Prolégomènes à une critique du génie logiciel

Partie II : Qualité et mesures des produits

par Ivan Maffezzini, Alice Premiana et Bernardo Ventimiglia

Croyez-vous qu'il faille, pour être agronome, avoir soi-même labouré la terre ou engraissé des volailles ? Mais il faut connaître plutôt la constitution des substances dont il s'agit, les gisements géologiques, les actions atmosphériques, la qualité des terrains, des minéraux, des eaux, la densité des différents corps et leur capillarité ! GUSTAVE FLAUBERT, Madame Bovary.

Nous avons besoin de théories convaincantes et sophistiquées pour avoir la capacité d'expliquer les observations empiriques. NORMAN FENTON, A critique of Software Defect Prediction Models.

La quantification d'attributs qui étaient auparavant considérés comme dichotomiques est une des conséquences du progrès de la science. MARIO BUNGE, Treatise on Basic Philosophy, Vol, 3 Ontology I.

Pour nous la dichotomie de base de chaque ensemble d'objets ce n'est pas celle entre individus et ensemble mais celle entre objets physiques et objets conceptuels. Ibid.

Il est frappant même de constater qu'à peu d'exceptions près, sinon toujours, la valeur esthétique absolue est en proportion directe de l'adéquation de la forme à la fonction. (…) il est possible en effet de se demander (...) si la qualité fonctionnelle des œuvres humaines, au lieu d'être figurative, n'est pas l'invagination pure et simple, dans le champ humain, d'un processus absolument naturel. A. LEROI-GOURHAN, le Geste et la Parole.

Résumé : Le présent article est le deuxième d'une série de quatre où nous tâchons de montrer que le *génie logiciel* (GL) ne peut aspirer à devenir l'un des centres – robuste – des activités d'automatisation que si les branches hypertrophiées par les vicissitudes historiques et les conflits économiques sont élaguées. Dans le premier nous avons « arbitrairement » fixé les principes qui soutiennent notre démarche. Dans cet article-ci nous abordons les « problèmes » reliés à la qualité et en particulier aux métrologies de qualité en introduisant le modèle proposé par la norme ISO 9126-2001. Les mesures des attributs de la facilité de maintenance nous permettront de douter du bien fondé des définitions actuelles de la maintenance. Nous considérerons ensuite le couplage comme un exemple de la difficulté de transformer une idée intuitivement valide en un concept mesurable. Nous terminerons en proposant une piste pour fonder la métrologie de qualité en se fondant sur une relecture de la théorie ontologique de Mario Bunge. Quatre encadrés (notes bibliographiques, terminologie, la physique vue par le GL et un dialogue sur la complexité cyclomatique) complèteront l'article.

1.Préambule

Nous ne commencerons pas à parler de qualité en présentant les *mythes* qui existent autour d'elle comme c'est souvent le cas dans les numéros spéciaux des revues qui traitent de la qualité des produits logiciels. Nous ne le ferons pas d'une part parce que nous respectons trop les mythes qui préparèrent l'entrée de la raison scientifique dans le monde et de l'autre parce qu'il est souvent facile de les inventer ou

de les ressusciter comme un moyen rhétorique pour valoriser nos mythes à nous[1]. Nous n'essayerons pas non plus de définir la qualité, car l'idée informelle que chaque lecteur en a est plus que suffisante pour suivre notre démarche.

Il suffit de consulter les articles qui traitent de mesures et de qualité dans des revues comme *IEEE Software*, *Transaction on Software Engineering*, *Communications of the ACM*, *Annals of Software Engineering*, *Software Quality Journal*… pour s'apercevoir qu'on y retrouve tout et le contraire de tout. L'auteur A confirme les résultats de B qui infirme ceux de D qui à son tour montre que A n'a pas considéré les données de E qui a vérifié le modèle pour de gros projets de F sur un échantillon de dix étudiants en première année d'université…

Même si dans cette cacophonie il est difficile d'entendre une ligne mélodique quelconque, loin de nous l'idée que tout ce bruit soit dû à l'incapacité ou la mauvaise volonté des auteurs. Encore plus loin l'idée qu'on ne puisse pas un jour filtrer le bruit et retrouver une mélodie. Les causes de cette situation sont sans doute nombreuses et variées. Nous nous arrêterons à celle qui nous semble la plus facilement contrôlable : l'absence d'un cadre théorique clair, formel et partagé par les intervenants dans la production du logiciel[2]. Cadre qu'il n'est pas facile de donner et que nous n'avons pas la prétention de donner. Ce qui par contre est plus facile et sans doute plus nécessaire c'est de s'entendre sur une terminologie commune. La norme ISO 9126 est un premier pas mais nous avons l'impression qu'elle ne pourra pas avoir le succès qu'eut le modèle OSI pour les protocoles à cause de certaines faiblesses que nous essayerons de montrer.

Avant de terminer ce préambule, il nous semble important de souligner que l'opposition entre praticiens et théoriciens, qui est souvent avancée comme un élément qui freine le progrès du génie logiciel, est loin d'être une caractéristique de ce dernier. Dans toutes les disciplines, même en physique ! il y a des différences significatives entre ceux qui préparent les instruments théoriques et ceux qui, à l'aide de ces mêmes instruments et d'instruments concrets, œuvrent pour comprendre et modifier le réel. Ce qui, par contre, est caractéristique du GL, à notre avis, c'est que les instruments théoriques n'ont souvent pas le minimum de rigueur requis.

NOTE : comme dans l'article précédent nous devons tout à tous. Nous avons donc choisi de joindre une très courte bibliographie commentée au lieu de donner de trop nombreuses références. Dans un article sur la qualité et les mesures qui se targue de montrer que la sortie du tunnel dans le domaine est encore très lointaine, il y a aussi une motivation épistémologique pour ne pas donner de références bibliographiques. Si la période préscientifique (surtout le Moyen-Âge européen) fut caractérisée par le IPSE DIXIT, il serait encore moins rationnel, après quelques siècles de succès de la raison scientifique, de s'appuyer sur des réseaux de petits IPSE DIXITS[3] pour valoriser nos affirmations. Les lecteurs peuvent consulter, à l'adresse suivante
http://www.trempet.uqam.ca/trempet/membres/Maffezzini/Articles/ArticlesGL/BiblioPourQualite.htm
les quelques dizaines d'œuvres dernièrement consultées et qui nous ont le plus aidés à réfléchir sur les mesures de qualité.

[1] Un bon exemple de résurrection est donné par le sixième mythe présenté par Jeffrey Voas dans le numéro de septembre/octobre de la revue *IEEE Software* : « *les tests sortent du pétrin n'importe quel produit* ». Depuis des décennies, il n'existe pas un seul ingénieur du logiciel qui croit à ce mythe. Éventuellement le mythe actuel consiste en une affirmation à peu près contraire : « Les tests sont inutiles pour livrer un produit de qualité quand il n'a pas été bien conçu et bien vérifié dès le début de son cycle de vie ».

[2] D'autres causes sans doute importantes mais sur lesquelles il est pratiquement impossible d'intervenir sont les suivantes : la jeunesse de la discipline, les impératifs économiques qui négligent le long terme, un style de recherche qui met au centre le nombre des publications, des réseaux d'idées paresseuses.

[3] Les *Ipse dixi*t ne sont pas petits parce que les personnes qui ont « dit » sont « petites » mais parce que chaque nœud du réseau est « petit » par rapport à l'ensemble des nœuds, ensemble qui est l'équivalent d'Aristote pour les savants du Moyen-Âge.

2. Introduction

2.1. Langue naturelle

Dans un article paru dans le numéro 66 de *Génie logiciel*, nous avons présenté un certain nombre de principes qui nous semblent essentiels pour jeter un regard différent sur le génie logiciel et un modèle que nous reproduisons ci-dessous.

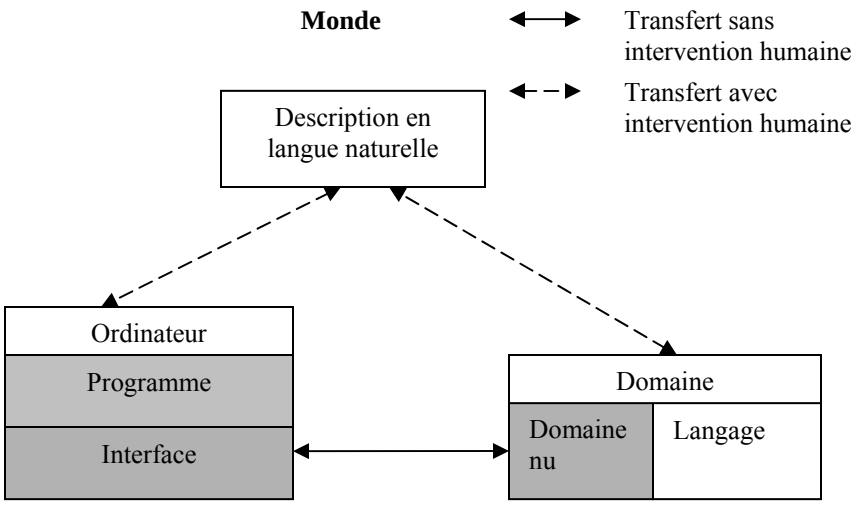

Figure 2.1 : Échange entre ordinateur et domaine

La description en langue naturelle est au centre du processus d'automatisation : elle est l'artefact principal créé par les experts du domaine — aidés éventuellement par des personnes dotées de bonnes capacités d'écriture, d'écoute, de synthèse et d'analyse. Cet artefact est la base du travail des ingénieurs du logiciel, dont les fonctions, selon notre approche, se « réduisent » à combler le fossé qui sépare l'ordinateur de la description en langue naturelle. Nous parlons de processus d'automatisation plutôt que d'informatisation pour mettre en évidence que le résultat du processus est une interaction « automatisée » entre les interfaces matérielles de l'ordinateur et le domaine nu (le domaine sans le langage).

Au moins trois motifs militent pour donner une importance particulière à la langue naturelle dans le processus d'automatisation :

- Le domaine est, par définition, une partie du réel inscrit dans le langage naturel et l'extraction des concepts utiles à l'automatisation ne peut donc que passer par le langage. Même dans des domaines très formels comme les mathématiques, les langages formels (les formules) nécessitent un tissu connectif naturel constitué de la langue « de tous les jours ».
- Les intervenant parlent, échangent et évaluent dans une langue naturelle. On suppose que les projets logiciels qui demandent une approche d'ingénierie soient d'une complexité telle qu'ils nécessitent l'intervention de plus qu'une personne. Les « petits » projets, ceux qui font intervenir un seul développeur, quand le projet dure plus de quelques mois, devraient être traités comme ceux qui emploient plusieurs personnes.
- Les « besoins » ne peuvent qu'être exprimés en langue naturelle (il est clair que l'on peut par la suite les traduire dans une langue formelle, mais cette traduction est, nécessairement, un appauvrissement. Appauvrissement qui est d'un tout autre ordre par rapport à celui qui est créé par la transcription des besoins en langue naturelle).

Mais la langue naturelle en tant qu'ensemble de signes est « tiraillée » entre la signifiance et la référence. La référence ancre les signes au réel physique hors discours et la signifiance est créée par les

échanges symboliques dans la culture. La figure suivante montre une situation « idéale » du point de vue de la vie quotidienne où le signe est équidistant du réel physique et du culturel.

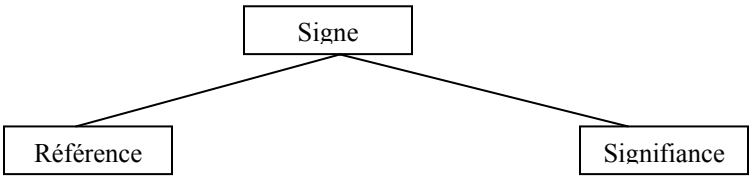

Figure 1.1 : Le signe en équilibre entre référence et signifiance

Les deux figures suivantes montrent les deux cas extrêmes qui permettent de mieux définir la différence entre la science et les discours sur les discours[4]. La première figure représente la science comme connaissance parfaite du réel physique : le signe complètement collé sur la référence est loin de la signifiance. Dans la deuxième, le signe est collé sur la signifiance et est loin de la référence : cette situation pourrait être exemplifiée par la rhétorique, la littérature et certaines sciences dites humaines.

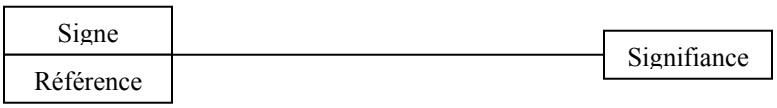

Figure 1.2 : Le signe déplacé vers la référence

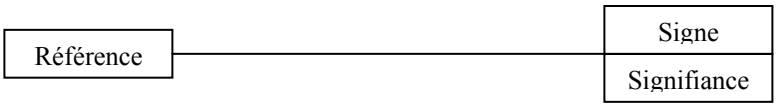

Figure 1.3 : Le signe déplacé vers la signifiance

Le GL ayant comme but la production de machines qui dialoguent avec le « réel physique » il faut que, dans le processus de développement, à partir d'un certain moment, le signe soit très proche de la référence. Mais à partir de quel moment ? Sans doute dès l'analyse, ce qui ne veut pas dire que l'analyse doit s'articuler autour d'éléments mathématiques mais seulement que les termes s'agencent pour donner des structures langagières solides mais perméables à la critique pour faciliter la validation. Mais peu importe le style de l'analyse et les artefacts générés, la distance entre le signe et la référence ne sera jamais objectivement mesurable.

NOTE : actuellement à cause du manque de définitions claires dans le processus de mesurage beaucoup de praticiens voient les mots des discours des qualiticiens qui s'occupent des mesures comme vides — loin de toute référence, comme dans la figure 1.3.

2.2. Formalisation

Mettre au centre le langage naturel comme nous venons de le faire n'est pas sans défaut, car cela pourrait faire oublier que, surtout là où l'on veut mesurer, on a besoin d'aller au-delà de la langue naturelle et de laisser parler les nombres.

[4] Il ne faudrait pas détecter dans cette répétition de « discours » un jugement de valeur, la répétition n'est qu'un moyen didactique pour souligner la différence.

La formalisation est surtout importante dans le domaine des mesures logicielles et, en particulier, des mesures de la qualité, là où le flou est le plus difficile à dissiper parce qu'il est pratiquement constitutif du concept même de qualité. La description en langue naturelle, les langues formelles et les mesures ont, toutes les trois, droit de cité dans le monde du GL. Leur importance relative est discutable, mais il n'est pas discutable que leur poids dépend du type d'application, des connaissances et des préférences des participants au projet.

Quel type de formalisation ? Nous croyons que la description du réel via les mathématiques proposée dans l'ontologie de Mario Bunge [Bun77] est une possibilité qui risque d'être riche en conséquences positives. Sa généralité et sa complétude devraient permettre d'aborder avec le même cadre théorique la conceptualisation du domaine et le mesurage de la qualité des artefacts. Nous avons aussi la prétention de croire que la centralité de la langue naturelle pourrait être le chaînon manquant qui permettrait à la théorie de Bunge de couvrir le processus de développement dès l'expression des besoins initiaux.

Dans le cadre de sa proposition d'une ontologie « *exacte et scientifique* », Mario Bunge écrit que celui qui s'intéresse à l'ontologie « *devrait délimiter les traits principaux du monde réel tel que connu via la science et procéder de manière claire et systématique. Il devrait reconnaître, analyser et mettre en relation les concepts qui lui permettent de produire un tableau unifié de la réalité* ». Même si on ne savait pas qu'un très grand nombre d'articles donnent parmi leurs références le livre de M. Bunge, il est clair qu'une telle définition ne peut que pousser à lire son œuvre. En lisant la phrase citée, n'a-t-on pas l'impression de lire la définition d'un GL idéal ? Pour se former une première idée de l'approche, voir l'encadré « Quelques concepts de l'ontologie de Bunge[5] ».

3. Mesures et qualité

Est-ce que l'affirmation de Tom deMarco si souvent citée est vraie : « *Vous ne pouvez pas maîtriser ce que vous ne mesurez pas* » ? Sans doute, même s'il nous semble plus correct de nuancer et de dire que la mesure *facilite* la maîtrise. Ce qui est par contre certain, c'est que quand les mesures sont employées sans un cadre théorique clair, non seulement elles sont inutiles mais elles peuvent créer de fausses assurances qui, dans le travail technique, peuvent facilement conduire à des catastrophes.

Depuis quelques siècles, depuis que la mathématisation de la physique, commencée par Galilée, a démontré sa supériorité sur la physique aristotélicienne et a permis l'avancement que nous connaissons de la technique, dans tous les domaines du savoir, on est obnubilé par la physique et ses méthodes fondées sur le mesurage. Ce qui fait que, souvent, l'introduction des mesures se fait trop hâtivement ou sans se soucier suffisamment du qualitatif préexistant et du cadre théorique qui, seul, donne signification aux grandeurs qui interviennent dans le processus de mesurage. Ces introductions hâtives créent des désillusions et un retour en force du « qualitatif » avec des conséquences fâcheuses pour le progrès technique. La sociologie est un bon exemple d'un domaine où « qualitatifs » et « quantitatifs » se livrent une guerre sans merci (et souvent sans intelligence). Dans le GL également, bien des conditions sont là pour que les escarmouches se transforment en une guerre de tranchée entre praticiens et théoriciens. Le dialogue qu'on peut lire en encadré entre un « qualitatif » et un « quantitatif » est un exemple des différends très légers qui pourraient facilement se transformer en oppositions radicales.

Même si la quantité d'articles publiés est énorme, la problématique des mesures en GL est loin d'être bien définie. Voici, par exemple, des questions très terre-à-terre pour des domaines scientifiques qui sont loin d'avoir des réponses claires et partagées par la communauté des praticiens et des chercheurs en GL :

- Veut-on mesurer pour évaluer la situation actuelle, pour prédire une situation future, pour mieux définir un concept ?
- Quoi mesurer ?
- Comment répéter le processus de mesurage pour comparer ce qui est comparable ?

[5] Nous avons eu l'impression que bien des auteurs citent Bunge sans vraiment l'avoir lu. Nous croyons que, pour bien maîtriser sa théorie et l'exploiter dans toutes ses potentialités, il faut lire non seulement le livre sur l'ontologie (Vol. 3) et celui sur les systèmes (Vol. 4) mais les quatre volumes qui composent son traité de philosophie.

- Quelle signification donner aux valeurs obtenues ?
- Comment les changements de valeurs influencent-ils d'autres phénomènes ou d'autres états de faits ?
- Comment employer les outils statistiques ?

Même si dans [Fen99] on ne s'intéresse qu'aux modèles de prédiction des défauts, cet article est un très bon point de départ pour une critique plus générale du mesurage.

Souvent, en lisant la littérature sur le sujet, on se demande si, lors du processus de mesurage, on a bien défini :

- l'*environnement*, c'est-à-dire le lieu où le processus de mesurage s'effectue avec toutes les connexions au phénomène sous mesure, l'organisation, les contraintes, etc.
- la *procédure* de mesurage : la suite des opérations à faire pour obtenir la mesure avec une description détaillée des données qui doivent être gardées constantes.
- les *instruments* employés pour permettre aux humains ou aux machines de lire des grandeurs et de les présenter sous un format conventionnel.
- les *sources* des données avec les procédures de modification.

Ce qui est certain, c'est que tout cela est rarement décrit, ce qui empêche de refaire l'expérience pour pouvoir comparer les résultats. Il faut dire que, dans le GL tel qu'il est considéré actuellement, il y a une composante humaine très importante qui rend le processus de mesurage plus difficile et moins objectif qu'en physique[6] et qui, au moins sous certains aspects, le rapproche davantage des sciences humaines.

À titre d'exemple considérons une mesure très simple tirée de la norme IEEE 982, le nombre de jours-défauts (FD : *Fault-Day number*)

- *Environnement* : environnement de développement (ordinateurs, base de données avec les données historiques et les documents de projet).
- *Procédure* : pour chaque défaut trouvé :
    - Stocker la date, l'heure et la phase d'apparition du défaut.
    - Trouver la date d'injection du défaut.
    - Calculer le nombre de jours pendant lesquels le défaut a « vécu » dans le système : FD_i.
    - $FD = \sum FD_i$ sommation des FD_i pour obtenir la valeur totale pour le système.
- *Instrument* : instruments de test.
- *Observateur* : celui qui teste.
- *Source de données* : horloge, données de projet.

Dans la procédure de mesurage, « Trouver la date d'injection du défaut » est loin d'être une action simple. Elle dépend des données stockées (et de leur fiabilité), du temps consacré à la recherche, de la personne qui a effectué la recherche… Avec toutes ces variables indépendantes, il est pratiquement impossible de connaître la marge d'erreur et donc de pouvoir répéter la mesure et faire des comparaisons, y compris dans la même entreprise. Mais même en imaginant qu'on puisse calculer FD avec une grande précision, il faudrait voir quelle est sa signification par rapport aux objectifs de qualité.

Si on a besoin de mesurer la température du four quand on prépare un gâteau, c'est parce qu'elle a un impact sur la qualité du gâteau. Quelle est la signification du nombre de jours-défauts par rapport au « gâteau » qualité du logiciel ? Un FD élevé indique une qualité mauvaise, mais la qualité de quoi ? Pour essayer de comprendre la signification de FD, considérons deux cas extrêmes :

1. FD élevé parce qu'il y a un petit nombre de défauts qui vivent longtemps dans le système.
2. FD élevé parce qu'un grand nombre de défauts qui vivent peu de temps.

[6] Dans la physique aussi, l'élément humain est présent en tant qu'observateur qui manipule les instruments de mesure. Cette présence influence éventuellement les phénomènes à échelle microscopique, mais ne rend pas la mesure moins objective.

Dans le premier cas, on peut considérer que les processus de vérification et de validation ne sont pas de bonne qualité. Dans le deuxième, il s'agit probablement d'une mauvaise conception (mauvais produit). Donc, la valeur numérique de FD ne nous dit pas grand-chose sur ce qu'on doit améliorer pour améliorer la qualité.

Vaut-il mieux avoir cette donnée qu'en avoir aucune ? Cela dépend si cette mesure nous enlève les ressources qu'on aurait pu consacrer à une autre mesure. Mais, même si on suppose qu'il s'agit de la meilleure mesure que l'on puisse prendre dans un contexte donné, le rapport entre les défauts et les pannes (la seule chose qui soit importante pour l'utilisateur) est loin d'être simple et directe [Fen99]. Pour résumer ces considérations, on pourrait dire que des trois fonctions des mesures, soit prédiction, évaluation et définition des concepts, c'est cette dernière fonction qui, à l'état actuel de l'évolution du GL, est la plus appropriée.

Nous continuons nos considérations sur les mesures en présentant le modèle de la norme ISO-9126-1. Nous présentons ce modèle, pas tellement parce qu'il est le seul intéressant ou le meilleur, mais parce qu'il est l'un des plus récents et que, comme tout ce qui est normalisé, il représente le résultat d'un compromis qui est intéressant pour évaluer l'état de l'art moyen, c'est-à-dire en évitant les pointes de la recherche très avancée qui risquent de se briser au contact de la dure réalité et les creux d'une pratique insouciante de toute théorie.

4. ISO 9126-1

ISO 9126 est constituée de quatre parties dont seule la partie 1 (ISO 9126-1 : 2001), qui présente le modèle de la qualité des produits, est une norme officielle en mars 2004.

4.1. Modèle d'interaction

La figure ci-dessous adaptée de la figure intitulée **Qualité dans le cycle de vie** tirée de [ISO01].

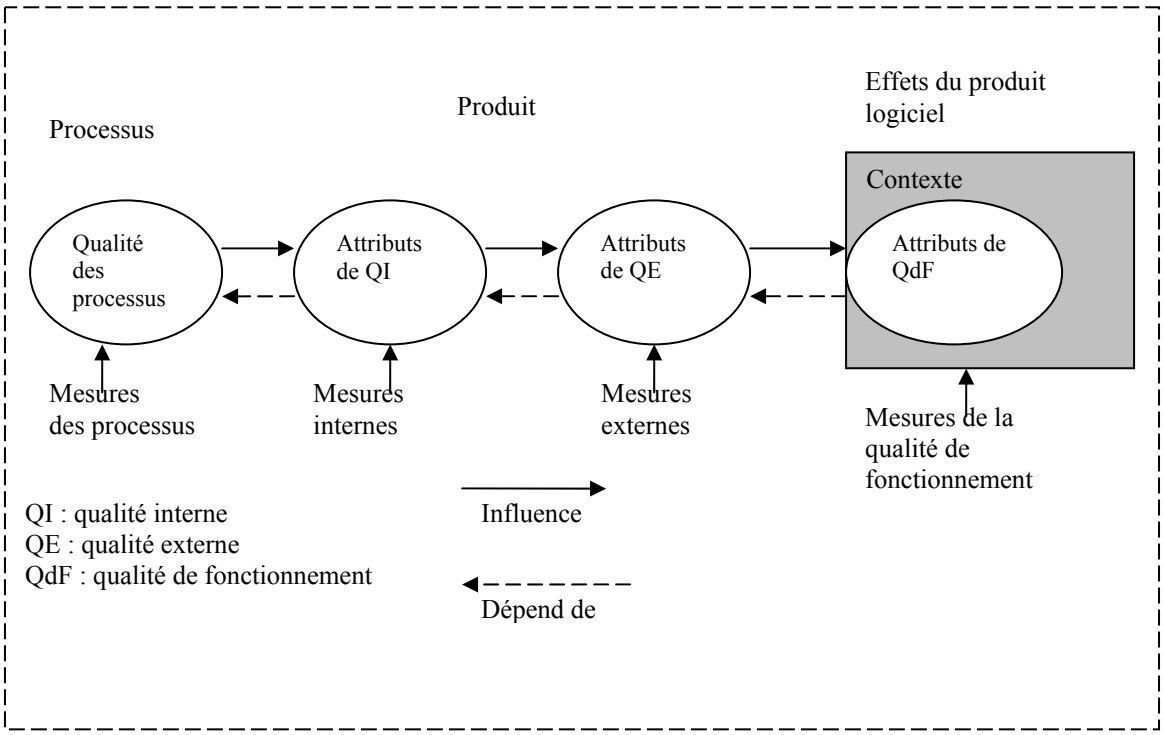

Figure 4.1 Qualité dans le cycle de vie

Cette description graphique montre que les attributs de qualité du logiciel s'exécutant dans son environnement de fonctionnement normal (*Qualité de fonctionnement : QdF*) sont influencés par les

attributs de la qualité mesurée pendant les essais (*Qualité externe : QE*). Les attributs de la qualité externe sont influencés par les attributs de la qualité des artefacts mesurés sans que le logiciel soit exécuté (*Qualité interne : QI*) qui, à son tour, est influencée par la qualité des processus.

Les attributs de la *qualité interne* sont mesurés sans l'exécution du programme. Les attributs de la *qualité externe* sont mesurés en exécutant le programme dans un environnement de test. Les attributs de la *qualité de fonctionnement* sont mesurés en exécutant le programme dans l'environnement des utilisateurs finaux.

Les attribut sont définis dans la norme comme : « *Une propriété physique ou abstraite mesurable d'une entité* ». Même si une entité peut être un élément d'un processus ou d'un produit, la norme est concernée seulement par les produits.

4.2. Caractéristiques et sous caractéristiques

Pour faciliter les processus de mesurage et d'organisation des attributs, la QE et la QI sont organisées en une hiérarchie dont les éléments du premier niveau sont appelés caractéristiques et ceux du deuxième niveau sous-caractéristiques. La figure suivante montre les six caractéristiques communes à QE et à QI.

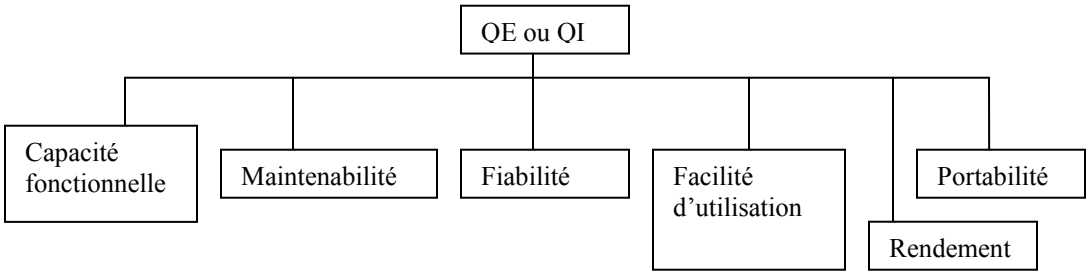

Figure 4.2 Hiérarchie de QE et QI

La QdF a quatre caractéristiques (*Efficacité*, *Sûreté*, *Productivité* et *Satisfaction*) qui ne sont pas ultérieurement structurées en sous-caractéristiques.

À titre d'exemple, nous considérons la maintenabilité qui est constituée des cinq sous-catégories : *facilité d'analyse*, *facilité de changement*, *testabilité*, *stabilité* et *conformité aux normes*. Parmi ces sous-caractéristiques, nous allons détailler la *facilité d'analyse*.

La facilité d'analyse est définie comme étant : « *La capacité du produit logiciel à faire l'objet d'un diagnostic des déficiences ou des causes de pannes du logiciel, ou des pièces à modifier* ». Cinq (5) mesures sont prévues pour les attributs de QE et deux (2) pour la QI.

- **QE**. Les cinq mesures de QE sont reliées à la présence de programmes qui facilitent les diagnostics. Comme exemple, on peut considérer la capacité d'analyse des pannes qui est censée répondre à la question « *est-ce que l'utilisateur peut identifier l'action précise qui a causé la panne* ? ». Pour ce faire, la norme propose de calculer le rapport entre le nombre de pannes que le responsable de l'entretien peut diagnostiquer en employant les programmes de diagnostic et le nombre total de pannes enregistrées.
- **QI**. Les deux mesures de QI sont reliées au nombre de fonctions mises en œuvres pour aider à établir les diagnostics : *activité de stockage* et *fonctions de diagnostic prêtes*. Cette dernière fonction est calculée comme suit : rapport entre le nombre de fonctions de diagnostic mises en œuvre et confirmées par une revue et le nombre de fonctions de diagnostic requises.

Les deux mesures présentées avec leur méthode de calcul, devraient aider à clarifier le concept de *facilité d'analyse* et en même temps permettre de mieux comprendre comment les attributs de QI influencent la QE.

4.3. Quelques considérations sur le modèle

Sans mettre en doute son utilité comme instrument de clarification de la métrologie en GL, nous présentons trois critiques, à notre avis fondamentales, à la norme.

4.3.1. Organisation du modèle

Dans la portée de la norme, il est indiqué que la partie 9126-1 : « *Décrit un modèle en deux parties : a) qualité interne et qualité externe, et b) qualité de fonctionnement* ». Il nous semble que cette division d'une part de la qualité des artefacts logiciels et du logiciel en test et de l'autre la qualité du logiciel s'exécutant dans l'environnement final est trop liée à l'organisation des projets et aux contingences des entreprises. Au fond, la QI et la QE sont liées entre elles pratiquement seulement parce que les mesures sont exécutées chez le fournisseur. La différence entre QE et QdF est secondaire par rapport à la différence entre QI et QE. Le fait de départager QE de QdF, conformément à la norme, est un élément important pour arriver à garantir que le produit exécute les bonnes fonctions mais c'est lié aux contingences du projet et aux contraintes contractuelles. Bien sûr, nous ne pensons pas que c'est sans importance pour la livraison d'un produit, mais que cela dépend davantage des processus que des produits.

Si on considérait d'une part la QI et de l'autre la QE et la QdF, on aurait un regroupement plus homogène. La QE et la QdF sont des qualités dynamiques qui impliquent une évaluation au niveau du système de la part des utilisateurs finaux, tandis que la QI est une qualité statique qui doit être évaluée par des ingénieurs du logiciel.

NOTE : même si la norme à propos de la QE fait souvent allusion à un responsable de la maintenance, il est clair que le responsable de la maintenance fait le travail de l'utilisateur final avant de lui passer la machine. Par contre, l'ingénieur du logiciel qui fait les corrections ne peut pas être assimilé à l'utilisateur final.

Nous croyons que ce serait bénéfique pour la clarté de l'introduction des mesures si le modèle était transformé comme suit :

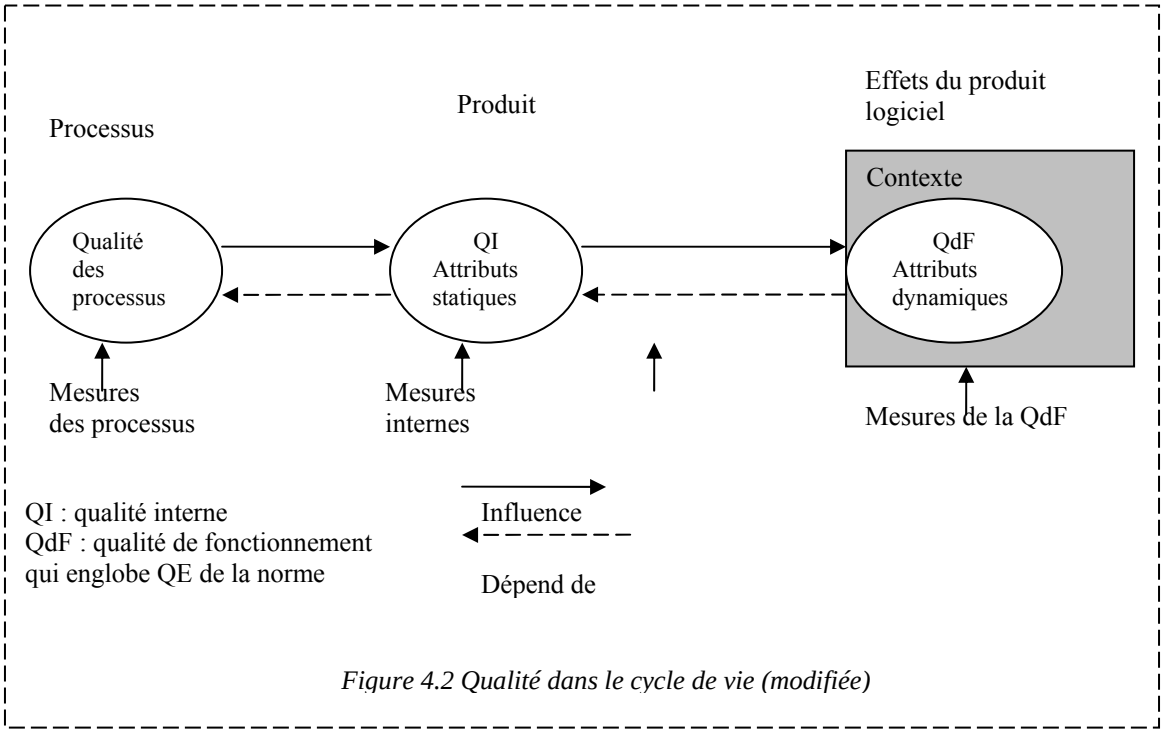

Figure 4.2 Qualité dans le cycle de vie (modifiée)

4.3.2. Évacuation des humains

Dans le modèle de qualité de la norme ISO 9126, il n'y a pas de place pour les humains, ce qui d'un certain point de vue est normal : on veut que la qualité du produit soit indépendante des humains et qu'elle dépende des « connaissances » et des méthodes de l'entreprise. Une entreprise qui ne maîtrise pas ses processus mais qui réalise le bon produit en respectant le budget et les coûts a nécessairement à son emploi de super-informaticiens (des héros) qui sacrifient leur vie au travail. Mais il serait très dangereux de se fier à une telle entreprise car les super-informaticiens sont aussi sur-convoités et sur-payés. Ils sont donc sur-mobiles…

Il faut limiter la présence des humains dans un modèle de qualité, surtout en tant qu'objets mesurés, mais il y a des activités dans les cycles de vie du logiciel où pratiquement le seul lien qui compte est celui qui existe entre la qualité de l'humain et celle du produit.

Une anecdote qui, même si elle ne passe pas le filtre de la scientificité, indique très bien que les ingénieurs du logiciel connaissent souvent mieux leur domaine et les humains que les qualitaticiens. Dans un projet d'un système de contrôle/commande de postes de transport d'électricité, nous avions introduit un indice de confiance, mesurable assez objectivement, comme élément pour caractériser la fiabilité d'une classe. Malgré cela, lorsque les programmeurs, suite à un panne, devaient chercher la classe « coupable », ils se fiaient pratiquement seulement au nom du programmeur qui avait codé la classe. C'est-à-dire que l'indice de confiance d'une classe était égal à l'indice de confiance qu'ils avaient dans le programmeur (indice qui dépendait du style de programmation, de l'intelligence…).

Il nous semble évident qu'un modèle de la qualité (surtout interne) où il n'y a pas de place pour l'humain est un modèle trop partiel.

4.3.3. Qualité des artefacts et satisfaction des exigences

Si, sans lui faire lire les métriques de la facilité d'analyse, on demandait à un ingénieur du logiciel « Qu'est-ce la facilité d'analyse, selon vous ? », il est fort probable qu'il répondrait qu'il s'agit de la facilité d'analyser le code, les documents de conception et les spécifications. Et il aurait raison. Et pourtant, dans la norme, il n'est fait mention de rien de tout cela. Dans la norme, même au niveau de la QI, on ne considère que les fonctions de diagnostics, c'est-à-dire que les métriques de la facilité d'analyse sont « orientés vers l'utilisateur final ». Mais pourquoi a-t-on oublié les mesures pour caractériser la qualité intrinsèque des artéfacts ? Une erreur ? Sans doute, mais alors pourquoi cette erreur ? Il n'est certainement pas facile de répondre, mais, ce qui est certain, c'est que le fait que la QI et la QE soient considérées comme constituant la « même partie » ne pouvait que faire déteindre la QE (celle qui est la plus proche de la QdF) sur la QI.

Si, à l'intérieur de la QI, au lieu de considérer la *facilité d'analys*e on considère la *facilité de changement,* on ne retrouve que la métrique suivante : « *rapport entre le nombre de modules/fonctions ayant des commentaires qui signalent les changements confirmés par les revues et le nombre de modules/fonctions changés à partir du code original* ». Ici, on se retrouve avec le même type de choix qu'aurait fait notre hypothétique[7] ingénieur du logiciel interrogé à propos de la facilité d'analyse. Cela indique qu'on n'a pas oublié toutes les qualités propres aux artefacts. Cette mesure, au lieu d'infirmer l'hypothèse que nous avions fait à propos de la facilité d'analyse, nous semble la confirmer dans le sens que, à cause du statut pas clair de la QI (ou de son double statut), une fois c'est un type de considération qui gagne et une fois une autre. Un peu au hasard.

Note : à noter non seulement l'emploi du terme module, mais aussi le fait que les changements sont considérés seulement par rapport au code, ce qui est loin de respecter l'esprit des processus de vérification et de validation tels que décrits dans ISO 12207.

[7] Pas tout à fait hypothétique. En réalité nous avons posé la question à deux collègues du département d'informatique de l'UQAM. Nous profitons de cette note pour remercier Louis Martin et Guy Tremblay pour les commentaires et les suggestions faites tout au long de notre long processus d'écriture.

5. Maintenance

La classification de la maintenance en maintenance corrective, adaptative et perfective est une classification canonique par rapport à laquelle on ne peut pas ne pas avoir un certain malaise, surtout si on a participé au développement et à l'entretien dans de gros projets. Il est difficile de voir la différence entre « maintenance perfective » et « nouveaux développements ».

La différence soulignée par Chris Kemerer dans le volume 1 des *Annals of Software Engineerin*g de 1995 *« Un facteur critique qui différencie la maintenance du logiciel d'un nouveau développement c'est le besoin que l'ingénieur du logiciel a d'interagir avec le logiciel existant et avec la documentation. »* même si elle représente la « pensée orthodoxe » nous semble excessivement simpliste et donc fausse (elle était vraie il y a une trentaine d'années : comme quoi les idées, une fois qu'elles ont été fixées sur papier — ou dans un disque dur — sont plus difficiles à changer que les objets du monde réel). Qu'il suffise de dire que la majorité des nouveaux développement s'inscrivent dans des environnements complexes, que la réutilisation implique des interactions complexes avec d'autres logiciels, que dans les gros projets il y a toujours des sous-systèmes terminés avant d'autres et avec lesquels il faut interagir…

La maintenance telle qu'elle est définie actuellement est le résidu historique d'une organisation du travail où les départements de maintenance étaient nés pour corriger les erreurs et progressivement ont pris en charge de nouveaux développements. Sans nier l'importance de l'organisation dans un projet, si on croit, comme nous le croyons, que la maintenabilité est l'une des caractéristiques de qualité les plus importantes, il faudrait envisager si, en changeant la classification de la maintenance, on ne pourrait pas faire un pas vers une nouvelle vision du GL. Par exemple : développer un nouveau produit comme s'il s'agissait d'un travail de maintenance.

Puisque que nous insistons sur le fait que la première fonction des mesures consiste à clarifier les concepts, dans les deux tableaux suivants, nous avons mis en correspondance les mesures de maintenabilité avec les trois types de maintenance de la norme.

Maintenabilité : qualité Interne

	Facilité d'analyse		Facilité de chang.	Facilité de test			Stabilité	
	AR	RDF	CR	CbT	AT	TPO	CI	MIL
Corr.	X	X	X	X	X	X	X	X
Adap.			X				?	?
Perf.			X				?	?

AR : *Activity recording*[8] (rapport entre le nombre d'éléments journalisés mis en oeuvre tel que spécifié et confirmé dans une revue et le nombre d'éléments à journaliser selon les spécifications) ; RDF : *Readness of Diagnostic Function* ; CR : *Change recordability* ; CbT : *Completeness of built-in test function* ; AT : *Autonomy of testability* ; TOP : *Test progress observability* ; CI : *Change Impact* ; MIL : *Modification impact localisation* ;

Maintenabilité : qualité externe

	F. d'analyse					F. de changement					F.de test			Stabilité	
	A	F	D	F	S	C	C	M	P	S	A	T	T	C	M

[8] Étant donné qu'il n'existe pas encore de traduction officielle de la norme, pour ne pas rendre la terminologie encore plus confuse, nous conservons les noms des mesures en anglais.

	F. d'analyse					F. de changement					F.de test			Stabilité	
	TC	AC	FS	AE	MC	CE	ET	C	M	CC	bT	RE	R	SR	IL
Corr.	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Adap.															
Perf.															

ATC : *Audit trail capability* (Rapport entre le nombre de données enregistrées pendant l'opération et le nombre de données qu'il aurait fallu enregistrer pour suivre l'état du logiciel) ; DFS : *Diagnostic function support* ; FAC : *Failure analysis capability* ; FAE : *Failure analysis efficiency* ; SMC : *Status monitor capability* ; CCE : *Change cycle efficiency* ; CET : *Change implementation elapsed time* ; MC : *Modification complexity* ; PM : *Parameterized modifiability* ; SCC : *Software change control capability* ; AbT : *Availability of built-in test function* ; RTE : *Re-test efficiency* ; TR : *Test restartability* ; CSR : *Change success ratio* ; MIL : *Modification impact localisation*.

Les tableaux n'ont pas besoin de commentaire : on mesure la maintenance corrective, c'est-à-dire la « vraie » maintenance.

Note : on n'avait pas besoin de mesures pour savoir que la maintenance, telle que définie en GL, est une maintenance inspirée des autres génies et qui ne se tient pas quand on considère les caractéristiques propres du GL, mais les mesures peuvent être d'un certain secours malgré tout.

6. Couplage

Nous avons choisi de parler du couplage, non pas tellement parce qu'il est important en soi ou plus important que d'autres concepts qu'on s'efforce de mesurer, mais parce qu'il nous semble paradigmatique de l'évolution de certains concepts en GL.

6.1. Origines

Pour montrer comment l'introduction de mesures sur des concepts empiriquement évidents, quand les termes ne sont pas assez clairement définis, peut conduire à des solutions pseudo scientifiques et trompeuses, nous suivrons en détail la naissance du concept de couplage tel que présenté dans le livre de E. Yourdon et L. Constantine sur la conception structurée [You75]. Nous avons choisi ce livre à cause de sa clarté et de son grand impact sur les praticiens de la fin des années 1970.

Dans la section *Fondements*, E. Yourdon et L. Constantine introduisent ce qu'ils appellent, ironiquement, le *Théorème fondamental du génie logiciel*. Le théorème, transposition dans le GL, de la maxime *dividi et impera* dit que le coût C de développement d'un programme P est supérieur au coût de développement de deux moitiés de programme :

$$C(P) > C(P/2) + C(P/2)$$

« Théorème » qui pourrait déclencher, chez les plus naïfs, une modularisation à outrance avec un module par instruction car puisque P/2 est encore une programme : $C(P/2) > C(P/4) + C(P/4)$…

Le bon sens nous dit que cette course à la division doit s'arrêter quelque part si on ne veut pas qu'on nous enferme dans un asile. Les auteurs améliorent donc leur « théorème » en introduisant les facteurs d'interaction I_1 et I_2 : I_1 représente l'interaction de la première moitié du programme qu'ils renomment P' avec la deuxième moitié renommé P" et I_2 représente l'interaction inverse. Ils obtiennent donc un coût

$$C(P) = C(P' + I_1 \times P'') + C(P'' + I_2 \times P')$$

qui est supérieur à $C(p/2) + C(P/2)$ si I_1 et I_2 ne sont pas 0 (comme il est intuitivement évident).

Suite à ses considérations, ils proposent deux courbes pour caractériser le lien entre le coût d'un programme et le nombre de modules qui le constituent :

- La première courbe, exponentielle à coefficient négatif, tient compte du fait qu'en augmentant le nombre de modules les coûts diminuent à cause de la plus grande facilité pour les humains de maîtriser un problème partagé en sous-problèmes.
- La deuxième courbe, exponentielle à coefficient positif, représente les coûts croissants de modularisation dus à l'augmentation des interfaces entre les modules.

Voilà donc que le « frein » à une modularisation excessive sont les facteurs I_i qui quantifient le coût des interactions et qui sont une « mesure » du couplage lorsqu'on considère P' et P" comme des modules. Les auteurs analysent ensuite le couplage qu'ils considèrent influencé par quatre facteurs qui capturent les difficultés (cognitives) des humains aux prises avec la modification des modules. :

- Le type de connexion entre les modules.
- La complexité de l'interface.
- Le type d'information qui transite à travers l'interface.
- Le moment du lien entre les modules.

Ils ne parlent pas de mesures sinon pour dire[9] :

> « *Si des valeurs numériques doivent être assignées pour des fins de recherche, de mathématisation ou de mystification, on suggère l'échelle suivante (…). Introduire de tels nombres actuellement (quand nous avons si peu d'expérience pour juger) pourrait créer un élément de magie dans tout le champ de la conception structurée. Ce qui concerne le plus les auteurs c'est le fait que des concepteurs/programmeurs introduits pour la première fois pourrait se sentir offensés par les tours de passe-passe de valeurs artificielles (…)* »

En résumé : Yourdon et Constantine guidés par leur bon sens et leur expérience proposent le concept de couplage comme un concept très important (avec celui quasi symétrique de *cohésion*) pour juger de la qualité d'une conception structurée. Mais le manque de mesures dans leur approche ne pouvait pas durer longtemps, surtout à une époque où on pensait encore que réduire la complexité était une « simple » affaire de mesures. Selon nos connaissances, l'un des premiers cris d'alarme est celui de Joseph Kearney *et autres* paru en 1986 dans *Communications of the ACM* (*Software Complexity Measurement*).

Nous faisons un saut d'une vingtaine d'années pour voir comment ces mêmes concepts sont abordés de nos jours dans la programmation par objets.

6.2. Approche objet

Nous considérerons les mesures proposées par Shyam Chidamber et Chris Kemerer [Chi94] pas tellement parce que, comme les auteurs affirment dans leur conclusion, leur article est « *la première proposition validée d'une métrologie formelle pour la conception orientée objets* » mais parce que plusieurs chercheurs et praticiens ont considéré ces mesures comme un bon corpus de départ.

Dans l'article ils présentent six mesures mais nous ne considérerons que WMC (*Weighted methods per class*), CBO (*Coupling between object classes*) et LOM (*Lack of Cohesion in Methods*)[10].

6.2.1. Définitions

WMC

Cette mesure est définie comme :

$$WMC = \sum c_i$$

où la sommation s'étend sur toutes les méthodes et c_i est la complexité de la ième méthode.

[9] Même si ces considérations sont tirées du chapitre sur la cohésion, elles s'appliquent pratiquement sans changement au couplage.

[10] Les autres trois étant : DIT (*Depth of inheritance tree*), NOC (*Number of children*) et RFC (*Response for a class*).

CBO
Cette mesure est définie comme le compteur du nombre d'autres classes auxquelles la classe est couplée.

LCOM
Soit I_i l'ensemble des variables d'instance employées par la méthode M_i. Soit P l'ensemble des intersections nulles entres I_i et I_j et Q l'ensemble des intersections non vides entres I_i et I_j.

$$\begin{aligned} LCOM &= |P| - |Q|, \text{ si } |P| > |Q| \\ &= 0 \text{ autrement} \end{aligned}$$

C'est-à-dire que LCOM est la différence entre le nombre d'intersections nulle et le nombre d'intersections non vides.

6.2.2. Quelques considérations

Quand ils introduisent WMC, les auteurs ne proposent aucune mesure de complexité pour les méthodes ce qui, selon eux, « *est une force et non une faiblesse* ». C'est une force parce que l'on peut choisir la mesure de complexité qu'on juge la plus apte à représenter la difficulté de modification. Mais, elle est aussi une faiblesse parce qu'elle risque d'être une enveloppe vide (pour ne pas dire un cadeau empoisonné) si les concepts que mesurent les c_i sont douteux. Ce qui est certain c'est que si on pose tous les coefficients c_i égaux à 1, WMC sera égal au nombre des méthodes. Ce qui risque de ne pas dire grand-chose par rapport aux difficultés de maintenance si, par exemple, les *Get* et les *Set* des variables sont mis sur le même plan que des méthodes avec du « vrai » traitement. Encore un exemple de comment les mesures qu'on croit « objectives » ont besoin de beaucoup plus de précision pour pouvoir être appliquées de manière uniforme dans des projets différents. Même si cette mesure ne semble pas faire référence au couplage il nous semble évident que plus la classe est complexe et plus elle risque d'être couplée de manière « complexe » du point de vue cognitif avec les autres classes.

CBO par contre parle explicitement de couplage. Il est clair que plus on a de classes couplées et plus on a de couplage. On pourrait même dire que CBO est la définition de couplage. Mais est-ce que CBO nous permettra de faire des prédictions ? d'évaluer ? de définir ? Nous avons des doutes mais surtout nous aimerions souligner que CBO est un énorme pas en arrière par rapport à la structuration du couplage par Yourdon et Constantine. Les éléments qui permettaient de différencier différentes types de couplages on été absorbés dans une seule catégorie.

Les considérations faites sur CBO valent aussi pour LCOM.

Les mesures de Kemerer considèrent la classe (ou plus précisément ses interfaces) comme l'entité sous mesure. Il s'agit d'une mesure à boîte fermée, des mesures qui ont donc tous les défauts des concepts qu'on applique à des entités sans les « ouvrir ». Imaginons de considérer la classe comme un agrégat de méthodes et de fonctions internes, ce qui est certainement valide autant que de considérer une classe comme un module. Dans ce cas le fait que les méthodes aient une grande cohésion entre elles implique un couplage fort entre les méthodes du point de vue d'un ingénieur du logiciel qui doit modifier la classe. Si, par exemple, trois méthodes et deux fonctions internes ont accès à une variable, qu'est-ce que le couplage entre ces modules sinon un couplage par « variable globale » ? Le couplage le plus dangereux donc. Ce que nous venons de souligner n'est qu'un élément pour montrer que le passage au mesurage n'a pas fait avancer beaucoup le génie logiciel par rapport au couplage empirique des années 1970. Ceci, *repatita iuvant*, parce que on s'est sans doute lancés trop vite dans les prédictions en oubliant de bien établir les points de départ.

6.3. Deux mots sur la conjecture de Boucles d'or (Goldilocks)

Nous ne savons pas si la conjecture dite de Boucles d'or[11] (il y a une dimension optimale des modules : ni trop grands ni trop petits) aura besoin de moins de temps pour être démontrée que celle de Fermat, ce qui est certain, c'est qu'elle est sur la bouche de tous ceux qui parlent de mesures. Le raisonnement qui permet de définir dans [You75] une valeur idéale de la dimension des modules n'est peut-être pas scientifique mais il est assez rigoureux et convaincant, surtout si on a une longue expérience de développement. Même si ce sont des chercheurs très sérieux qui mettent en doute la conjecture avec des données qui montrent que les modules les plus grands sont ceux qui ont la densité de défauts la plus faible, il ne faudrait pas se laisser convaincre trop facilement. Aux critiques de [Fen99] que nous partageons, nous voulons ajouter une considération qui nous semble dégonfler toutes les discussions sur la dimension optimale :
L'absence d'une définition unique et acceptée de « module » rend les comparaisons impossibles. Imaginez qu'il y ait dans une classe une méthode de mille lignes de long et supposez que la densité de défauts soit plus faible que celle des méthodes de longueur plus normale (disons de l'ordre de 83 instructions, pour faire un clin d'œil à ceux qui introduisirent la conjecture de Boucles d'or). Que pouvez-vous induire ? Rien, si vous n'ouvrez pas le module. Imaginez que le module soit composé de dix blocs de code bien séparés avec des commentaires au début et à la fin qui donnent les antécédents et les conséquents… Qu'est-ce que ces blocs de code sinon des modules ? Et sans doute le fait que les petits modules soient l'un après l'autre implique une cohésion au moins séquentielle, ce qui facilite le travail sur le « gros » module qui les contient et qui en fait n'est pas un… module.

7. Objets physiques et concepts

À titre de conclusion nous allons essayer d'introduire quelques éléments de l'ontologie de Bunge appliqués à un projet-jouet pour indiquer une piste de travail qui pourrait réserver de bonnes surprises.

Les considérations que nous avons faites sur le *couplage* et sur le processus de *maintenance* ne font qu'augmenter la cacophonie du domaine en ajoutant du bruit au bruit si on ne les encadre pas dans une structure qui permette d'envisager le GL différemment. Si, dans le premier article, nous avions parlé d'élagage, ici nous allons donner quelques indications d'un possible emploi de l'ontologie de Bunge. Notre proposition n'est qu'une première indication, certainement hâtive, sans doute prétentieuse, de ce que l'on peut essayer de faire en partant de la conceptualisation que Bunge fait du réel. Est-ce que cela permettra de mieux maîtriser le domaine de l'automatisation ? Nous croyons que oui, même si nous sommes conscients que la route est dangereuse et qu'elle pourrait se révéler n'être qu'un cul de sac.

Nous nous appuierons sur une simple et hypothétique application qui devrait répondre au besoin suivant : *Livrer une application qui affiche l'état d'un disjoncteur sur un ordinateur personnel. L'état du disjoncteur est « filé » sur une carte d'entrée numérique de l'ordinateur personnel.*

De la même manière que l'on a simplifié l'application, nous allons simplifier le cycle de vie du logiciel. Nous supposons que, suite à la définition des besoins, seulement deux artefacts ont été produits par les humains :

- *une spécification des exigences logicielles* (SEL) : un objet physique doté de ses propres propriétés[12] et qui contient la description du concept de disjoncteur (DISJ) ayant comme attribut un état de type booléen.
- *une classe* C++ : un objet physique comme la SEL avec la présentation du concept de disjoncteur dans un langage de programmation. Une variable booléenne contient l'état du disjoncteur.

Outre les deux artefacts créés par les humains, quatre autres objets concrets existent :

- *le code exécutable* : l'objet physique que l'ordinateur (son CPU) interprète et exécute et qui, à une certaine adresse mémoire, contient un octet qui lorsqu'il contient 0 indique *ouvert* et lorsqu'il contient une valeur différente de 0 indique *fermé*.

[11] Inspirée par la fable de la petite fille aux cheveux blonds et bouclée qui visite la maison de trois ours, l'un grand, l'un moyen et le dernier petit.
[12] Pour la différence entre propriété et attribut, voir l'encadré « Quelques concepts… ».

- *l'ordinateur* : objet physique qui, pour les besoins de la cause, n'est qu'une mémoire et un CPU. À une certaine adresse mémoire, il contient un octet qui représente l'état du disjoncteur (0 ouvert et une valeur différente de 0 fermé, par exemple).
- *l'écran* : objet physique qu'affiche l'état du disjoncteur (rouge fermé et vert ouvert, par exemple). Cette nouvelle exigence implique que, parmi les propriétés de l'écran, il y ait la possibilité d'avoir au moins deux couleurs.
- *le disjoncteur* : l'objet physique représenté à gauche, dans la figure suivante, par une icône est dont la seule propriété importante pour notre application est son État qui peu être *Ouvert* ou *Fermé*.

La figure 7.1 présente les cinq objets dont nous venons de parler avec des lignes numérotées qui représentent les transformations. Les lignes pointillées représentent les transformations qui demandent une intervention humaine et les lignes continues les transformations qui peuvent être faites automatiquement. Pour simplifier le dessin, nous n'avons pas représenté les humains qui génèrent la SEL et la classe C++, ni les programmes qui génèrent l'exécutable. Il est important de souligner que les cinq objets dont nous venons de parler sont cinq « objets physiques » et non pas des concepts ou des classes.

Ces précisions peuvent sembler banales et, d'un certain point de vue, elles le sont mais, d'un autre point de vue, elles nous permettent de considérer une SEL comme un écran ou comme un disjoncteur : c'est-à-dire qu'elle nous permet de parler de produits physiques indépendamment de leur fonction. À l'aide de cette représentation, nous espérons aussi mieux encadrer l'idée d'une restructuration du modèle de la norme ISO 9126.

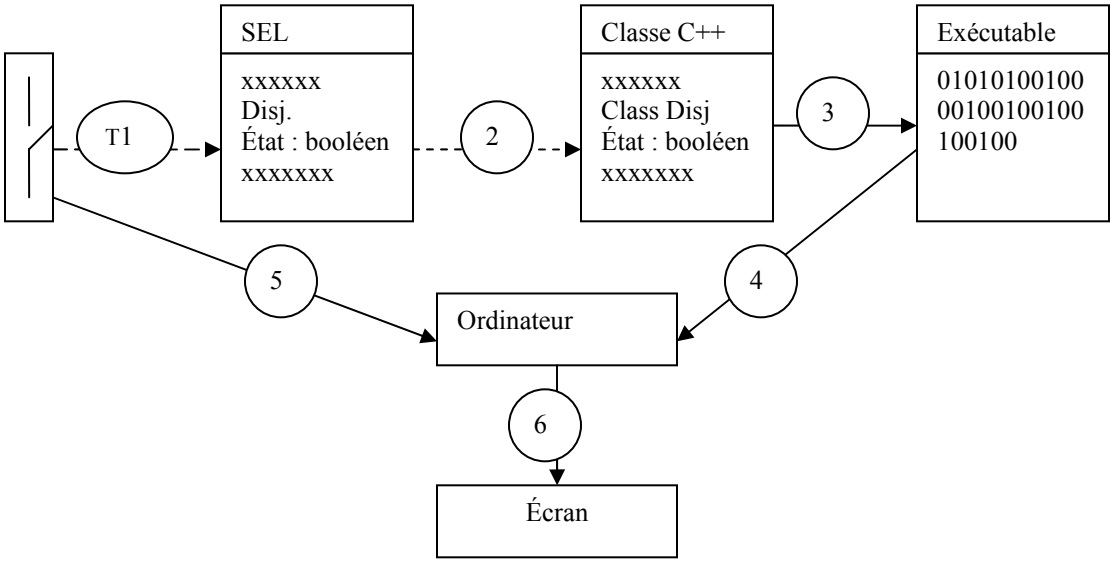

Figure 7.1 : Hiérarchie de QE et QI

Les attributs de la QI du logiciel ne concernent que trois objets : la SEL, la classe C++ et l'exécutable (attention : l'exécutable en tant qu'objet résidant sur un disque et non l'exécutable en exécution ! On pourrait dire l'exécutable et non l'« exécutant »).

Pour parler de la qualité des artefacts, il faut se demander ce qu'est une SEL en tant qu'objet physique, ce qui équivaut à se demander quelles sont ses propriétés. Le nombre de propriétés étant très élevé, il faut garder seulement celles qui sont importantes par rapport au GL et en particulier la maintenabilité. On peut considérer un premier groupe qui est facilement mesurable : le nombre de pages, le nombre de caractères, la langue, les outils employés, les auteurs, le nombre de concepts des domaines décrits, etc. Un deuxième

groupe pourrait être constitué de cinq (5) parmi les huit (8) caractéristiques d'une bonne SEL présentées dans la norme IEEE 830-1998 : traçable, modifiable, vérifiable, consistante, non ambiguë. Ce groupe n'a sans doute pas de mesures directes et, surtout, elles ont des composants subjectifs. Il est clair également que la valeur des mesures des propriétés du deuxième groupe dépendent de la valeur des propriétés du premier groupe. Toutes les propriétés que nous venons de présenter indépendamment du groupe sont des propriétés des artéfacts indépendantes du domaine décrit par la SEL (les mesures du premier groupe sont complètement indépendantes). Est-il possible de parler de la qualité d'une SEL sans parler du domaine ? Question on ne peut pas plus rhétorique !

Si, par exemple, dans la SEL de notre exemple on avait écrit qu'un disjoncteur avait quatre états possibles, on aurait eu beau avoir un document parfait du point de vue des propriétés présentées, il nous aurait conduit à un programme incorrect[13]. On a donc un troisième groupe de caractéristiques de la SEL qui ne concerne pas la SEL en tant qu'objet physique mais la SEL en tant que description des concepts du domaine. Une SEL qui est proche de la référence et dont la signification peut être vérifiée avec les experts du domaine. Ce genre de propriété de la SEL ne doit surtout pas être confondue avec les deux premiers groupes : là il s'agit de propriétés intrinsèques de l'objet physique SEL, ici des propriétés des objets du domaine représentés comme des concepts (des classes) dans la SEL.

Que le concept dans la SEL représente correctement l'objet du monde réel est fondamental (si cela n'est pas le cas, sur notre écran n'apparaîtra jamais le bon état du disjoncteur) mais cela est vérifiable seulement par des humains et sa mesure est théoriquement impossible car il s'agirait d'une mesure de signification.

Ces brèves considérations nous permettent de penser une QI différente de celle de la norme :

- QI de l'artefact indépendant du domaine (qui peut éventuellement être raffinée).
- QI des concepts que l'artefact décrit.

Pour terminer cette tentative de proposer une autre manière de regarder la qualité en s'inspirant lourdement de la différence entre concept et objet physique de Bunge, nous ajoutons deux considérations qui pourraient aider à éclaircir ce que nous venons d'esquisser.

Le processus de contrôle de la configuration n'est concerné que par les objets concrets, par les artefacts, indépendamment de la signification des concepts qu'ils transportent. C'est sans doute là la raison principale qui a permis un tel degré d'automatisation de ce processus. Il est à ce propos intéressant de considérer une mesure de la QI de la norme ISO-9126 : *utilisation de la mémoir*e. La propriété dimension en octets de l'objet physique « exécutable » nous donne la mesure de l'utilisation de la mémoire qui peut être éventuellement utile au processus de contrôle de la configuration pour décider où mettre l'objet physique.

8. Conclusion

Dans cet article, nous avons essayé de montrer que pour sortir de la cacophonie actuelle dans la métrologie de la qualité il faut, d'un part, employer les mesures comme un moyen de clarification conceptuelle avant de penser à des évaluations ou à des prédictions et de l'autre qu'il faut trouver des cadres théoriques assez vastes et solides (comme celui de Bunge, par exemple) qui permettent non seulement d'encadrer les processus de mesurage, mais tout le cycle de vie du logiciel.

Pour ce faire, nous nous sommes appuyés sur la norme ISO 9126 et sur l'historique du concept de couplage.
Dans la section 7, nous avons aussi essayé de regarder la qualité en nous appuyant fortement sur la différence entre objets concrets et concepts.

Étant données les critiques que nous avons faites des « avenues de recherches » dans l'encadré terminologique, nous ne conclurons pas en évoquant de nouvelles avenues, mais en nous demandant, *quoi faire* ? quoi faire, au-delà de la division recherche/pratique.

[13] Il est par contre évident que si la SEL était bien faite de tous les autres points de vue, on l'aurait modifiée plus facilement.

Par exemple :

- Plonger dans le traité de Bunge ;
- Revoir et faire une nouvelle classification des mesures de la norme ISO 9126 ;
- Insister sur les trois différentes possibilités d'emploi des mesures (clarification conceptuelle, prévision, contrôle) ;
- Se fonder sur des modèles statistiques plus sophistiqués ;
- Commencer à normaliser certains termes de manière plus précise que ne le fait la norme IEEE-610 ;
- Essayer de faire tomber les frontières les plus artificielles entre pratique et théorie.
- Élaguer le GL de tout ce qui concerne la sémantique du domaine.

Indications trop abstraites ? sans doute, mais si on ne passe pas par là, on risque de rester sur place pendant long, long temps…

Encadrés

Encadré No 1

Quelques concepts de l'ontologie de Bunge

Individu nu (*Bare*). Les individus nus sont sans propriétés sinon celle de s'associer pour former des individus composés. À titre d'exemple de l'emploi des mathématiques pour construire son ontologie, Bunge définit l'ensemble **S** des individus nus comme étant un *monoïde commutatif d'idempotents* ce qui équivaut à dire qui si x et y sont des individus nus, □ l'élément neutre (le seul élément caractérisé de l'ensemble) et ○ l'opération d'association :

X ○ Y = Y ○ X
X ○ □ = X
X ○ X = X
X ○Y est différent de □ si X ou Y sont différents de □.

Monde. Le monde est un individu tel que tout individu fait partie de lui. À propos du monde, Bunge souligne que « *le monde est un individu qui ne doit pas être confondu avec l'ensemble de tous les individus qui est un concept*. »

Les individus enrichis de propriétés deviennent des **objets**.

Objets physiques et objets conceptuels. « *Pour nous la dichotomie de base de chaque ensemble d'objets ce n'est pas celle entre individus et ensemble mais celle entre objets physiques et objets conceptuels.* » Et, en partant de cette séparation, lorsque Bunge parlera des éléments qui donnent de la chair aux individus, il insistera sur la différence entre les propriétés des objets physiques et les attributs des objets conceptuels. Cette remarque a l'air anodine mais elle est fondamentale pour le GL car, trop souvent, dans ce qu'on a la mauvaise habitude d'appeler modélisation, on confond les concepts avec les objets réels.

Propriété et attributs. Les objets physiques sont dotés de propriétés. Les propriétés n'ont pas d'existence autonome et les objets ne sont pas un simple agglomérat de propriétés. Les attributs des objets conceptuels sont l'équivalent des propriétés pour les objets physiques. Il ne faut donc surtout pas confondre les attributs avec les propriétés, car les attributs impliquent un sujet connaissant tandis que les propriétés ne l'impliquent pas.

Nous espérons aussi que ces quelques éléments mal présentés pousseront certains des lecteurs à étudier en profondeur les œuvres de Bunge[14].

Encadré No 2

Quelques termes

Voici quelques termes ou expressions qu'on retrouve souvent dans les articles à propos de la qualité et des mesures et qui mériteraient d'être sinon pensés et repensés au moins précisés.

Complexité. Ce terme fourre-tout à forte connotation psychologique est défini en IEEE Std 610.12 comme : *le degré de difficulté de compréhension et de vérification d'un système ou d'un composant*. Il n'est pas sûr que la *difficulté de compréhension* et de *vérification* puissent être mises sur le même plan avec nonchalance : cet aplatissement n'est pas sans conséquences sur les difficultés des mesures de complexité. Il est aussi évident qu'en partant d'une telle définition il n'est plus possible d'éliminer les caractéristiques des humains qui sont seuls juges de la difficulté. Avec une telle définition il est clair que toute tentative de réduire le calcul de la complexité à une formule est vouée à l'échec à moins d'opter pour une épistémologie opérationnaliste et dire que la formule est la définition.

Qualité. Il n'y a pratiquement pas d'article ou de livre qui ne souligne pas l'ambiguïté du terme. Nous n'ajouterons pas ambiguïté à ambiguïté.

Erreur (*Error*), **défaut** (*fault* ou *defect*) et **panne** (*failure*). Ces termes sont employés de façon très libre et confuse surtout quand on les mesure en les rapportant au temps ou aux lignes de code. En s'inspirant de la norme IEEE 982, on peut définir l'erreur comme une action humaine dans la chaîne de production du logiciel qui cause un défaut qui, à son tour, peut causer une panne. Il faut aussi ajouter qu'il y a des erreurs humaines (celles de l'utilisateur final) qui causent des pannes sans qu'il y ait de défaut logiciel, à moins d'interpréter défaut dans un sens plus large et d'y inclure un manque de robustesse.

Maintenance. Il est désormais classique de considérer la maintenance comme constituée de trois parties : une qui concerne la correction des défauts[15] (corrective), une autre qui concerne le changement d'environnement (adaptative) et une troisième qui consiste en l'ajout de nouvelles fonctionnalités (perfective). Nous sommes loin de penser que la maintenance ne soit pas importante, vu que, dans notre article précédent, nous avons introduit un principe qui affirme que « *La maintenabilité est la qualité intrinsèque principale pour toute approche d'ingénierie au développement du logiciel* », mais il est clair que les trois composantes sont tellement différentes qu'il vaudrait la peine de se pencher sur l'utilité d'un concept de maintenance qui chapeaute le tout. Il est clair que ce principe s'applique à la maintenance perfective, qui est très difficilement séparable du développement « normal », si ce n'est du point de vue organisationnel. En considérant les mesures proposées par ISO 9126, on s'aperçoit que pratiquement toutes les mesures ne s'appliquent qu'à la maintenance corrective.

Avenue de recherche. Cela fait partie de l'orthodoxie que de terminer un article en présentant des avenues (des pistes pour les plus humbles) de recherche. S'il est vrai que, par définition, la recherche doit être spécialisée et donc s'attaquer à des détails qui semblent sans intérêt, il ne reste par contre pas moins vrai que les pistes deviennent des labyrinthes inutiles s'il n'existe pas quelques Ariane avec leurs écheveaux.

Ceteris paribus. Innombrables sont les *ceteris paribus* qui agrémentent les articles sur les mesures mais,

[14] Tâche que nous-mêmes sommes très loin d'avoir accomplie.
[15] Ou des erreurs ?

à moins de préciser quels sont les *ceteri,* on reste à un niveau intéressant du point de vue rhétorique mais beaucoup moins intéressant du point de vue scientifique. *Ceteris paribus* peut être employé sans trop de peur dans des disciplines comme la physique où il existe un cadre conceptuel solide mais, dans ces débuts du GL, l'expression n'est d'aucune utilité pour ceux qui aimeraient refaire les expériences.

Créativité. Dans pratiquement tous les textes, il y a une courte référence à la créativité des individus qui entraîne des difficultés dans la proposition de mesures objectives, mais cette courte référence est sans conséquence sur le reste. Est-ce parce qu'il est impossible de l'intégrer ou est-ce que l'intégration demande un autre cadre conceptuel ?

Module. Selon la deuxième définition du dictionnaire d'IEEE « U*ne partie logiquement séparée d'un programme* ». Définition tellement générale qu'on peut en faire et on en fait ce que l'on veut. Mais cela n'est pas sans conséquences sur l'interprétation des mesures. Est-il vrai que pour certains auteurs « module » est l'équivalent de « fichier » ? Nous avons l'impression que oui. Une classe est-elle un module ? et une méthode ? et une partie d'une méthode bien isolée avec un nom en commentaire ? Nous croyons qu'actuellement la définition de modules de [You75] est encore celle qui pourrait être la plus intéressante : « *Une séquence d'énoncés contigus du point de vue lexical, délimitée par des éléments frontaliers et ayant un identificateur d'agrégat* ».

Encadré No 3

Dialogue plein de bon sens entre un « quantitatif » et un « qualitatif » à propos de la complexité cyclomatique.

Selon la norme IEEE 982, « *Cette mesure est employée pour déterminer la complexité structurelle du code d'un module. L'emploi de cette mesure est conçu pour limiter la complexité d'un module, le rendant ainsi plus facile à comprendre* » et peut être obtenue en additionnant le nombre des sommets de partage (sélections et itéractions)[16]. Cette mesure donne une indication de la qualité interne du module (complexité structurelle) qui est censée influencer la qualité externe (maintenabilité). Toujours dans la même norme, il est indiqué que « *pour un module typique, une complexité maximum idéal est 10 (…).* »

QUALITATIF : Tu devrais m'expliquer quelque chose. Pour toi c'est sans doute très simple mais, pour moi, c'est complètement flou : qu'est-ce que la complexité ?
QUANTITATIF : Simple. Je cite par cœur la définition du Std. IEEE 610.12 : « Le degré de difficulté de compréhension et de vérification de la conception ou de la mise en œuvre d'un système ou d'un composant. » C'est assez clair, n'est-ce pas ?
QUALITATIF : C'est une difficulté d'ordre cognitif : elle donne une indication de la difficulté qu'un humain a à saisir quelque chose.
QUANTITATIF : Parfaitement dit !
QUALITATIF : Merci, mais comment mesures-tu la complexité ?
QUANTITATIF : On vient de le voir. En considérant le nombre de sommets, etc… en peu de mots, avec la CC (complexité cyclomatique).
QUALITATIF : Ce qui veut dire qu'on a prouvé expérimentalement qu'il y a une loi qui relie le nombre de sélections dans un programme avec la difficulté psychologique de comprendre.
QUANTITATIF : Je ne connais pas de références précises, mais cela me semble évident. Il est bien plus facile de comprendre un programme sans IF qu'un programme avec une trentaine d'IF imbriqués !
QUALITATIF : Oui
QUANTITATIF : Oui.
QUALITATIF : Oui mais, étant donné qu'un module est en relation avec d'autres modules, il est possible qu'un module hypercomplexe « absorbe » toute la complexité et permette ainsi d'avoir une complexité au niveau du système plus faible.
QUANTITATIF : Oui. C'est possible. Mais il ne faut pas faire dire à la CC plus que ce qu'elle veut dire. Et puis si tu veux que je te dise le fond de ma pensée, la CC est surtout utile pour déterminer le nombre minimal de cas à tester pour une unité.
QUALITATIF : Est-ce une autre mesure de la norme 982 ?

QUANTITATIF : Oui, la A17.
QUALITATIF : Je ne suis pas très difficile à convaincre, quand les choses sont simples. Imaginons que je veuille connaître la CC d'une méthode d'une classe, que cette classe commence avec une séquence d'une dizaine d'instructions, qu'il y ait ensuite un *While* avec en son intérieur une séquence de 50 instructions. Puisque j'ai un seul *While,* ma CC vaut 2. Ce qui veut dire que cette méthode a la même complexité qu'une méthode avec une instruction suivie d'un *While* qui contient à son tour une seule instruction !
QUANTITATIF : Non ! Elle n'a pas la même complexité, elle a la même CC !
QUALITATIF : Donc la complexité d'une méthode est quelque chose de différent de la CC.
QUANTITATIF : Certes. La CC donne une indication de la complexité structurelle de ta méthode. Imagine que tu introduises un IF à l'intérieur de ton *While*, ta complexité augment de 1.
QUALITATIF : Si j'ajoute un autre IF, elle augmente encore de 1. J'ai bien compris. Mais… mais… ta CC ne tient pas compte des niveaux d'imbrication. Selon toi, les deux programmes suivants ont-ils la même complexité ?

Programme avec deux IF non imbriqués	**Programme avec deux IF imbriqués**

BEGIN
A := 87
IF A > x
  ALORS A := 37
IF A < y
  ALORS A := 16
END
	BEGIN
	A := 87
	IF A > x
	  ALORS A := 37
	  SINON **IF** A < y
	  ALORS A := 16
	END

QUANTITATIF : Je comprends où tu veux en venir. Tu veux me faire dire que le programme avec les IF imbriqués est plus complexe du point de vue cognitif et que la CC ne le montre pas.
QUALITATIF : Tu as très bien saisi. Mais, moi aussi je crois d'avoir bien saisi. La CC a une certaine corrélation avec la complexité structurelle qui, à son tour, a une certaine corrélation avec la complexité cognitive.
QUANTITATIF : Parfait ! Ce n'est pas la solution de tous nos problèmes de complexité mais… Mais, c'est un petit pas en avant.
QUALITATIF : D'accord. D'accord si on reste avec les pieds sur terre et qu'on ne se laisse pas prendre par la fascination des nombres.

Encadré No 4

La physique et le GL

La physique pour le GL, comme pour beaucoup de « quantitatifs » dans les sciences humaines, est un modèle de clarté et de rigueur qui inspire, qui est inatteignable et, parce qu'inatteignable, inconnu. Étant inconnue, elle est employée comme une arme pour défendre ses préjugés sans que les réactions des autres, aussi « ignorants », soient à craindre. Nous donnerons ici un exemple qui nous semble particulièrement dangereux.
On s'appuie sur les conférences de Feymann pour dire qu'il faut se méfier de l'intuition, de l'expérience et du bon sens : comprendre, dans le domaine sacré de la mécanique quantique, signifie abandonner les schémas de pensée qui nous guident dans la vie quotidienne pour s'abandonner au rythme enivrant des équations différentielles. Ceux qui tiennent de tels propos oublient, les pauvres, que les artefacts produits tout au long du CVL n'ont rien de microscopique, qu'ils ne sont pas gérés par une quelconque équation de Schrödinger. Les artefacts, contrairement au spin des particules, appartiennent au monde de notre perception. Cela fait très savant, quand on a du mal à faire concorder les chiffres avec les données de notre expérience, de jeter l'expérience par-dessus bord… mais cette désinvolture n'est pas sans risques !

Encadré No 5

Six références commentées.

[Bri59] Percy W. Bridgman*, The Way Things Are,* Harvard University Press, 1959. Le regard que ce prix Nobel de la physique, chef de file de l'opérationnalisme, jette sur le monde est étonnant par son efficacité et son humilité. L'insistance sur le fait que les concepts, même ceux qui semblent les plus clairs, peuvent se dissoudre face à des mesures ne peut qu'être d'une très grande utilité en génie logiciel. La lecture des œuvres de Bridgman pourrait pousser « Qualiticiens » et « métrologistes » à faire quelques pas en arrière pour mieux sauter : c'est-à-dire à mieux considérer l'humble aspect des mesures qui aide à clarifier les concepts plutôt que de se lancer hâtivement dans la prédiction.

[Bun77] Mario Bunge*, Treatise on Basic Philosophy, Vol. 3 Ontology I,* Reidel, 1977. Après s'être libéré en quelques pages de pratiquement toutes les approches traditionnelles à l'ontologie, Bunge construit au fil du texte une ontologie exacte et scientifique fondée sur les mathématiques. Peu importe qu'on soit d'accord ou non avec sa vision philosophique du monde, il est certain que, dans ce livre, il a une vision du rapport entre concepts et choses concrètes et une définition des propriétés des objets qui peuvent se révéler fort utiles en génie logiciel. Même si ce texte est l'un des plus cités dans les articles de recherche sur la qualité et les mesures, il n'est sans doute pas inutile de se demander si l'on en n'a extrait tout ce qu'on pouvait, assez facilement, en extraire.

[Chi94] Shyam Chidamber, Chris Kemerer*, IEEE Transaction on Software Engineering,* Vol. 20, No 6 « A Metric Suite for Object Oriented design »*.* Un article qui est devenu un classique dans les mesures de la programmation par objets et qui, malgré les critiques, reste une lecture incontournable. Après une présentation très concise des éléments de la théorie de Mario Bunge dont ils se servent comme base théorique, les auteurs présentent six mesures pour chacune desquelles, après la définition, ils présentent la base théorique, une évaluation analytique selon la liste des propriétés proposées par Weyuker dans un article de *Transaction on Software Engineering* de 1988, des données empiriques et leur interprétation.

[ISO 01] ISO/IEC, *International Standard 9126-1*, *Software engineering – Product quality, Part 1: Quality mode*, 2001. Une courte norme qui se lit très bien, avec un modèle de qualité assez simple et qui fixe une terminologie qui peut faciliter les échanges entre praticiens et chercheurs. Trois rapports de recherches la complètent : le premier (TR 9126-2) présente 112 mesures pour la qualité du logiciel exécuté dans un environnement de test, le deuxième (TR 9126-3) présente 70 mesures pour la qualité des artefacts et le troisième (TR 9126-4) présente 15 mesures pour la qualité de fonctionnement dans l'environnement de l'utilisateur final.

[Fen 99] Norman E. Fenton, Martin Neil, *IEEE Transaction on Software Engineering,* Vol. 25, No 5 « A Critique of Software Defect Prediction Models »*.* Avec une lucidité et une maîtrise du domaine impressionnante les auteure mettent en évidence les points faibles de pratiquement tous les modèles connus qui devraient aider à prédire les défauts qui restent à découvrir dans un programme. Il ne s'agit pas de critiques de détail mais de critiques qui touchent aux fondements des approches plus connues à la prédiction des défauts. L'article se termine en proposant une approche moins simpliste soutenue, du point de vue statistique, par les *Réseaux de croyance baysiens*.

[You75] Edward Yourdon, Larry Constantine, *Structured Design*, Yourdon Press, 1975. Un livre qui, à cause de ses fondements pratiques et du « bon sens » des auteurs, fruit d'expériences réfléchies, continue à garder un grand intérêt même s'il s'agit de conception structurée. L'ironie et le brio permettent aux lecteurs d'entrer dans des thèmes « sérieux » avec une facilité dont ils seront les premiers à s'étonner.