normative et où l'on présente des exemples, on peut de demander « à quoi bon l'appeler norme ? ». La seule conformité envisageable est une conformité que l'on pourrait qualifier de formelle… sur les titres de chapitres, et encore !

Il serait intéressant de se demander pourquoi, parmi les normes de IEEE, il n'y a pas de norme sur le codage et sur la documentation des modules. Parce qu'elles dépendent trop des particularités de l'entreprise ? Parce que l'on pourrait être très précis ? Parce que les membres des comités sont trop éloignés de la programmation ? Peu importe la réponse, ce qui est certain, c'est que la partie du logiciel la plus étroitement liée à l'un des artefacts finaux et qui est l'une des plus faciles à normaliser n'a pas été normalisée.

2.2 OMG

La spécification de UML est un bon exemple d'une norme qui a eu un impact énorme en GL. Son impact n'est pas tellement dû à la notation elle-même, mais à l'approche de

[4] Il devient par contre un élément très important pour une analyse philosophique, philologique, historique et, pourquoi pas ? politique de la notion de « norme ».

développement qui la soutient. Ce qui apparaît clairement à la section 3.1 de la spécification de UML[5] [5] où, après avoir écrit que « [UML] est essentiellement une norme pour une notation de modélisation et non une norme pour le processus. » et après avoir souligné que « UML doit être appliqué dans le contexte d'un processus », les auteurs écrivent que, selon leur expérience, « les différentes organisations et les problèmes du domaine requièrent des processus adaptés » et que « par conséquent les efforts ont été concentrés en premier lieu sur un méta-modèle commun […] et ensuite sur une notation commune […]. ». Le paragraphe se termine avec une phrase qui a comme effet d'effacer ce qui a été écrit auparavant « Les auteurs de UML encouragent un processus de développement piloté par les cas d'utilisation, centré sur l'architecture, itératif et incrémentiel. » Cet encouragement a eu beaucoup de succès, certainement beaucoup trop, car bien des ingénieurs du logiciel associent très étroitement UML au développement piloté par les cas d'utilisation. Mais si, comme les auteurs de UML l'écrivent « les différentes organisations et les problèmes du domaine requièrent des processus adaptés », il est fort peu probable que les cas d'utilisation constituent la panacée. Voilà un effet pervers d'une norme qui est censée aider à réaliser un logiciel de qualité et qui met l'ingénieur du logiciel, bon gré mal gré, dans un processus trop général qui risque de faire perdre tous les bienfaits de la langue de modélisation.

Que dire de SysML[6], l'une des dernières spécifications adoptées par OMG ? Qu'il s'agit d'une spécification dont l'objectif explicite est de faciliter la spécification des exigences système ? Que l'objectif tacite est d'étendre la « famille » UML et les outils associés ? Que la réduction de l'exigence à un simple texte sans certains attributs comme *stabilité*, *nécessité*, *priorité, antécédents*… fait reculer le génie des exigences ? Et, pour terminer cette série de questions : *Cui prodest* ? Sans doute aux constructeurs d'outils. À moins que cela ne constitue qu'une indication selon laquelle les ingénieurs du logiciel, par crainte de devenir des cordonniers mal chaussés, chaussent des inutiles souliers… abstraits

Au-delà

Assimiler la normalisation à la bureaucratisation est sans doute l'un des freins psychologiques les plus importants à l'introduction des normes dans l'industrie du logiciel, surtout dans les petites entreprises. Cette assimilation est favorisée par le très grand nombre de normes. Ce qui est certain, c'est qu'à cause de la jeunesse relative du GL et, surtout, de la possibilité d'informatiser n'importe quel domaine, la prolifération des normes risque de se poursuivre pendant des années. Comment vivre avec cela ? Comment une entreprise peut-elle ne pas trop se contraindre et gonfler outre mesure les contrôles mais en même temps donner assez des règles pour que chaque individu ne doive pas re-inventer la roue quotidiennement ? La réponse à ces questions dépend du domaine, des exigences du client, des dimensions de l'entreprise, de la « culture » de celle-ci, etc. Il n'y a pas UNE réponse[7]. Il serait très naïf de vouloir trouver une réponse

[5] Cette spécification est aussi disponible à ISO comme ISO/IEC 19501.

[6] Qu'il s'agisse de « système » et non de logiciel n'est d'aucune importance car les exigences n'appartiennent ni au logiciel ni au système mais — au moins avant que l'on ne conçoive la machine — au domaine.

[7] Il ne faut pas avoir honte de s'unir au chœur de ceux qui affirment que trop souvent en GL on croit avoir LA réponse.

valable pour tous les cas en tout temps, à moins que l'on ne donne une réponse tellement générale qu'elle devienne pratiquement inutilisable. Par exemple : « il faut des règles ».

Il faut avant tout que les normes aient à tous les niveaux (international, entreprise et projet) une possibilité d'adaptation. Possibilité d'adaptation qui devrait être maximale au niveau international et presque inexistante au niveau du projet. Adaptation envisagée par pratiquement toutes les normes de IEEE.

On doit tenir compte d'un deuxième élément : si les machines rognent toujours plus de terrain par rapport aux humains, les normes doivent viser un formalisme toujours plus grand pour passer de la catégorie *échange entre êtres humains* à celle d'*échanges entre machines* là où il n'y a plus de place pour l'ambiguïté. La complexité de la normalisation dans le GL provient aussi du fait qu'une norme qui règle le comportement être les humains peut un jour devenir une norme pour la communication entre machines.

C'est bien parce qu'il n'y a pas UNE réponse que l'on préfère terminer avec quelques conseils à des ingénieurs du logiciel d'une nouvelle petite entreprise d'informatique pour qu'ils ne tombent ni dans une acceptation acritique des normes ni dans un refus enfantin.

1. Ne pas commencer à penser en termes de modèles de maturité (CMMI ou autre). En raison de la complexité et des détails présents dans ces modèles, le risque de se dégoûter à jamais de pratiques pourtant essentielles est très élevé.
2. Fixer une norme de programmation (et de documentation du code).
3. Considérer un ensemble de normes assez cohérent et complet (IEEE par exemple) et s'assurer de comprendre le but de chacune d'entre elles et la cohérence de certains sous-ensembles de normes (comme les normes qui traitent des plans ou les normes qui concernent les processus).
4. En choisir trois ou quatre au maximum et les simplifier pour les adapter à l'entreprise. Ces normes deviennent le noyau autour duquel en fonction du type de projet, des exigences de qualité et des goûts personnels, lentement, les autres normes s'agencent. Le tableau 1 présente un choix de quatre normes ainsi que les raisons qui motivent leur choix.

Tableau 1 Quelques normes IEEE

Norme	Raisons
IEEE std 12207.0 (Processus du cycle de vie du logiciel)	Pour fixer la signification de certains termes et choisir le processus prioritaire. La norme peut être considérée comme un dictionnaire et une liste de vérification pour la gestion de projet. Elle pourrait permettre de construire un squelette avec les types de taches pour un logiciel de gestion de projet.
IEEE std 1362 (Guide pour la rédaction du document Principe d'opération)	Pour faciliter l'écriture des études d'opportunité. Pour aider à faire une « photo » de la situation actuelle, à définir la transition vers un nouveau système et décrire les problèmes et les besoins les plus importants. Cette norme aide à intégrer dans la culture de l'entreprise l'importance d'avoir des approches structurées et rigoureuses dès le début des projets. Elle aide aussi à bien comprendre que les cas d'utilisation du vieux système peuvent

	servir à éclaircir les concepts du domaine même s'ils sont complètement différents de ceux du nouveau système.
IEEE std 830 (Pratique recommandée pour les spécifications des exigences logicielles)	Pour que l'entreprise ne se laisse pas entraîner vers une approche techniciste où seuls comptent l'architecture, les cadres de conception et l'environnement de programmation. Cette norme (selon les domaines) pourrait être étroitement intégrée avec des approches de validation par prototype. La création de gabarits avec des documents qui s'intègrent aux prototypes diminuerait le sentiment initial de « sur-documentation ».
IEEE std 1028 (Norme pour les revues logicielles)	Pour que la vérification et la validation entrent sans trop de lourdeur. Il s'agit d'une norme facile à adapter à tous les contextes et qui permet donc en très peu de temps (quelques jours) d'avoir « sa norme ».

Pourquoi ne pas considérer une norme aussi importante que celle qui concerne la gestion de la configuration ? Pour les mêmes raisons pour lesquelles on n'a pas besoin de connaître la spécification de UML : les outils intègrent déjà en partie des normes et les ingénieurs du logiciel sont censés connaître les outils qui concernent la gestion de la configuration.

Pourquoi pas la norme IEEE 730 concernant l'assurance de la qualité ou la norme IEEE pour la vérification et la validation, s'il est vrai que la qualité est tellement importante ? Parce qu'il s'agit de normes qui risquent de donner l'impression « d'avoir tout ce qu'il faut » parce que l'on a un soi-disant plan s'assurance qualité ou un plan de vérification et validation.

Pourquoi pas les normes concernant les tests ? Parce que l'importance des tests n'est jamais sous-évaluée comme le sont les vérifications au début du cycle de vie…

Le choix des « pourquoi » et, surtout, des « parce que » est sans doute très arbitraire mais, quand il s'agit de normes, l'arbitraire n'est pas à craindre car, comme toute règle ou loi, les normes sont fondées sur l'arbitraire, ce qui ne leur enlève aucun droit car ce sont elles qui font et qui fondent le droit.

Références

[1] IEEE, *IEEE Standard Software Engineering*, volume 1, 1999.

[2] ECMA, *The Meaning of Conformance to Standards*, **TR/18**, September 1983. (Disponible à http://www.ecma.org/xxxx)

[3] OMG, *OMG System Modeling Language (OMG SysML) Specification: Final Adopted Specification*, May 2006.
Disponible à (http://www.omg.org/technology/documents/spec_catalog.htm)
[4] OMG, *Unified Modeling Language Specification*, Version 1.4.2, January 2005
(Disponible à http://www.omg.org/technology/documents/spec_catalog.htm)
[5] Naur P. "Programming as Theory Building", pp. 37-48, *Computing: A Human Activity*, ACM Press 1992.

[6] IEEE, *IEEE Standard for Software Test Documentation - std. 829-1998.*