ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
M. Ing.

PAR
BERNARD GUAY

ANALYSE COMPARATIVE DU GUIDE SWEBOK ET DES PRINCIPES FONDAMENTAUX DU GÉNIE LOGICIEL

MONTRÉAL, LE 7 JUILLET 2004

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Pierre Bourque, directeur de mémoire
Département de génie électrique à l'École de technologie supérieure

M. Alain Abran, président du jury
Département de génie électrique à l'École de technologie supérieure

M. George Wilkie, membre du jury
University of Ulster, Northern Ireland, United Kingdom

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 2 JUIN 2004

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ANALYSE COMPARATIVE DU GUIDE SWEBOK ET DES PRINCIPES FONDAMENTAUX DU GÉNIE LOGICIEL

Bernard Guay

RÉSUMÉ

Le génie logiciel est une discipline récente et il y a maintenant un besoin d'améliorer ses bases pour encadrer son évolution et atteindre un certain niveau de maturité. Cette analyse comparative est motivée par l'amélioration de ces bases. Elle utilise la liste des principes fondamentaux suggérés dans l'article « Fundamental Principles of Software Engineering – A Journey » et présente une méthodologie aidant à analyser le niveau et l'étendue de la correspondance d'un principe fondamental par rapport aux connaissances généralement reconnues du « Guide to the Software Engineering Body of Knowledge » (Guide SWEBOK). Les résultats de l'analyse facilitent l'identification de notions permettant de produire une description d'accompagnement des principes fondamentaux de l'étude, de présenter les citations justifiant la présence de ces notions et de produire les descriptions d'accompagnement. Les résultats de l'analyse permettent aussi d'identifier des pistes pour améliorer le contenu du Guide SWEBOK ou les connaissances de la discipline.

ANALYSE COMPARATIVE DU GUIDE SWEBOK ET DES PRINCIPES FONDAMENTAUX DU GÉNIE LOGICIEL

Bernard Guay

SOMMAIRE

Le génie logiciel est une discipline récente et il y a maintenant un besoin d'améliorer ses bases pour encadrer son évolution et atteindre un certain niveau de maturité. Cette analyse comparative est motivée par l'amélioration de ces bases. Elle utilise la liste des principes fondamentaux suggérés dans l'article « Fundamental Principles of Software Engineering – A Journey » et présente une méthodologie aidant à analyser le niveau et l'étendue de la correspondance d'un principe fondamental par rapport aux connaissances généralement reconnues du « Guide to the Software Engineering Body of Knowledge » (Guide SWEBOK). Les résultats de l'analyse facilitent l'identification de notions permettant de produire une description d'accompagnement des principes fondamentaux de l'étude, de présenter les citations justifiant la présence de ces notions et de produire les descriptions d'accompagnement. Les résultats de l'analyse permettent aussi d'identifier des pistes pour améliorer le contenu du Guide SWEBOK ou les connaissances de la discipline, pour les pistes d'améliorations identifiées qui n'ont pas trait aux connaissances déjà généralement reconnues.

COMPARATIVE ANALYSIS BETWEEN THE GUIDE TO THE SWEBOK AND THE FUNDAMENTAL PRINCIPLES OF SOFTWARE ENGINEERING

Bernard Guay

ABSTRACT

Software engineering is a relatively novel discipline and there is still a need to improve upon its foundations in order to promote its evolution and foster its maturity. The motivation behind this analysis is the clarification of those foundations. The current analysis is based on the fundamental principals list as suggested by the article "Fundamental Principles of Software Engineering – A Journey" and utilizes a methodology allowing for the analysis of the correlation between the level and the extent of a fundamental principle and the generally accepted industry knowledge as stated by the Guide to the *Software Engineering Body of Knowledge* (Guide to the *SWEBOK*). The results extrapolated from this analysis allow for the identification of notions to be included in the fundamental principle's complementary description as well as the quotation, which illustrates these notions. Furthermore, those same results point to possible avenues of improvement to the Guide or to the body of knowledge of the discipline since those avenues are not yet integral part or the generally accepted knowledge of the discipline.

REMERCIEMENTS

Je voudrais remercier Pierre Bourque de m'avoir offert ce projet et pour les innombrables heures qu'il a dû y consacrer à lire et relire ces centaines de pages. Il m'a aussi enseigné comment ajouter de la rigueur à un travail d'une telle envergure.

Je voudrais remercier le IEEE de permettre la réalisation d'une telle étude à la seule présentation de cette notice concernant les droits d'auteur dans le Guide to the Software Engineering Body of Knowledge :

> "Copyright and Reprint Permissions: This document may be copied, in whole or in part, in any form or by any means, as is, or with alterations, provided that (1) alterations are clearly marked as alterations and (2) this copyright notice is included unmodified in any copy. Any other use or distribution of this document is prohibited without the prior express permission of IEEE."

Je voudrais aussi remercier Nicole pour m'avoir poussé à aller jusqu'au bout, et de s'être occupée des enfants durant tous ces mois où j'ai dû m'enfermer dans mon bureau. Je remercie aussi Émilie et Maxime pour leur compréhension vis-à-vis un père qui était moins présent pour eux au cours de la dernière année.

Finalement, je voudrais remercier Denise Huot et Martine Buczinsky qui ont consacré de nombreuses heures à corriger mes textes.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ACRONYMES

Acronyme	Description
ACM	Association for Computer Machinery
AQL	Assurance qualité du logiciel
CCB	Configuration Control Board
CH	Chapitre
CMM	Capability Maturity Model
FIG	Figure
IEEE	Institute of Electrical and Electronics Engineers
KA	Knowledge Area
P	Page
PEB	Process Experience Base
PF	Principe fondamental
PMBOK	Project Management Body Of Knowledge
PMI	Project Management Institute
S	Section
SC	Software Construction
SCI	Software Configuration Item
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SCR	Software Change Request
SCSA	Software Configuration Status Accounting
SD	Software Design
SEM	Software Engineering Management
SEP	Software Engineering Process
SEPG	Software Engineering Process Group
SETM	Software Engineering Tools and Methods

SM	Software Maintenance
SQ	Software Quality
SQA	Software Quality Assurance
SR	Software Requirements
SRS	Software Requirements Specification
ST	Software Testing
SWEBOK	Software Engineering Body Of Knowledge
V&V	Verification and Validation

INTRODUCTION

Le standard IEEE 610.12 [1] de l'Institute of Electrical and Electronics Engineers (IEEE) définit le génie logiciel comme étant l'application d'une approche systématique, disciplinée et quantifiable pour le développement, l'opération et la maintenance du logiciel; c'est-à-dire l'application du génie au logiciel. Mais, le génie logiciel est une discipline encore jeune et il existe encore des zones grises dans sa reconnaissance comme discipline du génie. Comme le souligne Gabrini [2], lorsqu'il est question de développement de logiciel, il y a manipulation d'idées abstraites et le logiciel n'est pas aussi contraint que les ponts par les lois de la physique et du monde qui l'entoure. Il y a aussi Kruth [3] qui suggère que le développement de logiciel est plus un art qu'une science.

En fait, la reconnaissance du génie logiciel comme discipline du génie n'est que toute récente [4] et exige que des efforts significatifs soient investis pour en améliorer les bases. En particulier, un besoin a été identifié pour découvrir, et graduellement reconnaître, des principes spécifiques au domaine qui doivent devenir l'un des fondements du génie logiciel [5]. L'identification des connaissances généralement reconnues en génie logiciel a déjà produit un ouvrage présentant les différents domaines de connaissances de la discipline [6]. Par contre, en raison de la dynamique des changements techniques et technologiques, et de la jeunesse de la discipline, les résultats obtenus de cette identification doivent continuellement être révisés pour refléter la réalité du temps. Cette base en mouvement de la discipline confirme ce besoin d'effectuer des travaux de recherche pour définir ce qui doit constituer la base de la discipline, pour en faciliter l'évolution et la maturation.

Les principes fondamentaux sont l'une des pièces maîtresses qui doivent composer cette base de la discipline. Comme le présente Bourque et al. [5], l'identification des principes fondamentaux du génie logiciel est influencée par les principes fondamentaux

plus généraux s'appliquant à l'ensemble des disciplines du génie. Les principes fondamentaux du génie logiciel soutiennent à leur tour les pratiques et les normes de la discipline.

Mais, il faut comprendre que le génie logiciel ne possède pas actuellement une liste de principes fondamentaux reconnue par l'ensemble des intervenants de la discipline. Celle-ci (la liste) n'étant pas reconnue, la discipline repose sur une base en mouvement. Certains ouvrages ont été écrits dans le but d'identifier des principes devant faire partie de cette liste. Ainsi, des principes fondamentaux candidats ont été suggérés par Bourque et al. [5] pour répondre à cette préoccupation. Ceci et d'autres ouvrages identifiés dans Séguin [7], représentent une avancée importante et suggèrent davantage d'investigation et de validation dans la discipline du génie logiciel pour définir une liste de principes fondamentaux reconnus.

Tel que le décrit Séguin [7], en référence aux bases du génie logiciel, l'établissement d'une liste de principes fondamentaux du génie logiciel permettrait d'obtenir un cadre d'analyse pour les normes et les pratiques, et faciliterait leur reconnaissance par la discipline. Il existe déjà des normes qui ont été définies par différents organismes et le corpus de connaissances du génie logiciel fait l'objet d'un consensus d'envergure international. Le « Guide to the Software Engineering Body of Knowledge » (Guide SWEBOK) [6] constitue ce consensus et peut servir de base de comparaison pour réaliser une validation préliminaire des principes fondamentaux identifiés en [5] et pour effectuer certains ajustements à cette liste. Lorsqu'une liste de principes fondamentaux aura été reconnue par la discipline, il sera possible d'en vérifier l'application dans les pratiques et les normes existantes pour identifier des améliorations possibles à celles-ci. C'est au niveau de la validation préliminaire des principes fondamentaux que se situe l'analyse présentée dans ce document.

Pour réaliser notre étude, nous utiliserons le cadre de Basili [8] adapté pour la recherche exploratoire en génie logiciel. Le premier chapitre du document présente cette adaptation. Chacun des chapitres suivants est structuré pour refléter les différentes phases du cadre de Basili alignées sur les objectifs de l'étude. Le chapitre 2 présente la définition du projet et la chapitre 3, la planification de l'étude. Le chapitre 4 contient le détail des activités d'analyse constituant le cœur du projet. Au cours de ces activités, nous comparons les quinze (15) principes fondamentaux identifiés par Bourque et al. [5], par rapport aux connaissances généralement reconnues de la discipline du Guide SWEBOK [6], pour :

a. évaluer le niveau et l'étendue de la correspondance de ces principes par rapport aux connaissances généralement reconnues;
b. identifier les notions dans les connaissances généralement reconnues pouvant servir de base à la production d'une description d'accompagnement pour chacun des principes fondamentaux;
c. produire les descriptions d'accompagnement pour les principes fondamentaux de Bourque et al. [5];
d. identifier des pistes d'améliorations au Guide SWEBOK ou à la discipline.

Le chapitre 5 présente l'interprétation des résultats en fonction du contexte dans lequel l'étude a été produite. Nous y présentons aussi les travaux qui pourront être réalisés à l'aide des résultats de l'étude ou pour en améliorer la qualité. Nous complétons en présentant nos conclusions dans le chapitre 6.

Tout au long du document, nous allons présenter des extraits des deux références de l'étude (Bourque et al. [5] et le Guide SWEBOK [6]) en conservant la langue anglaise, pour ainsi éviter les erreurs d'interprétation qui pourraient survenir lors de la traduction des extraits.

CHAPITRE 1

MÉTHODE DE RECHERCHE

Pour mieux aborder les problèmes reliés à la recherche exploratoire, une adaptation du cadre de Basili [8], pour la recherche expérimentale en génie logiciel, est utilisée pour structurer notre projet. Le cadre de Basili a été proposé par Basili, V. et .R, Selby, et D. Hutchens en juillet 1986. Autant le cadre original que l'adaptation pour la recherche expérimentale a fait ses preuves (Bourque et Côté [9], Brouillette [10], Belkebir [11], Abran, Bourque et Laframboise [12]) et permet de baliser le travail de recherche en génie logiciel, sa présentation et l'échange de ses résultats. Cette étude est une recherche exploratoire du fait que les problèmes soulevés par le projet ne sont pas bien connus et qu'il y a peu d'informations disponibles sur des études similaires. Une liste de ces études est identifiée dans Séguin [7].

Le cadre de Basili voit à répondre aux préoccupations de quête de données sans avoir défini correctement ce qui doit être mesuré; d'interprétation statistique sans vraiment savoir ce qui est recherché; et d'interprétation partielle des résultats. Il permet de bien définir le but et les objectifs de l'étude; de réfléchir à l'implantation du processus qui sera suivi au cours de l'exécution de l'étude; de définir les données à récolter en identifiant les mesures pertinentes à l'étude; et de valider les données récoltées avant de passer à l'analyse de celles-ci. Il voit à baser l'étude sur une méthode scientifique d'analyse.

Le tableau I présente l'adaptation de la structure du cadre de Basili qui est divisée en 4 phases. Les adaptations, à l'intérieur de chaque phase, ont été réalisées du fait que les étapes reliées à la conception des échantillons et aux analyses statistiques pour la collecte formelle des données des analyses statistiques, ne s'appliquent pas dans les recherches exploratoires comme la nôtre.

Tableau I

Structure pour la recherche exploratoire en génie logiciel
(adapté de Basili, Selby et Hutchens [8])

I. Définition du projet			
Motivation	Objet	Objectif	Utilisateurs
II. Planification du projet			
Étapes du projet		Intrants	Biens Livrables
III. Exécution			
Développement des livrables		Révision par les experts	Amélioration des livrables
IV. Interprétation			
Contexte d'interprétation		Extrapolation	Travaux subséquents

1.1 Définition du projet

La phase de définition comprend les 4 composants suivants :

a. la motivation;
b. l'objet;
c. l'objectif;
d. les utilisateurs.

La motivation est la raison qui pousse l'auteur à entreprendre le projet. Elle identifie la grande question à laquelle doit répondre l'étude. L'objet définit l'entité principale à être étudiée. L'objectif est le but précis du projet; il constitue le véritable problème à résoudre. Les utilisateurs sont les personnes susceptibles d'utiliser les résultats de la recherche.

1.2 Planification du projet

La phase de planification définit les 3 composants suivants :

a. les étapes du projet;
b. les intrants;
c. les livrables.

La planification du projet voit à présenter la démarche suivie au cours de l'étude. Un plan de projet est développé, identifiant les étapes du projet et la façon dont sera exécutée chacune de ces étapes. Les intrants et les produits des différentes étapes du projet, y sont aussi identifiés. Ils peuvent être composés de littérature scientifique ou de rapports sur les expériences connues dans l'industrie. Les biens livrables doivent être clairement définis pour bien gérer les attentes des divers intervenants.

1.3 Exécution du projet

La phase d'exécution comprend les 3 étapes suivantes :

a. le développement des biens livrables;
b. la révision par les experts;
c. l'amélioration des livrables.

Les étapes du projet identifiées dans le plan de projet sont réalisées dans la phase d'exécution où les livrables proposés sont développés, en se basant sur une revue approfondie de la littérature. La qualité de ces livrables est ensuite améliorée grâce aux révisions des experts provenant du monde académique et de l'industrie.

La révision par les experts et l'amélioration des livrables ne seront pas exécutées au cours de cette étude. Les résultats seront utilisés dans le cadre d'un atelier avec différents intervenants de la discipline, après la livraison de l'étude.

1.4 Interprétation

La phase d'interprétation comprend les 3 composants suivants :

a. le contexte d'interprétation;
b. l'extrapolation;
c. les travaux subséquents.

L'interprétation des résultats du projet constitue la dernière phase du projet. La section du contexte d'interprétation permet de vérifier si les objectifs identifiés dans la phase de définition du projet, ont été atteints, en utilisant les résultats de la recherche. On y présente aussi les particularités de l'étude qui ont pu être identifiées au cours de la phase d'exécution, et qui amèneront, entre autres, à la présentation de l'extrapolation des résultats. Dans l'extrapolation, les résultats de la recherche sont étudiés pour voir s'ils

peuvent être utilisés dans d'autres circonstances. Dans les travaux subséquents, la réalisation de recherches plus poussées ou des travaux d'application dans le même domaine d'étude, sont discutés.

CHAPITRE 2

DÉFINITION DU PROJET

2.1 Motivation

Les chercheurs de la discipline travaillent à définir des bases solides au génie logiciel pour assurer une pleine reconnaissance de la discipline comme discipline du génie. Plusieurs spécialistes du domaine ont produit des ouvrages tentant d'identifier des principes fondamentaux qui constitueraient une partie de cette base. Notre étude s'intègre dans les efforts d'amélioration des bases de la discipline en s'intéressant tout particulièrement au lien entre les connaissances généralement reconnues du Guide SWEBOK [6] et les travaux de Bourque et al. [5], pour l'identification d'une liste de principes fondamentaux du génie logiciel. Elle est motivée par l'amélioration de la base de la discipline du génie logiciel pour qu'elle soit pleinement reconnue comme discipline du génie.

2.1.1 Principes fondamentaux

Pour bien saisir ce qu'est un principe fondamental, il faut comprendre les deux termes, « principes » et « fondamentaux ». Le Petit Larousse [13] définit ainsi « principe » : « *Connaissance, règle élémentaire d'une science, d'un art, d'une technique, etc.* »; et « fondamental » : « *Qui est à la base; qui se rapporte à l'essentiel* ». Le mot « fondamentaux » pris seul se définit comme suit : « *Principes, idées constituant le fondement et l'essence d'une science, d'une doctrine, d'un art, etc.* ».

Ainsi, une combinaison valable donnerait une définition de « principes fondamentaux » du génie logiciel ressemblant à ceci : « *Connaissances, règles élémentaires constituant le fondement et l'essence du génie logiciel* ». Les principes fondamentaux doivent donc se retrouver à la base de l'ensemble de la discipline. Ils constituent la portion de la

discipline la plus stable à travers le temps. Mais, la discipline du génie logiciel est encore jeune et gagne encore en maturité d'année en année. Des normes sont définies par différents organismes permettant de systématiser les pratiques des professionnels de l'industrie. Par contre, la façon dont évoluent les normes, fait en sorte qu'elles ne sont pas toujours intégrées et qu'il peut être difficile de s'y retrouver pour les praticiens en entreprise. Moore [14] exprime bien cette difficulté :

> « Today's user of software engineering standards is presented with a bewildering array of choices, more than 300 standards developed and maintained by more than 50 different organizations, all approaching software engineering from different viewpoints and contexts. The standards are inconsistent, overlapping, and occasionally contradictory. »

Il manque à la discipline du génie logiciel un ensemble de principes scientifiques et d'ingénierie qui, comme pour les autres disciplines du génie où les principes sont souvent d'ordre physique, permettrait de fournir des balises à l'établissement des pratiques et des normes. Dans la figure 1, Bourque et al. [5] présente les principes fondamentaux comme cette base de la discipline. Cette figure identifie les rôles des principes fondamentaux du génie logiciel. Ces derniers sont influencés par des principes fondamentaux plus généraux s'appliquant à l'ensemble des disciplines du génie. Ils soutiennent à leur tour les pratiques et les normes de la discipline.

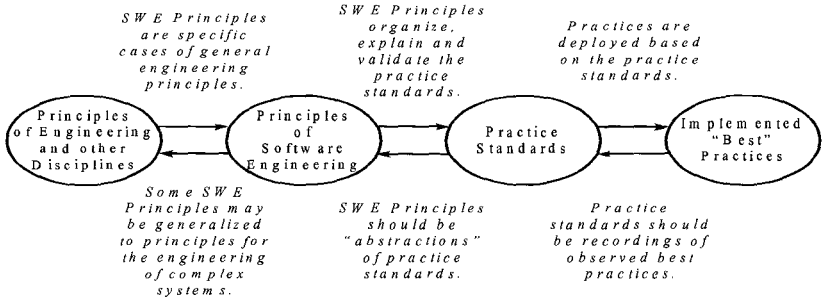


Figure 1 Liens entres les principes fondamentaux et les pratiques
(Bourque et al. [5] p.61)

Les liens ne sont pas unidirectionnels car la discipline est influencée à la fois par les changements à la base du génie (à gauche) que par l'évolution des technologies et des pratiques (à droite). Les pratiques ont d'ailleurs plus de chance d'être affectées par les changements technologiques que par les principes. Les changements aux pratiques affecteront ou non les normes, et à la limite, pourront influencer les bases de la discipline.

L'évolution de la technologie n'a pas toujours un effet de changement sur les pratiques et les normes, et par ricochet, sur les principes fondamentaux du génie logiciel. Dans le mouvement de transition de la discipline (déplacement de droite à gauche dans la figure 1), l'effet d'un changement technologique s'amenuise au passage d'une bulle vers une autre du fait que les éléments de gauche sont beaucoup plus stables dans le temps que ceux de droite.

La création d'une liste de principes fondamentaux reconnue doit donc être une priorité pour répondre aux besoins de stabiliser les bases de la discipline et d'assurer une intégration cohérente des nouvelles technologies, des pratiques et des normes dans la

discipline. Séguin [7], dans sa proposition de recherche, présente une revue de la littérature portant sur les principes fondamentaux du génie logiciel. Ces ouvrages couvrent environ les trente-cinq (35) dernières années de la discipline, de Winston Royce [15] en 1970, à Ghezzi et al. [16] en 2003. La liste produite par Bourque et al. [5], l'un des deux ouvrages à la base de notre étude, fait partie de cette revue.

Pour mieux situer les travaux de Bourque et al. [5] dans les ouvrages identifiés par Séguin [7], nous présentons ici les autres écrits de cette revue de la littérature. Nous y ajoutons les éléments importants soulevés par Séguin [7] lors de son analyse, par rapport à l'identification des principes fondamentaux présentés dans les différents ouvrages :

a. Royce [15] propose cinq (5) règles qui devraient être suivies pour avoir du succès dans les projets de grande envergure.
b. Ross et al. [17] identifient sept (7) principes du génie logiciel qu'ils affirment faisant l'objet d'un consensus auprès de la communauté du génie logiciel.
c. Mills [18] identifie deux (2) principes s'appliquant à la conception. Il propose un regroupement des principes en trois (3) catégories : conception, développement et gestion.
d. Lehman [19] propose sept (7) lois qui se rapprocheraient de celles de la physique et de la biologie. Il souligne l'importance de développer une compréhension globale des principes à la base du génie logiciel.
e. Boehm [20] identifie sept (7) principes qui seraient à la base de la discipline. Il les décrit pour en faciliter le sens et la portée. Il propose aussi deux (2) critères permettant d'identifier les principes.
f. Booch et Bryan [21] présentent les mêmes principes identifiés par Ross et al. [17] et précisent les relations entre ceux-ci. Ils présentent aussi certains critères de qualité du logiciel à associer à ceux-ci.
g. Davis [22] propose 201 énoncés de principes qu'il regroupe en huit (8) catégories recoupant les activités du développement et des activités de support. Il suggère,

entre autres, au génie logiciel de développer ses propres principes de base. Il identifie, dans un article [23] paru quelque temps avant son ouvrage principal, les quinze (15) principes les plus importants de sa liste.

h. Buschmann et al. [24] identifient onze (11) principes qui seraient, selon eux, les plus importants pour le volet de l'architecture du logiciel.
i. Wiegers [25] identifie quatorze (14) principes qui influenceraient la culture du génie logiciel. Il précise que ces principes seraient la fondation de l'amélioration des processus et des produits.
j. Wasserman [26] présente une liste de huit (8) concepts fondamentaux du génie logiciel. Il souligne que ces concepts fondamentaux sont au cœur même des meilleures pratiques de l'industrie.
k. Maibaum [27] précise que si le génie logiciel est une discipline du génie, elle devrait avoir des sources mathématiques comme pour les autres disciplines du génie. Il base son article sur les six (6) catégories de connaissances de Vincenti [28].
l. Taylor [29] interprète, en transposant à la façon du génie logiciel, les dix (10) principes de design de Mayall [30] dans un texte de nature académique destiné aux étudiants en conception.
m. Meyer [31] présente et commente treize (13) principes ou concepts fondamentaux qui seraient les fondements de la connaissance qu'un professionnel en génie logiciel doit posséder.

Le fait d'utiliser la liste des quinze (15) principes fondamentaux de Bourque et al. [5] plutôt qu'une autre, est justifié par :

a. leur démarche méthodologique de recherche qui est bien définie et qui implique l'opinion de plusieurs spécialistes de la discipline;
b. l'utilisation de définitions claires (définitions des termes et critères de sélection) pour l'identification des principes fondamentaux;

c. la mesure d'évaluation du degré de consensus des principes fondamentaux retenus, à partir d'une liste qui était plus imposante au départ; soixante-cinq (65) principes fondamentaux ont été suggérés lors de la première des deux études « Delphi ».

Dans le cadre de ses travaux de doctorat, Séguin veut analyser les principes fondamentaux suggérés par les différents auteurs, pour produire une liste plus restreinte, à l'aide d'un cadre conceptuel qu'il aura à définir. Son projet de recherche verra à documenter le sous-ensemble de principes fondamentaux qu'il identifiera, pour en faciliter la compréhension et pour en définir la portée. Enfin, il vérifiera le degré de support de ce sous-ensemble par rapport aux connaissances généralement reconnues du Guide SWEBOK, par rapport à un sous-ensemble de normes du génie logiciel et par rapport au curriculum du génie logiciel. Cette vérification permettra de confirmer son identification du sous-ensemble de principes fondamentaux et favorisera l'amélioration de l'infrastructure (normes, corpus de connaissances et curriculum) du génie logiciel.

Notre étude se situe au niveau de la vérification du degré de support des principes fondamentaux par rapport aux connaissances généralement reconnues du Guide SWEBOK. Séguin pourra utiliser notre méthode d'analyse de la correspondance pour étendre cette étude de la correspondance aux autres éléments de l'infrastructure du génie logiciel qu'il a ciblés (normes et curriculum) par rapport à la liste de principes fondamentaux qu'il produira. Il pourra aussi utiliser les notions des descriptions d'accompagnement de notre étude pour documenter et définir la portée de son sous-ensemble de principes fondamentaux.

2.1.2 Connaissances généralement reconnues

Le corpus de connaissances existe dans la littérature du génie logiciel. Le but du Guide SWEBOK [6] est de décrire quelle portion de cette littérature est « généralement reconnue », d'organiser cette portion et de fournir un accès par sujet à celle-ci. Cinq (5)

objectifs ont permis de définir le contenu du Guide SWEBOK (voir [CH1P2]) :

a. Promote a consistent view of software engineering worldwide.
b. Clarify the place – and set the boundary – of software engineering with respect to other disciplines such as computer science, project management, computer engineering, and mathematics.
c. Characterize the contents of the software engineering discipline.
d. Provide a topical access to the Software Engineering Body of Knowledge.
e. Provide a foundation for curriculum development and individual certification and licensing material.

L'organisation hiérarchique du Guide SWEBOK a été définie pour répondre à ces objectifs et pour être compatible avec les principales écoles de pensée de l'industrie, la littérature et les standards de l'ingénierie du logiciel. La taxonomie présentée est celle nécessaire pour permettre une compréhension de la nature de la connaissance « généralement reconnue » des sujets et pour faciliter la recherche d'informations lorsque le guide est utilisé comme ouvrage de référence.

Quant à la profondeur du traitement des sujets, il a fallu faire la distinction entre la connaissance « généralement reconnue », la connaissance « avancée » et provenant de la recherche (associée à la maturité du sujet), et la connaissance « spécialisée » (associée à la généralisation de l'application). Une définition produite par le Project Management Institute (PMI), a été choisie à ce niveau pour définir la connaissance « généralement reconnue » (voir la référence dans le Guide SWEBOK [CH1P3]) : « *The generally accepted knowledge applies to most projects most of the time, and widespread consensus validates its value and effectiveness* ».

La figure 2, extraite du site Internet du Guide SWEBOK [32], permet de situer les connaissances généralement reconnues par rapport aux autres catégories de

connaissances. Cette partie de la définition de « connaissance généralement reconnue » provient du Guide to the Project Management Body of Knowledge (Guide PMBOK) [33]. Le Guide PMBOK est aussi un standard du IEEE [34].

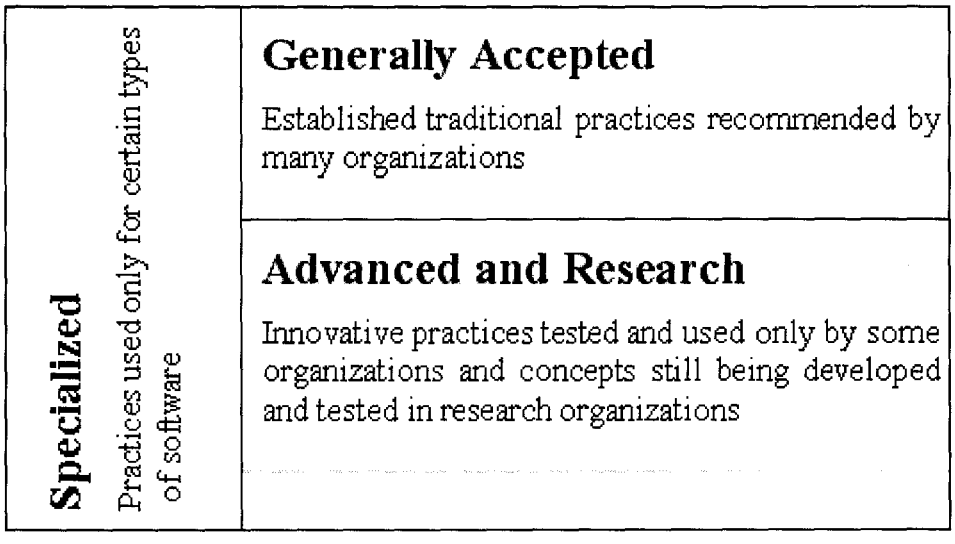


Figure 2 Catégories de connaissances

Enfin, la taxonomie des domaines de connaissances a été définie en considérant que les connaissances identifiées ne sont pas généralement reconnues qu'au présent, mais pour une période pouvant aller de trois (3) à cinq (5) ans.

2.1.3 Principe fondamental vs Connaissance généralement reconnue

Il faut comprendre que c'est par les ressemblances entre les concepts de « principes fondamentaux » et de « connaissances généralement reconnues » que l'étude est rendue possible. Les définitions de « principe fondamental » et de « connaissance généralement reconnue » sont assez similaires même si des méthodes différentes de recherche les ont produites. Les principes fondamentaux de l'étude et le Guide SWEBOK ont été produits grâce à l'implication d'un grand nombre d'intervenants de la discipline. Les processus utilisés dans les deux cas facilitent l'atteinte d'un consensus permettant le regroupement de spécialistes de la discipline dans la réalisation du projet.

Pour produire la liste des principes fondamentaux de l'étude, Bourque et al. [5] ont suivi le processus suivant :

a. recommandations pour l'identification de principes fondamentaux du génie logiciel lors d'un premier atelier de travail tenue dans une conférence internationale lors d'un premier atelier de travail tenue dans une conférence internationale;
b. définition des critères pour l'identification et l'évaluation des principes proposés;
c. soumission de 65 principes candidats dans une première ronde d'étude « Delphi » (cotation sur l'importance à accorder à chacun des principes et vote sur l'acceptation de la moyenne de cotation obtenue par 14 participants);
d. évaluation des principes fondamentaux candidats par des experts internationaux de la discipline, lors d'un deuxième atelier de travail également tenu dans une conférence internationale [35];
e. soumission de la liste améliorée de principes fondamentaux avec les recommandations des experts, pour une deuxième ronde d'étude « Delphi » (72 participants);
f. évaluation des principes fondamentaux candidats par les membres du Technical Council on Software Engineering du IEEE Computer Society à l'aide d'un sondage par Internet (574 participants).

Pour réaliser la version du Guide SWEBOK à l'étude (« Trial Version »), il y a eu 3 phases de production impliquant 488 différents réviseurs :

a. phase « strawman » : production d'une liste suggérée des domaines de connaissances et d'une liste suggérée des disciplines connexes, production des chapitres par des spécialistes de chacun des domaines de connaissances identifiés (version 0.1);
b. phase « stoneman » (Trial Version) : production des chapitres par des spécialistes de chacun des domaines de connaissances identifiés et 3 rondes de révision du document par 1) un nombre limité d'experts des différents domaines (33 réviseurs

pour la version 0.1), 2) des utilisateurs sélectionnés (195 réviseurs pour la version 0.5) et 3) la communauté (378 réviseurs pour la version 0.7);

c. phase « ironman » : 2 sous-phases d'amélioration du document pour 1) développer la version "Ironman" (2004 Version) du Guide basée sur un processus de révision similaire aux révisions de la phase « stoneman » (573 réviseurs), et 2) pour endosser le Guide SWEBOK (Industrial Advisory Board et IEEE Computer Society Board of Governors).

Pour faciliter la compréhension des ressemblances et des différences entre les principes fondamentaux et les connaissances généralement reconnues, nous présentons dans le tableau II, les critères utilisés pour la reconnaissance des principes fondamentaux, par rapport à la définition de « connaissance généralement reconnue » du Guide SWEBOK.

Tableau II

Principe fondamental vs Connaissance généralement reconnue

Critères d'identification des principes fondamentaux de Bourque et al. [S2.3P61]	**Définition de « connaissance généralement reconnue » du Guide SWEBOK**
Fundamental principals are less specific than methodologies and techniques, i.e. specific methodologies and techniques may be selected, within a particular technological context, to accomplish the intent of fundamental principles.	*The generally accepted knowledge applies to most projects most of the time, and widespread consensus validates its value and effectiveness.*

Tableau II (suite)

Critères d'identification des principes fondamentaux de Bourque et al. [S2.3P61]	**Définition de « connaissance généralement reconnue » du Guide SWEBOK**
Fundamental principles are more enduring than methodologies and techniques, i.e. fundamental principles should be phrased in a way that will stand the test of time, rather than in the context of current technology.	*We applied a criterion of generally accepted knowledge, which we had to distinguish from advanced and research knowledge (on grounds of maturity) and from specialized knowledge (on the grounds of generality of application).*
Fundamental principles are typically discovered or abstracted from practice and should have some correspondence with best practices.	
A fundamental principle should not contradict more general fundamental principles.	
A fundamental principal should not conceal a tradeoff. By that we mean that a fundamental principle should not attempt to prioritize or select from among various qualities of a solution; the engineering process should do that. Fundamental principles should identify or explain the importance of the various qualities among which the engineering process will make trades.	

Tableau II (suite)

Critères d'identification des principes fondamentaux de Bourque et al. [S2.3P61]	***Définition de « connaissance généralement reconnue » du Guide SWEBOK***
... but, there may be tradeoffs in the application of fundamental principles.	*Additionally, the KA Descriptions are somewhat forward-looking – we're considering not only what is generally accepted today but also what could be generally accepted in three to five years.*
A fundamental principle should be precise enough to be capable of support or contradiction.	
A fundamental principle should relate to one or more underlying concepts.	

Même si ce n'est pas un critère d'identification pour les principes fondamentaux, il est clair que comme pour la connaissance généralement reconnue, il faut que le principe fondamental soit reconnu par la discipline. Que ce soit pour l'un ou pour l'autre, le praticien doit en connaître la teneur mais n'est pas toujours tenu de le considérer. Il est en effet possible qu'un compromis doive être fait entre un principe fondamental par rapport à un autre, ou entre l'utilisation d'une connaissance généralement reconnue par rapport à une autre.

Que ce soit dans la reconnaissance d'un principe fondamental ou d'une connaissance généralement reconnue, il y a une question de temps, de durée de vie qui s'applique. Par contre, les principes fondamentaux sont plus durables que les connaissances généralement reconnues. Les principes fondamentaux ont aussi une application plus générale que les connaissances généralement reconnues. Ainsi, un changement technologique majeur pourra affecter les connaissances généralement reconnues sans affecter les principes fondamentaux à la base de la discipline.

2.1.4 Apport de l'étude

C'est au niveau de la dernière partie du travail de recherche de Séguin [7] que se situe le premier apport de notre étude. La démarche de notre étude permet d'identifier des notions pour produire des descriptions d'accompagnement des principes fondamentaux de Bourque et al. [5] et de vérifier le niveau de correspondance de ces principes fondamentaux dans le Guide SWEBOK.

Les résultats de la démarche de l'étude pourront être réutilisés pour appuyer la vérification du degré de support du sous-ensemble de principes fondamentaux que Séguin identifiera. Elle lui fournira une proposition pour la méthode d'établissement de la correspondance des principes fondamentaux avec l'infrastructure (Guide SWEBOK, normes, curriculum) du génie logiciel.

En évaluant le niveau de correspondance avec les connaissances généralement reconnues du Guide SWEBOK, nous ajoutons des éléments de réflexion par rapport à la liste des principes fondamentaux de Bourque et al. [5]. Nos résultats permettent de faciliter une avancée de la discipline par l'élaboration d'une liste reconnue de principes fondamentaux du génie logiciel.

2.2 Objet de l'étude

Deux publications seront utilisées pour effectuer l'étude. Il s'agit du Guide to the Software Engineering Body of Knowledge [6] et de l'article « Fundamental Principles of Software Engineering – A Journey » [5] qui présente une liste de quinze (15) principes fondamentaux suggérés à la base du génie logiciel.

2.2.1 Guide SWEBOK

Le but du Guide SWEBOK est d'être un ouvrage de références pour les connaissances généralement reconnues qui composent la discipline du génie logiciel. Sachant que l'ingénierie du logiciel est encore jeune et même immature à certains niveaux, nous nous devons d'identifier et de catégoriser les « sous-disciplines » qui collectivement constituent la discipline. Même si de nouvelles technologies émergent et que de nouvelles méthodes et techniques sont développées, plusieurs des principales constituantes de l'ingénierie du logiciel sont connues aujourd'hui. La taxonomie proposée par le Guide SWEBOK vise à identifier ces constituantes qui ne varient pas ou qui ont peu de chance d'être modifiées dans un avenir rapproché, et permet l'intégration des connaissances émergeantes.

Chacun des dix (10) domaines de connaissances du Guide SWEBOK est présenté indépendamment dans un chapitre du Guide SWEBOK. La figure III, extraite du Guide SWEBOK [CH1P7], présente les domaines de connaissances et les sujets principaux de leur taxonomie respective. À l'intérieur de chacun des chapitres du guide, on retrouve une figure représentant le découpage de la taxonomie pour les sujets principaux du domaine et de leurs subdivisions.

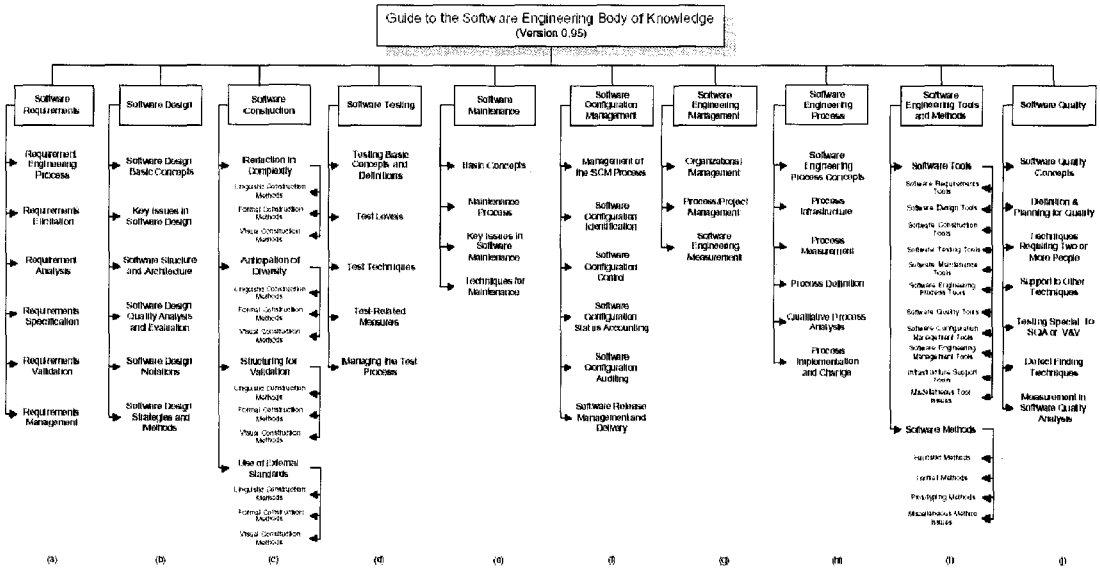


Figure 3 Domaines de connaissance du Guide SWEBOK [CH1P7]

Le tableau III liste les domaines de connaissances du Guide SWEBOK et l'identificateur qui sera utilisé pour y référer au cours de l'étude. Les identificateurs sont aussi présentés dans la liste des acronymes au début du document.

Tableau III

Identificateurs des domaines de connaissances du Guide SWEBOK

Identificateur	**Domaine de connaissances**
SR	Software Requirements
SD	Software Design
SC	Software Construction
ST	Software Testing
SM	Software Maintenance
SCM	Software Configuration Management
SEM	Software Engineering Management
SEP	Software Engineering Process
SETM	Software Engineering Tools and Methods
SQ	Software Quality

Le Guide SWEBOK est déjà cité dans différentes publications ou utilisé par des organismes ou organisations. Il a entre autres été utilisé pour le développement de la méthodologie qui se retrouve dans l'outil « CxOne » de la compagnie Construx [36] :

> « CxOne is based on IEEE's SWEBOK (Software Engineering Body of Knowledge). The SWEBOK defines the universe of software engineering knowledge and is a basis for defining engineering licensing and certification in the software industry. »

Il est aussi utilisé par certaines universités pour définir le contenu de programme(s) en ingénierie du logiciel. Un exemple d'utilisation par l'Université de Sunderland est présenté en [37].

Mais, le Guide SWEBOK a aussi ses détracteurs. Des commentaires sont disponibles sur certains sites Internet à ce sujet. Par exemple, ce texte recueilli sur le site de Wikipedia [38] parlant de la controverse autour de l'acceptation du guide :

> « Note that this effort is quite controversial. Cem Kaner and Grady Booch are two of many who have publicly stated that the document is misguided, and the ACM has withdrawn from its initial participation in the project. The document may not accurately reflect the community's view of software engineering, but more fundamentally, its primary goal is to enable the licensing of programmers – a goal which ignores the continuing (and unsolved) discussion as to whether programming is really analogous to other engineering fields. »

Kaner [39] présente d'ailleurs une critique très virulente par rapport au domaine de connaissances du ST, en relation avec les méthodes « agiles ». Certains membres de l'Association for Computing Machinery (ACM) croient qu'il ne devrait pas y avoir d'exigence à l'obtention d'un permis d'ingénieur comme le modèle adopté au Texas. Elle s'est donc retirée du processus conjoint de production du Guide SWEBOK en juin 2000 [40].

Il est possible de consulter les commentaires recueillis au cours des différentes phases de révision du Guide SWEBOK, ainsi que de leurs résolutions, à l'adresse Internet http://www.swebok.org/reviewers/reviewresults.html. Plusieurs statistiques y sont présentées comme la distribution géographique des différents intervenants ou réviseurs du projet.

2.2.2 Fundamental Principles of Software Engineering – A Journey

Le but visé par Bourque et al. [5] au cours des travaux ayant permis d'identifier la liste des principes fondamentaux à l'étude, était de fournir une structure permettant de définir les relations entre les différents groupes de standards. Certains standards sont en effet isolés, sans lien avec le reste de la discipline en raison de leur spécialisation. L'identification d'une liste de principes fondamentaux permettrait aussi de caractériser les activités de la discipline qui se distinguent par rapport aux autres disciplines du génie ou des autres disciplines reliées à l'informatique. Elle permettrait de mieux définir les programmes de formation et offrirait un cadre d'analyse et d'amélioration à la littérature produite, comme le Guide SWEBOK.

C'est en utilisant les critères de l'identification des principes fondamentaux présentés dans le tableau IV, que les deux ateliers de travail, les deux études « Delphi » et le sondage par Internet ont permis de produire la liste de principes fondamentaux à la base de notre étude. Il y a quinze (15) principes fondamentaux qui ont été suggérés dans l'article de Bourque et al. [5]. Le tableau présente également l'identificateur qui sera utilisé pour y référer au cours de l'étude.

Tableau IV

Liste des principes fondamentaux suggérés
par Bourque et al. [5]

Identificateur	**Principe Fondamental**
(A)	Apply and use quantitative measurements in decision-making
(B)	Build with and for reuse
(C)	Control complexity with multiple perspectives and multiple levels of abstraction
(D)	Define software artifacts rigorously
(E)	Establish a software process that provides flexibility
(F)	Implement a disciplined approach and improve it continuously
(G)	Invest in the understanding of the problem
(H)	Manage quality throughout the life cycle as formally as possible
(I)	Minimize software component interaction
(J)	Produce software in a stepwise fashion
(K)	Set quality objectives for each deliverable
(L)	Since change is inherent to software, plan for it and manage it
(M)	Since tradeoffs are inherent to software engineering, make them explicit and document them
(N)	To improve design, study previous solutions to similar problems
(O)	Uncertainty is unavoidable in software engineering. Identify and manage it

L'article de Bourque et al. [5] fait partie des dix (10) articles ayant été le plus souvent téléchargés à partir du site Internet du *Journal of Systems and Software* [41] pour l'année 2003. Étant un ouvrage assez récent, il commence à peine à être cité comme référence.

2.3 Objectifs de l'étude

Cette étude veut atteindre les trois (3) objectifs suivants :

a. Établir le niveau de correspondance entre les principes fondamentaux et le contenu du Guide SWEBOK

   La première partie de l'analyse permettra de reconnaître les principes fondamentaux dans les chapitres du guide présentant les dix (10) domaines de connaissances. Cette reconnaissance d'une correspondance pourra se retrouver à tous les niveaux d'un domaine de connaissances, à l'exception des références présentées à la fin de chacun des domaines qui ne font pas partie de l'étude. La reconnaissance de la correspondance se fera dans une seule direction : des principes fondamentaux vers le contenu du Guide SWEBOK.

b. Produire une version initiale des descriptions d'accompagnement pour chacun des principes fondamentaux

   En utilisant les résultats de l'analyse de correspondance, la deuxième partie de l'étude voit à produire une description d'accompagnement associée au libellé de chacun des principes fondamentaux. L'objectif ici n'est pas de produire une description définitive, mais d'identifier les notions de base qui permettront de la produire dans des travaux ultérieurs. Ces descriptions pourront notamment servir d'intrants à un atelier de spécialistes sur les principes fondamentaux. La création éventuelle d'une description associée à chacun des principes fondamentaux limitera le nombre d'interprétations pouvant ressortir à la seule lecture du libellé.

c. Identifier des pistes d'améliorations au Guide SWEBOK

Le résultat de l'analyse de la correspondance entre les principes fondamentaux et le contenu du Guide SWEBOK permettra aussi d'identifier des pistes d'amélioration au Guide ou aux connaissances de la discipline, pour les pistes d'améliorations identifiées n'ayant pas trait aux connaissances déjà généralement reconnues aux connaissances généralement reconnues. Ces éléments seront présentés soit comme des pistes d'amélioration, soit comme des commentaires qui ont comme objectif de stimuler une réflexion sur le sujet apporté. Tous ces éléments seront transmis, après la livraison de l'étude, à l'équipe du Guide SWEBOK pour qu'ils soient évalués et puissent faire partie d'une version ultérieure du Guide SWEBOK.

En se rapportant à la figure 4, un sous-ensemble de la figure 1 présentée précédemment, il est possible de situer l'apport de notre étude par rapport aux principes fondamentaux. La flèche du haut correspond au travail de reconnaissance du niveau de la correspondance des principes fondamentaux vers les connaissances généralement reconnues du Guide SWEBOK. Alors que la flèche du bas correspond au travail de production des descriptions d'accompagnement pour les principes fondamentaux à partir des correspondances reconnues dans les connaissances généralement reconnues. Pour les fins de cette figure, nous considérons que les connaissances généralement reconnues sont au même niveau que les pratiques et les standards. Ce ne sont pas des standards, mais elles sont reconnues par la discipline et se retrouvent donc au même niveau que ceux-ci.

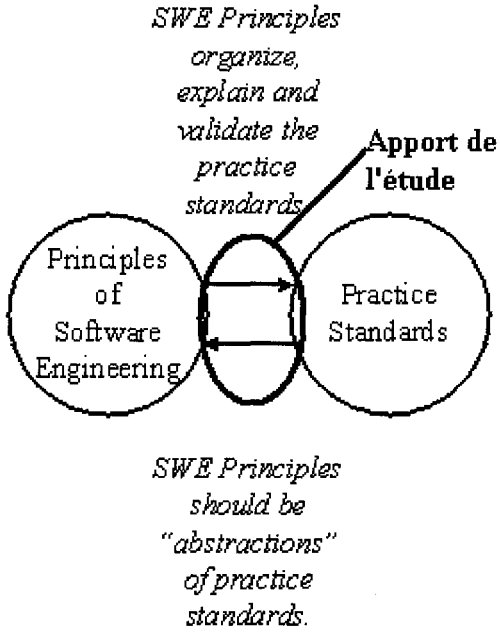


Figure 4 Apport de l'étude face aux principes fondamentaux

2.4 Utilisateurs

Les résultats de l'étude seront présentés et pourront être utilisés par l'équipe du Guide SWEBOK pour des améliorations aux versions futures. Ils seront aussi importants pour la recherche de Séguin sur l'établissement d'une liste structurée et documentée de principes fondamentaux en génie logiciel. Ils pourront satisfaire les chercheurs du domaine intéressés par l'identification des principes fondamentaux, à la base du génie logiciel, et par tous les chercheurs, que l'amélioration de la discipline en général préoccupent. Le tableau V présente la synthèse de la définition du projet selon le cadre de Basili, qui inclut les motivations à la base du projet, les objets du projet, les objectifs du projet et les utilisateurs du livrable du projet.

Tableau V

Définition du projet

I. Définition du projet			
Motivation	Objet(s)	Objectif(s)	Utilisateur(s)
Améliorer la base de la discipline du génie logiciel pour qu'elle soit reconnue comme discipline du génie.	• Les quinze (15) principes fondamentaux suggérés par Bourque et al. [5] • Guide SWEBOK [6]	• Établir le niveau de correspondance entre les principes fondamentaux et les domaines de connaissances du Guide SWEBOK. • Produire des descriptions d'accompagnement pour chacun des principes fondamentaux. • Identifier des pistes d'amélioration au Guide SWEBOK ou à la discipline en général.	• Normand Séguin • L'équipe du Guide SWEBOK • Les chercheurs du domaine intéressés par l'identification d'une liste reconnue de principes fondamentaux du génie logiciel • Les chercheurs intéressés par l'amélioration de la discipline du génie logiciel

CHAPITRE 3

PLANIFICATION DU PROJET

Cette partie de l'étude explique la démarche suivie pour atteindre les objectifs fixés à la section 2.3 (Objectifs de l'étude). En premier lieu, nous identifions et analysons la correspondance des principes fondamentaux à l'intérieur du Guide SWEBOK. Cette analyse de la correspondance permet de déterminer le niveau de correspondance des citations retenues. Par la suite, nous utilisons les citations retenues pour produire une description d'accompagnement pour chacun des principes fondamentaux. Enfin, l'auteur présente des pistes d'amélioration au Guide SWEBOK ou à la discipline, selon que ces pistes d'amélioration ont trait ou non aux connaissances déjà généralement reconnues. Ces pistes correspondront à des suggestions de l'auteur par rapport à l'analyse de la correspondance et seront basées sur l'étude du niveau de correspondance maximum reconnu pour les différents principes fondamentaux à l'intérieur du Guide SWEBOK.

Les différentes étapes de la démarche utilisent l'hypothèse que les principes fondamentaux de Bourque et al. [5] ont été reconnus comme tel par la discipline, même si ce n'est pas encore le cas. Il y a quand même un certain consensus d'un bon nombre de spécialistes ayant participé à l'une ou l'autre des différentes activités ayant produit cette liste. La dernière étape du projet, une évaluation des principes fondamentaux candidats à l'aide d'un sondage par Internet à laquelle 574 membres du IEEE ont répondu.

3.1 Étapes du projet

La phase d'exécution du projet est divisée en trois (3) étapes :

a. analyse de la correspondance;
b. production d'une description d'accompagnement par principe fondamental;
c. présentation de pistes d'amélioration au Guide SWEBOK ou à la discipline.

3.1.1 Analyse de la correspondance

La première étape de l'étude consiste à réaliser l'analyse de la correspondance entre les principes fondamentaux et le Guide SWEBOK. Cette analyse permet d'identifier les citations à l'intérieur du Guide SWEBOK où une correspondance avec le principe fondamental est reconnue et est présentée sous forme de tableaux de correspondances. La correspondance reconnue peut être partielle : ne représentant qu'une partie du libellé du principe fondamental; ou complète : représentant l'ensemble du libellé. Pour chacun des principes à l'étude, nous indiquons la façon dont la correspondance a été interprétée au cours de l'analyse de la correspondance. Ces indications, présentées sous la forme d'un paragraphe d'introduction au tableau de la correspondance, spécifient comment la reconnaissance du principe fondamental a été effectuée, que ce soit pour les correspondances partielles ou les correspondances complètes.

Un exemple de correspondance partielle ou complète peut être faite avec le principe fondamental « Build with or for reuse ». La correspondance partielle de ce principe peut être reconnue par la production du logiciel en réutilisant des composants déjà existants (Build with reuse) ou par la production des composants du logiciel en fonction d'une réutilisation future (Build for reuse).

La correspondance complète est reconnue si une citation présente les deux parties du principe fondamental. Les trois (3) façons de reconnaître une correspondance avec ce principe fondamental sont présentées dans les figures 5 et 6.

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH2 S2.3 P12	Architectural design is a skill that is driven by many factors such as the recognition of reusable architectural "patterns" or the existence of off-the shelf components.	• Build for reuse : Création de "patterns" • Build with reuse : Utilisation de composants d'une tierce partie (off-the shelf components)	DEF	Overview of requirement analysis

Figure 5 Exemple de correspondance complète

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SC	CH4 S2.5 P56	Automated construction is also reuse-intensive construction, since by limiting human options it allows the controlling software to make more effective use of its existing store of effective software problem solutions.	Cette correspondance a été reconnue avec « Build with reuse ». Il est question de produire des outils automatisant la construction de certaines parties du logiciel.	DEF	Manual and Automated Construction/The Spectrum of Construction Techniques **Build with reuse**
	CH4 S2.5 P56	When simple selections from a list of options will not suffice, software engineers often can still develop application specific tool kits (that is, sets of reusable parts designed to work with each other easily) to provide a somewhat lesser level of control.	Cette correspondance a été reconnue avec « Build for reuse » : l'ingénieur logiciel peut développer des ensembles d'outils logiciels réutilisables, s'emboîtant les uns dans les autres.	DEF	Manual and Automated Construction/The Spectrum of Construction Techniques **Build for reuse**

Figure 6 Exemples de correspondance partielle

Les correspondances reconnues sont présentées sous forme de tableaux; il y a un tableau produit par principe fondamental à l'étude. Ces tableaux sont formés des colonnes suivantes :

a. Domaine

Cette colonne contient l'acronyme identifiant le domaine de connaissances du Guide SWEBOK. Lorsque plus d'une correspondance est identifiée dans un domaine de connaissances, seule la première citation identifie le domaine comme présenté à la figure 6 pour le SC.

b. Repère

Cette colonne contient l'endroit dans le Guide SWEBOK où la correspondance avec le principe fondamental a été reconnue. Chacun des repères est présenté selon le format suivant :

CH*x* : Chapitre *x* du Guide SWEBOK

S*y.z* : Section *y.z* à l'intérieur de la taxonomie du domaine de connaissances

P*v* : Page *v* du Guide SWEBOK

Par exemple, pour faire référence à un texte se trouvant dans la section 2.3 du SR (chapitre 2), à la page 12, la notation suivante est utilisée pour le repère :

CH2S2.3P12

Lorsqu'il n'a pas été possible de reconnaître une correspondance avec le principe fondamental dans un domaine de connaissances, le terme « Aucune », pour « Aucune citation reconnue », est inscrit dans cette colonne du tableau. Les autres colonnes sont alors laissées vides.

c. Citation

Cette colonne présente la citation du Guide SWEBOK où la correspondance avec le principe fondamental a été reconnue. La citation se trouve dans la section et la page identifiée dans la colonne « Repère », pour la version du Guide SWEBOK à l'étude. Une citation peut se retrouver dans plus d'un tableau de correspondances si une correspondance y est reconnue avec plus d'un principe fondamental.

d. Interprétation

Cette colonne contient les textes d'interprétation des correspondances reconnues, lorsqu'il sera nécessaire d'interpréter la citation présentée. Il est possible que cette colonne ne contienne aucun texte si l'auteur a jugé que la citation présentée n'exige aucune interprétation, la correspondance identifiée étant assez explicite pour la reconnaître à sa seule lecture. Elle permet aussi d'identifier les correspondances partielles au libellé du principe fondamental ou de justifier le niveau de correspondance présenté.

e. Niveau

Cette colonne contient le niveau de correspondance entre le principe fondamental et une citation du Guide SWEBOK. Les niveaux possibles sont :

- DOM

La correspondance peut être reconnue par l'application de l'ensemble des sujets d'un domaine de connaissances.

- INTRO

La correspondance provient de la section « Introduction » du domaine de connaissances.

- DEF

La correspondance provient de la section « Définition » du domaine de connaissances.

- SUJET

  La correspondance provient d'un sujet principal du domaine (premier niveau de la taxonomie d'un domaine).

- N1

  La correspondance provient du premier niveau de décomposition d'un sujet principal du domaine.

- N+

  La correspondance provient d'un niveau inférieur au niveau « N1 » dans la taxonomie du domaine.

- TEXTE

  La correspondance provient d'un texte à l'intérieur du domaine, sans relation avec la taxonomie dans laquelle le texte a été retrouvé.

f. Titre de la section

   Cette colonne présente le titre de la section du Guide SWEBOK correspondant à la référence identifiée dans la colonne « Repère ».

Les figures 7, 8 et 9 présentent comment le niveau de correspondance est reconnu au cours de l'analyse de la correspondance. Il faut que la citation identifiée fasse partie du texte décrivant la notion (connaissance) présentée dans la taxonomie pour être identifiée dans le niveau correspondant (DOM, SUJET, N1, N+), autrement, le niveau de la correspondance peut être « INTRO », pour une citation se trouvant dans la section d'introduction du domaine, « DEF », pour une citation se trouvant dans la section des définitions du domaine, ou « TEXTE ».

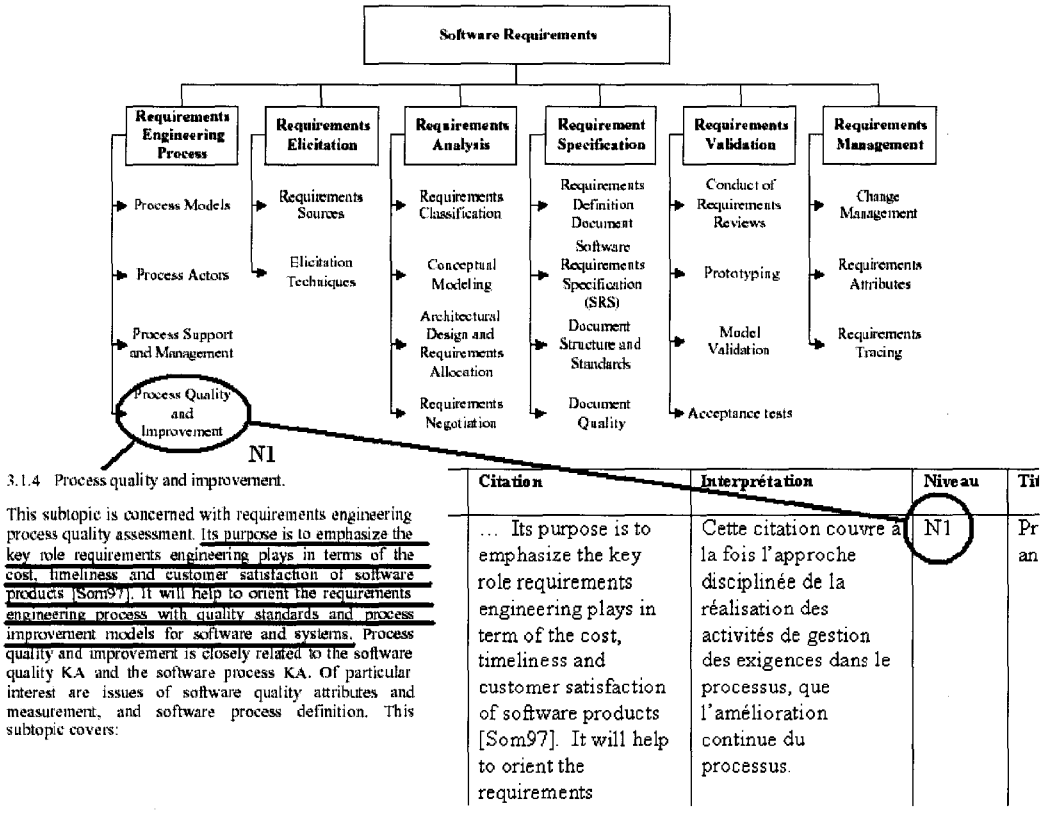
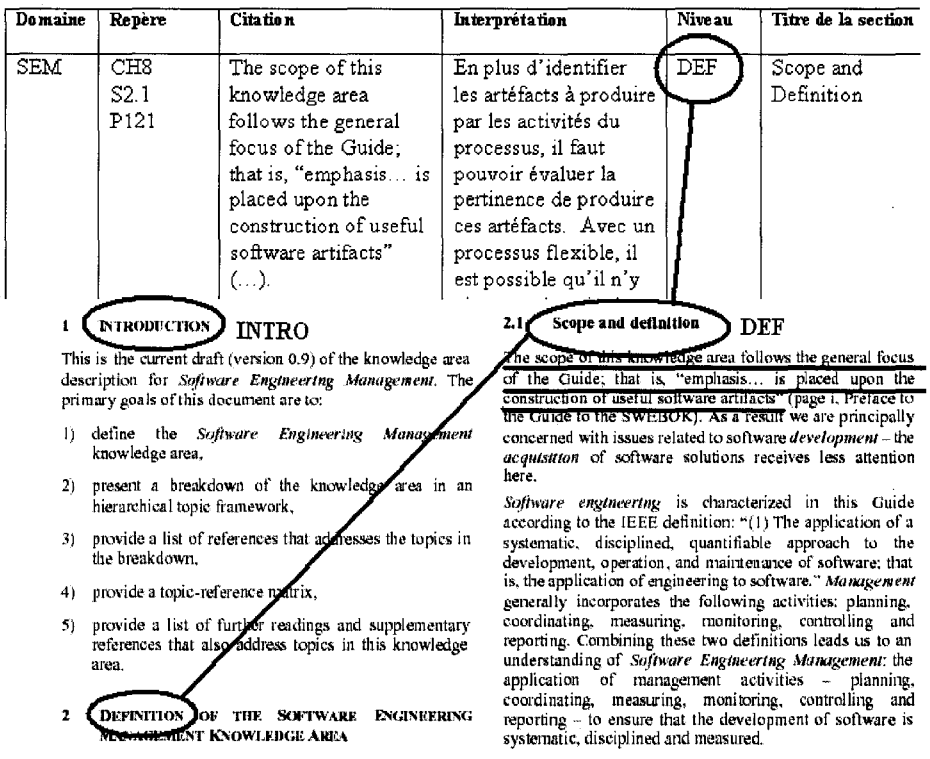


Figure 7 Exemple d'une correspondance de niveau « N1 »



Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SEM	CH8 S2.1 P121	The scope of this knowledge area follows the general focus of the Guide; that is, "emphasis... is placed upon the construction of useful software artifacts" (...).	En plus d'identifier les artéfacts à produire par les activités du processus, il faut pouvoir évaluer la pertinence de produire ces artéfacts. Avec un processus flexible, il est possible qu'il n'y	DEF	Scope and Definition

1 INTRODUCTION INTRO

This is the current draft (version 0.9) of the knowledge area description for *Software Engineering Management*. The primary goals of this document are to:

1) define the *Software Engineering Management* knowledge area,
2) present a breakdown of the knowledge area in an hierarchical topic framework,
3) provide a list of references that addresses the topics in the breakdown,
4) provide a topic-reference matrix,
5) provide a list of further readings and supplementary references that also address topics in this knowledge area.

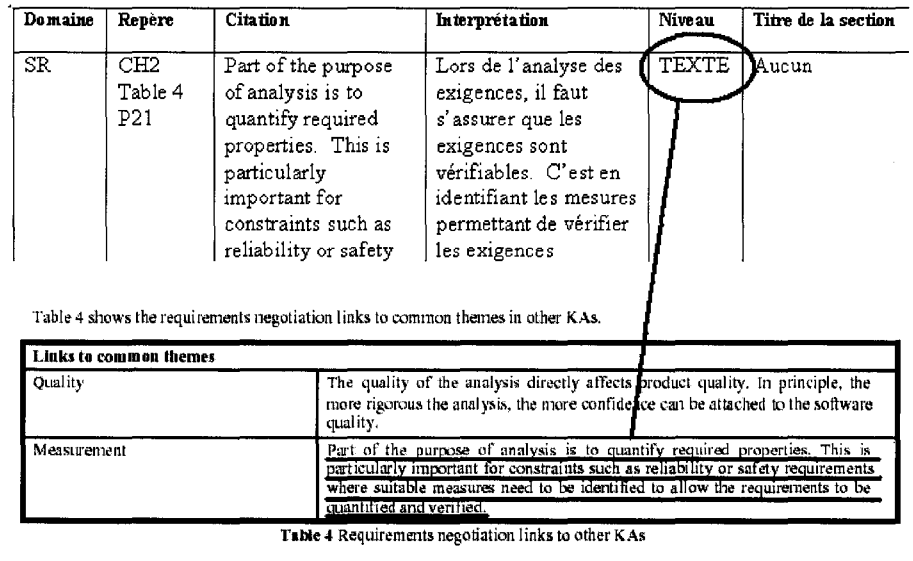
2 DEFINITION OF THE SOFTWARE ENGINEERING MANAGEMENT KNOWLEDGE AREA

2.1 Scope and definition DEF

The scope of this knowledge area follows the general focus of the Guide; that is, "emphasis... is placed upon the construction of useful software artifacts" (page i, Preface to the Guide to the SWEBOK). As a result we are principally concerned with issues related to software *development* – the *acquisition* of software solutions receives less attention here.

Software engineering is characterized in this Guide according to the IEEE definition: "(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software." *Management* generally incorporates the following activities: planning, coordinating, measuring, monitoring, controlling and reporting. Combining these two definitions leads us to an understanding of *Software Engineering Management*: the application of management activities – planning, coordinating, measuring, monitoring, controlling and reporting – to ensure that the development of software is systematic, disciplined and measured.

Figure 8 Niveaux de correspondance « DEF »



Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SR	CH2 Table 4 P21	Part of the purpose of analysis is to quantify required properties. This is particularly important for constraints such as reliability or safety	Lors de l'analyse des exigences, il faut s'assurer que les exigences sont vérifiables. C'est en identifiant les mesures permettant de vérifier les exigences	TEXTE	Aucun

Table 4 shows the requirements negotiation links to common themes in other KAs.

Links to common themes	
Quality	The quality of the analysis directly affects product quality. In principle, the more rigorous the analysis, the more confidence can be attached to the software quality.
Measurement	Part of the purpose of analysis is to quantify required properties. This is particularly important for constraints such as reliability or safety requirements where suitable measures need to be identified to allow the requirements to be quantified and verified.

Table 4 Requirements negotiation links to other KAs

Figure 9 Exemple de correspondance au niveau du texte

Toutes les correspondances reconnues seront identifiées pour présenter l'étendue de la couverture du principe fondamental dans chacun des domaines de connaissances. Le résultat final de l'analyse sera rédigé sous forme de quinze (15) tableaux de correspondances; un pour chacun des principes fondamentaux de l'étude.

3.1.2 Production des descriptions d'accompagnement

La deuxième phase consiste à analyser les résultats de l'analyse de correspondance pour identifier les notions qui doivent se retrouver dans le texte de la description d'accompagnement de chacun des principes fondamentaux. Une sélection sera faite à l'intérieur des différentes citations où la correspondance avec le principe fondamental a été reconnue et à partir des textes commentant les différents principes fondamentaux dans Bourque et al. [5].

Il faut noter que ce ne sont pas toutes les citations de l'analyse de correspondance présentant la notion identifiée qui sont présentées. Le but n'est pas de répéter tout le travail de la première étape du projet. La sélection des citations de présentation des notions est faite pour faire comprendre au lecteur comment la notion doit être interprétée. La description d'accompagnement, quant à elle, contient des citations de l'analyse de correspondance représentant le mieux la notion.

Pour assurer une continuité dans le texte de la description d'accompagnement, l'auteur ajoute à l'occasion des éléments complémentaires basés sur son expérience personnelle de la discipline. Ces éléments assurent une liaison entre les différentes notions identifiées pour le principe fondamental.

Un tableau présentera les citations sélectionnées en introduction aux descriptions produites. Chacun des tableaux sera composé des colonnes suivantes :

a. Notion

   Cette colonne identifie la notion qui synthétise ce qui se trouve dans les colonnes suivantes du tableau.

b. Citation

   Cette colonne présente les citations dans lesquelles on retrouve la notion identifiée. Ce ne sont pas toutes les citations de l'analyse de la correspondance, présentant la notion, qui sont présentées dans cette colonne. Nous en avons sélectionné quelques-unes car il y a des cas où une grande partie des citations de l'analyse de la correspondance présentent la notion identifiée.

c. Repère(s)

   Cette colonne indique les repères des citations présentées. Il peut s'agir de la colonne « Repère » des tableaux de l'analyse de la correspondance ou de la référence au texte commentant les principes fondamentaux dans Bourque et al. [5]. Dans ce dernier cas, le repère sera présenté comme suit : [BourqueSx.yPzz] (voir figure 10).

Pour chacun des principes, il y a une version anglaise et une version française de la description d'accompagnement qui est produite. La version anglaise servira d'intrant à un atelier sur les principes fondamentaux, alors que la version française répond au besoin des évaluateurs de l'étude. La figure 10 démontre comment les descriptions d'accompagnement sont produites en utilisant les citations de l'analyse de la correspondance et les textes commentant les principes fondamentaux dans Bourque et al. [5].

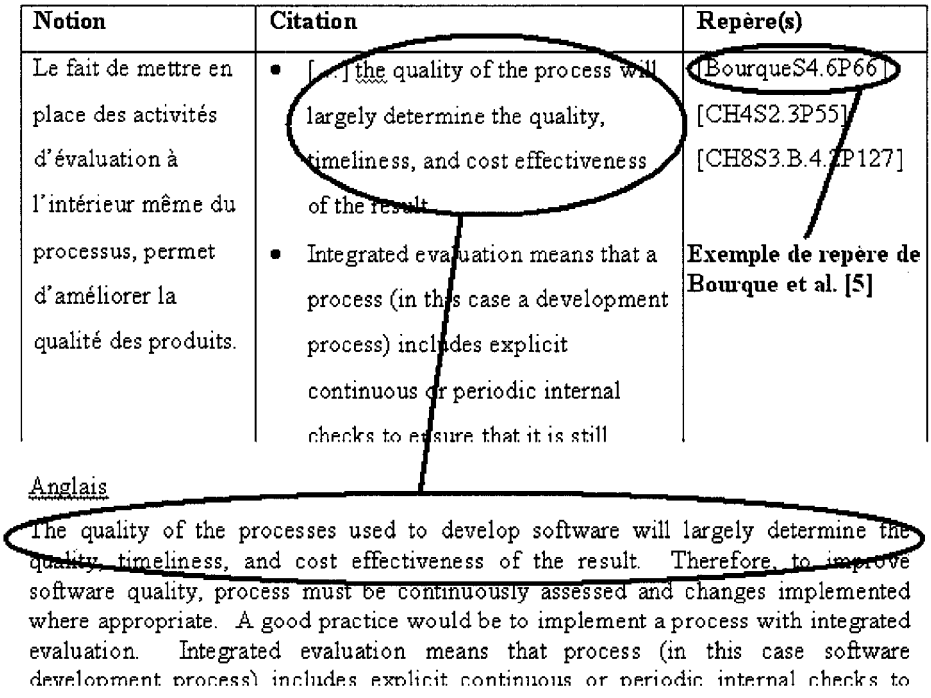


Figure 10 Production d'une description d'accompagnement

3.1.3 Présentation de pistes d'amélioration

La dernière phase consiste à identifier des pistes d'amélioration au Guide SWEBOK ou aux connaissances de la discipline. Pour identifier ces pistes, nous utilisons l'analyse de la correspondance pour produire des tableaux présentant les plus hauts niveaux de correspondance atteints pour chacun des principes fondamentaux. L'auteur utilise les données présentées dans les différents tableaux pour justifier les pistes d'amélioration qu'il aura identifiées.

L'hypothèse à la base de cette section est que les principes fondamentaux de l'étude sont reconnus. Il faut aussi savoir que les pistes présentées constituent le point de vue de l'auteur suite à l'analyse du niveau de correspondance de ces principes fondamentaux par rapport aux différents domaines de connaissances. Il arrive que certaines références

soient citées pour appuyer ces pistes mais ce n'est pas dans la majorité des cas. Il faudrait une analyse plus approfondie pour confirmer les suggestions de l'auteur. Même si certaines pistes ciblent directement un domaine de connaissances, il est possible que la piste ne puisse être acceptée du fait que la discipline ne serait pas en mesure de la reconnaître comme généralement reconnue.

Les tableaux produits dans cette partie de l'étude ont le format suivant :

a. Domaine :

   Cette colonne présente l'acronyme du domaine de connaissances du Guide SWEBOK.

b. Plus haut niveau reconnu :

   Cette colonne présente le niveau de correspondance le plus élevé ayant été reconnu à l'intérieur du domaine de connaissances, lors de l'analyse de correspondance.

c. Nombre de citations :

   Cette colonne présente le nombre de citations où une correspondance a été reconnue à l'intérieur du domaine de connaissances.

La figure 11 illustre comment les tableaux de l'analyse du niveau de correspondance seront présentés dans cette partie de l'étude.

Domaine	Plus haut niveau reconnu	Nombre de citations
SR	TEXTE	1
SD	N1	1
SC	DEF	1
ST	SUJET	3
SM	N1	2
SCM	N1	4
SEM	SUJET	4
SEP	DEF	1
SETM	N1	1
SQ	SUJET	4

Nombre de citations reconnues dans le domaine de connaissances

Plus haut niveau de correspondance reconnu dans la colonne du niveau de correspondance, pour le domaine de connaissances.

Figure 11 Tableau de l'analyse du niveau de correspondance

3.2 Les intrants

Pour réaliser ce projet, les ouvrages suivants sont utilisés comme intrants :

a. Guide to the Software Engineering Body of Knowledge : Trial Version (*Trial Version 1.00 – May 2001*) [6]

b. Fundamental Principles of Software Engineering – A Journey [5]

3.3 Les livrables

Tableau VI

Planification du projet

II. Planification du projet		
Étape 1	Intrants	Livrables
• Analyse de la correspondance	• Guide to the Software Engineering Body of Knowledge : Trial Version (*Trial Version 1.00 – May 2001*) [6] • Fundamental Principles of Software Engineering – A Journey [5]	• 15 tableaux décrivant la correspondance reconnue entre les 15 principes fondamentaux et le Guide SWEBOK
• Production d'une description d'accompagnement par principe fondamental	• 15 tableaux de correspondance entre les principes fondamentaux et le Guide SWEBOK • Fundamental Principles of Software Engineering – A Journey [5]	• 15 descriptions d'accompagnement pour les principes fondamentaux à l'étude, en anglais et en français
• Présentation de pistes d'amélioration au Guide SWEBOK ou à la discipline	• 15 tableaux de correspondance entre les principes fondamentaux et le Guide SWEBOK	• Des pistes d'amélioration au Guide SWEBOK ou à la discipline

CHAPITRE 4

EXÉCUTION DU PROJET

Le présent chapitre présente la phase d'exécution du projet. Elle comprend les trois (3) étapes identifiées à la section 3.1, c'est-à-dire :

a. l'analyse de la correspondance;
b. la production d'une description d'accompagnement par principe fondamental;
c. la présentation de pistes d'amélioration au Guide SWEBOK ou à la discipline.

4.1 Analyse de la correspondance

Pour cette phase du projet, nous étudions la correspondance des principes fondamentaux de Bourque et al. [5] par rapport aux connaissances généralement reconnues des dix (10) domaines de connaissances du Guide SWEBOK. Les principes fondamentaux et les domaines de connaissances du Guide sont identifiés dans les tableaux III et IV (section 2.2). Chacune des sous-sections présente le détail de l'analyse de correspondance pour un principe fondamental. Dans la dernière sous-section, une figure par domaine de connaissances est présentée pour montrer les endroits où les différents principes fondamentaux ont été reconnus dans la taxonomie de chacun des domaines de connaissances.

4.1.1 Apply and use quantitative measurements in decision-making

Tous les domaines de connaissances identifient des types de mesures à considérer au cours de leurs activités. Ces mesures sont habituellement un prélude à un besoin d'amélioration dans le processus en place ou pour produire des estimations lors de la planification du projet. Elles permettent aussi d'offrir une vue du processus et de la qualité des produits aux gestionnaires et un meilleur suivi des projets.

Pour reconnaître une correspondance avec ce principe, nous avons considéré les citations qui liaient les mesures et la prise de décisions par rapport aux mesures. Il faut bien distinguer ce principe de celui où il est question d'identifier les objectifs de qualité pour chacun des biens livrables (K), et de celui traitant de la gestion de la qualité tout au long du cycle de vie du logiciel (H). Le premier identifie les objectifs de qualité et par ricochet, les mesures qui y seront attachées. Le second est plutôt lié à l'identification des activités qui permettront de prendre les mesures nécessaires à la validation des objectifs de qualité identifiés préalablement. C'est suite à la complétion des activités permettant de produire ces mesures que les décisions seront prises.

Il y a donc une certaine séquence dans le temps où ces trois principes doivent être considérés à l'intérieur du cycle de vie d'un logiciel. La figure 12 présente cette séquence.

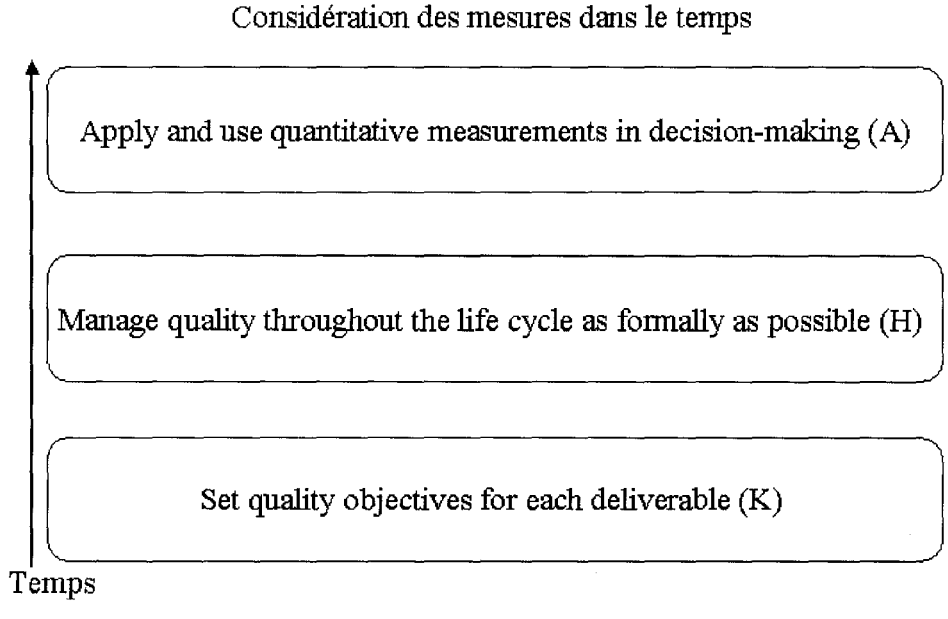


Figure 12 Évolution des mesures par rapport aux principes fondamentaux

Le tableau VII présente les citations du Guide SWEBOK où la correspondance avec l'utilisation des mesures pour la prise de décision a été reconnue. Pour ce principe fondamental, il n'y a pas de correspondance partielle.

Tableau VII

Analyse de la correspondance avec le principe fondamental (A)

Domaine	**Repère**	**Citation**	**Interprétation**	**Niveau**	**Titre de la section**
SR	CH2 Table 4 P21	Part of the purpose of analysis is to quantify required properties. This is particularly important for constraints such as reliability or safety requirements where suitable metrics need to be identified to allow the requirements to be quantified and verified.	Lors de l'analyse des exigences, il faut s'assurer que ces dernières sont vérifiables. C'est en identifiant les mesures permettant de vérifier les exigences que le l'ingénieur logiciel intervient. L'utilisation de ces mesures servira ensuite à décider du moment où un produit répond aux exigences de sécurité identifiées	TEXTE	Aucun

Tableau VII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			par exemple. L'identification de ce type de mesures est primordiale lorsqu'un logiciel est impliqué dans un système où la vie humaine peut être en jeu.		
SD	CH3 S3.IV P39	Measures: Formal measures (aka. Metrics) can be used to estimate, in a quantitative way, various aspects of the size, structure or quality of a design.	Les mesures peuvent être utilisées pour choisir le type de design qui répondra le mieux aux besoins exprimés.	SUJET	Software Design Quality Analysis and Evaluation
SC	CH4 S2.3 P55	Tools for extracting measurements of code quality and structure can also be used during construction, although such measurement tools are commonly applied during integration of		DEF	The Role if Integrated Evaluation in Construction

Tableau VII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		large suites of software units. When collecting measurements, it is important that the measurements collected be relevant to the goals of the development process.			
ST	CH5 S3.D P79	Measurement is instrumental to quality analysis. Indeed, product evaluation is effective only when based on quantitative measures. Measurement is instrumental also to the optimal planning and execution of tests, and several process metrics can be used by the test manager to monitor progress.	Il est clairement indiqué dans cette citation qu'une planification et une évaluation optimales des tests passent par la prise et l'utilisation des mesures par le gestionnaire responsable des tests.	SUJET	Test related measures

Tableau VII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH5 S3.D1 P79	To make testing more effective it is important to know which types of faults could be found in the application under test, and the relative frequency with which these faults have occurred in the past. This information can be very useful to make quality predictions as well as for process improvement.		N+	Evaluation of the program under test
	CH5 S3.E1 P81	Termination A critical task of the test manager is to decide how much testing is enough and when a test stage can be terminated. Thoroughness measures such as achieved code coverage or functional		N+	Management concerns

Tableau VII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		completeness, as well as estimates of fault density or of operational reliability, provide useful support, but are not sufficient by themselves.			
SM	CH6 S3.3.3.2 P93	Maintenance cost estimates are affected by many technical and non-technical factors. Primary approaches to cost estimating include use of parametric models and experience. Most often a combination of these is used to estimate costs.	Pour planifier les activités de maintenance, il est nécessaire de produire des estimés dont ceux des coûts. Ces estimés permettront d'identifier les activités à déployer pour maximiser l'efficacité de la maintenance du produit.	N+	Cost estimation
	CH6 S3.3.4 P94	Software life cycle costs are growing and a strategy for maintenance is	Des mesures sont utilisées pour identifier les activités du cycle de	N1	Software Maintenance Measurement

Tableau VII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		needed. Software measurement needs to be a part of that strategy. [...] Software measures are vital for software process improvement but the process must be measurable.	vie où des améliorations sont souhaitables. Que ce soit pour le développement ou la maintenance d'un logiciel, l'organisation les utilisera pour identifier les priorités d'amélioration.		
SCM	CH7 S3.I.C.3 P106	Different types of tool capabilities, and procedures for their use, support the SCM activities. [...] The use of tools in these areas increases the potential for obtaining product and process measurements to be used for project management and process improvement purpose.		N1	Tool Selection and Implementation

Tableau VII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH7 S3.I.E.1 P107	SCM metrics can be designed to provide specific information on the evolving product or to provide insight into the functioning of the SCM process. [...] Analysis of the measurements may produce insights leading to process changes and corresponding updates to the SCMP.		N1	SCM Metrics and Measurement
	CH7 S3.IV.A P110	Various information and measurements are needed to support the SCM process and to meet the configuration status reporting needs of management, software engineering, and other related activities.	Les mesures utiles aux gestionnaires sont par exemple les déviations par rapport aux plans ou les annulations de changement. Par exemple, le nombre de demandes de changements, par type de changement,	N1	Software Configuration Status Information

Tableau VII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			permet d'identifier les priorités dans les projets à venir.		
	CH7 S3.VI.B P111	Given that product changes can be occurring on a continuing basis, one issue for release management is determining when to issue a release. The severity of the problems addressed by the release and measurements of the fault densities of prior releases affect this decision.		N1	Software Release Management
SEM	CH8 S2.1 P121	The *SEM* knowledge area therefore addresses the management of software development and the measurement and modeling of software development.	Dans le domaine du SEM, le lien entre la prise de mesures et la gestion est fortement présenté. Il y est question de gestion et de mesure tout au long du	DEF	Scope and Definition

Tableau VII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			chapitre.		
	CH8 S2.5 P122	[...], the *SEM* knowledge area consist of both the management process and measurement / metrics sub-areas. [...] In essence, management without measurement, qualitative and quantitative, suggests a lack of rigor, and measurement without management suggests a lack of purpose context. [...] Effective management requires a combination of both numbers and stories.	Sans mesure, il n'est pas possible d'avoir un suivi adéquat du projet par les gestionnaires.	DEF	Management and Measurement
	CH8 S3.B.2.4 P125	Effort, schedule and cost estimation – breakdown of tasks, inputs and outputs, the expected effort range based on the required		N+	Effort, schedule and cost estimation

Tableau VII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		for each is determined using a calibrated estimation model based on historical size-effort data where available and relevant [...].			
	CH8 S3.C P127	3.C Software engineering measurement	Tout ce qui entoure la gestion des mesures est présenté dans cette section du guide, dont l'utilisation des mesures dans la prise de décision.	SUJET	Software engineering measurement
SEP	CH9 S3.1 P140	Process Measurement: This is concerned with quantitative techniques to diagnose software processes; to identify strengths and weaknesses. This can be performed to initiate process improvement and change, and afterwards		DEF	Software Engineering Process Concepts

Tableau VII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		to evaluate the consequences of process implementation and change.			
	CH9 S3.3 P141	Process measurement, as used here, means that quantitative information about the process is collected, analyzed, and interpreted. Measurement is used to identify the strengths and weaknesses of processes, and to evaluate processes after they have been implemented and/or changed [...].	Les mesures sont utilisées pour vérifier le processus en place ou pour confirmer tout changement à ce dernier.	N1	Process Measurement
SETM	CH10 S3.I P159	I. Software Engineering Management Tools Management tools are subdivided into three	Les outils à utiliser pour la gestion de projet logiciel incluent les mesures.	N1	Software Engineering Management Tools

Tableau VII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		categories: project planning and tracking, risk management, and measurement.			
SQ	CH11 S1 P165	Measurement of product quality at all levels of the project will in the future become more important [...]. [...], the questions of quality go beyond whether the system works or not, to how well it achieves measurable quality goals.		INTRO	Introduction
	CH11 S2.2 P169	SQA and V&V also provide management with visibility into the quality of products at each stage in their development or maintenance. The visibility comes from data and	Les mesures permettent d'offrir une visibilité aux gestionnaires sur la qualité des produits résultant de chacune des activités du cycle de vie du logiciel. Il leur est	SUJET	Purpose and Planning of SQA and V&V

Tableau VII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		measurements produced through the performance of tasks to assess and measure quality of the outputs of any software life cycle processes as they are developed.	ainsi plus facile d'évaluer les besoins d'améliorations des produits ou des processus en place.		
	CH11 S2.5.1 P174	Measurement programs are considered useful if they help project stakeholders (1) understand what is happening during their processes, and (2) control what is happening in their projects.	Lorsqu'il est question de contrôler le processus, cela inclut les décisions de réaliser des changements aux activités en cours ou à venir (ex. : modifi-cations au plan du projet).	N1	Fundamentals of Measurement
	CH11 S2.5.2 P175	If they are designed properly metrics can support software quality (among other aspects of the software engineering process) in multiple ways.		N+	Metrics

Tableau VII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		They can help management decision-making.			

4.1.2 Build with and for reuse

Pour reconnaître la correspondance avec ce principe fondamental, nous avons utilisé les deux parties du principe séparément ou en combinaison. Les deux parties du principe sont 1) la production du logiciel en réutilisant des composants déjà existants (Build with reuse), et 2) la production des composants du logiciel en fonction d'une réutilisation future (Build for reuse). La correspondance complète sera reconnue si une citation présente les deux parties du principe fondamental..

Les domaines où le principe est surtout reconnu, sont le SD, le SC et le ST; des domaines où sont décrites les activités du processus de développement du logiciel. Le tableau VIII présente les correspondances reconnues dans le Guide SWEBOK avec la réutilisation lors de la production d'un logiciel.

Tableau VIII

Analyse de la correspondance avec le principe fondamental (B)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SR	CH2 S2.2 P11	System developers – these have a legitimate interest in profiting from developing the system by, for example, reusing components in different products. If, in this scenario, a customer of a particular product has specific requirements that compromise the potential for component reuse, the developer must carefully weigh their own stake against those of the customer.		DEF	System requirements and process drivers
	CH2 S2.3 P12	Architectural design is a skill that is driven by many factors such as the recognition of	• Build for reuse : Création de "patterns"	DEF	Overview of requirement analysis

Tableau VIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		reusable architectural "patterns" or the existence of off-the shelf components.	• Build with reuse : Utilisation de composants d'une tierce partie (off-the shelf components)		
SD	CH3 S2 P36	[...], FP-design (Family Pattern design, whose goal is to establish exploitable commonalities over a family of systems) [...]	La correspondance est reconnue par rapport au « Build for reuse » seulement.	DEF	Definition of Software Design
	CH3 S3.III P38	But they can also be useful for designing generic systems, leading to the design of families of systems [...]. Interestingly, most of these notions can be seen as attempts to describe, and thus reuse, generic design knowledge.	Certaines notions utilisées en architecture permettent de créer des systèmes génériques. Ces notions voient à décrire, et même à réutiliser des connaissances génériques identifiées au moment du design.	SUJET	Software Structure and Architecture

Tableau VIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			Cette section du guide énumère certaines de ces notions. Certaines ciblent la production de logiciel en fonction de la réutilisation (« Architectural styles » et « Design patterns »), alors que d'autres ciblent plutôt la réutilisation de composants déjà existants (« Families of programs » et « frameworks »).		
	CH3 S3.IV P39	An interesting distinction is the one between quality attributes discernable at run-time (e.g., [...]), those not discernable at run-time (e.g., modifiability, portability, reusability,	L'analyse de la qualité du design évalue le degré de réutilisation à l'intérieur du logiciel ou entre les familles de logiciel. Il est donc reconnu qu'une des qualités d'un	SUJET	Software Design Quality Analysis and Evaluation

Tableau VIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		integrability and testability) [...].	logiciel est la production basée sur la réutilisation. La mesure de la réutilisation ne fait pas de distinction entre le « with » et le « for » du principe fondamental, mais il serait possible de le faire en fonction de critères distinguant le « with » et le « for ».		
SC	CH4 S2.5 P56	Automated construction is also reuse-intensive construction, since by limiting human options it allows the controlling software to make more effective use of its existing store of effective software problem solutions.	Cette correspondance a été reconnue avec « Build with reuse ». Il est question de produire des outils automatisant la construction de certaines parties du logiciel.	DEF	Manual and Automated Construction/ The Spectrum of Construction Techniques

Tableau VIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH4 S2.5 P56	When simple selections from a list of options will not suffice, software engineers often can still develop application specific tool kits (that is, sets of reusable parts designed to work with each other easily) to provide a somewhat lesser level of control.	Cette correspondance a été reconnue avec « Build for reuse » : l'ingénieur logiciel peut développer des ensembles d'outils logiciels réutilisables, s'emboîtant les uns dans les autres, pour limiter les efforts de contrôles.	DEF	Manual and Automated Construction/ The Spectrum of Construction Techniques
	CH4 S2.6 P56	*Toolkit languages* are use to build applications out of toolkits (integrated sets of application-specific reusable parts), and are more complex than configuration languages.	Correspondance avec « Build with reuse ».	DEF	Construction Languages
	CH4 S3.1.2.1 P58	Generalization is the process of recognizing how a few specific	Correspondance avec « Build for reuse ».	N+	Generalization

Tableau VIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		problem cases fit together as part of some broader framework of problems, and thus can be solved by a single overarching software construction in place of several isolated ones.			
ST	CH5 S3.E1 P81	To carry out testing or maintenance in an organized and cost/effective way, the means used to test each part of the system should be reused systematically. At all levels of testing, test scripts, test cases, and expected results should be carefully defined and documented so that they may be reused.	La correspondance est d'abord faite avec « Build for reuse » pour qu'une fois le logiciel implanté, son évolution soit faite en fonction du « Build with reuse ».	N+	Management Concerns

Tableau VIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SM	CH6 S3.4.2 P94	Re-engineering is defined as the examination and alteration of the subject system to reconstitute it in a new form, and the subsequent implementation of the new form.	La ré-ingénierie voit à examiner les composants d'un système pour le reconstituer dans une nouvelle forme et les futures implantations de cette nouvelle forme. Il s'agit d'une correspondance avec « Build with reuse ».	N1	Re-engineering
SCM	CH7 S3.I P104	The environment for software engineering includes such things as the: [...] - Software reuse processes, [...].	Cette correspondance avec la réutilisation n'a pas de lien précis avec le domaine de connaissances en question. C'est un texte qui a été recueilli à l'intérieur du SCM au sujet de la réutilisation des processus.	SUJET	Management of the SCM Process
SEM	CH8 S3.C.4 P129	Collection data – [...] The possibility of reusing metrics collected for other	C'est une correspondance avec « Build with reuse ». Il est possible de	N+	Collection of data

Tableau VIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		purposes is also considered as part of the collection process.	réutiliser les mesures de projets antérieurs pour les besoins de projets futurs ou pour d'autres besoins identifiés après avoir défini les besoins en mesure pour les activités du projet.		
SEP	Aucune				
SETM	CH10 S2 P155	Tools allow repetitive, well-defined actions to be automated, thus reducing the cognitive load on the software engineer.	Correspondance avec « Build with reuse ».	DEF	Definition of the Software Engineering Tools and Methods Knowledge Area
SQ	CH11 S2.1.4 P167	Other considerations of software systems are known to affect the software engineering process while the system is being built and during its future evolution or modification, and	Correspondance avec les deux parties du principe fondamental.	N+	Special Types of Systems and Quality Needs

Tableau VIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		these can be considered elements of product quality. These software qualities include, but are not limited to: [...] - Code and object reusability [...]			

4.1.3 Control complexity with multiple perspectives and multiple levels of abstraction

Dans la majorité des domaines de connaissances, la complexité du logiciel est mentionnée. Mais, le contrôle de cette complexité n'est pas toujours explicite. C'est souvent par l'interprétation d'une citation que nous pouvons l'identifier. Pour effectuer la correspondance avec ce principe, nous avons identifié les citations selon deux points de vue par rapport au contrôle de la complexité : 1) selon les différentes perspectives ou 2) selon les niveaux d'abstraction. Il y a donc des citations qui ne font référence qu'à l'une des deux vues, alors que d'autres couvrent les deux. Pour qu'une citation soit considérée, il faut qu'il y ait plus d'une perspective ou plus d'un niveau d'abstraction de représenté dans celle-ci.

Le tableau IX identifie les citations où il a été possible de reconnaître une correspondance avec le contrôle de la complexité de multiples perspectives et/ou par plusieurs niveaux d'abstraction.

Tableau IX

Analyse de la correspondance avec le principe fondamental (C)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SR	CH2 S2.3 P12	The requirements allocated to components that are complex systems in themselves will need to undergo further cycles of analysis in order to add more detail, and to interpret the domain-oriented system requirements for developers [...]	L'une des façons utilisées, par l'ingénieur logiciel pour traiter la complexité, est d'effectuer plusieurs cycles d'analyses des exigences pour détailler au maximum les besoins et ainsi faciliter la compréhension du domaine du système par les développeurs. Il s'agit donc d'une correspondance partielle par rapport aux niveaux d'abstraction.	DEF	Overview of requirements analysis

Tableau IX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH2 S3.3.3 P19	Allocation is important to permit detailed analysis of requirements. Hence, for example, once a set of requirements have been allocated to a component, they can be further analyzed to discover requirements on how the component needs to interact with other components in order to satisfy the allocated requirements. In large projects, allocation stimulates a new round of analysis for each subsystem.	Correspondance avec « control complexity with multiple levels of abstraction ».	N1	Architectural design and requirements allocation
SD	CH3 S2 P35	Software design plays an important role in the development of a software system in that it allows the developer to produce	La production de différents modèles permet de présenter la solution visée selon plusieurs points de vue (perspectives).	DEF	Definition of Software Design

Tableau IX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		various models that form a kind of blueprint of the solution to be implemented. These models can be analyzed and evaluated to determine if they will allow the various requirements to be fulfilled. Various alternative solutions and tradeoffs can also be examined and evaluated.			
	CH3 S3.I P36	In the context of software design, two key abstraction mechanisms are abstraction by parameterization and by specification, which in turn lead to three major kinds of abstraction: procedural	Cette partie du domaine du SD présente explicitement les niveaux d'abstraction par lesquels la complexité du logiciel est analysée.	SUJET	Software Design Basic Concepts

Tableau IX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		abstraction, data abstraction and control (iteration) abstraction [...].			
	CH3 S3.III P38	[...] "a software architecture is a description of the subsystems and components of a software system and the relationships between them" [...]	En décomposant le logiciel en sous-systèmes, puis en composants, le travail de compréhension du système se fait à plusieurs niveaux d'abstraction.	SUJET	Software Structure and Architecture
	CH3 S3.VI P40	Various general strategies can be used to help guide the design process [...]. By contrast with general strategies, methods are more specific in that they generally suggest and provide I) a set of notations to be used with the method; ii) a description of the process to be used	Cette section du guide présente des stratégies et méthodes pour le design logiciel. Il est possible d'utiliser plusieurs méthodes pour contrôler la complexité du logiciel selon de multiples perspectives. Il y a aussi des exemples de stratégies comme	SUJET	Software Design Strategies and Methods

Tableau IX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		when following the method; iii) a set of heuristics that provide guidance in using the method [...].	« diviser et conquérir » et « raffinement par étape » (« stepwise refinement ») qui permettent de contrôler la complexité à plusieurs niveaux d'abstraction.		
SC	CH4 S2.1 P54	Software design methods are used to express a global solution as a set of smaller solutions and can be applied repeatedly until the resulting parts of the solution are small enough to be handled with confidence by a single developer.	Correspondance avec la contrôle de la complexité selon les niveaux d'abstraction.	DEF	Software Construction and Software Design
	CH4 S2.1 P54	In design the emphasis is on how to partition a complex problem effectively, while in	Cette citation démontre comment la complexité du logiciel est contrôlée	DEF	Software Construction and Software Design

Tableau IX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		construction the emphasis is on finding a complete and executable solution to a problem.	du design à la construction. Il y a le contrôle à différents niveaux d'abstraction en passant du détail du design à l'exécutable présentant la solution finale. Puis le contrôle de différentes perspectives où le design produit des modèles, alors que la construction produit des librairies et un exécutable.		
	CH4 S3.1.1 P57	This need for simplicity in the human-to-computer interface leads to one of the strongest drivers in software construction : reduction of complexity. [...]	Il s'agit du contrôle de la complexité selon plusieurs perspectives.	SUJET	Reduction of Complexity

Tableau IX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		There are three main techniques for reducing complexity: 3.1.1.1 Removal of complexity [...] 3.1.1.2 Automation of complexity [...] 3.1.1.3 Localization of complexity [...]			
	CH4 S3.1.3 P59	This modularity allows both improved analysis and thorough unit-level testing of such components before they are integrated into higher levels in which their errors may be more difficult to identify.	En effectuant des tests à différents niveaux du produit, on évalue les composants avec différents objectifs. Ces différents objectifs permettent de contrôler la complexité à différents niveaux d'abstraction.	SUJET	Structuring for Validation
	CH4 S3.3.1 P62	3.3.1 Reduction in complexity 3.3.1.1 Linguistic Construction Methods [...]	Cette section présente des méthodes (linguistique, formelle et visuelle)	SUJET	Reduction in Complexity

Tableau IX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		3.3.1.2 Formal Construction Methods [...] 3.3.1.3 Visual Construction Methods [...]	de construction permettant de réduire la complexité selon plusieurs perspectives.		
ST	CH5 S3.B1 P75	Testing of large software systems usually involves more steps [...]. Three big test stages can be conceptually distinguished, namely Unit, Integration and System.	En testant le logiciel à plusieurs niveaux d'abstraction, l'ingénieur logiciel peut contrôler la complexité de son produit.	N1	The target of the test
	CH5 S3.B2 P76	Testing of a software system (or subsystem) can be aimed at verifying different properties. [...]	Les objectifs fixés pour le déroulement des tests permettent de tester le système selon plusieurs points de vue. Chaque type de test étant réaliser pour atteindre un objectif lui étant propre contre le même système. Le	N1	Objectives of testing

Tableau IX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			but est donc de contrôler la complexité du logiciel selon de multiples perspectives comme l'installation, les impacts des changements ou la fiabilité, par exemple.		
	CH5 S3.C P77	The first classification, [...], is based on how tests are generated, i.e., respectively from : tester's intuition and expertise, the specifications, the code structure, the (real or artificial) faults to be discovered, the field usage or finally the nature of application, [...]	Cette section du domaine du ST présente les techniques de tests. Y sont présentées deux méthodes de classification qui permettent de contrôler la complexité du logiciel selon plusieurs perspectives ou selon plusieurs niveaux d'abstraction. La	SUJET	Test Techniques

Tableau IX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		The second classification is the classical distinction of test techniques between black-box and white-box.	classification des techniques de tests, basée sur la façon dont les tests sont produits, démontre un moyen de contrôler la complexité des multiples points de vue. La classification entre « black-box » et « white-box » est plutôt un moyen de contrôler la complexité selon le niveau d'abstraction. Les tests de type « black-box » dépendent du comportement du système selon les entrées/sorties (une vue externe), alors que les tests de type « white-box » se concentrent sur la		

Tableau XI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			façon dont le design et le code ont été réalisés (une vue interne).		
SM	CH6 S3.1.5 P90	The four major aspects that maintenance focuses on are [Pfl98] : Maintaining control over the system's day-to-day functions. • Maintaining control over system modification. • Perfecting existing acceptable functions. • Preventing system performance from degrading to unacceptable levels. Accordingly, software must evolved and be maintained.	En regardant attentivement cette citation, nous pouvons voir que l'ensemble des activités du SM est là pour contrôler la complexité du système au cours de son évolution. Il s'agit de contrôler la complexité selon plusieurs perspectives.	N1	Need for Maintenance

Tableau IX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH6 S3.1.6 P90	The categories of maintenance defined by ISO/IEC are as follows: • Corrective maintenance. [...] • Adaptative maintenance. [...] • Perfective maintenance. [...] • Preventive maintenance. [...]	Les catégories de maintenance identifient des approches selon lesquelles le logiciel est contrôlé au cours de la maintenance.	N1	Categories of Maintenance
	CH6 S3.4.3 P95	Reverse engineering is the process of analyzing a subject to identify the system's components and their relationships and to create representations of the system in another form or at higher levels of abstraction.	Au cours des activités de maintenance, il peut être nécessaire de visualiser le logiciel de façon graphique, à l'aide d'outils spécialisés, en partant du code ou des données. Cette façon de faire permettra à l'ingénieur logiciel de mieux comprendre	N+	Reverse Engineering

Tableau IX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			la complexité du système, à différents niveaux d'abstraction.		
SCM	Aucune				
SEM	CH8 S3.C.3.2 P128	Structure measurement – a diverse range of measures of software product structure may be applied to both high- and low-level design and code artifacts to reflect control-flow [...], data-flow [...], nesting [...], control structures [...], and modular structure and interaction [...].	La mesure de la structure du logiciel peut se faire selon les niveaux d'abstraction ou selon différentes perspectives.	N+	Structure Measurement
SEP	Aucune				
SETM	Aucune				
SQ	CH11 S2.3.1.2 P172	An individual generally applies analytic techniques.	Des techniques d'analyse permettent de supporter les	N+	Analytic Techniques

Tableau IX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		An individual generally applies analytic techniques. [...] The support group of techniques also includes various assessments as part of overall quality analysis. [...] An example of a support technique is complexity analysis, useful for determining that the design or code may be too complex to develop correctly, to test or maintain;	Des techniques d'analyse permettent de supporter les activités de SQA et de V&V. L'une de ces techniques est l'analyse de la complexité qui est l'une des perspectives de contrôle de la complexité.		

4.1.4 Define software artifacts rigorously

Il y a deux points de vue à considérer lorsqu'il est question d'une identification « rigoureuse » des artéfacts. Il y a la vue d'identification de l'artéfact même : le contenant – le produit résultant de l'activité – et le contenu – ce qui s'y retrouve. Puis, il y a la vue des activités permettant de produire l'artéfact et de contrôler le suivi des règles

définies (méthodes et standards) pour la production de l'artéfact. La correspondance avec le principe fondamental a été réalisée à partir de ces deux points de vue.

Pour ce qui est du contenant et du contenu (l'artéfact même), il y a une grande dépendance avec les méthodes et les standards en place. Ce sont les méthodes et les standards qui prescrivent le format et la forme à l'intérieur de l'artéfact. De son côté, le processus doit définir les artéfacts à produire au cours de ses activités et les activités pour en vérifier et valider le contenu.

En général, le guide SWEBOK définit rigoureusement les différents artéfacts à produire au cours des activités définies à l'intérieur des domaines de connaissances. Le tableau X présente les citations du Guide SWEBOK où il a été possible de reconnaître une correspondance avec la définition rigoureuse des artéfacts.

Tableau X

Analyse de la correspondance avec le principe fondamental (D)

Domaine	**Repère**	**Citation**	**Interprétation**	**Niveau**	**Titre de la section**
SR	CH2 S2.2 P11	The elicitation and analysis of system requirements needs to be driven by the need to achieve the overall project aims. To provide this focus, a business case should be made which clearly	Il s'agit d'une correspondance avec la production de l'artéfact.	DEF	System requirements and process drivers

Tableau X (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		defines the benefits that the investment must deliver.			
	CH2 S2.5 P13	Good requirements engineering requires that the products of the process – the deliverables – are defined. [...] A document that specifies the system requirements. This is sometimes known as the requirements definition document, user requirements document or, [...], the concepts of operations (ConOps) document. [...] A document that specifies the software requirements. This is sometimes known as the software requirements	Il s'agit d'une correspondance complète avec le principe fondamental. Il est d'abord question de définir les artéfacts résultants des activités du processus. Puis, on y présente des artéfacts avec ce qui doit s'y retrouver.	DEF	Products and deliverables

Tableau X (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		specification (SRS).			
	CH2 S2.5 P14	The requirements document(s) must be subject to validation and verification procedures. [...] It is also important to verify that a requirements document conforms to company standards, and is understandable, consistent and complete.	Il s'agit d'une correspondance avec la production rigoureuse de l'artéfact par rapport aux activités de contrôle du suivi des standards de l'organisation.	DEF	Products and deliverables
	CH2 S3.4 P21	This may take the form of two documents, or two parts of the same document with different readership and purposes (see 2.5): the requirements definition document and the software requirements specification.	Il s'agit d'une correspondance avec l'identification rigoureuse des artéfacts à produire.	SUJET	Software requirements specification

Tableau X (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SD	CH3 S2 P35	Software design plays an important role in the development of a software system in that it allows the developer to produce various models that form a kind of blueprint of the solution to be implemented. These models can be analyzed and evaluated to determine if they will allow the various requirements to be fulfilled.	Il s'agit d'une correspondance avec le contrôle du contenu de l'artéfact.	DEF	Definition of Software Design
	CH3 S3.I P36	The output of this process is a set of models and artifacts that record the major decisions that have been taken [...].	Il s'agit d'une correspondance avec les artéfacts à produire pour les activités du processus.	SUJET	Software Design Basic Concepts
	CH3 S3.III P38	The key idea is that a software design is a multi-faceted artifact	Il s'agit d'une correspondance avec l'identification des	SUJET	Software Structure and Architecture

Tableau X (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		produced by the design process and generally composed of relatively independent and orthogonal views.	artéfacts produits par les activités du processus.		
SC	CH4 S2.1 P54	This definition also recognizes the distinction that while software construction necessarily produces executable software, software design does not necessarily produce any executable products at all.	Il s'agit d'une correspondance avec l'identification des artéfacts produits par les activités de SC.	DEF	Software Construction and Software Design
ST	CH5 S1 P69	Indeed, planning for testing should start since the early stages of requirement analysis, and test plans and procedures must be systematically and continuously refined as the development proceeds.	Il s'agit d'une correspondance avec l'identification de l'artéfact à produire pour une activité du processus.	INTRO	Introduction

Tableau X (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH5 S3.E1 P80	Documentation is an integral part of the formalization of the test process. [...] Test documents includes, among others, Test Plan, Test Design Specification, Test Procedure Specification, Test Case Specification, Test Log and Test Incident or Problem Report. [...] Test documentation should be produced and continually updated, at the same standards as other types of documentation in development.	Il s'agit d'une correspondance avec l'identification des artéfacts à produire par les activités de ST.	N+	Management Concerns
SM	CH6 S3.2 P91	Each of the ISO/IEC 14764 primary software maintenance activities is further broken down into	Il s'agit d'une correspondance avec l'identification des artéfacts à produire par les activités du	N1	Maintenance Process

Tableau X (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		tasks as follows: [...] Develop maintenance plans and procedures [...] Develop a migration plan [...] Develop a retirement plan [...]	processus.		
	CH6 S3.2.2.2.2 P92	It is not sufficient to simply hope that increased quality will result from the maintenance of software. It must be planned and processes implemented to support the maintenance process. [...] This is implemented by developing and following SQA and V&V plans and procedures.	Pour être rigoureux dans la production des artéfacts, il faut mettre en place des activités permettant de vérifier et de valider ces artéfacts.	N+	Quality

Tableau X (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH6 S3.2.2.2.3 P92	Once the maintenance concept is determined, the next step is to develop the maintenance plan. The maintenance plan should be prepared during software development and should specify how users will request modifications or report problems.	Il s'agit d'une correspondance complète où l'on identifie le produit de l'activité de planification de la maintenance, et ce qui doit se retrouver à l'intérieur de l'artéfact.	N+	Maintenance Planning Activity
SCM	CH7 S3.I.C P106	The results of planning activity are recorded in a Software Configuration Management Plan (SCMP). The SCMP is typically subject to SQA review and audit.		N1	Planning for SCM
	CH7 S3.I.D	The results of SCM planning for a given project are recorded in a Software Configuration		SUJET	Software Configuration Management Plan

Tableau X (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		Management Plan (SCMP). The SCMP is a "living document" that serves as a reference for the SCM process. It is maintained [...] as necessary during the software life cycle.			
	CH7 S3.II P108	A first step in controlling change is to identify the software items to be controlled. [...] A variety of items, in addition to the code itself, are typically controlled by SCM. Software items with potential to become SCIs include plans, specifications and design documentation, testing materials, software tools, source and executable code,	Il s'agit d'une correspondance avec l'identification des artéfacts produits par les différentes activités du processus.	SUJET	Software Configuration Identification

Tableau X (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		code libraries, data and data dictionaries, and documentation for installation, maintenance, operations and software use.			
	CH7 S3.VI.B P111	Software release management encompasses the identification, packaging and delivery of the elements of a product, for example, the executable, documentation, release notes, and configuration data.	La gestion du déploiement du logiciel inclut l'identification des éléments du produit qui seront implantés en plus des programmes (exécutables) du logiciel. Il s'agit d'une correspondance avec l'identification des artéfacts produits par les différentes activités du processus.	N1	Software Release Management
SEM	CH8 S2.1	The scope of this knowledge area	En plus d'identifier les artéfacts à	INTRO	Introduction

Tableau X (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	P121	follows the general focus of the Guide; that is, "emphasis [...] is placed upon the construction of useful software artifacts" [...].	produire par les activités du processus, il faut pouvoir évaluer la pertinence de produire ces artéfacts.		
	CH8 S3.B.2.3 P125	Determine deliverables – the product(s) of each task (e.g. high level architectural design, inspection report) are specified and characterized.	Cette citation est une correspondance avec l'identification des artéfacts devant être produits par les différentes activités du processus et par le besoin de définir le contenant et le contenu de ces artéfacts.	N+	Determine deliverables
SEP	CH9 S3.1.1 P139	At the center of the cycle is the "Process Experience Base". This is intended to capture lessons from past iterations of the cycle (e.g. previous evaluations, process	Le PEB constitue le principal artéfact du SEP. Cette base d'informations sur le processus sert pour définir les changements à réaliser pour	N1	Themes

Tableau X (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		definitions, and plans). Evaluation lessons can be qualitative or quantitative. No assumptions are made about the nature or technology of this "Process Experience Base", only that it be a persistent storage.	améliorer le processus supportant le cycle de vie du logiciel. Son implantation permet d'identifier les lacunes du processus et les artéfacts produits par les différentes activités.		
SETM	CH10 P3.I P156	I. Software tools	Les outils peuvent être des artéfacts ou servir à produire des artéfacts.	SUJET	Software Tools
SQ	CH11 S2.2.2 P170	The SQA plan defines the processes and procedures that will be used to ensure that software developed for a specific product meets its requirements and is of the highest quality possible within project constraints. [...] It must consider management,	Pour que le plan du SQA soit développé rigoureusement, il doit considérer les plans de maintenance, de développement et de gestion du projet.	N1	The SQA Plan

Tableau X (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		development and maintenance plans for the software.			
	CH11 S2.2.3 P171	The V&V plan is the instrument to explain the requirements and management of V&V and the role of each technique in satisfying the objectives of V&V. [...] The plan for V&V addresses the management, communication, policies and procedures of the V&V activities [...].	Il s'agit d'une correspondance complète.	N1	The V&V Plan
	CH11 S2.5.1 P174	The measurement process and its implementation should be documented in the form of a measurement plan.	C'est une correspondance avec l'identification des artéfacts à produire par les activités du processus. Le niveau de correspondance reconnu est « DEF » car, même si la	DEF	Fundamentals of Measurement

Tableau X (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			citation a été reconnue à l'intérieur de la taxonomie de niveau « N1 », il s'agit d'une section présentant des éléments fondamentaux du domaine de connaissances.		

4.1.5 Establish a software process that provides flexibility

La flexibilité du processus n'est pas un sujet dont la correspondance est simple à retrouver. La terminologie retrouvée dans le guide ne parle pas explicitement de mettre en place un processus flexible. C'est plutôt en expliquant que certaines activités pourraient être adaptées selon le contexte de l'organisation, du projet ou du produit que la correspondance a pu se faire. C'est en identifiant des citations, où le Guide SWEBOK suggère d'adapter le processus ou les artéfacts à produire selon le contexte du projet ou du produit, que nous avons reconnu les correspondances citées.

Le tableau XI présente les citations où il a été possible de reconnaître une correspondance dans le Guide SWEBOK avec l'établissement d'un processus logiciel permettant de la flexibilité.

Tableau XI

Analyse de la correspondance avec le principe fondamental (E)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SR	CH2 S2.4 P13	In practice, therefore, it is almost always impractical to implement requirements engineering as a linear, deterministic process where system requirements are elicited from the stakeholders, baselined, allocated and handed over to the software development team.	Le processus de gestion des exigences doit être flexible pour permettre plus d'une ronde d'analyses. La complexité des systèmes fait que ceux-ci sont rarement entièrement compris du premier coup, et libres de tout problème après la première ronde d'analyse.	DEF	Requirements engineering in practice
	CH2 S3 P15	We have chosen the process-based breakdown to reflect the fact that requirements engineering, if it is to be successful, must be considered as a	Le découpage du domaine du SR a été défini pour démontrer que l'ingénierie des exigences ne produit pas un résultat dans un processus linéaire,	DEF	Breakdown of Topics for Software Requirements

Tableau XI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		process with complex, tightly coupled activities (both sequential and concurrent) rather than as a discrete, one-off activity at the outset of a software development project.	mais qu'il s'agisse plutôt d'un processus itératif qui se doit d'être flexible pour obtenir une qualité et un détail permettant de prendre des décisions pertinentes. Il faut donc que le processus permette cette flexibilité de réaliser plusieurs rondes d'analyses selon la complexité du produit à développer.		
	CH2 S3 P16	The different activities in requirements engineering are repeated until an acceptable requirements specification document is produced or until external factors such as	Il est primordial que le processus d'ingénierie des exigences soit flexible pour permettre l'itération à travers ses activités jusqu'à l'atteinte des critères de sortie du processus.	DEF	Breakdown of Topics for Software Requirements

Tableau XI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		schedule pressure or lack of resources cause the requirements engineering process to terminate.			
	CH2 S3.1.1 P16	[...] requirements engineering process: [...] - will need to be tailored to the organization and project context.	Le processus d'ingénierie du logiciel se doit d'être flexible pour pouvoir s'adapter au contexte de l'organisation ou du projet.	N1	
	CH2 S3.1.1 P16	In particular, the subtopic is concerned with how the activities of elicitation, analysis, specification, validation and management are configured for different types of project and constraints.		N1	Process models
SD	CH3 S2 P35	[...] it (SD) allows the developer to produce various models that	Le processus de SD doit être assez flexible pour	DEF	Definition of Software

Tableau XI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		form a kind of blueprint of the solution to be implemented. These models can be analyzed and evaluated to determine if they will allow the various requirements to be fulfilled. Various alternative solutions and tradeoffs can also be examined and evaluated.	permettre d'identifier des solutions alternatives ou des modèles présentant des compromis par rapport aux exigences soulevées. L'échéancier du projet a un impact important sur les options offertes concernant l'identification de solutions alternatives.		Design
SC	CH4 S2 P53	An important part of software engineering is to make a rational choice of development style for a given software project.		DEF	Definition of Software Construction
	CH4 S2.1 P54	Firstly, software construction is influenced by the scale or size of the software product being .	Dans un processus flexible, les activités de SD et de SC pourraient être combinées dépendant	DEF	Software Construction and Software Design

Tableau XI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		constructed. Very small projects in which the design problems are already "construction size" may neither require nor need an explicit design phase, [...]	de la taille et de la complexité du projet.		
ST	CH5 S3.B1 P75	Testing of large software systems usually involves more steps [...]. [...] Depending on the development model followed, these tree stages will be adopted and combined in different paradigms, and quite often more than one iteration between them is necessary.	Le processus de ST doit pouvoir s'adapter à la taille du logiciel à évaluer. Par exemple, les trois niveaux de tests (unitaire, intégré et de système) peuvent être combinés pour un changement mineur contrôlé.	N1	The target of the test
	CH5 S3.B2 P76	Note that some kinds of testing are more appropriate for custom made packages (e.g.	Le processus du ST doit permettre d'adapter les types de tests à effectuer au	N1	Objectives of Testing

Tableau XI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		installation testing), while others for generic products (e.g., beta testing).	type de produit. Il s'agit de définir les objectifs des tests à réaliser.		
SM	Aucune				
SCM	CH7 S3.I.B P105	Policies and procedures set forth at corporate or other organizational levels might influence or prescribes the design and implementation of the SCM process for a given project.	Le processus du SCM doit pouvoir s'adapter au contexte du projet.	N1	Constraints and Guidance for SCM
	CH7 S3.III.A.1 P110	The authority for accepting or rejecting proposed changes rests with an entity typically known as a Configuration Control Board (CCB). In smaller projects, this authority actually may reside with the responsible leader or an assigned individual,	Le processus du SCM doit permettre d'adapter la taille du CCB à la taille du projet.	N1	Software Configuration Control Board

Tableau XI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		rather than a multi-person board.			
SEM	CH8 S2.1 P121	The scope of this knowledge area follows the general focus of the Guide; that is, "emphasis [...] is placed upon the construction of useful software artifacts" [...].	En plus d'identifier les artéfacts à produire par les activités du processus, il faut pouvoir évaluer la pertinence de produire ces artéfacts. Avec un processus flexible, il est possible qu'il n'y ait pas un besoin de produire un certain artéfact, selon le contexte du projet.	DEF	Scope and Definition
	CH8 S3.B.3.4 P126	Control process – the outcomes of the process monitoring activities provide the basis on which action decisions are taken. Where appropriate, and where impact and associated risks are	Les résultats des activités de contrôle ayant lieu à l'intérieur du processus permettent d'identifier les changements à réaliser. Le processus doit être .	N+	Control Process

Tableau XI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		modeled and managed, changes can be made to the process/project.	assez flexible pour effectuer les changements nécessaires. Ces changements peuvent être permanents ou temporaires. Par exemple, un changement temporaire pourrait être nécessaire en raison du contexte d'un projet.		
SEP	CH9 S3.4 P144	It should be noted also that the context of the project and organization will determine the type of process definition that is most important. Important variables to consider include the nature of the work [...], the application domain, the structure of the delivery process	Pour le développement de logiciels, il faut un processus flexible pouvant s'adapter à différents contextes (organisation, grosseur du projet, type de logiciel, etc.).	SUJET	Process Definition

Tableau XI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		[...], and the maturity of the organization.			
SETM	Aucune				
SQ	CH11 S2.3 P171	Static and dynamic techniques are used in either SQA or V&V. Their selection, specific objectives and organization depend on project and product requirements.	Le processus doit être assez flexible pour permettre une sélection des techniques à employer au cours de ses activités, selon le contexte du projet et du produit à développer.	SUJET	Activities and techniques for SQA and V&V

4.1.6 Implement a disciplined approach and improve it continuously

Pour produire la correspondance entre ce principe fondamental et le guide SWEBOK, nous avons ciblé le contenu du guide qui démontrait un lien avec l'amélioration du processus. Nous avons considéré que l'approche disciplinée dont il est question dans le libellé du principe fondamental, correspondait aux différents processus déployés pour couvrir l'ensemble du cycle de vie du logiciel. Nous n'avons pas divisé le libellé en deux comme pour certains autres principes fondamentaux. Ce choix a limité le nombre de correspondances, mais permet de faire la distinction entre ce principe et celui traitant de la production du logiciel par étapes logiques (J).

Il fallait aussi considérer ce qui est utilisé par l'organisation pour identifier ce qui doit être amélioré. Ainsi, tout ce qui a rapport à l'évaluation du processus a aussi été reconnu comme une correspondance avec le principe fondamental. Le tableau XII présente les correspondances reconnues par rapport à l'implantation d'une approche disciplinée devant être améliorée continuellement.

Tableau XII

Analyse de la correspondance avec le principe fondamental (F)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SR	CH2 S3.1.4 P17	[...] Its purpose is to emphasize the key role requirements engineering plays in term of the cost, timeliness and customer satisfaction of software products [Som97]. It will help to orient the requirements engineering process with quality standards and process improvement models for software and systems.	Cette citation couvre à la fois l'approche disciplinée de la réalisation des activités de la gestion des exigences dans le processus, que de l'amélioration continue du processus.	N1	Process quality and improvement

Tableau XII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH2 Table 2 P17	[...] If these indicate room for improvement (as they inevitably will) it is possible to measure the extent and rigor with which requirements 'good practice' is used in a process.	Les mesures prises au cours des activités de gestion des exigences permettent d'identifier les lacunes du processus de SR. Il est alors possible d'y appliquer les changements nécessaires pour en améliorer la qualité.	TEXTE	Aucun
SD	Aucune				
SC	CH4 S2.2 P54	In software engineering, a tool is a hardware or software device that is used to support performing a process. An effective tool is one that provides significant improvements in productivity and/or quality. [...] Good tools also improve software quality by .	Les outils, en ingénierie logiciel, permettent d'améliorer le processus en améliorant la qualité et la productivité. L'implantation d'un outil peut permettre d'améliorer le processus en place.	DEF	The Role of Tools in Construction

Tableau XII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		allowing people to avoid repetitive or precise work for which a computer is better suited.			
	CH4 S2.3 P55	Integrated evaluation means that a process (in this case a development process) includes explicit continuous or periodic internal checks to ensure that it is still working correctly. These checks usually consist of evaluations of intermediate work products [...], but they also look at characteristics of the development process itself.	L'évaluation des produits intermédiaires permet de vérifier la qualité des produits des différentes activités du processus de développement. Les résultats servent à identifier les faiblesses du processus et les améliorations pouvant y être apportées.	DEF	The Role of Integrated Evaluation in Construction
ST	CH5 S3.E1 P81	Evaluation of test phase reports is often combined with root cause analysis to	L'évaluation des rapports des phases de tests permet d'évaluer le	N1	Management concerns

Tableau XII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		evaluate test process effectiveness in finding faults as early as possible.	processus de test à trouver les anomalies le plus tôt possible. Les résultats de l'évaluation permettent d'identifier les activités du processus du ST à améliorer.		
	CH5 S3.E2 P81	Defect tracking information is used to determine what aspects of system development need improvement and how effective have been previous analyses and testing.		N+	Test Activities
SM	CH6 S3.3.4 P94	Software measures are vital for software process improvement but the process must be measurable.	Pour qu'un processus puisse être amélioré, il faut qu'il soit mesurable. Les mesures permettent d'identifier les faiblesses du	N+	Software Maintenance Measurement

Tableau XII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			processus et de cibler les améliorations possibles.		
SCM	CH7 S3.I.B P105	Guidance for designing and implementing an SCM process can also be obtained from 'best practice' as reflected in the standards on software engineering issued by the various standards organizations. [...] Best practice is also reflected in process improvement and process assessment models [...].	L'implantation des meilleures pratiques en génie logiciel, à l'intérieur du processus de SCM, permet d'améliorer les activités du processus.	N1	Constraints and Guidance for SCM
	CH7 S3.I.E.1 P107	SCM metrics can be designed to provide specific information on the evolving product or to provide insight into the functioning of the		N+	SCM Metrics and Measurement

Tableau XII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		SCM process. A related goal of monitoring the SCM process is to discover opportunities for process improvement.			
SEM	CH8 S3.B.3.4 P126	Control process – the outcomes of the process monitoring activities provide the basis on which action decisions are taken. Where appropriate, and where the impact and associated risks are modeled and managed, changes can be made to the process/project.	Il faut contrôler le processus de développement. Les données recueillies au cours des activités de contrôle du processus, permettent de cibler les possibilités d'amélioration. Avant d'effectuer des changements au processus, il faut valider l'impact de ceux-ci sur les projets et les produits en développement ou à développer.	N+	Control Process
	CH8 S3.B.4.2	The various methods, tools and techniques	Il est question de l'évaluation	N+	Reviewing and

Tableau XII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	P127	employed are evaluated for their effectiveness and appropriateness, and the process itself is systematically and periodically assessed for its relevance, utility and efficacy in the process/project context [...].	périodique du processus.		evaluating performance
	CH8 S3.B.5 P127	Once closure is established, archival, post mortem and process improvement activities are performed.	Des activités d'amélioration du processus doivent être prévues pour le projet, une fois le projet complété. Ces activités permettent d'identifier les lacunes du processus ayant été utilisé au cours du projet de développement.	N1	Closure
	CH8 S3.C.1 P127	Each measurement endeavor should be guided by	Un programme de mesure doit avoir comme but	N1	Determining the goals of a measurement

Tableau XII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		organizational objectives and driven by an over-riding goal that has organizational improvement at its foundation. […] 2. Software process improvement goals – organizational objectives are translated into specific software-related goals that, if achieved, can assist the organization in attaining its objectives […]	d'améliorer le processus en place, selon les objectifs organisationnels.		Program
SEP	CH9 S3.1.1 P138	[…] "All process work is ultimately directed at 'software process assessment and improvement'".		N1	Themes
	CH9 S3.2.1 P141	The SEPG is intended to be central focus for process improvement within an organization.	En mettant en place un groupe dont le mandat est le suivi du processus, l'organisation	N1	The Software Engineering Process Group

Tableau XII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			s'assure que des objectifs et des activités permettant l'amélioration du processus seront déployés.		
	CH9 S3.3 P141	Measurement is used to identify the strengths and weaknesses of process, and to evaluate processes after they have been implemented and/or changed [...].	Ce sont les mesures qui permettent d'identifier les forces et les faiblesses du processus.	SUJET	Process Measurement
SETM	CH10 S2 P155	Development methods impose structure on the software development activity with the goal of making the activity systematic and ultimately more to be successful [...].	Les méthodes de développement permettent de structurer le processus. Elles assurent donc une approche disciplinée du développement et facilitent le suivi du processus.	DEF	Definition of Software Engineering Tools and Methods

Tableau XII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH10 S3.II P160	Software Development Methods	Les méthodes de développement permettent de structurer le processus. Elles assurent donc une approche disciplinée du développement et facilitent le suivi du processus.	SUJET	Software Development Methods
SQ	CH11 S2 P166	Sections on the processes – SQA and V&V – that focus on software quality follow, the discussion on software in a given project. The quality-focused processes help to ensure better software in a given project. They also provide, as a by-product, general information to management that can improve the quality of		DEF	Breakdown of topics for Software Design

Tableau XII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		the entire software and maintenance processes.			
	CH11 S2.1.4 P167	These quality attributes can be viewed as satisfying organizational or project requirements for the software in the effort to improve the overall performance of the organization or project.	La mesure de la qualité du produit permet d'identifier ce qui doit être amélioré dans les activités du processus.	N1	Special Types of Systems and Quality Needs
	CH11 S2.2 P169	As described in this KA, the SQA and V&V processes are closely related processes that can overlap and are sometimes even combined. [...] They should also produce feedback that can improve the software engineering process.	Les processus du SQA et du V&V jouent un rôle important pour informer les intervenants des éléments à améliorer dans le processus en place.	SUJET	Purpose and Planning of SQA and V&V

4.1.7 Invest in the understanding of the problem

Au cours des activités du processus de développement du logiciel, c'est au niveau du SR où la majorité des efforts de compréhension du problème ont lieu. Par contre, dans la majorité des cas, la compréhension des exigences continue d'évoluer pendant le SD et les autres étapes du processus de développement. Elle constitue aussi la majorité des efforts au cours des activités de SM.

Il est important de bien définir ce qu'est le « problème » dans le libellé du principe fondamental. Il est certain que dans chacune des situations de la vie courante où l'on se retrouve, et même dans un projet de développement de logiciel, qu'un problème puisse se poser. Mais dans ce cas-ci, le problème se rapporte plutôt aux besoins exprimés par le client. Par exemple, la citation « [...], *many of the techniques of software design also apply to software construction, since dividing problems into smaller parts is just as much a part of construction as it is design* » [CH3S3.IP36] constitue une réponse à un problème d'un autre ordre que celui du problème soulevé par le client. Le problème soulevé par le client doit être explicité et analysé pour être compris et se transformer au bout du compte en un logiciel. Dans la citation présentée, il s'agit plutôt d'une réponse conceptuelle à la compréhension précédemment faite du problème soulevé.

Nous avons donc été en mesure d'identifier des correspondances avec ce principe fondamental dans les domaines de connaissances du SR, SD, SC, ST, SM et SCM. Le tableau XIII présente les correspondances reconnues par rapport à l'investissement dans la compréhension du problème.

Tableau XIII

Analyse de la correspondance avec le principe fondamental (G)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SR	CH2 P9	The knowledge area is concerned with the acquisition, analysis, specification, validation and management of software requirements.	Tout le domaine de connaissances du SR est lié à l'investissement dans la compréhension des besoins (problèmes soulevés) des différents intervenants pour le logiciel à produire.	DOM	Software Requirements
	CH2 S2.3 P11	Once the aims of the project have been established, the work of eliciting, analyzing and validating the system requirements can commence. This is crucial to gaining a clear understanding of the problem for which the system is to provide a solution and its likely costs.	Le travail d'explicitation, d'analyse et de validation des exigences de l'ingénieur logiciel permet d'obtenir une bonne compréhension du problème exposé. Ces activités doivent avoir lieu après l'identification des	DEF	Overview of requirements analysis

Tableau XIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			objectifs du projet.		
	CH2 S2.4 P13	In almost all cases requirements understanding continues to evolve as design and development proceeds.	La compréhension des exigences se poursuit tout au long du cycle de développement. Les activités de SD et de SC permettent de pousser plus loin cette compréhension.	DEF	Requirements engineering in practice
	CH2 S3.2 P17	Requirements elicitation is the first stage in building an understanding of the problem the software is required to solve.		N1	Requirements elicitation
	CH2 S3.3.2 P19	The development of models of the problem is fundamental to requirements analysis (see 2.4). The purpose is to aid understanding of the problem rather than to initiate design of the solution. … Note that in almost all		N+	Conceptual modeling

Tableau XIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		cases, it is useful to start by building a model of the system context. The system context provides an understanding between the intended system and its external environment.			
	CH2 S3.5.2 P22	Prototyping is commonly employed for validating the requirements engineer's interpretation of the system requirements, as well as for eliciting new requirements.	La production d'un prototype permet à l'ingénieur logiciel de valider sa compréhension de l'interprétation du problème.	N+	Prototyping
SD	Aucune				
SC	CH4 S3.1.2.2 P58	Experimentation means using early [...] software constructions in as many different user contexts as possible, and as early in the development	L'expérimentation peut être utilisée pour confirmer la compréhension du problème soulevé, tout en permettant d'identifier les	N+	Experimen-tation

Tableau XIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		process as possible, for the explicit purpose of collecting data on how to generalize the construction.	contextes d'utilisation génériques pour la construction du logiciel.		
ST	CH5 S3.E2 P81	• Defect tracking [...] defects should be analyzed to determine when they were introduced into the systems, what kind of error caused them to be created [...], and when they could have been first observed in the system.	Les anomalies doivent être analysées pour voir si elles proviennent de la mauvaise compréhension du problème soulevé par le client. Les résultats des tests peuvent permettre de découvrir qu'une exigence a besoin d'être mieux définie.	N+	Test Activities
SM	CH6 S3.1.4 P89	[...], Lehman stated that maintenance is really evolutionary developments and that maintenance decisions are aided by understanding what	Pour faciliter l'évolution du logiciel, il faut une bonne compréhension de ce dernier à travers le temps. Il est donc	N1	Evolution of Software

Tableau XIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		happens to systems (and software) over time.	nécessaire pour l'équipe veillant à la maintenance d'un logiciel, d'investir à la compréhension du problème auquel répond le logiciel.		
	CH6 S3.2.1 P91	Problem and Modification tasks are : • Perform initial analysis • Verify the problem • Develop options for implementing the modification • Document the results • Obtain approval for modification option.	Au cours des activités de maintenance, une analyse initiale du problème soulevé (anomalie ou évolution) permet à l'équipe de maintenance d'en comprendre toute l'étendue.	N+	Maintenance Process Models
	CH6 S3.3.1.1 P92	Practitioners and researchers indicate that some 40% to 60%	La maintenance d'un logiciel étant régulièrement	N+	Limited understanding

Tableau XIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		of the maintenance effort is devoted to understanding the software to be modified.	effectuée par une équipe indépendante de celle l'ayant initialement développé, une bonne partie des efforts de la maintenance est consacrée à la compréhension (analyse) du problème devant être résolu par le logiciel. Cette compréhension facilitera l'évolution du logiciel.		
	CH6 S3.4.1 P94	Programmers spend considerable time in reading and comprehending programs in order to implement changes.	Pour l'implantation de changements au logiciel, les développeurs doivent investir beaucoup de temps dans la compréhension du problème auquel le logiciel doit répondre.	N1	Program Compre-hension

Tableau XIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH6 S3.4.4 P95	Impact analysis identifies all systems and system products affected by a change request, and develops an estimate of the resources needed to accomplish the change.	En investissant du temps dans la compréhension du problème résolu par le logiciel, il devient plus simple de réaliser une analyse de l'impact d'un changement au logiciel, sur les autres systèmes en périphérie.	N1	Impact Analysis
SCM	CH7 S3.III.A P109	The software change request process (...) provides formal procedures for submitting and recording change requests, evaluating the potential cost and impact of proposed change, and accepting, modifying or rejecting the proposed change.	Le processus de demande de changement au logiciel inclut des activités d'analyse d'impact pour les changements demandés. Ces activités permettent de déterminer si un changement est accepté ou non. Il est important pour l'évolution du	N1	Requesting, Evaluating and Approving Software Changes

Tableau XIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			logiciel d'investir les efforts nécessaires à ce niveau pour s'assurer que les problèmes initialement soulevés, seront encore résolus suite à l'évolution du logiciel, à moins que le besoin ait lui aussi évolué.		
SEM	Aucune				
SEP	Aucune				
SETM	Aucune				
SQ	CH11 S2.5.4 P175	SQA and V&V discover defects. Characterizing those defects enables understanding of the product, facilitates corrections of the process or the product, and informs the project management or customer of the status	En caractérisant les anomalies, il est possible d'identifier les parties du logiciel où il y avait une déficience sur la compréhension du problème devant être résolu par le logiciel.	N1	Defect Characteri-zation

Tableau XIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		of the process or product.			

4.1.8 Manage quality throughout the life cycle as formally as possible

La gestion de la qualité est couverte dans la plupart des domaines de connaissances du Guide SWEBOK. Le guide étant découpé de façon à présenter les différents processus du cycle de vie du logiciel et des activités les supportant, il fallait donc la reconnaître dans les domaines de connaissances du Guide les présentant (SR, SD, SC, ST et SM) pour que la première partie du principe fondamental – « Manage quality throughout the life cycle » – soit reconnue.

Pour qu'elle soit formelle, la gestion de qualité doit être formulée avec précision. La citation doit exprimer clairement que des activités sont réalisées dans le processus pour gérer la qualité du produit ou du processus. Nous avons reconnu une correspondance dès qu'il était question de gestion de la qualité de façon formelle dans les différents domaines de connaissances. Pour ce qui est du domaine de connaissances du SQ, les activités pouvant être réalisées au cours du cycle de vie du logiciel pour en gérer la qualité, y sont présentées. Il répond donc à lui seul à l'ensemble du principe fondamental.

Le tableau XIV présente les correspondances reconnues par rapport à la gestion de la qualité tout au long du cycle de vie du logiciel, le plus formellement possible.

Tableau XIV

Analyse de la correspondance avec le principe fondamental (H)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SR	CH2 S2.1 P10	An essential property of all requirements is that they should be verifiable. … The requirements engineering and V&V personnel must ensure that the requirements can be verified within the available resource constraints.	Tout au long du cycle de vie, il faut s'assurer que les exigences formulées sont vérifiables. Le fait qu'elles soient vérifiables permet de pouvoir évaluer de façon formelle la qualité du produit par rapport aux exigences formulées.	DEF	What is a requirement
	CH2 Table 4 P21	The quality of the analysis directly affects the product quality. In principle, the more rigorous the analysis, the more confidence can be attached to the software quality.	Un processus rigoureux de l'analyse des exigences permet, en principe, de produire un logiciel de meilleure qualité. En contrôlant la qualité de cette analyse, l'ingénieur logiciel a beaucoup plus de	TEXTE	Aucun

Tableau XIV (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			chance de produire un logiciel de bonne qualité.		
	CH2 S3.5.3 P22	The quality of the models developed during analysis should be validated.	La qualité des différents modèles produits au cours des activités de l'analyse doit être validée. Cette validation doit être formelle pour assurer que la qualité du produit développé correspond aux exigences de qualité exprimées.	N+	Model Validation
SD	CH3 S3.IV P39	Quality analysis and evaluation tools : There exists a variety of tools and techniques that can help ensure the quality of a design.	Il existe des outils et des techniques pour aider à atteindre le niveau de qualité souhaité pour le produit.	SUJET	Software Design Quality Analysis and Evaluation
SC	CH4 S2.2 P55	Good tools also improve software quality by allowing people to avoid	Une bonne façon d'assurer la qualité de la construction d'un logiciel est	DEF	The Role of Tools in Construction

Tableau XIV (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		repetitive or precise work for which a computer is better suited.	d'utiliser des outils qui permettent d'éviter le travail répétitif ou exigeant beaucoup de précision.		
	CH4 S2.3 P55	Another important theme of software engineering is the evaluation of software products. This includes such diverse activities as peer review of code and test plan, testing, software quality assurance, and metrics [...].	L'évaluation des produits du logiciel peut se faire de plusieurs façons. Elle permet d'améliorer la qualité du résultat final. La production d'un plan de SQA et de son suivi permettent aux activités qui seront réalisées pour contrôler la qualité du produit développé, de jouer adéquatement leur rôle.	DEF	The Role in Integrated Evaluation in Construction
	CH4 S3.3.3 P63	3.3.3 Structuring for validation	En structurant la construction du logiciel pour faciliter	SUJET	Structuring for Validation

Tableau XIV (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			sa validation – lors de la révision par les pairs, par exemple – l'équipe de développement assure une meilleure gestion de la qualité du produit.		
ST	CH5 S1 P69	Testing is an important, mandatory part of software development; it is a technique for evaluating product quality and also for indirectly improving it, by identifying defects and problems.	Les activités de tests sont importantes pour évaluer la qualité du logiciel. Un processus de développement rigoureux obligera l'équipe de développement à réaliser les activités de tests du logiciel. Ces activités de tests assureront que les exigences de qualité du logiciel ont été atteintes par la solution offerte.	INTRO	Introduction
	CH5	Software testing is	Pour faciliter la	INTRO	Introduction

Tableau XIV (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	S1 P69	nowadays seen as an activity that should encompass the whole development process, and is an important part itself of the actual product construction.	gestion de la qualité des produits du logiciel, les tests font aujourd'hui partie de l'ensemble du processus de développement.		
	CH5 S2.1 P70	Software testing is usually performed at different levels along the development process. [...] The testing is conducted in a view of specific purpose (test objective),[...].	Les objectifs fixés pour les tests tout le long du processus de développement et l'étendue de ces tests (composant, fonctionnalité, l'ensemble du système, etc), sont définis selon les critères de qualité identifiés au cours du processus de SR.	DEF	Conceptual Structure of the Breakdown
	CH5 S3.D P79	Measurement is instrumental to quality analysis. [...] Measurement is instrumental also to the optimal planning	Certaines mesures produites au cours des projets permettent de fixer certains objectifs de qualité pour les	SUJET	Test related measures

Tableau XIV (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		and execution of tests, and several process metrics can be used by the test manager to monitor progress.	projets ou les activités du projet à venir. Elles sont entre autres nécessaires pour réaliser une planification optimale des tests à exécuter. Une fois les objectifs de qualité fixés, la prise des mesures est ensuite nécessaire pour la validation du niveau de qualité atteint par les différents produits.		
SM	CH6 S3.2.2.2.2 P92	It is not sufficient to simply hope that increased quality will result from maintenance of software. It must be planned and process implemented to support the	Le processus déployé doit permettre de planifier pour l'accroissement de la qualité, que ce soit pour le processus de maintenance ou pour tous les autres processus du cycle	N+	Quality

Tableau XIV (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		maintenance process.	de vie du logiciel. Il sera alors plus facile de gérer la qualité des produits du logiciel.		
SCM	CH7 S3.I.A P104	SCM is closely related to the software quality assurance (SQA) activity. The goals of SQA can be characterized [Humphrey] as monitoring the software and its development process, ensuring compliance with standards and procedures, and ensuring that product, process, and standards defects are visible to management. SCM activities help in accomplishing these SQA goals.	Les activités de SQA permettent de faire la gestion de la qualité tout au long du cycle de vie du logiciel. Ces activités sont supportées par le processus de SCM.	N1	Organiza-tional Context for SCM

Tableau XIV (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SEM	CH8 S2.4 P122	Software Quality, as quality is constantly a goal of management and is an aim of many activities that must be managed.	La qualité est un but fixé par les gestionnaires et doit être gérée dans les différentes activités du cycle de vie du logiciel pour en faciliter l'atteinte.	DEF	Relationship to other Guide to the SWEBOK knowledge areas and standards
	CH8 S3.B.2 P125	Comprehensive quality management processes are determined as part of the planning process in the form of procedures and responsibilities for quality assurance, verification and validation (...).	Les activités de gérance de la qualité du logiciel font partie intégrante de sa validation et de sa vérification continue. Les activités de ce processus sont planifiées et suivies pour assurer l'atteinte des objectifs de qualité fixés par les gestionnaires et/ou les clients.	N1	Planning
SEP	CH9 S3.5 P145	The objective of qualitative process analysis is to identify	La qualité est un but fixé par la direction et est un objectif à	DEF	Qualitative Process Analysis

Tableau XIV (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		the strengths and weaknesses of the software process. It can be performed as a diagnosis before implementing or changing a process.	atteindre dans chacune des activités du processus de développement. Pour atteindre le niveau de qualité visé, il faut s'assurer que le processus en place permet de l'atteindre. C'est en analysant les forces et les faiblesses du processus qu'il est possible d'améliorer la qualité du processus et des produits l'utilisant.		
SETM	CH10 S3.I.G P159	G. Software Quality Tools	Des outils peuvent être utilisés pour vérifier la qualité des produits du logiciel. Il faut que ces outils soient identifiés et utilisés dans le processus de développement pour	N1	Software Quality Tools

Tableau XIV (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			assurer une bonne gestion de la qualité.		
SQ	CH11 S1 P165	The discussion of the purpose and planning of SQA and V&V is a bridge between the discussion of quality and the activities and techniques discussion for SQA and V&V, but it is also an important activity in itself. In the planning process, the activities are designed to be fitted to the product and its purposes, including the quality attributes in the requirements.	Dans le processus de planification, les activités de SQA et V&V sont ajustées au produit et aux objectifs de qualité fixés au cours des activités du processus de SR. Elles sont donc planifiées et suivies tout au long du processus de développement.	DOM	Software Quality
	CH11 S2.2 P169	SQA and V&V also provide management with visibility into the quality of products at each stage in the development or	Tout au long du cycle de vie du logiciel, la prise de mesures permet de donner de la visibilité sur la	SUJET	Purpose and Planning of SQA and V&V

Tableau XIV (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		maintenance. The visibility comes from the data and measurements produced through the performance of tasks to assess and measure quality of the outputs of any software life cycle processes as they are developed.	qualité des produits. Elle informe ainsi la direction de la qualité du processus en place et de chacune des activités de ce processus, et des artéfacts produits au cours de ces activités.		

4.1.9 Minimize software component interaction

Les activités de SD sont celles où l'ingénieur logiciel est en mesure de contrôler l'interaction entre les composants du logiciel. Il produit les modèles présentant l'architecture du produit à tous les niveaux de détails. Lorsqu'il est question d'interaction des composants du logiciel, il est donc surtout question des modèles produits lors du SD et du code réalisé dans le logiciel pour appliquer ces modèles (SC). Il n'est donc pas surprenant que la correspondance avec ce principe se limite à ces deux domaines de connaissances du Guide SWEBOK. Le principe est aussi effleuré au niveau du SQ.

Le tableau XV présente les correspondances reconnues dans les domaines de connaissances du Guide SWEBOK par rapport au fait de minimiser l'interaction entre les composants du logiciel.

Tableau XV

Analyse de la correspondance avec le principe fondamental (I)

Domaine	**Repère**	**Citation**	**Interprétation**	**Niveau**	**Titre de la section**
SR	Aucune				
SD	CH3 S3.I P36	Coupling and cohesion: whereas coupling measures the strength of the relationship that exists between modules, cohesion measures how the elements making up a module are related [...].	Le "coupling" mesure le niveau du lien entre les modules. À l'aide de cette mesure, l'ingénieur logiciel peut identifier les endroits où il faut travailler à minimiser l'interaction entre les composants.	SUJET	Software Design Basic Concepts
	CH3 S3.I P36	Decomposition and modularization: the operation of decomposing a large system into a number of smaller independent ones, usually with the	Au cours des activités du SD, l'ingénieur logiciel voit à décomposer le logiciel en modules pour isoler les fonctionnalités ou les	SUJET	Software Design Basic Concepts

Tableau XV (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		goal of placing different functionalities or responsibilities in different components.	responsabilités dans différents composants à l'intérieur du produit. Cette façon de faire permet de minimiser l'interaction entre les composants.		
	CH3 S3.III P38	In its strict sense, "a software architecture is a description of the subsystems and components of a software system and the relationship between them" [...].	C'est au cours des activités du SD que l'architecture du logiciel est définie. On y décrit les sous systèmes, les composants et les relations entre-eux. C'est à ce niveau que l'ingénieur logiciel travaille à minimiser le « coupling ».	SUJET	Software Structure and Architecture
SC	CH4 S3.1.1.3 P57	Classical design admonitions such as the goal of having "cohesion" within modules and to minimize "coupling"	En minimisant le "coupling", il devient plus simple de comprendre un module.	N+	Localization of Complexity

Tableau XV (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		are also fundamentally localization of complexity techniques, since they strive to make the number and interaction of parts within a module easy for a person to understand.			
	CH4 S3.1.3 P59	One important implication of structuring for validation is that software must generally be modular in at least one of its major representation spaces, such as in the overall layout of the displayed or printed test of a program. This modularity allows both improved analysis and thorough unit-level testing of	En divisant le logiciel en modules lors de sa construction, on facilite la réalisation des activités de tests. Moins il y a d'interaction entre les composants développés, plus il est facile de les tester séparément et de trouver les problèmes qui pourraient être plus complexes à isoler	SUJET	Structuring for Validation

Tableau XV (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		such components before they are integrated into higher levels in which their errors may be more difficult to identify.	lorsque les composants seront intégrés à un niveau plus élevé. C'est l'une des raisons poussant l'ingénieur logiciel à minimiser l'interaction entre les composants du logiciel.		
ST	Aucune				
SM	Aucune				
SCM	Aucune				
SEM	Aucune				
SEP	Aucune				
SETM	Aucune				
SQ	CH11 S2.1.4 P167	Other considerations of software systems are known to affect the software engineering process while the system is being built and during its future evolution or modification, and these can be	L'indépendance des composants et la modularité du code sont deux qualités du logiciel qui peuvent affecter le processus de développement et de maintenance du logiciel. C'est en mesurant le	N1	Special Types of Systems and Quality Needs

Tableau XV (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		considered elements of product quality. These software qualities include, but are not limited to: [...]. • Modularity of code and independence of modules.	« couplage » et la « cohésion » du logiciel que ces qualités sont évaluées. Ces mesures permettent d'identifier les ajustements nécessaires au logiciel pour minimiser l'interaction entre les composants de ce dernier.		

4.1.10 Produce software in a stepwise fashion

Nous avons identifié deux vues possibles pour établir la correspondance avec ce principe fondamental. Il peut d'abord s'agir de produire un logiciel à l'aide d'un processus où les activités définies suivent une séquence précise (ex : planification, réalisation, évaluation et livraison). Il peut aussi s'agir de produire le logiciel en versions où les principales exigences sont livrées dans la (les) phase(s) initiale(s) du projet global de développement du logiciel; le but étant de répondre le plus rapidement possible aux besoins les plus criants du client.

Pour la correspondance avec la production du logiciel à l'aide d'un processus, il faut voir le processus de développement comme étant subdivisé en sous processus contenant chacun leurs propres activités. Selon ce point de vue, le Guide SWEBOK présente en effet les processus de développement du logiciel comme étant le SR, le SD, le SC, le ST et le SM. Il y a quand même plusieurs processus présentés dans les 5 autres domaines de connaissances mais ceux-ci correspondent moins bien à la notion de « stepwise fashion » présentée ici. Ces activités sont plutôt diluées à l'intérieur des processus de développement du logiciel et ne permettent pas de produire un bien livrable associé au logiciel en tant que tel.

Le tableau XVI présente les correspondances reconnues à l'intérieur du Guide SWEBOK avec la production d'un logiciel par étapes logiques.

Tableau XVI

Analyse de la correspondance
Avec le principe fondamental (J)

Domaine	**Repère**	**Citation**	**Interprétation**	**Niveau**	**Titre de la section**
SR	CH2 S3 Figure 1 P16	Figure 1 A spiral model of the requirements engineering process	Il s'agit d'une correspondance avec la production du logiciel par version au niveau du processus d'ingénierie des exigences. Cette figure peut aussi	TEXTE	Breakdown of topics for Software Requirements

Tableau XVI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			s'appliquer sur l'ensemble du processus de développement.		
	CH2 S3.1 P15	In particular, the subtopic is concerned with how the activities of elicitation, analysis, specification, validation process to terminate.	Le processus du SR se découpe en activités couvrant dans l'ordre, l'explicitation des exigences, l'analyse des exigences, la spécification des exigences et la validation des exigences. Il s'agit d'une correspondance avec les activités définies suivant une séquence précise.	N1	Process models
SD	CH3 S2 P35	[...], software design consist of two activities that fit between software requirements analysis and software coding	Il s'agit d'une correspondance avec les activités définies suivant une séquence précise.	DEF	Definition of Software Design

Tableau XVI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		and testing: i) software architectural design – […] ; ii) software detailed design – […] .			
	CH3 S3.I P36	The software design process: Software design is generally considered a two steps process: architectural design describes how the system is decomposed and organized into components (the software architecture), whereas detailed design describes the specific behavior of these components […].	Il s'agit d'une correspondance avec les activités définies suivant une séquence précise.	SUJET	Software Design Basic Concepts
SC	CH4 S2 P53	Software construction is linked to all other Kas, perhaps most strongly to Design, and Testing. This is because the	À l'intérieur du processus de développement, le SC utilise les produits du SD et fournit à son tour ses	DEF	Definition of the Software Construction

Tableau XVI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		construction process consumes the output of the Design process (KA3) and itself provides one of the inputs to the Testing process (KA5).	produits au ST. Il s'agit d'une correspondance avec les activités définies suivant une séquence précise.		
	CH4 S2 P54	Software construction is a fundamental act of software engineering: the construction of working, meaningful software through a combination of coding, validation, and testing (unit testing) by a programmer.	Le processus de SC est divisé en étapes : code, validation et tests unitaires. Il s'agit d'une correspondance avec les activités définies suivant une séquence précise.	DEF	Definition of the Software Construction
ST	CH5 S3.E2 P81	E2. Test Activities • Planning [...] • Test case generation [...] • Test environment development [...] • Execution [...] • Tests results	Le processus de tests est divisé en plusieurs activités. Il s'agit d'une correspondance avec les activités définies suivant une séquence précise.	N1	Test Activities

Tableau XVI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		evaluation [...] • Problem reporting/Test log [...] • Defect tracking [...].			
SM	CH6 S2 P88	Software maintenance sustains the software product throughout its life cycle. Modification requests are logged and tracked, the impact of proposed changes is determined, code is modified, testing is conducted, and a new version of the software product is released. Training is provided to users.	Le processus du SM est aussi composé de plusieurs activités devant suivre une certaine séquence. Il s'agit d'une correspondance avec les activités définies suivant une séquence précise.	DEF	Definition of the Software Maintenance
	CH6 S3.1.1 P88	ISO/IEC 12207 identifies the primary activities of software maintenance as:	Les activités du processus de SM sont identifiées dans la norme ISO/IEC	N1	Definitions and Terminology

Tableau XVI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		process implementation; problem and modification analysis; modification implementation; maintenance review/acceptance; migration; and retirement.	12207. Ces activités sont un processus par étapes pour en arriver finalement au retrait du produit. Il s'agit d'une correspondance avec les activités définies suivant une séquence précise.		
	CH6 S3.2.1 Figure 2 P90 CH6 S3.2.1 Figure 3 P91	Figure 2 The IEEE Maintenance Process Activities Figure 3 ISO/IEC 14764 Maintenance Process Activities	Les figures 2 et 3 présentent deux processus provenant de normes connues (IEEE et ISO/IEC) pour la SM. Ce sont deux processus par étapes. Il s'agit d'une correspondance avec les activités définies suivant une séquence précise.	N1	Maintenance Process Models
	CH6 S3.2.2 P91	Maintenance activities are similar to those of software development.	Les activités pour la maintenance des logiciels sont les	N1	Maintenance Activities

Tableau XVI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		Maintainers perform analysis, design, coding, testing, and documenting. Maintainers must track requirements just as they do in development.	mêmes qu'en développement. Il s'agit d'une correspondance avec les activités définies suivant une séquence précise.		
SCM	CH7 S3.II.A.6 P109 CH7 S3.II.A.6 Figure 4 P109	Software configuration items are placed under SCM control at different times; i.e. they are incorporated into a particular baseline at a particular point in software life cycle.	Les SCI sont intégrés à différentes périodes dans le temps, en relation avec le cycle de vie du logiciel. Cela implique que le logiciel soit développé par étape, et que les SCI soient planifiés et intégrés au moment prévu dans le cycle de vie. Cette citation couvre les deux vues du principe fondamental. L'essentiel est de définir à quel	N+	Acquiring Software Configuration Items

Tableau XVI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			moment un SCI doit être livré, que ce soit par rapport à une version du produit ou à la suite d'une activité du processus.		
	CH7 S3.III.A P109	The first step in managing changes to controlled items is determining what changes to make. The software change request process (see Figure 5) provides formal procedures for submitting and recording change requests, evaluating the potential cost and impact of proposed change, and accepting, modifying or rejecting the proposed change.	Les changements au logiciel sont aussi réalisés par étape. Il s'agit d'une correspondance avec les activités définies suivant une séquence précise.	N1	Requesting, Evaluating and Approving Software Changes
SEM	CH8 S3.B P124	B. Process/Project Management	La gestion de projet ou de processus est réalisée par étapes.	SUJET	Process/ project management

Tableau XVI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		1. Initiation and scope definition – […] 2. Planning – […] 3. Enactment – […] 4. Review and evaluation – […] 5. Closure – […]	Il y a l'initiation du projet, la planification, la réalisation, l'évaluation et la fermeture du projet. Il s'agit d'une correspondance avec les activités définies suivant une séquence précise.		
SEP	CH9 S3.4.2 P144	These framework models serve as a high level definition of the phases that occur during development. They are not detailed definitions, but only the high level activities and their interrelationships.	Les modèles structurés servent à définir le processus de développement. Ils permettent de produire le logiciel par étapes bien définies ou un logiciel en versions, selon le gabarit (« framework ») du processus utilisé.	N1	Life Cycle Framework Models
	CH9 S3.4.3	The IEEE standard on development life cycle	Le standard "IEEE Std 1074-1991"	N1	Software Life Cycle Process

Tableau XVI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	P144	processes also provides a list of processes and activities for development and maintenance [...], and provides examples of mapping them to life cycle framework models.	fournit une liste de processus et d'activités pour le développement et la maintenance du logiciel par étapes. Il s'agit d'une correspondance avec les activités définies suivant une séquence précise.		Models
SETM	CH10 S3.I P156	The first five subsections correspond to the five Knowledge Areas (Requirements, Design, Construction, Testing, and Maintenance) that correspond to a phase lifecycle, [...].	La présentation des différents types d'outils pouvant être utilisés au cours du cycle de vie du logiciel, suggère que le développement de logiciels se fait par étapes. Ces étapes contiennent des activités pour le SR, le SD, la SC, le ST et la SM. Il s'agit d'une correspondance avec	SUJET	Software Engineering Management Tools

Tableau XVI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			les activités définies suivant une séquence précise.		
SQ	CH11 S2.2.1 P170	Planning for software quality involves (1) defining, the required product in terms of its quality attributes and (2) planning the processes to achieve the required product.	Pour s'assurer que la qualité du logiciel atteigne le niveau défini par les exigences des différents intervenants, il faut que la qualité à atteindre soit identifiée puis, que cette qualité soit planifiée par le biais de la mise en place des activités de SQA et V&V. Il s'agit d'une correspondance avec les activités définies suivant une séquence précise.	N1	Common Planning Activities

4.1.11 Set quality objectives for each deliverable product

Les objectifs de qualité du logiciel sont obtenus auprès des principaux intéressés (« stakeholders ») et par le processus de développement logiciel en place. Une fois identifiés, ils doivent s'intégrer aux activités de planification du projet car ils sont affectés par le contexte dans lequel se retrouve le projet (échéancier, disponibilité des ressources, etc.). Les exigences de qualité sont généralement identifiées à l'intérieur du processus de SR. Elles établissent des cibles de qualité pour des livrables produits au cours des activités couvrant l'ensemble du cycle de vie du logiciel.

Nous avons identifié une correspondance avec ce principe fondamental dès qu'une citation permettait de reconnaître l'identification d'une cible (objectif) de qualité pour un livrable. Par livrable, nous entendons tout produit ou sous-produit du logiciel produit au cours du cycle de vie du logiciel.

Le tableau XVII présente les correspondances reconnues pour l'identification des objectifs de qualité pour chacun des livrables à produire, avec le contenu du Guide SWEBOK.

Tableau XVII

Analyse de la correspondance
Avec le principe fondamental (K)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SR	CH2 S2.1 P10	Non-functional requirements are sometimes known as	Les exigences non fonctionnelles sont une partie des	DEF	What is a requirement

Tableau XVII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		constraints or quality requirements.	objectifs de qualité identifiés.		
	CH2 S2.4 P13	[...], requirements engineers are necessarily constrained by project management plans and must therefore take steps to ensure that the requirements' quality is as high as possible given the available resources.	Les contraintes du projet obligent l'ingénieur logiciel à identifier les objectifs de qualité maximum pouvant être atteints selon ces contraintes.	DEF	Requirements engineering in practice
	CH2 S3.4.4 P21	A number of quality indicators have been developed that can be used to relate the quality of an SRS to other project variables such as cost, acceptance, performance, schedule, reproductivity, etc.	L'utilisation de certains indicateurs permet de valider l'atteinte des objectifs fixés pour la qualité du SRS.	N1	Document quality
SD	CH3 S3.II	A number of key issues must be dealt	Des objectifs de qualité doivent être	SUJET	Key Issues in Software

Tableau XVII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	P37	with when designing software systems. Some of these are really quality concerns that must be addressed by all systems, for example, performance.	fixés, comme la performance du produit et ce qui en affectera le design.		Design
	CH3 S3.IV P38	Quality attributes: Various attributes are generally considered important for obtaining a design of good quality, e.g., various "ilities" [...], various "nesses" [...], including "fitness of purpose" [...].		SUJET	Software Design Quality Analysis and Evaluation
SC	CH4 S2.3 P55	Tools for extracting measurements of code quality and structure can also be used during construction, [...]. When collecting measurements, it is important that the measurements	Des outils permettant de mesurer la qualité et la structure du code peuvent être également utilisées tout au long de la programmation du logiciel. Ces mesures doivent	DEF	The Role of Integrated Evaluation in Construction

Tableau XVII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		collected be relevant to the goals of the development process.	évidemment tenir compte de l'atteinte des objectifs de qualité du logiciel en construction.		
	CH4 S2.5 P56	That is, when selection from a simple set of options is all that is really required to make software work for a business or system, then the goal of software engineers should continually be to make their systems come as close to that level of simplicity as possible. This not only makes software more accessible, but also makes it safer and more reliable by removing opportunities for error.	L'un des objectifs de qualité que devrait toujours viser un ingénieur logiciel, est de produire le système le plus simple possible. Ainsi, le logiciel devient plus facile à maintenir et plus sécuritaire en éliminant les possibilités d'erreur. Cet objectif de qualité devrait généralement se retrouver comme exigence à l'intérieur de l'équipe de projet.	DEF	Manual and Automated Construction/ The Spectrum of Construction Techniques
ST	CH5 S2.1	The test target and test objective together	Les objectifs de tests sont déterminés	DEF	Conceptual Structure of

Tableau XVII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	P70	determine how the test set is identified; both with regard to its consistency – [...] – and its composition – [...].	selon les objectifs de qualité visés pour le projet. Ces objectifs permettent d'identifier la liste des tests à réaliser, selon sa consistance par rapport aux objectifs de qualité du projet, et sa composition reliée à ces mêmes objectifs.		the Breakdown
	CH5 S3.D P79	Measurement is instrumental to quality analysis. [...] Measurement is instrumental also to the optimal planning and execution of tests, and several process metrics can be used by the test manager to monitor progress.	Certaines mesures accumulées au cours des projets permettent de fixer certains objectifs de qualité pour les projets à venir. Ils sont entre autres nécessaires pour réaliser une planification optimale des tests à exécuter. Une fois les objectifs de	SUJET	Test related measures

Tableau XVII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			qualité fixés, la prise des mesures est ensuite nécessaire pour la validation du niveau de qualité atteint par les différents produits.		
SM	CH6 S3.2.2.2.2 P92	It must be planned and processes implemented to support the maintenance process. The activities and techniques for Software Quality Assurance (SQA) and V&V must be selected in concert with all other processes to achieve the level of quality desired.	Les activités de SQA et V&V doivent être sélectionnées selon les objectifs de qualité définis lors de l'explicitation des exigences. Ce qui se reflètera dans le plan du projet. Ces activités auront comme intrants les objectifs de qualité fixés pour chacun des livrables des activités du processus de développement ou de maintenance.	N+	Quality

Tableau XVII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH6 S3.2.2.2.3 P92	Maintenance planning should begin with the decision to develop a new system and should consider quality objectives.		N+	Maintenance Planning Activity
SCM	Aucune				
SEM	CH8 S3.B.2 P125	Comprehensive quality management processes are determined as part of the planning process in the form of procedures and responsibilities for quality assurance, verification and validation [...].	Les objectifs de qualité pour le projet sont déterminés au cours de la planification du projet et se reflètent par les activités définies pour l'AQL et le V&V.	N1	Planning
SEP	CH9 S3.3.1 P142	A number of guides for measurement are available [...]. All of these describe a goal-oriented process for defining measures. This means that one should start from	Cette correspondance est implicite. Il y est question de définir les besoins en mesures selon le processus ou selon le projet et non l'inverse. Pour	N1	Methodology in Process Measurement

Tableau XVII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		specific information needs and then identify the measures that will satisfy these needs, rather than start from specific measures and try to use them.	évaluer la qualité du produit ou du processus, il faut donc définir les objectifs de qualité pour déterminer les mesures qui seront utilisées pour vérifier l'atteinte ou non de ces objectifs.		
SETM	Aucune				
SQ	CH11 S1 P165	In the planning process, the activities are designed to be fitted to the product and its purpose, including the quality attributes in the requirements.	Au cours des activités de planification du projet de développement d'un produit, il y a des activités qui voient à extraire les attributs de qualité (exigences de qualité) désirés pour le produit. Ces attributs permettent de définir les objectifs de qualité visés pour le produit.	INTRO	Introduction

Tableau XVII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH11 S2.1 P166	The software dictates the performance of the system, and is therefore important to the system quality. Much thought must therefore go into the value to place on each quality attribute desired and on the overall quality of the system.	Une bonne partie de la qualité d'un système provient de la qualité du logiciel en faisant partie. En fixant les objectifs de qualité du système, il faut considérer les objectifs de qualité qui seront attribués au logiciel en faisant partie, et les sous produits du logiciel.	SUJET	Software quality concepts
	CH11 S2.1 P166	The important point is that requirements define the required quality of the respective software, the measurement methods and acceptance criteria for the attributes, finished product, but maybe relevant to the achievement of overall quality.	Les exigences définissent les objectifs de qualité à atteindre pour le produit et les sous produits à développer.	N1	Software quality concepts

Tableau XVII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH11 S2.2 P169	The SQA process provides assurance that the software products and processes in the project life cycle conform to their specified requirements by planning a set of activities to help build quality into the software.	Les objectifs de qualité sont fixés au cours des activités de l'explicitation des exigences.	SUJET	Purpose and Planning of SQA and V&V
	CH11 S2.2.1 P170	Planning for software quality involves (1) defining, the required product in terms of its quality attributes and (2) planning the processes to achieve the required product.	La planification de la qualité du logiciel voit à définir les objectifs de qualité à atteindre, selon les attributs de qualité obtenus lors de l'explicitation des exigences.	N1	Common Planning Activities
	CH11 S2.2.2 P170	The SQA plan defines the processes and procedures that will be used to ensure that the software developed for a specific product	Pour produire le plan de qualité, il faut d'abord que l'ingénieur logiciel s'assure que les objectifs de qualité	N1	The SQA Plan

Tableau XVII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		meets the requirements and is of the highest quality possible within project constraints. To do so, it must first ensure that the quality target is clearly defined and understood.	sont bien définis et que ces objectifs sont bien compris.		

4.1.12 Since change is inherent to software, plan for it and manage it

La planification et la gestion des changements sont particulièrement bien couvertes dans l'ensemble du Guide SWEBOK. Pour reconnaître la correspondance avec ce principe fondamental, nous avons divisé le principe en trois (3) parties. Il a fallu d'abord considérer les citations qui identifiaient l'inévitabilité du changement au cours du cycle de vie du logiciel (Since change is inherent to software). Ensuite, il y avait les citations indiquant que les changements demandés sont planifiés (plan for it). Enfin, il a fallu identifier les citations où il était question de gérer les activités permettant de réaliser le(s) changement(s) planifié(s) (manage it).

Plus les activités permettant de gérer les changements sont planifiées tôt, moins les changements auront d'impact sur le projet. C'est donc au cours des activités de SR que la majorité des activités de l'identification des changements auront lieu. Nous avons

donc reconnu plusieurs fois le principe fondamental dans cette partie du Guide. C'est aussi au cours de la maintenance du logiciel que des activités d'analyse des exigences ont lieu. Des activités similaires à celles du SR doivent donc se retrouver au niveau du SM pour planifier et gérer les changements au logiciel.

Ce principe fondamental cible le produit. Le tableau XVIII présente les correspondances reconnues avec la planification et la gestion des changements dans le Guide SWEBOK.

Tableau XVIII

Analyse de la correspondance avec le principe fondamental (L)

Domaine	**Repère**	**Citation**	**Interprétation**	**Niveau**	**Titre de la section**
SR	CH2 S2.3 P12	Tracing is crucial to requirements management because it allows, for example, the impact of any subsequent changes to the requirements to be assessed.	La "traçabilité" des exigences facilite la gestion des changements aux exigences et l'analyse de l'impact de ces changements.	DEF	Overview of requirements analysis
	CH2 S2.4 P13	Perhaps the most crucial point of understanding about requirements engineering is that a significant proportion of the requirements .	Cette portion du guide démontre une forte correspondance avec le libellé du principe fondamental. Il y a reconnaissance de	DEF	Requirements engineering in practice

Tableau XVIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		will change. [...] Whatever the cause, it is important to recognize the inevitability of change and adopt measures to mitigate the effects of change.	l'inévitabilité du changement et des mesures devant être adoptées pour en limiter les effets.		
	CH2 S3.3 P19	Volatility/stability. Some requirements will change during the life-cycle of the software and even during the development process itself. It is useful if some estimate of the likelihood of a requirement changing can be made.	Au cours de l'analyse des exigences, il faut identifier les exigences qui ont le plus de chance de changer. C'est la première étape de la planification de l'éventualité d'un changement.	N+	Requirements analysis
	CH2 S3.3 P19	Flagging requirements that may be volatile can help the software engineer establishing a design that is more tolerant to change.	Le design doit être bâti pour prévoir les changements possibles à l'application. Cette préparation se fait à	DEF	Requirements analysis

Tableau XVIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			l'aide des résultats de l'analyse de la volatilité des exigences.		
	CH2 S3.4.2 P21	The benefits of the SRS includes: [...] It forces a rigorous assessment of requirements before design can begin and reduces later redesign.	L'un des bénéfices du SRS est de documenter la compréhension que l'ingénieur logiciel a du problème à résoudre, pour la confirmer avec le client et ainsi éviter des changements importants dans les activités subséquentes du projet. Le travail de production du document l'oblige à effectuer une analyse rigoureuse pour bien démontrer cette compréhension du problème.	N1	The software requirements specification (SRS)

Tableau XVIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH2 S3.6 P23	Requirements management is an activity that spans the whole software life-cycle. It is fundamentally about change management and the maintenance of the requirements… Tracing is fundamental to performing impact analysis when requirements change.	Ce sujet du domaine traite essentiellement de la gestion des changements et de l'utilisation de la traçabilité pour faciliter l'analyse d'impact des changements prévus.	SUJET	Requirements management
SD	CH3 S3 P35	Various alternative solutions and tradeoffs can also be examined and evaluated.	C'est une correspondance implicite avec la planification des changements. L'analyse de solutions alternatives voit à définir la conception qui facilitera l'évolution du logiciel. Des compromis pourront	DEF	Definition of Software Design

Tableau XVIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			être faits où le risque d'un changement est le moins probable.		
	CH3 S3.IV P39	[...], those not discernable at run time (e.g., modifiability, portability, reusability, integrability and testability) [...].	Il est possible de mesurer la qualité d'un logiciel selon sa capacité à être modifiable (modifiability), c'est à dire la facilité avec laquelle il sera possible de faire des changements à ce dernier dans le futur. C'est une correspondance avec la gestion des changements.	N1	Software Design Quality Analysis and Evaluation
SC	CH4 S2.5 P56	For example, even a concept as simple as defining a constant at the beginning of a software module reflects the automation theme, since such constant "automate"	Dans la partie des définitions concernant l'automatisation, il est question de l'utilisation d'une méthode de programmation	DEF	Manual and Automated Construction/ The Spectrum of Construction Techniques

Tableau XVIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		the appropriate insertion of new values for the constant in the event that changes to the program are necessary.	permettant de faciliter les changements au programme. Dans ce cas-ci, c'est l'utilisation de constantes.		
	CH4 S3.1.2 P58	*There is no such thing as an unchanging software construction.* Any useful software construction will change in various ways over time, and the *anticipation* of change drives nearly every aspect of software construction.	L'un des sujets du SC est l'anticipation de la diversité, c'est-à-dire, prévoir les façons dont pourra être utilisé un composant pour pouvoir développer ce composant en fonction des changements possibles à ce dernier au cours du cycle de vie du logiciel.	SUJET	Anticipation of Diversity
	CH4 S3.1.2 P58	There are three main techniques for anticipating change during software construction :	Cette partie du SC donne des techniques pour faciliter la gestion des changements dans le	N1	Anticipation of Diversity

Tableau XVIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		3.1.2.1 Generalization [...] 3.1.2.2 Experimentation [...] 3.1.2.3 Localization [...]	programme. La localisation permet d'isoler les parties du logiciel les plus susceptibles d'être modifiées pour limiter les impacts sur l'ensemble du produit. C'est une correspondance avec la planification des changements.		
ST	CH5 S3.B2 P76	[...], regression testing is the "selective retesting of a system or component to verify that modifications have not caused unintended effects [...]". In practice, the idea is to show that previously passed tests, still do.	Les tests de régression permettent de s'assurer que les changements effectués au programme n'ont pas d'impact sur les résultats obtenus par le programme avant ces changements. C'est un moyen utilisé pour gérer les changements au programme.	N+	Objectives of testing

Tableau XVIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH5 S3.E2 P81	Test Reports are also an input to the Change Management system (which is part of the Configuration Management system).	Les rapports produits à la suite de tests permettent de compléter l'analyse d'impact du ou des changements effectués.	N1	Test Activities
SM	CH6 S3.1.1 P88	[...] describes maintenance as the process of a software product undergoing "modification to code and associated documentation due to a problem or the need for improvement. The objective is to modify existing software product while preserving its integrity."	Dans le processus suivi par la SM, il faut s'assurer de conserver l'intégrité de l'application lors d'un changement. Cela exige une bonne planification et une bonne gestion des changements.	DEF	Definitions and Terminology
	CH6 S3.1.3 P88	[...] maintenance has a broader scope than development, with more changes to track and control.	Dans la section expliquant la nature de la maintenance, il est précisé que comme pour le	N1	The Nature of Maintenance

Tableau XVIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			développement, il faut identifier les changements possibles et les gérer même au cours de la maintenance du produit.		
	CH6 S3.2.2.1 P91 CH6 S3.3.3.1 P93 CH6 S3.4.4 P95	Impact analysis identifies all systems and system products affected by a change request and develops an estimate of the resources needed to accomplish the change [Art88]. Additionally, the risk of making the change is determined. [...] Maintainers must also be concerned about the "ripple effect" of any proposed change.	Une importante activité de la maintenance est l'analyse d'impact, où les membres de l'équipe de maintenance vérifient les effets possibles sur l'application de chacun des changements proposés et les risques d'effectuer ces changements. L'analyse d'impact est l'une des méthodes utilisées pour gérer les	N+	Unique Activities

Tableau XVIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			changements.		
	CH6 S3.2.2.2.1 P92	The software product and any changes made to it must be controlled.	La gestion de la configuration permet de contrôler les changements au produit logiciel.	N+	Configuration management
SCM	CH7 S2 P103	Configuration Management (CM), then, is the discipline of identifying the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration and maintaining the integrity and traceability of the configuration throughout the system life cycle [...].	La gestion de la configuration constitue le moyen de contrôler les changements au système à travers le temps.	DEF	Definition of the SCM
	CH7 S3.I P104	From a management perspective, SCM controls the evolution	Une gestion de configuration logicielle réussie,	SUJET	Management of the SCM Process

Tableau XVIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		and integrity of a product by identifying its element, managing and controlling change, and verifying, recording and reporting on configuration information. From the developer's perspective, SCM facilitates the development and change implementation activities. A successful SCM implementation requires careful planning and management.	constitue le meilleur moyen de planifier et de gérer les changements au logiciel, que ce soit pour les gestionnaires ou pour les développeurs. Elle permet de contrôler l'évolution du système et de conserver l'intégrité du système à la suite de ces changements.		
	CH7 S3.I.C.5 P107	When a software item will interface with another software or hardware item, a change to either item	La gestion de configuration logicielle voit même à identifier l'impact d'un changement à	N1	Interface Control

Tableau XVIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		can affect the other. The planning for the SCM process considers how the interfacing items will be identified and how changes to the items will be managed and communicated.	l'extérieur du logiciel, pour les items communiquant avec l'item modifié.		
	CH7 S3.II.A P108	A first step in controlling change is to identify the software item to be controlled.	Il est ici question de l'identification des items qui pourraient changer pour faciliter la gestion des changements. C'est l'étape initiale du processus de contrôle du changement.	N1	Identifying Items to be Controlled
	CH7 S3.III P109	"Software configuration control is concerned with managing changes during the software life cycle. It covers the process for determining what	Cette partie du domaine identifie clairement le principe fondamental traité. On y décrit le processus à mettre en place pour la planification et la	SUJET	Software Configuration Control

Tableau XVIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		changes to make, the authority for approving certain changes, support for the implementation of those changes, and the concept of formal deviations and waivers from project requirements."	gestion des changements au logiciel.		
SEM	CH8 S3.B.1.3 P124	Process for the review and revision of requirements – given the inevitability of change, it is vital that agreement among stakeholders is reached at this early point as to the means by which scope and requirements are to be reviewed and revised (e.g. via agreed change management procedures).	Dans la gestion de projet, le processus mis en place pour gérer les changements doit convenir à l'ensemble des intervenants. Le processus de révision des changements est lui-même planifié et géré.	N+	Process for the review and revision of requirements

Tableau XVIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH8 S3.B.2.8 P126	Plan management – in an environment where change is an expectation rather than a shock, it is vital that plans are themselves managed.	Tout comme pour la gestion du processus de révision des changements, les changements aux différents plans doivent aussi être gérés.	N+	Plan management
SEP	Aucune				
SETM	CH10 S3.A P156	Requirements traceability tools are becoming increasingly important as the complexity of software systems grow, and since traceability tools are relevant also in other lifecycle phases, they have been separated from the other tools for requirements.	Certains outils facilitent la gestion des changements.	N+	Software Requirements Tools
	CH10 S2.H P159	Tools for configuration management have been categorized as	Les outils de gestion de configuration facilitent la planification et la	N1	Software Configuration Management Tools

Tableau XVIII (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		related to tracking issues associated with a particular software product, management of multiple versions of a product or to managing the task of software release and build.	gestion des changements en gérant les tâches associées à la production des versions et la production des différentes versions du logiciel.		
SQ	CH11 Table 1. P168	Maintainability: Characteristics related effort needed to make modifications, including corrections, improvements or adaptation of software to changes in environment, requirements and functional specifications.	Ce tableau présente les caractéristiques et les attributs de la qualité du logiciel. La « maintenabilité » d'une application est la caractéristique qui guide le gestionnaire dans son effort pour réaliser les changements. Il permet de prévoir d'où viendront les changements et les raisons les amenant.	TEXTE	Aucun

4.1.13 Since tradeoffs are inherent to software engineering, make them explicit and document them

Le fait qu'il faille négocier avec différents intervenants au cours d'un projet, surtout par rapport aux différentes exigences soulevées par ceux-ci, est bien reconnu à l'intérieur du Guide SWEBOK. Par contre, il n'est pas toujours explicite que l'identification des compromis doit être documentée.

Pour notre analyse, nous avons considéré les trois (3) parties composant le principe fondamental : 1) la reconnaissance de l'inévitabilité du compromis, 2) l'identification de ces compromis pour qu'ils soient visibles dans l'ensemble des processus, et 3) la documentation des ententes de compromis. Nous avons reconnu une correspondance avec ce principe fondamental dès qu'une de ces trois parties était identifiée.

Ce principe fondamental cible à la fois le produit et le processus. Le tableau XIX présente les correspondances reconnues de la reconnaissance, de l'identification et de la documentation des compromis.

Tableau XIX

Analyse de la correspondance avec le principe fondamental (M)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SR	CH2 S1 P9	One of the fundamental tenets of good software engineering is that there is good	L'un des types de compromis qui doit être négocié au cours de l'analyse des exigences, est le	INTRO	Introduction

Tableau XIX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		communication between system users and system developers. It is the requirements engineer who is the conduit for this communication. They must mediate between the domain of the system users (and other stakeholders) and the technical world of the software engineer.	domaine de l'utilisateur du système par rapport au monde technique de l'ingénieur logiciel. Il s'agit d'une correspondance avec la reconnaissance que des compromis devront être négociés.		
	CH2 S2.3 P12	Once identified, the system requirements should be validated by the stakeholders and tradeoffs negotiated before further resources are committed to the project.	Une fois les exigences du système identifiées, il faut négocier le contenu du produit à développer. Il s'agit d'une correspondance avec l'identification des compromis.	DEF	Overview of requirements analysis
	CH2 S2.3	[...], it may take considerable effort to	Lors de la négociation des	DEF	Overview of requirements

Tableau XIX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	P12	reach consensus on requirements priorities because one stakeholder's essential requirement may have only cosmetic value to another. In practice, the existence of sufficient resources will allow some non-essential requirements to be satisfied, while insufficient resources may force even strongly advocated requirements to be excluded.	exigences devant être comblées par le logiciel, la disponibilité des ressources constitue un élément majeur sur ce qui fera partie du produit développé. La négociation du contenu doit tenir compte des ressources disponibles. Il s'agit d'une correspondance avec l'identification des compromis.		Analysis
	CH2 S3.1.2 P17	It will not be possible to perfectly satisfy the requirements of every stakeholder and the requirements engineer's job is to negotiate a compromise that is	En plus de devoir négocier l'ensemble des exigences auprès des différents intervenants, d'autres éléments peuvent influencer la sélection des	N1	Process actors

Tableau XIX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		both acceptable to the principal stakeholders and within budgetary, technical, regulatory and other constraints.	exigences, comme le budget disponible, les limitations techniques, les lois, etc. Il s'agit d'une correspondance avec l'identification des compromis.		
	CH2 S3.2.1 P18	Domain knowledge. The requirements engineer needs to acquire or to have available knowledge about the application domains. This enables them to infer tacit knowledge that the stakeholders do not articulate, assess the tradeoffs that will be necessary between conflicting requirements and sometimes to act as a "user" champion.	L'ingénieur logiciel responsable de l'explicitation des exigences, doit avoir une bonne connaissance du domaine des affaires pour pouvoir agir comme « facilitateur » auprès des différents intervenants lors de la négociation du contenu du logiciel. Il s'agit d'une correspondance avec la reconnaissance	N+	Requirements sources

Tableau XIX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			que des compromis devront être négociés.		
	CH2 S3.3 P19	This subtopic is concerned with the process of analyzing requirements to: ▪ Detect and resolve conflicts between requirements ▪ [...].	À l'intérieur des activités d'analyse des exigences, il faut détecter les exigences entrant en conflit les unes contre les autres, et négocier des compromis. Il s'agit d'une correspondance avec la reconnaissance que des compromis devront être négociés.	SUJET	Requirements analysis
	CH2 S3.3.4 P20	Another name commonly used for this subtopic is 'conflict resolution'. It is concerned with resolving problems with requirements where conflicts occur;	La négociation des exigences contient les activités de résolution de conflit entre les exigences. Il y est mentionné que la résolution de conflit doit pouvoir	N1	Requirements negotiation

Tableau XIX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		between two 'stakeholders' requiring mutually incompatible features, or between requirements and resources or between capabilities and constraints, [...]. It is often important for contractual reasons that such decisions are traceable back to the customer.	être retracée du produit aux exigences, et des exigences au client, pour des raisons contractuelles. Le but est de conserver la trace du résultat des négociations du contenu du produit, tel qu'entendu avec le client. Il s'agit d'une correspondance avec l'identification des compromis et la documentation des ententes de compromis.		
SD	CH3 S2 P35	These models can be analyzed and evaluated to determine if they will allow the various requirements to be fulfilled. Various alternative	Le SD permet à l'ingénieur logiciel d'analyser les exigences à un niveau d'abstraction beaucoup plus bas. Il permet de vérifier	DEF	Definition of Software Design

Tableau XIX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		solutions and tradeoffs can also be examined and evaluated.	les différentes alternatives qui s'offrent à lui et d'examiner les conflits possibles pour ainsi dégager une solution qui fera consensus. Il s'agit d'une correspondance avec l'identification des compromis.		
SC	Aucune				
ST	CH5 S2 P70	Clearly, "enough" testing should be performed to provide reasonable assurance. Indeed, testing always implies a tradeoff between limited resources and schedules, and inherently unlimited test requirements.	Pour déterminer le nombre de cas de tests qu'il faudra développer et tester au cours des activités de tests, il faut négocier entre le nombre limité de ressources à la disposition de l'ingénieur logiciel et le besoin d'offrir une assurance	DEF	Definition of the Software Testing

Tableau XIX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			raisonnable que le logiciel réponde aux différentes exigences exprimées. Les objectifs de qualité doivent permettre de définir le « raisonnable ». Ces objectifs découlent de l'analyse des exigences et font partie de ce qui doit être négocié avec le client. Il s'agit d'une correspondance avec l'identification des compromis.		
SM	Aucune				
SCM	Aucune				
SEM	CH8 S3.B.1.1 P124	Determination and negotiation of requirements methods of requirements engineering, elicitation [...], analysis [...], specification, and	Les processus suivis au cours d'un projet de développement doivent être sélectionnés selon le contexte du projet. Il faut déterminer et	N+	Determi-nation and negotiation of requirements

Tableau XIX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		validation [...] must be selected and applied in cognizance of various stakeholder perspectives.	négocier le choix des méthodes à utiliser pour chacune des activités du processus. Les choix doivent se faire selon les ressources et les perspectives offertes par les différents intervenants. Il doit y avoir une entente avec le client sur le choix des méthodes et techniques pour s'assurer qu'elles répondent à ses exigences d'affaires et de qualité. Il s'agit d'une correspondance avec l'identification et la négociation des compromis.		
SEP	Aucune				
SETM	Aucune				

Tableau XIX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
SQ	CH11 S2.1 P166	Quality attributes may be present or absent, or may be present in greater or lesser degree, with tradeoffs among them, with practicality and cost as major considerations.	Comme avec d'autres exigences, il est possible que des compromis soient faits entre les différents objectifs de qualité. Il s'agit d'une correspondance avec la reconnaissance que des compromis devront être négociés même au niveau des exigences de la qualité.	SUJET	Software quality concepts

4.1.14 To improve design, study previous solution to similar problems

Dans le libellé de ce principe fondamental, il est question de l'étude de la conception de logiciels existants, devant résoudre un problème similaire à celui auquel doit faire face l'ingénieur logiciel. Il y avait donc une attente à retrouver une correspondance à l'intérieur du domaine de connaissances du SD dans le Guide SWEBOK. Lorsqu'il est question d'étudier des solutions à des problèmes similaires, nous pouvons reconnaître un fort lien avec le « Build with reuse » du principe fondamental (B). Nous avons donc reconnu une correspondance dès qu'il était question d'étudier un composant existant en

fonction d'une réutilisation dans la production du logiciel. Il y a aussi le processus de SM où l'évolution du logiciel demande une étude de solutions existantes pour régler les problèmes soulevés.

Le tableau XX présente les correspondances reconnues dans le Guide SWEBOK par rapport à l'amélioration du design par l'étude de solutions existantes à des problèmes similaires.

Tableau XX

Analyse de la correspondance avec le principe fondamental (N)

Domaine	**Repère**	**Citation**	**Interprétation**	**Niveau**	**Titre de la section**
SR	CH2 S2.3 P12	Architectural design is a skill that is driven by many factors such as the recognition of reusable architectural "patterns" or the existence of off-the shelf components.	Lorsqu'il est question d'étudier des solutions à des problèmes similaires, il est souvent question de composants d'une tierce partie (off-the shelf components) .	DEF	Overview of requirements analysis
SD	CH3 S3.III P38	But they can also be useful for designing generic systems, leading to the design of families of systems [...]. Interestingly,	Cette section du guide énumère certaines notions ciblant la production du logiciel en réutilisant des	SUJET	Software Structure and Architecture

Tableau XX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		most of these notions can be seen as attempts to describe, and thus reuse, generic design knowledge.	composants déjà existants (« Families of programs » et « frameworks » par exemple).		
SC	CH4 S2.6 P56	*Toolkit languages* are use to build applications out of toolkits (integrated sets of application-specific reusable parts), and are more complex than configuration languages.	Les langages « outils » permettent de réutiliser des ensembles de parties spécifiques déjà existantes pour solutionner un problème. Leur utilisation doit faire l'objet d'une étude du besoin pour s'assurer que les outils pourront effectivement être utilisés.	DEF	Construction Languages
ST	Aucune				
SM	CH6 S3.4.2 P94	Re-engineering is defined as the examination and alteration of the subject system to	La ré-ingénierie voit à examiner les composants d'un système pour le reconstituer dans une	N1	Re-engineering

Tableau XX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		reconstitute it in a new form, and the subsequent implementation of the new form.	nouvelle forme et les futures implantations de cette nouvelle forme. Elle permet donc d'utiliser une version d'un système pour bâtir un nouveau système.		
SCM	Aucune				
SEM	Aucune				
SEP	Aucune				
SETM	CH10 S3.E P159	Re-engineering tools allow translation of a program to a new programming language, or a database to a new format. Reverse engineering tools assist the process by working backwards from an existing product to create abstract artifacts such as design and specification	Il existe des outils qui permettent de traduire un programme pour en produire une version plus évoluée ou d'en développer un nouveau basé sur l'existant. Cette assistance se fait par l'étude de la solution source.	N1	Software Maintenance Tools

Tableau XX (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		descriptions, which then can be transformed to generate a new product from an old one.			
SQ	CH11 S2.2.1 P170	Failure history of similar systems may also help in identifying which activities will be most useful in detecting faults and assessing quality.	En étudiant l'historique des anomalies de systèmes similaires, il est possible d'isoler les éléments à corriger dans le système à développer pour éviter de répéter les mêmes erreurs, et ainsi d'augmenter la qualité du produit.	N1	Common Planning Activities

4.1.15 Uncertainty is unavoidable in software engineering, identify and manage it

En effectuant l'analyse de la correspondance avec ce principe fondamental, nous avons constaté que l'identification et la gestion de l'incertitude faisaient partie intégrante de l'identification et de la gestion des risques. Nous avons donc identifié les citations où il y avait une correspondance avec le risque, plutôt que de limiter cette dernière avec le

seul concept d'incertitude. Il serait peut-être nécessaire d'ajuster la terminologie du libellé en utilisant le risque plutôt que l'incertitude car le risque inclut l'incertitude.

Pour notre analyse de la correspondance avec ce principe fondamental, nous avons identifié les citations où il était question d'une des trois (3) parties suivantes : 1) la reconnaissance que le risque est inévitable en génie logiciel, 2) l'identification du risque, ou 3) la gestion du risque. Le risque peut se situer au niveau du processus (Process risk) ou au niveau du produit (Product risk).

Le tableau XXI présente les correspondances reconnues par rapport à la reconnaissance, l'identification et la gestion du risque.

Tableau XXI

Analyse de la correspondance avec le principe fondamental (O)

Domaine	**Repère**	**Citation**	**Interprétation**	**Niveau**	**Titre de la section**
SR	CH2 S3.4.2 P21	The benefits of the SRS include: ▪ [...] ▪ It provides a realistic basis for estimating product costs, risks and schedules. ▪ [...].	Le SRS permet de documenter les risques que l'ingénieur logiciel identifie au cours de l'explicitation et de l'analyse des exigences. Il s'agit d'une correspondance avec l'identification du	N1	The software requirements specification (SRS)

Tableau XXI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			risque.		
SD	CH3 S3.II P38	▪ Error and exception handling and fault tolerance: how to prevent and tolerate faults and deal with exceptional conditions [...].	En mettant en place, lors du design du logiciel, une structure permettant de prendre en charge les erreurs ou les exceptions, l'ingénieur logiciel gère les risques possibles du système. Cette façon de procéder est primordiale notamment pour les systèmes où la vie humaine ou des conséquences économiques majeures sont en jeu.	SUJET	Key Issues in Software Design
SC	CH4 S2.1 P54	[...], effective design techniques instead acknowledge risk, work to reduce it, and help make sure that effective alternatives	Les techniques efficaces du SD permettent d'identifier les situations à risque et de travailler pour en	DEF	Software Construction and Software Design

Tableau XXI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		will be available when some choices eventually prove wrong, [...].	réduire l'occurrence. Elles voient aussi à mettre en place des moyens pour les contrer lorsqu'elles surviennent ou lorsqu'une solution s'avère inefficace. Il s'agit d'une correspondance avec les trois vues du principe fondamental.		
	CH4 S3.1.3 P59	Structuring for validation means building software in such a fashion that such errors and omissions can be ferreted out more easily during unit testing and subsequent testing activities.	En structurant le logiciel pour en faciliter la validation, l'ingénieur logiciel permet de faire ressortir plus facilement les erreurs ou les manquements aux exigences lors des activités de tests. Cette façon de faire permet de gérer plus efficacement les	N1	Structuring for Validation

Tableau XXI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			risques potentiels.		
	CH4 S3.1.4 P59	This is a complex issue, however, because the selection of an external standard may need to take account of such difficult-to-predict issues as the long-term economic viability of a particular software company or organization that promotes that standard.	L'utilisation de standards externes doit aussi passer au travers une analyse de risque. Il s'agit d'une correspondance avec l'identification du risque.	N1	Use of External Standards
	CH4 S3.3.3.2 P63	Such methods can also be used to "instrument" programs to look for failures based on sets of preconditions. • Assertion-based programming (static and dynamic) • State machine	Les méthodes formelles permettent de gérer le risque en mettant en place des pré-conditions pour réagir à des événements spécifiques à l'intérieur du logiciel. Il s'agit d'une	N+	Formal Construction Methods

Tableau XXI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		logic • Redundant systems, self-diagnosis, and fail-safe methods • Hot-spot analysis and performance tuning • Numerical analysis	correspondance avec la gestion du risque.		
ST	CH5 S1 P69	Testing is an important, mandatory part of software development; it is a technique for evaluating product quality and also for indirectly improving it, by identifying defects and problems.	En effectuant les différentes activités de tests à l'intérieur du projet, l'ingénieur logiciel utilise la façon la plus courante de gérer le risque. Par exemple, en testant le logiciel dans ses limites (stress tests), les situations comportant des risques de mauvais fonctionnement pourront être identifiées.	INTRO	Introduction

Tableau XXI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH5 S2 P70	• Selected: … How to identify the most suitable selection criterion under given conditions is a very complex problem; in practice risk analysis techniques and test engineering expertise are applied;	Pour déterminer les tests devant être effectués, une analyse de risque est effectuée par l'ingénieur logiciel. Cette analyse permet d'identifier les parties du logiciel où le logiciel est le plus vulnérable, que ce soit en raison des compromis qui ont dû y être faits au cours de la conception ou de la construction, ou parce qu'il y a une forte probabilité d'un changement dans le futur. Les cas de tests devront être produits en considérant tous ces risques. Il s'agit d'une	DEF	Definition of the Software Testing

Tableau XXI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			correspondance avec la reconnaissance du risque et avec l'identification du risque.		
	CH5 S3.A2 P75	The obvious reason is that complete testing is not feasible in real systems. Because of this, testing must be driven based on risk, i.e., testing can also be seen as a risk management strategy.	Les tests sont aussi une façon de gérer les risques. Une analyse du risque permet d'identifier les meilleurs tests à utiliser selon les contraintes du projet et les compromis faits au niveau du produit.	N+	Theoretical foundations
SM	CH6 S3.1.6 P90	Preventive maintenance. Modification of a software	La maintenance préventive est un des moyens utilisés pour gérer les risques en éliminant les possibilités de défaillance du logiciel avant qu'elles puissent survenir.	N+	Categories of Maintenance

Tableau XXI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH6 S3.2.2.1 P91	Impact analysis identifies all systems and system products affected by a change request and develops an estimate of the resources needed to accomplish the change [...]. Additionally, the risk of making the change is determined.	Pour chaque changement demandé au système, l'ingénieur logiciel doit évaluer le risque à effectuer le changement demandé sur l'ensemble du système. Il s'agit d'une correspondance avec l'identification du risque.	N+	Unique Activities
	CH6 S3.3.1.3 P93	Impact analysis is necessary for risk abatement.	Les analyses d'impact permettent de réduire le risque dû aux changements au logiciel. C'est la méthode privilégiée par l'équipe de maintenance du logiciel pour identifier les risques reliés aux changements à	N+	Impact Analysis

Tableau XXI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
			effectuer.		
SCM	CH7 S3.III.A P109	The software change request process [...] provides formal procedures for submitting and recording change requests, evaluating the potential cost and impact of a proposed change, and accepting, modifying or rejecting the proposed change.	Au cours du processus de demande de changement, l'évaluation de l'impact du changement proposé permet d'identifier les risques potentiels sur l'ensemble du logiciel.	N1	Requesting, Evaluating and Approving Software Changes
SEM	CH8 S3.B.2.6 P125	Risk management – risk identification and analysis [...], critical risk assessment [...], risk mitigation and contingency planning [...] are all undertaken. Risk assessment methods [...] should be used in order to highlight and evaluate risks.	La gestion du risque fait partie intégrante des activités de planification du processus de gestion de projet.	N+	Risk management

Tableau XXI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	CH8 S3.B.3.3 P126	Monitor process – [...] Risk exposure and leverage are recalculated and decision trees, simulation and so on are re-run in the light of new data.	Lorsque le projet est en cours, il faut continuellement réévaluer son exposition aux différents risques identifiés lors de la planification du projet. Il s'agit d'une correspondance avec la gestion du risque.	N+	Monitor process
SEP	CH9 S3.2.2 P141	• Lessons learned (e.g., risks associated with an Ada development).	Les leçons retenues des projets passés, permettent d'identifier les risques des projets à venir, dans un contexte similaire.	N+	The Experience Factory
SETM	CH10 S3.I P159	Risk management tools	Il existe des outils qui permettent de gérer les risques d'un projet de développement logiciel.	N+	Software Tools
SQ	CH11 S2.2	SQA seeks to retain the quality throughout	Les activités de SQA permettent	SUJET	Purpose and Planning of

Tableau XXI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
	P169	the development and maintenance of the product by execution of a variety of activities at each stage that can result in early identification of problems, which are almost inevitable in any complex activity.	d'identifier les problèmes tôt au cours du processus de développement logiciel. L'identification de problèmes au logiciel constitue une façon de gérer les risques du produit.		SQA and V&V
	CH11 S2.3.1.1 P172	Management reviews determine adequacy of and monitor progress or inconsistencies against plans and schedules and requirements. These reviews may be exercised on products such as audit reports, progress reports, V&V reports and plans of many types including risk management, software safety, and risk assessment,	Les révisions de la gestion du projet voient, entre autres, à vérifier la consistance et les progrès de la gestion des risques au cours du projet.	N+	People-Intensive Techniques

Tableau XXI (suite)

Domaine	Repère	Citation	Interprétation	Niveau	Titre de la section
		among others.			
	CH11 S2.5.4 P175	SQA and V&V processes discover defects. [...] When tracking defects, the software engineer is interested not only in count of defects, but the types. Without some classification, information will not really be useful in identifying the underlying causes of the defects because no one will be able to group specific types of problems and make determination about them.	En caractérisant les défauts découverts au cours des activités de SQA et de V&V, l'ingénieur logiciel effectue, d'une certaine façon, des activités de gestion du risque.	N1	Defect Characteri-zation

4.1.16 Présentation de la correspondance dans les domaines connaissances

Les figures présentées dans cette section permettent de visualiser les endroits dans la taxonomie (DOM, SUJET, N1 et N+) de chacun des domaines de connaissances où la correspondance avec les différents principes fondamentaux a pu être reconnue. Les figures de base proviennent du Guide SWEBOK; il y a une figure par domaine de connaissances. La lettre d'identification des principes fondamentaux est utilisée dans les figures pour les identifier. Lorsqu'une correspondance se retrouve à un niveau inférieur à ce qui est présenté comme taxonomie dans la figure, le signe « - » est utilisé à côté de la lettre d'identification du principe fondamental. Les correspondances au niveau de l'introduction, des définitions et des textes ne sont pas présentées dans les figures car elles sont considérées comme étant à l'extérieur de la taxonomie des domaines de connaissances.

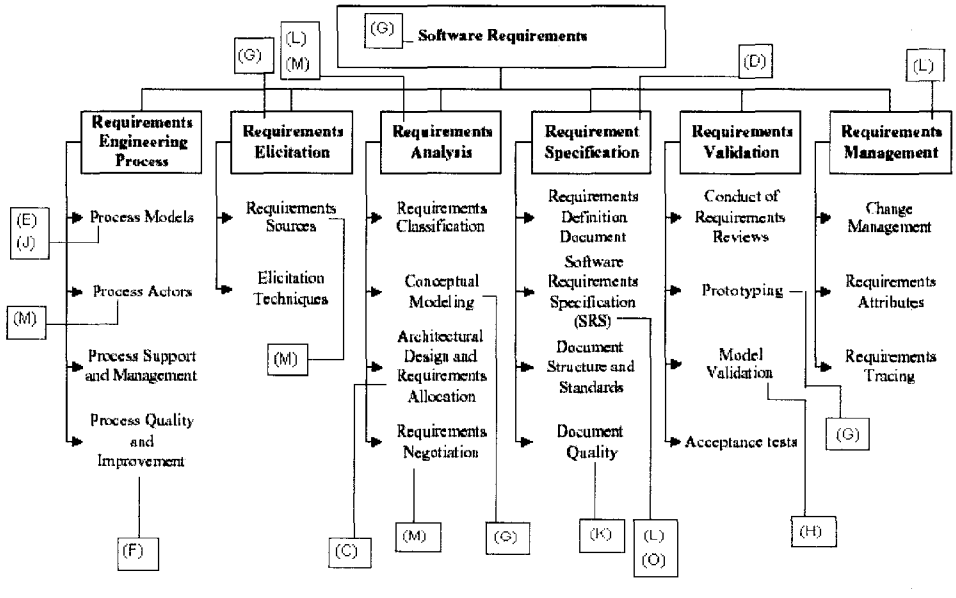


Figure 13 Correspondances reconnues dans le SR

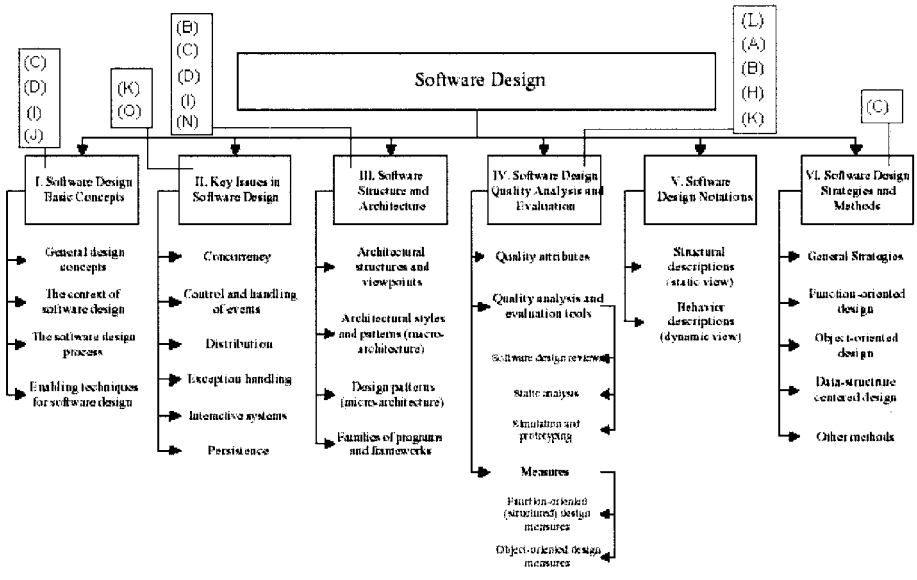


Figure 14 Correspondances reconnues dans le SD

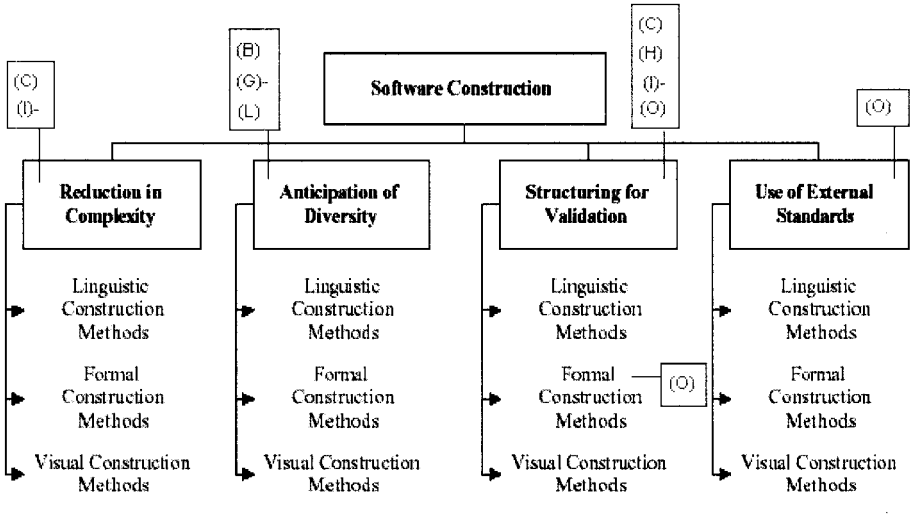


Figure 15 Correspondances reconnues dans le SC

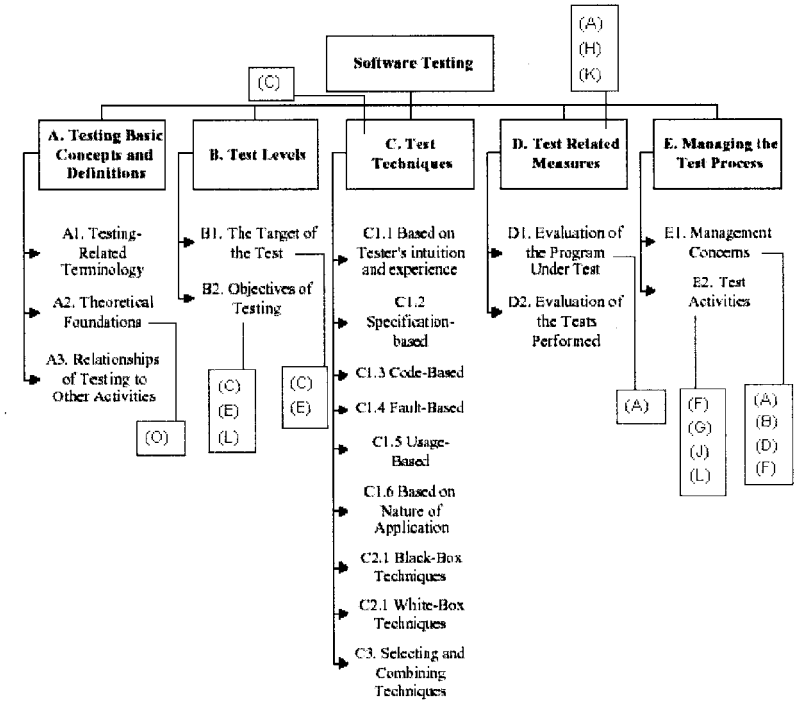


Figure 16 Correspondances reconnues dans le ST

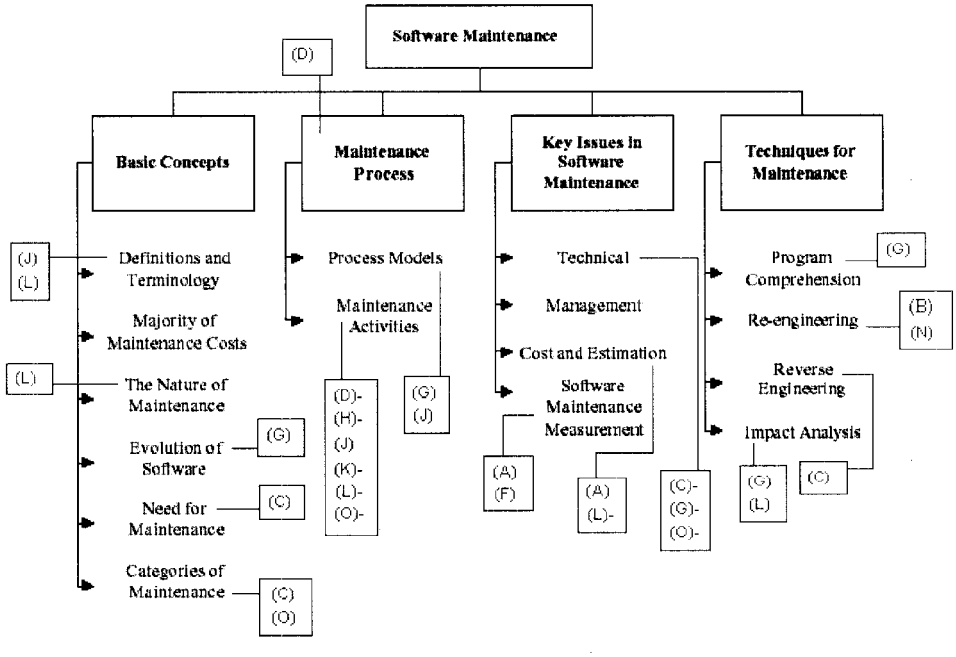


Figure 17 Correspondances reconnues dans le SM

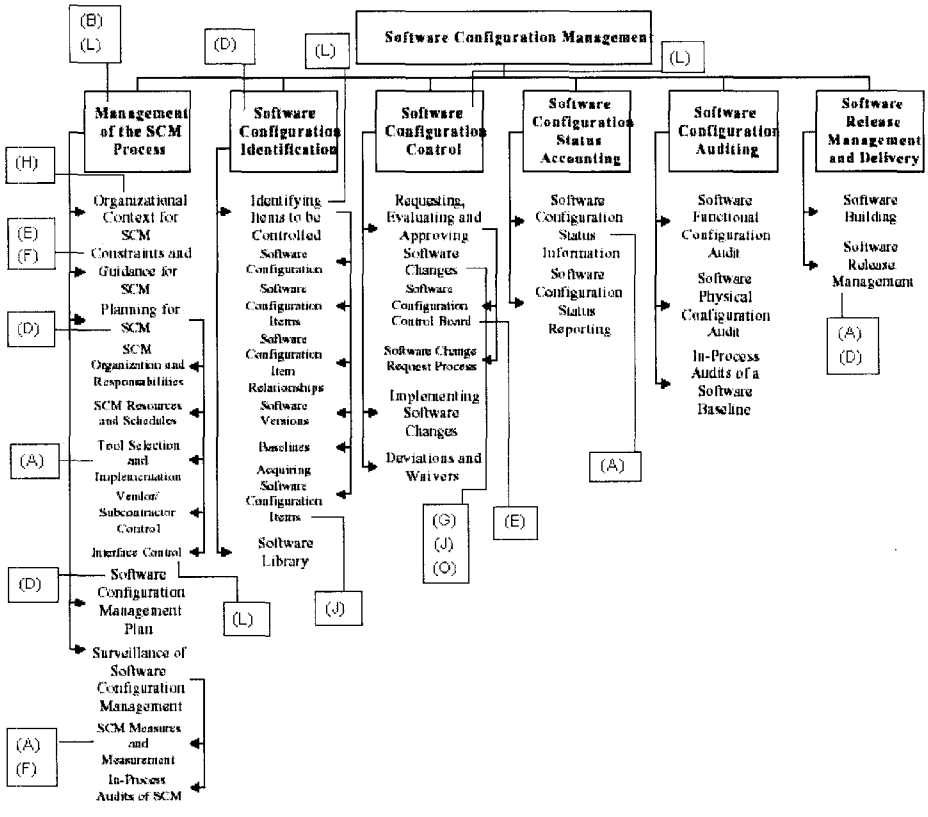


Figure 18 Correspondances reconnues dans le SCM

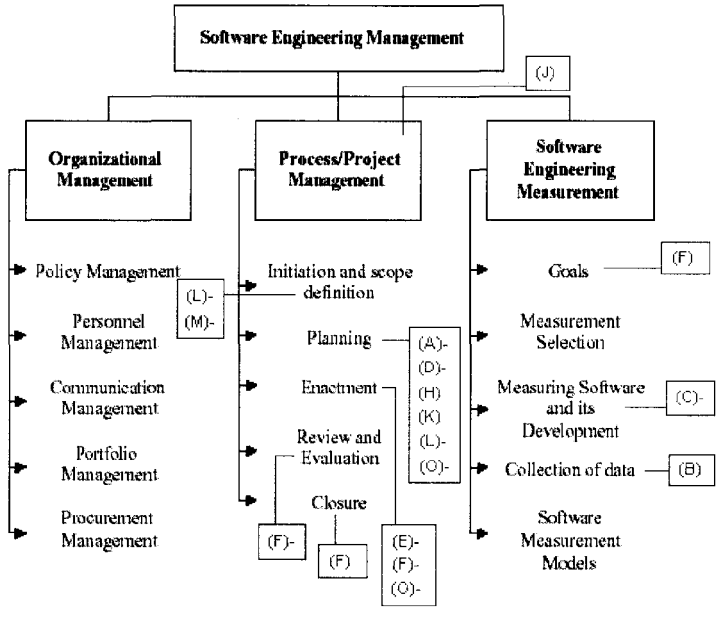


Figure 19 Correspondances reconnues dans le SEM

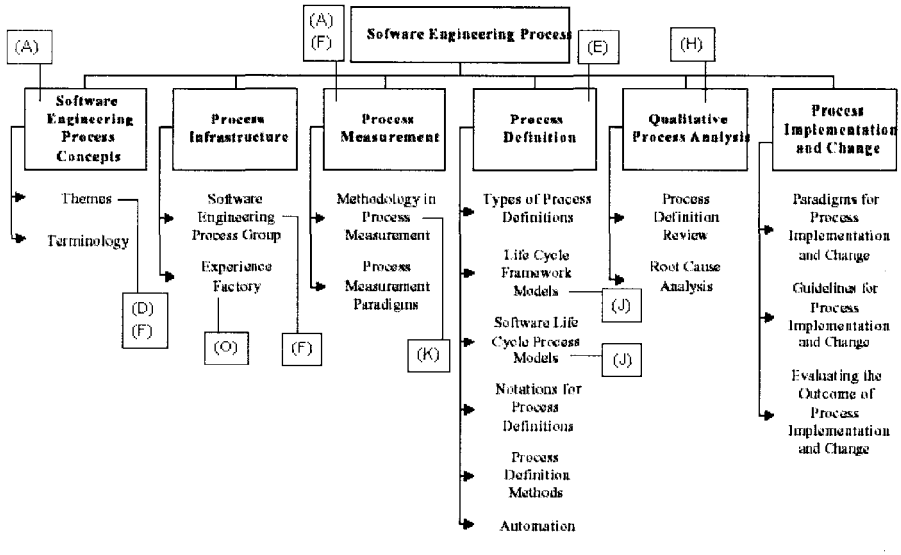


Figure 20 Correspondances reconnues dans le SEP

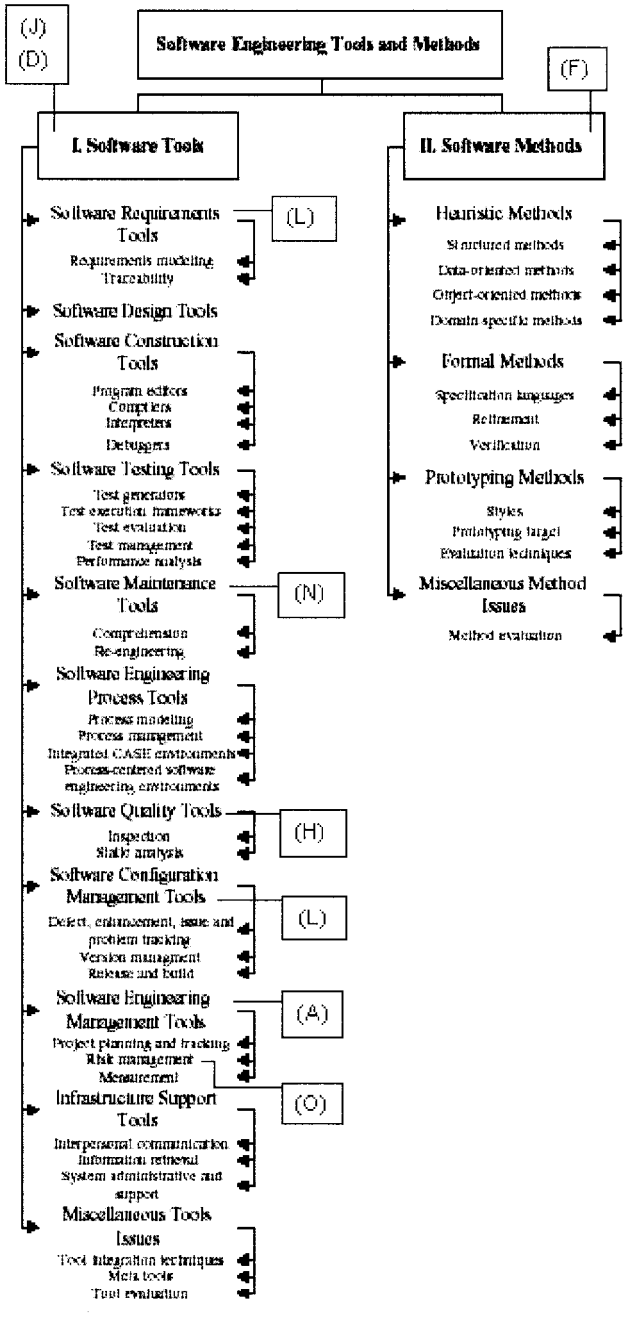


Figure 21 Correspondances reconnues dans le SETM

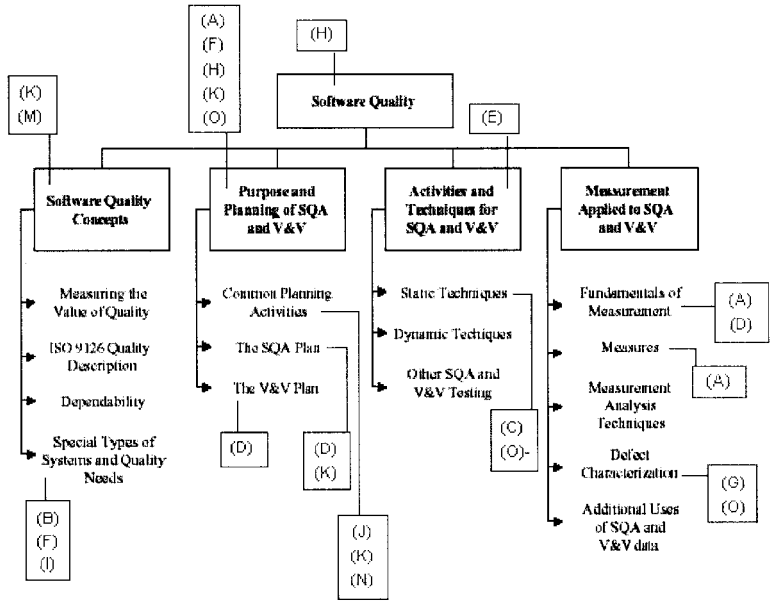


Figure 22 Correspondances reconnues dans le SQ

4.2 Production d'une description d'accompagnement par principe fondamental

Les tableaux produits lors de l'analyse de la correspondance entre les principes fondamentaux et le contenu des différents domaines de connaissances du Guide SWEBOK (section 4.1.1), sont utilisés comme intrants à cette seconde partie de l'étude. Les différentes citations présentées dans ces tableaux nous ont permis de déterminer les notions devant être couvertes par le principe fondamental. Ces notions sont présentées sous forme de tableaux. Elles permettent de solidifier l'interprétation à donner aux libellés des principes fondamentaux, en produisant une ébauche de description pour chacun d'eux.

Il faut noter que ce ne sont pas toutes les citations de l'analyse de correspondance présentant la notion identifiée qui sont présentées. La sélection des citations de présentation des notions est faite pour faire comprendre au lecteur comment la notion doit être interprétée.

Chacune des sous-sections qui suivent, présente un tableau identifiant les notions couvertes par le principe fondamental, et une version française et anglaise de la description suggérée pour appuyer le libellé de chacun des principes fondamentaux à l'étude.

4.2.1 Apply and use quantitative measurements in decision-making

Tableau XXII

Notions du principe fondamental (A)

Notion	Citation	Repère(s)
Les mesures doivent être basées sur les objectifs de l'organisation, du processus et du projet.	• [...] When collecting measurements, it is important that the measurements collected be relevant to the goals of the development process.	[CH5S3.DP79]
Les mesures fournissent les informations nécessaires aux gestionnaires pour la prise de décision.	• Effort, schedule and cost estimation – [...] • [...] they can help management decision-making.	[CH8S3.B.2.4P125] [CH11S2.5.2P175]
Les mesures doivent être planifiées et permettre d'évaluer	• [...] Software measures are vital for software process improvement but the process must be measurable.	[CH6S3.3.4P94] [CH7S3.I.E.1P107] [CH11S2.5.1P174]

Tableau XXII (suite)

Notion	Citation	Repère(s)
le projet et/ou le processus en vue de leur évolution.	• SCM metrics can be designed to provide specific information on the evolving product or to provide insight into the functioning of the SCM process. • Measurement programs are considered useful if they help project stakeholders (1) understand what is happening during their processes, and (2) control what is happening in their projects.	

Anglais

Measurements are an integral part of engineering. Therefore, in order for the discipline known as software engineering to be considered part of the engineering family it needs to be able to clearly define a series of recognized measuring principles. But with software, there are no silver bullet measures; measurement programs must be tailored to the organization and the process/project. They need to be planned for and based upon the objectives as defined by the organization. Measurements will only be deemed useful should they assist stakeholders (1) understand what occurs over the course of the process, and allows them to control (2) the outcome of the project. Clearly, they assist management in reaching better decisions. Notwithstanding, properly designed measures can support software quality in multiple ways. Measuring product quality throughout the project goes beyond simply determining whether the system works or how will it achieve measurable quality goals.

Estimates are an integral part of decision-making. They assist management by providing valuable quantifiable information with regard to deadline evaluation and project risk assessment. Estimated values need to be based upon previous projects with similar contexts and processes. They are essential to the evaluation of project progression. They allow project managers to confirm or revise projected goals. Finally, they allow for the identification of project deviations during the post-mortem review.

Software measurements are essential to the improvement of the software process but in order to do so the process must be measurable. Process measurement generally entails the use of diagnostic quantitative techniques to identify software processes strengths and weaknesses. Analysis of the measurements may provide insights into necessary process adjustments.

Français

Les mesures sont fondamentales pour l'ingénierie en général. Pour être considéré comme l'un des champs du génie, l'ingénierie logicielle doit considérer les mesures à sa base. Mais avec le logiciel, il n'y a pas de mesures répondant à tous les cas; les programmes de mesure doivent être ajustés à l'organisation et au processus/projet. Ils doivent être planifiés et basés sur les objectifs fixés par l'entreprise. Les mesures seront considérées utiles si elles aident les intervenants du projet à (1) comprendre ce qui se passe à l'intérieur des processus, et (2) à contrôler ce qui se passe dans leurs projets. En clair, elles peuvent aider la direction dans ses prises de décisions. Si elles sont conçues convenablement, les mesures peuvent supporter la qualité du logiciel de plusieurs façons. La mesure de la qualité du produit à chaque étape du projet, va plus loin que la seule considération du fonctionnement ou non du système; elle permet d'identifier le niveau de qualité atteint.

L'estimation est une partie importante de la prise de décisions. Elle donne des informations sur l'évaluation des échéanciers et sur les risques du projet. Les estimations doivent être basées sur les mesures prises au cours de projets passés, dans un contexte similaire ou utilisant le même processus. Les mesures sont nécessaires pour évaluer l'avancement du projet. Elles permettent au gestionnaire de projet de confirmer ou de réviser les cibles prévues. Enfin, elles permettent d'identifier les écarts ayant eu lieu au cours du projet lors de la révision post-mortem.

Les mesures du logiciel sont vitales pour l'amélioration du processus logiciel, mais le processus doit être mesurable. La mesure du processus est normalement concernée par les techniques quantitatives de diagnostique des processus logiciels pour en identifier les forces et les faiblesses. L'analyse des mesures recueillies permet d'identifier des éléments à modifier dans le processus.

4.2.2 Build with and for reuse

Tableau XXIII

Notions du principe fondamental (B)

Notion	Citation	Repère(s)
Il y a plusieurs façons au cours du cycle de vie du logiciel de bâtir en fonction de la réutilisation.	• System developers – these have a legitimate interest in profiting from developing the system by, for example, reusing components in different products.	[CH2S2.2P11] [CH2S2.3P12] [CH5S3.E1P81]

Tableau XXIII (suite)

Notion	Citation	Repère(s)
	• Architectural design is a skill that is driven by many factors such as the recognition of reusable architectural "patterns" or the existence of off-the shelf components. • At all levels of testing, test scripts, test cases, and expected results should be carefully defined and documented so that they may be reused.	
Il y a des moments où la réutilisation n'est pas souhaitable ou difficile à planifier et à réaliser.	• If, in this scenario, a customer of a particular product has specific requirements that compromise the potential for component reuse, the developer must carefully weigh their own stake against those of the customer. • [...] though reuse is desirable and important, it is not always pertinent. It depends on many factors, notably the economic context and the delivery schedule.	[CH2S2.2P11] [BourqueS4.2P65]
La réutilisation dans le logiciel peut être identifiée et mesurée.	• An interesting distinction is the one between quality attributes discernable at run-time (e.g., [...]), those not discernable at run-time	[CH11S2.1.4P167]

Tableau XXIII (suite)

Notion	Citation	Repère(s)
	(e.g., modifiability, portability, reusability, integrability and testability) [...].	

Anglais

Software building is not limited to code writing. The IEEE defines software as the computer programs, the procedures, and possibly the associated documentation and data pertaining to the operation of a computer system [1]. It covers the entire software life cycle. In software production, the creation of reusable artifacts leads to making them available to other programs, software modules or systems or even an intermediary product. The goal is to take advantage of the company or even a third party's past experiences and adapts them to the situation at hand by recycling them once proven effective.

Often times, software developers attempt to automate construction, to develop toolkits or use third party components to accelerate software construction. Software architects identify architectural reusable components by creating frameworks to resolve similar problems. Software testers create artifacts used for testing that can be reused systematically over the entire software life cycle.

When considering building, one of software engineers' most important requirements is reusability. However, this may lead to trading-off activities whereby these concepts go against customer requirements. Should a particular product have specific requirements, which may compromise the reusability of the component, it may be that the software engineer will need to carefully weigh its own stake against that of the customer. Even tough reusability is the most desirable outcome; it may not always be pertinent. This may be influenced by a variety of factors, notably the economic context and the delivery schedule.

As defined by the IEEE, reusability is the quality factor that can be measured to determine the extent of which a software module or other work product can be used in more than one computer program or software system [1].

Français

La production d'un logiciel n'est pas seulement l'écriture du code. Le IEEE définit le logiciel comme des programmes informatiques, des procédures, et possiblement de la documentation associée et des données permettant l'opération du système informatique [1]. Le cycle de vie complet du logiciel est traité. Pour le logiciel, la production en vue d'obtenir des artéfacts réutilisables, vise à rendre disponible pour utilisation dans un autre programme informatique ou système logiciel, un module du logiciel ou un autre produit intermédiaire. Le but est de prendre avantage de l'expérience passée de l'entreprise ou d'une tierce partie, de l'adapter aux besoins présents et de la réutiliser une fois qu'elle est jugée efficace.

Les développeurs logiciels tentent le plus souvent possible d'automatiser la construction, de développer des outils ou d'utiliser des composants de tierces parties pour accélérer la construction du logiciel. Les architectes logiciels identifient des composants réutilisables en créant une structure permettant de résoudre les problèmes du même ordre. Les testeurs logiciels créent des éléments utilisés pour les tests pouvant être réutilisés systématiquement à travers tout le cycle de vie du logiciel.

Produire avec ou en fonction de la réutilisation est l'une des plus importantes exigences de l'ingénieur logiciel. Mais, elle peut amener à des activités de négociation lorsqu'elle va à l'encontre des exigences d'un client. Si, dans le scénario où pour un produit particulier ayant des exigences compromettant le potentiel de réutilisation de composants, l'ingénieur logiciel doit bien peser le pour et le contre de ses propres besoins face à ceux du client. Ainsi, la réutilisation est désirable et importante, mais elle n'est pas toujours pertinente. Elle dépend de plusieurs facteurs, notamment le contexte économique et la cédule de livraison.

Telle que définie par le IEEE [1], la « réutilisabilité » est un facteur de qualité mesurable. Ce facteur peut informer du degré avec lequel un composant logiciel ou un autre produit de travail peut être utilisé par plus d'un programme ou système.

4.2.3 Control complexity with multiple perspectives and multiple levels of abstraction

Tableau XXIV

Notions du principe fondamental (C)

Notion	Citation	Repère(s)
L'analyse de la complexité permet de déterminer si un produit est trop complexe.	• An example of a support technique is complexity analysis, useful for determining that the design or code may be too complex to develop correctly, to test or maintain.	[CH11S2.3.1.2P172]
Le contrôle de la complexité selon plusieurs perspectives.	• There are three main techniques for reducing complexity : 3.1.1.1 Removal of complexity [...] 3.1.1.2 Automation of complexity [...] 3.1.1.3 Localization of complexity [...] • 3.3.1 Reduction in complexity 3.3.1.1 Linguistic Construction Methods [...] 3.3.1.2 Formal Construction Methods [...] 3.3.1.3 Visual Construction	[CH4S3.1.1P57] [CH4S3.3.1P62]

Tableau XXIV (suite)

Notion	Citation	Repère(s)
	Methods [...]	
Le contrôle de la complexité à plusieurs niveaux d'abstraction.	• The requirements allocated to components that are complex systems in themselves will need to undergo further cycles of analysis in order to add more detail, and to interpret the domain-oriented system requirements for developers [...]. • Software design methods are used to express a global solution as a set of smaller solutions and can be applied repeatedly until the resulting parts of the solution are small enough to be handled with confidence by a single developer.	[CH2S2.3P12] [CH4S2.1P54]
Les tests pour contrôler la complexité selon les perspectives et les niveaux d'abstraction.	• The first classification, [...], is based on how tests are generated, i.e., respectively from: tester's intuition and expertise, the specifications, the code structure, the (real or artificial) faults to be discovered, the field usage or finally the nature of application, [...] • The second classification is the	[CH5S3.CP77] [CH5S3.B1P75]

Tableau XXIV (suite)

Notion	Citation	Repère(s)
	classical distinction of test techniques between black-box and white-box. • Testing of large software systems usually involves more steps [...]. Three big test stages can be conceptually distinguished, namely Unit, Integration and System.	

Anglais

Complexity can be defined as the degree by which a designed system or component is difficult to understand and verify [1]. The can be determined by effecting complexity analysis.

There are multiple ways to reduce software complexity in terms of perspective over the course of the software development life cycle. This can be accomplished via localization and removal of the complexity, by automating those parts where it has been identified or by isolating it to better control it. All or many of these solutions may be applied while performing software requirements, software design or software construction activities.

The software analyst's goal is to delve deeper and deeper into the details of the problem, going further in the abstraction levels to achieve a complete understanding of the requirements. Complex requirements need to undergo multiple cycles of analysis, each new one achieving a higher level of understanding. The software architect has the same goal when analyzing the highest level of the system architecture in order to create the

detailed software design blueprint. When creating the detailed design, emphasis is put on successfully partitioning a complex problem. Software design methods are used to extract a global solution as a series of smaller solutions to which conception methods may be repeatedly applied until the resulting parts are small enough that they can be handled with confidence by a single developer.

Testing activities may also help the software engineer controls complexity. This can be achieved with multiple levels of abstraction, from unit testing to complete system testing. One could also accomplish this via the application of multiple perspectives using various testing methods (e.g. white-box versus black-box) during software test activities.

Français

La complexité peut être définie comme le degré avec lequel un système ou un composant conçu ou implanté est difficile à comprendre et à vérifier [1]. La complexité peut être évaluée en réalisant une analyse de la complexité.

Il y a plusieurs façons de réduire la complexité du logiciel en terme de perspectives durant le cycle de développement du logiciel. Cela peut être en la localisant pour la retirer, en automatisant les parties où elle est identifiée ou en l'isolant dans des composants pour faciliter son contrôle. Toutes ou plusieurs de ces solutions peuvent être appliquées au cours des activités d'ingénierie des exigences, de design ou de construction.

Les exigences complexes doivent passer par plusieurs cycles d'analyse pour ajouter du détail à leur compréhension. Ainsi, le but de l'analyste logiciel est d'aller de plus en plus loin dans le détail du problème, allant de l'avant dans les niveaux d'abstraction pour obtenir une compréhension complète des exigences. L'architecte logiciel a le même but

quand il analyse le niveau le plus élevé de l'architecture du système, pour ensuite créer le plan détaillé de la conception du logiciel. Lors de la création de la conception détaillée, l'accent est mis sur la manière de diviser efficacement un problème complexe. Des méthodes de conception logicielle sont utilisées pour exprimer une solution globale en un ensemble de solutions de petite taille sur lesquelles les méthodes de conception peuvent être appliquées répétitivement jusqu'à ce que les portions résultantes soient assez petites pour être résolues sans problème par un seul développeur.

Les activités de tests peuvent aussi aider l'ingénieur logiciel à contrôler la complexité. Ce contrôle peut être fait à plusieurs niveaux d'abstraction, en testant de façon unitaire un composant jusqu'au test complet du système. Ou, il peut être fait en plusieurs perspectives par l'application de différentes méthodes de tests (ex. : « white-box » versus « black-box ») au cours des activités de tests du logiciel.

4.2.4 Define software artifacts rigorously

Tableau XXV

Notions du principe fondamental (D)

Notion	Citation	Repère(s)
Il faut identifier les artéfacts à produire par les différentes activités du processus, selon le contexte.	• The scope of this knowledge area follows the general focus of the Guide; that is, "emphasis… is placed upon the construction of useful software artifacts" […]. • Determine deliverables – the product(s) of each task […] are	[CH8S2.1P121] [CH8S3.B.2.3P125] [BourqueS4.5P66]

Tableau XXV (suite)

Notion	Citation	Repère(s)
	specified and characterized. • [...] the desired level of rigor is dependent upon the situation and , notably, the level of criticality of the software.	
Il faut être en mesure de valider et de vérifier les artéfacts produits face aux exigences, aux standards ou aux autres facteurs de qualité.	• The requirements document(s) must be subject to validation and verification procedures. [...] It is also important to verify that a requirements document conforms to company standards, and is understandable, consistent and complete. • These models can be analyzed and evaluated to determine if they will allow the various requirements to be fulfilled.	[CH2S2.5P14] [CH3S2P35]
Il existe plusieurs types d'artéfacts.	• Documentation is an integral part of the formalization of the test process. [...] • Software items with potential to become SCIs include plans, specifications and design documentation, testing materials, software tools, source and executable code, code libraries,[...]	[CH5S3.E1P80] [CH7S3.IIP108]

Tableau XXV (suite)

Notion	Citation	Repère(s)
Il faut être en mesure de suivre les artéfacts dans leur évolution.	• A first step in controlling change is to identify the software items to be controlled.	[CH7S3.IIP108]

Anglais

For each activity realized during the many processes of the software life cycle, inputs and outputs need to be clearly defined. By that we meant that the products of the activity – the deliverables – are defined, i.e. the product(s) of each task are specified and characterized. Emphasis should be placed upon the production of *useful* software artifacts. Thus, the list of deliverables to be produced must be independent from the situation and particularly the software's criticality level.

Each deliverable must be subject to rigorous verification procedures to determine its conformity to company standards. Each artifact should be understandable, consistent and complete. Documentation is an integral part of the formalization of the software development process. It may include among others things, plans, procedures, specification documents, logs, reports, etc. Other types of process products include models, source code, code libraries, tools, executables or test cases.

With regard to consistency and completeness, the resulting products of the software development processes should be produced and updated as the software evolves, and the requirements should be traceable forward or backward within all the artifacts resulting from each different activity. Each item, or artifacts, must be controlled over the software's life cycle and as a result they need to be constantly and systematically refined as the software development process progresses.

Français

Pour chacune des activités réalisées au cours des différents processus du cycle de vie du logiciel, les entrées et les sorties doivent être clairement définies; c'est-à-dire, que les produits des activités – les biens livrables – sont définis, i.e. qu'ils sont spécifiés et caractérisés. L'accent devrait être mis sur la production d'artéfacts logiciels *utiles*. Donc, la liste des biens livrables à produire devrait être dépendante de la situation et notamment, du niveau de criticité du logiciel.

Pour être rigoureux, chaque bien livrable devrait être vérifié pour assurer sa conformité face aux standards de la compagnie, et qu'il est compréhensible, consistant et complet. La documentation est une partie intégrante de la formalisation du processus de développement logiciel. Elle inclut, entre autres, les plans, les procédures, les documents de spécifications, les journaux de bord, les rapports, etc. D'autres types de produits sont les modèles, le code, les librairies de codes, les outils, les programmes exécutables et les cas de tests.

Lorsqu'il est question de consistance et de complétude, les produits résultants des processus du développement logiciel devraient être produits et mis à jour avec l'évolution du logiciel, et les spécifications devraient pouvoir être retracées à l'intérieur de tous les artéfacts résultant des différentes activités. Tous les items – artéfacts – devraient être contrôlés tout au long du cycle de vie du logiciel, et systématiquement et continuellement raffinés avec l'avancement du développement du logiciel.

4.2.5 Establish a software process that provides flexibility

Tableau XXVI

Notions du principe fondamental (E)

Notion	Citation	Repère(s)
Le processus doit être choisi selon le contexte du projet/produit.	• […] requirements engineering process: […] - will need to be tailored to the organization and project context. • Firstly, software construction is influenced by the scale or size of the software product being constructed. • It should be noted also that the context of the project and organization will determine the type of process definition that is most important.	[CH2S3.1.1P16] [CH4S2.1P54] [CH9S3.4P144]
Le processus doit être adapté selon les contraintes organisationnelles ou du projet.	• The different activities in requirements engineering are repeated until an acceptable requirements specification document is produced or until external factors such as schedule pressure or lack of resources cause the requirements engineering process to terminate.	[CH2S3P16] [CH7S3.I.BP105]

Tableau XXVI (suite)

Notion	Citation	Repère(s)
	• Policies and procedures set forth at corporate or other organizational levels might influence or prescribes the design and implementation of the SCM process for a given project.	
Les exigences logicielles influencent la sélection du processus de développement et l'adaptation à laquelle il devra être soumis.	• [...] it is almost always impractical to implement requirements engineering as a linear, deterministic process [...] • Static and dynamic techniques are used in either SQA or V&V. Their selection, specific objectives and organization depend on project and product requirements.	[CH2S2.4P13] [CH11S2.3P171]

Anglais

An important part of software engineering is to make a informed choice of software process framework for a given software project. Process must be tailored to the organization and project contexts. There is no all-encompassing process, which can be applied to all projects, and for all organizations. Process choice depends upon the scale or size of the software product to be constructed. It should also be noted that the context of the project and organization could determine the nature of the most appropriate type of process. Important variables to consider include the types of work, the application domain, the delivery process structure and the organization's maturity level.

Process activities should be configured for different types of project and constraints. Some activities could be repeated until an acceptable deliverable is produced or until such times as external factors such as schedule pressure or lack of resources cause the process to terminate. Assigned tasks and responsibilities may vary according to the scale of the project or the organization. Corporate level policies and procedures may influence or prescribe the software development process of any given project. For example, project managers could also consider the use of additional standards selected to be suitable to the specific characteristics of the project.

Requirements engineering is an important part of the software development process. It is unrealistic to implement requirements engineering as a linear, deterministic process. Such processes should allow for flexibility since you rarely achieve a complete view of customer's needs on the first elicitation round. If it is to be successful, requirements engineering must be considered as a process with complex, tightly coupled activities (both sequential and concurrent) rather than as a discrete, one-off activity at the onset of a software development project. Selection of SQA and V&V techniques, specific quality objectives and organization hinge on project and product requirements.

Français

Une partie importante de l'ingénierie du logiciel est de faire un choix judicieux de la structure du processus pour un projet de développement logiciel donné. Le processus doit être adapté au contexte de l'organisation et du projet. Il n'existe pas de processus miracle pouvant être appliqué à tous les projets, pour toutes les organisations; son choix est notamment influencé par la taille du logiciel à développer. Il faut aussi considérer que le contexte du projet et de l'organisation pourrait déterminer le type de processus qui est le plus approprié. Des variables importantes également à considérer pour le choix du processus sont la nature du travail, le domaine des affaires, la structure du processus de livraison, et de la maturité de l'organisation.

Les activités du processus devraient être configurées pour différents types de projets et de contraintes. Certaines activités pourraient être répétées jusqu'à ce qu'un livrable acceptable puisse être produit ou jusqu'à ce que des facteurs externes, comme la pression de l'échéancier ou le manque de ressources, obligent de les considérer comme étant complétées. Les tâches assignées et les responsabilités peuvent aussi différer selon la taille du projet ou de l'organisation.

Les politiques et procédures au niveau corporatif ou de l'organisation, peuvent influencer ou prescrire le processus de développement logiciel pour un projet donné. Par exemple, le chargé de projet peut considérer l'utilisation de standards additionnels pour un projet ayant des caractéristiques spécifiques.

Une des parties importantes du processus de développement logiciel est l'ingénierie des exigences. Il est pratiquement impossible d'implanter un processus d'ingénierie des exigences en un processus linéaire, prédéterminé; il doit permettre de la flexibilité car il est rare d'avoir une vue complète des besoins du client à la première ronde de l'explicitation des exigences. Pour avoir du succès, l'ingénierie des exigences doit être considérée comme un processus ayant des activités complexes, fortement liées les unes aux autres (autant séquentielles que concurrentes) plutôt qu'une activité unique, avec une entrée et une sortie, se trouvant au début du projet de développement logiciel. La sélection des techniques d'AQL et de V&V, les objectifs de qualité spécifiques et l'organisation dépendent du projet et des exigences du produit à développer.

4.2.6 Implement a disciplined approach and improve it continuously

Tableau XXVII

Notions du principe fondamental (F)

Notion	Citation	Repère(s)
Le fait de mettre en place des activités d'évaluation à l'intérieur même du processus, permet d'améliorer la qualité des produits.	• [...] the quality of the process will largely determine the quality, timeliness, and cost effectiveness of the result. • Integrated evaluation means that a process (in this case a development process) includes explicit continuous or periodic internal checks to ensure that it is still working correctly. [...] • [...] the process itself is systematically and periodically assessed for its relevance, utility and efficacy in the process/project context [...].	[BourqueS4.6P66] [CH4S2.3P55] [CH8S3.B.4.2P127]
Les mesures sont primordiales pour évaluer et améliorer le processus en place.	• Software measures are vital for software process improvement but the process must be measurable. • Control process – the outcomes of the process monitoring activities provide the basis on which action	[CH6S3.3.4P94] [CH8S3.B.3.4P126] [CH8S3.C.1P127]

Tableau XXVII (suite)

Notion	Citation	Repère(s)
	decisions are taken. Where appropriate, and where the impact and associated risks are modeled and managed, changes can be made to the process/project. • Each measurement endeavor should be guided by organizational objectives and driven by an over-riding goal that has organizational improvement at its foundation.	
Il y a plusieurs façons d'assurer l'amélioration continue du processus.	• [...] Best practice is also reflected in process improvement and process assessment models [...] • The various methods, tools and techniques employed are evaluated for their effectiveness and appropriateness, [...] • The SEPG is intended to be central focus for process improvement within an organization. • [...] the disciplined approach or process itself could become the focus of attention at the expense of the product to be developed, [...]	[CH7S3.I.BP105] [CH8S3.B.4.2P127] [CH9S3.2.1P141] [BourqueS4.6P66]

Anglais

The quality of the processes used to develop software will largely determine the quality, timeliness, and cost effectiveness of the result. Therefore, to improve software quality, process must be continuously assessed and changes implemented where appropriate. An accepted practice is to implement a process with integrated evaluation. Integrated evaluation means that the process (in this case, the software development process) includes explicit continuous or periodic internal checks to ensure that it is still running properly, particularly once a change has been made. These checks usually consist of evaluations of intermediate work products, but they can also look at characteristics of the software development process itself. Ultimately, all process work is directed at 'Software process assessment and improvement'.

Software measures are vital for software process improvement but the process must be measurable. Measurement programs are necessary to process assessment. Even if they are geared towards providing specific information about the product, they can also contain a supplementary function that of monitoring the process to discover avenues for process improvement. The outcomes of the process monitoring activities provide the basis upon which action decisions are taken. By measuring the resulting products of the development process, the organization obtains information about the process itself. Product quality evaluation can also act as a means to improving the process. Where appropriate, and where impact and associated risks are modeled and managed, process changes can be made. Defect tracking information can therefore be used to determine which aspects of system development require improvement as well as determining the validity of previously effected tests and analysis. Evaluation of test phase reports is often combined with root cause analysis to evaluate test process effectiveness in finding defects as early as possible.

Tools can improve software quality by allowing development personel to avoid repetitive or precise work for which a computer is better suited. The various tools, methods and techniques employed should be evaluated for their effectiveness and appropriateness according to the process/project context. Software engineering process group (SEPG) is intended to be the focus for process improvement within an organization. By naming a group with a "mandate" to improve processes, the organization ensures that objectives and activities for process improvement will be deployed. Best practices are reflected in process improvement and process assessment models. Nevertheless, the focus of the entire organization must not be solely process and process improvement at the expense of the product to be developed.

Français

La qualité du processus utilisé pour développer le logiciel détermine la qualité, l'échéancier et les coûts du résultat. Alors, pour améliorer la qualité du logiciel, le processus doit être continuellement évalué et les changements implantés où ils doivent s'appliquer. Une bonne pratique serait d'implanter un processus avec une évaluation intégrée. L'évaluation intégrée consiste à inclure dans le processus (dans ce cas-ci, le processus de développement logiciel) des points de contrôle pour s'assurer de son bon fonctionnement, surtout après un changement. Ces points de contrôle consistent généralement en une évaluation des produits intermédiaires, mais ils peuvent aussi vérifier les caractéristiques du processus même. Tout le travail en ce qui concerne le processus, vise, à la fin, l'évaluation et l'amélioration du processus logiciel.

Les mesures sont vitales pour l'amélioration du processus logiciel, mais le processus doit être mesurable. Les programmes de mesure font partie de l'évaluation du processus. Même s'ils ont comme but de fournir des informations spécifiques au sujet du produit, ils peuvent aussi avoir un but supplémentaire de surveiller le processus pour découvrir des opportunités d'amélioration du processus. Les résultats des activités de surveillance

du processus fournissent une base avec laquelle les décisions d'amélioration sont prises. En mesurant les produits résultant du processus de développement, l'organisation obtient des informations sur le processus même. L'évaluation de la qualité du produit sert aussi le but d'amélioration du processus. Où il est approprié, et où l'impact et les risques associés sont modélisés et gérés, les changements au processus peuvent être appliqués. L'information obtenue par l'analyse des anomalies est utilisée pour déterminer quels aspects du développement du système nécessitent une amélioration et l'efficacité des analyses et des tests précédemment réalisés. L'évaluation des rapports des différentes phases de tests est souvent combinée avec l'analyse des causes fondamentales, pour évaluer l'efficacité du processus de tests à découvrir les fautes le plus tôt possible.

Les outils peuvent améliorer la qualité du logiciel en permettant au personnel de développement d'éviter le travail répétitif ou de précision pour lequel un ordinateur est mieux adapté. Les différents outils, méthodes et techniques employés devraient être évalués pour leur efficacité et pour savoir s'ils sont appropriés dans le contexte du processus/projet. Le travail du « Software Engineering Process Group » (SEPG) se concentre sur l'amélioration du processus à l'intérieur de l'organisation. En mandatant un groupe pour améliorer le processus, l'organisation s'assure que les objectifs d'amélioration seront atteints et que les activités du processus d'amélioration seront déployées. Les meilleures pratiques se retrouvent aussi dans les modèles d'amélioration du processus et les modèles d'évaluation du processus. Mais, il ne faut pas que l'ensemble de l'organisation se concentre uniquement sur le processus et l'amélioration du processus au dépend du produit à développer.

4.2.7 Invest in the understanding of the problem

Tableau XXVIII

Notions du principe fondamental (G)

Notion	Citation	Repère(s)
L'ingénierie des exigences est à la base de la compréhension du problème soulevé.	• Once the aims of the project have been established, the work of eliciting, analyzing and validating the system requirements can commence. [...] • In almost all cases requirements understanding continues to evolve as design and development proceeds. • Prototyping is commonly employed for validating the requirements engineer's interpretation of the system requirements, as well as for eliciting new requirements.	[CH2S2.3P11] [CH2S2.4P13] [CH2S3.5.2P22]
L'étude approfondie des anomalies d'un système permet d'identifier les zones où il y a eu des lacunes au niveau de la compréhension du	• [...] defects should be analyzed to determine when they were introduced into the systems, what kind of error caused them to be created. • SQA and V&V discover defects. Characterizing those defects enables	[CH5S3.E2P81] [CH11S2.5.4P175]

Tableau XXVIII (suite)

Notion	Citation	Repère(s)
problème.	understanding of the product, facilitates corrections of the process or the product, and informs the project management or customer of the status of the process or product.	
Il faut s'assurer de bien comprendre les problèmes à la base de la création du produit pour en faciliter l'évolution.	• [...] maintenance is really evolutionary developments and that maintenance decisions are aided by understanding what happens to systems (and software) over time. • some 40% to 60% of the maintenance effort is devoted to understanding the software to be modified. • Programmers spend considerable time in reading and comprehending programs in order to implement changes. • Impact analysis identifies all systems and system products affected by a change request, and develops an estimate of the resources needed to accomplish the change.	[CH6S3.1.4P89] [CH6S3.3.1.1P92] [CH6S3.4.1P94] [CH6S3.4.4P95]

Anglais

Once the aims of the project have been established, the work of eliciting, analyzing and validating the system requirements can take place. This procedure is decisive in gaining a clear understanding of the problem to which the system is to provide a solution and its associated costs. Understanding of the problem is a constantly evolving concept. It continues throughout the software life cycle. The development of models, prototypes or other kinds of simulated products is essential in providing the development team insight into completing the requirements analysis.

The organization should implement a defect tracking process whereby activities are identified to track and analyze the cause behind the defects before attempting to remove them from the system. Characterization of identified defects enables a deeper understanding of the product, allows for corrections of the process or the product, and advises project management or the customer of the status of the process or product.

Maintenance is an evolutionary development process whereby decisions are aided by understanding of happening within the system (and software) over time. In reality, the majority of efforts are devoted to understanding the software to be modified. Programmers spend a considerable amount of time reading and gaining a clear understanding of the programs in order to implement changes. Impact analysis is one way to get a clear picture of what will be affected by a required change.

Français

Une fois ciblés les objectifs du projet, le travail d'explicitation, d'analyse et de validation des exigences du système peut commencer. Ceci est crucial pour obtenir une compréhension claire du problème auquel le système offrira une solution et les coûts qui y sont associés. La compréhension du problème est un concept en constante évolution.

Elle continue tout au long du cycle de vie du logiciel. Le développement de modèles, de prototypes ou autres types de produits simulés, est fondamental pour offrir des opportunités à l'équipe de développement de compléter l'analyse des exigences.

L'organisation devrait implanter un processus de suivi des anomalies où les activités sont identifiées pour suivre et pour analyser les raisons des anomalies avant de les retirer du système. La caractérisation des anomalies découvertes permet une meilleure compréhension du produit, facilite les corrections du processus ou du produit, et informe le gestionnaire du projet ou le client du statut du processus ou du produit.

La maintenance est un processus de développement centré sur l'évolution où les décisions sont facilitées par la compréhension de ce qui se passe dans le système (et le logiciel) à travers le temps. Dans la pratique, la majeure partie des efforts est consacrée à la compréhension du logiciel à modifier. Les programmeurs passent un temps considérable à la lecture et à la compréhension des programmes pour implanter les changements. L'analyse d'impact est une façon d'obtenir une compréhension claire de ce qui sera affecté par le changement demandé.

4.2.8 Manage quality throughout the life cycle as formally as possible

Tableau XXIX

Notions du principe fondamental (H)

Notion	Citation	Repère(s)
Pour assurer l'atteinte du niveau de qualité voulu	• It is not sufficient to simply hope that increased quality will result from maintenance of software. It	[CH6S3.2.2.2.2P92] [CH8S3.B.2P125]

Tableau XXIX (suite)

Notion	Citation	Repère(s)
pour un produit, il est primordial de planifier cette qualité et d'en faire un suivi tout au long du cycle de vie du logiciel.	must be planned and process implemented to support the maintenance process. • Comprehensive quality management processes are determined as part of the planning process in the form of procedures and responsibilities for quality assurance, verification and validation [...].	
Il y a plus de chance d'atteindre les objectifs de qualité visés en rendant visible le processus auprès des différents intervenants. Ces objectifs de qualité doivent être alignés avec les exigences du produit.	• An essential property of all requirements is that they should be verifiable. [...] The requirements engineering and V&V personnel must ensure that the requirements can be verified within the available resource constraints. • The quality of the analysis directly affects the product quality. In principle, the more rigorous the analysis, the more confidence can be attached to the software quality. • SQA and V&V also provide management with visibility into the quality of products at each stage in the development or maintenance.	[CH2S2.1P10] [CH2Table 4P21] [CH11S2.2P169]

Tableau XXIX (suite)

Notion	Citation	Repère(s)
Il y a plusieurs façons de gérer la qualité au cours du cycle de vie d'un logiciel.	• Testing is an important, mandatory part of software development; it is a technique for evaluating product quality and also for indirectly improving it, by identifying defects and problems. • Software testing is nowadays seen as an activity that should encompass the whole development process, [...]. • Quality analysis and evaluation tools: There exist a variety of tools and techniques that can help ensure the quality of a design.	[CH3S3.IVP39] [CH5S1P69]

Anglais

When developing software, it is not sufficient to simply hope that increased quality will result from maintenance or from the expertise of the development team members. It must be planned and a SQA process should be implemented to support it. Comprehensive quality management processes are determined as part of the planning process in the form of procedures and responsibilities for quality assurance, verification and validation.

SQA and V&V provide management with visibility into quality of products at each stage in the development or maintenance. The visibility comes from the data and measurements produced through the performance of tasks to assess and measure quality

of outputs of any software development activities. Another important theme of quality management is the evaluation of software products. This includes such diverse activities as peer review of code, test plan, testing, SQA and metrics. To manage quality efficiently, the development team must ensure that all deliverables – software products – can be verified within the available resource constraints.

It is during the software requirements process activities that the quality objectives to be managed are identified. A good requirements engineer will make sure that all requirements will be verifiable, so the quality of the deliverables will be measurable. The more rigorous will be the requirements analysis, the more confidence can be attached to the software quality.

Software testing is seen as an activity that should encompass the whole development life cycle, and is an important part itself of the quality management. Good tools also improve software quality by allowing development team members to avoid repetitive or precise work for which a computer is better suite. Implementing the right tools, based on the context of the project/process, will help managing the quality of the products throughout the software development process.

Français

Lors du développement d'un logiciel, il n'est pas suffisant de simplement espérer que l'amélioration de la qualité résultera de la maintenance du logiciel ou de l'expertise des membres de l'équipe de développement. Elle doit être planifiée et un processus d'assurance qualité du logiciel (AQL) devrait être implanté pour la supporter. Des processus complets de gestion de la qualité sont intégrés au processus de planification sous la forme de procédures et de responsabilités pour l'AQL, la vérification et la validation (V&V).

L'AQL et la V&V fournissent à la gestion une visibilité sur la qualité des produits à chacune des étapes du développement et de la maintenance. Cette visibilité provient des données et des mesures obtenues par la réalisation de tâches pour mesurer la qualité des livrables de toutes les activités du développement logiciel. Un autre important thème de la gestion de la qualité est l'évaluation des produits logiciels. Ceci inclut diverses activités comme la révision par les pairs, la création du plan de tests, les tests, l'AQL et les mesures. Pour gérer efficacement la qualité, l'équipe de développement doit s'assurer que tous les livrables – produits du logiciel – sont vérifiables à l'intérieur des ressources disponibles et des contraintes.

C'est au cours des activités du processus des exigences logicielles que les objectifs de qualité devant être gérés sont identifiés. Un bon ingénieur des exigences fera en sorte que toute les exigences sont vérifiables, donc que la qualité des livrables sera mesurable. Plus l'analyse des exigences sera rigoureuse, plus il y aura une confiance de la qualité du logiciel produit.

Les tests sont vus comme une activité qui doit couvrir l'ensemble du cycle de vie du développement logiciel et est une part importante de la gestion de la qualité. De bons outils améliorent aussi la qualité du logiciel en permettant au personnel de développement d'éviter le travail répétitif ou de précision pour lequel un ordinateur est mieux adapté. Implanter les bons outils, basés sur le contexte du projet/processus, aidera à gérer la qualité des produits tout au long du processus de développement logiciel.

4.2.9 Minimize software component interaction

Tableau XXX

Notions du principe fondamental (I)

Notion	Citation	Repère(s)
Qu'est-ce qui motive le besoin de produire le logiciel en minimisant l'interaction entre ses composants ?	• Classical design admonitions such as the goal of having "cohesion" within modules and to minimize "coupling" are also fundamentally localization of complexity techniques, since they strive to make the number and interaction of parts within a module easy for a person to understand. • [...] This modularity allows both improved analysis and thorough unit-level testing of such components before they are integrated into higher levels in which their errors may be more difficult to identify.	[CH4S3.1.1.3P57] [CH4S3.1.3P59]
La gestion de l'interaction entre les composants se fait en grande partie au cours des activités de SD.	• Decomposition and modularization: [...] • In its strict sense, "a software architecture is a description of the subsystems and components of a software system and the relationship between them" [...].	[CH3S3.IP36] [CH3S3.IIIP38]

Tableau XXX (suite)

Notion	Citation	Repère(s)
Il est possible de mesurer l'interaction entre les composants d'un logiciel.	• Coupling and cohesion: whereas coupling measures the strength of the relationship that exists between modules, cohesion measures how the elements making up a module are related [...]. • [...] while the system is being built and during its future evolution or modification, and these can be considered elements of product quality. These software qualities include, but are not limited to: [...] Modularity of code and independence of modules	[CH3S3.IP36] [CH11S2.1.4P167]

Anglais

The first thing to consider when attempting to minimize software component interaction is the reason why it is done in the first place as well as the benefits the product will draw from it. The answer is not straightforward, but major elements should be well thought-out. The prime reason for accomplishing this is the facility with which the organization will be able to carry out tests and track defects within testing activities. Furthermore, it will allow the development team to effortlessly produce software products and systems than can readily evolve. When proceeding with a system change, if the interaction between software components is well controlled, the impact of the change will be evenly distributed over the entire software.

Software architecture (system blueprint) is the artifact produced by the software design process and is where the software engineer (software architect to be exact) works at dissecting and modularizing the system. The objective will be to place various functionalities or responsibilities in different components. Strictly speaking, software architecture is a description of software subsystems and components and their relationships.

Coupling and cohesion are the two main measures allowing the software engineer to verify the interaction between software components. Whereas coupling measures the strength of the relationship between modules, cohesion measures how the elements making up a module relate to each other. The intent here is to achieve a high level of cohesion within modules, with a low coupling level between modules.

Français

La première chose à considérer lorsqu'il est question de minimiser l'interaction entre les composants logiciels, est la raison pour laquelle il faut le faire et les bénéfices qu'en tirera le produit. La réponse n'est pas nécessairement évidente, mais des éléments importants doivent être bien analysés. La première raison pour le faire est la facilité avec laquelle l'organisation sera en mesure de tester et de suivre les anomalies au cours des activités de tests, suivie par la facilité avec laquelle l'équipe de développement sera en mesure de produire des logiciels et des systèmes pouvant facilement évolués. En procédant à un changement au système, si l'interaction entre les composants du logiciel est bien contrôlée, l'impact du changement sera minimisé sur l'ensemble du logiciel.

L'architecture logicielle (le bleu du système) est l'artéfact produit par le processus de design et est l'endroit où l'ingénieur logiciel (l'architecte logiciel pour être plus précis) travaille à décomposer et à diviser en modules le système. Le but sera de placer différentes fonctionnalités ou responsabilités du logiciel dans différents composants. Au sens strict, l'architecture logiciel est une description des sous-systèmes et composants d'un système logiciel, et de leurs relations.

Deux mesures permettent à l'ingénieur logiciel de vérifier l'interaction entre les composants logiciels : le couplage et la cohésion. Alors que le couplage mesure la force des relations entre les modules, la cohésion mesure comment les éléments composant un module sont reliés. Les cibles de l'ingénieur logiciel pour son produit seront un fort niveau de cohésion à l'intérieur des modules, avec un faible couplage entre les modules.

4.2.10 Produce software in a stepwise fashion

Tableau XXXI

Notions du principe fondamental (J)

Notion	Citation	Repère(s)
Un processus est divisé en activités qui doivent s'adapter au contexte du projet/produit.	• These framework models serve as a high level definition of the phases that occur during development. They are not detailed definitions, but only the high level activities and their interrelationships. • The IEEE standard on	[CH9S3.4.2P144] [CH9S3.4.3P144] [CH11S2.2.1P170]

Tableau XXXI (suite)

Notion	Citation	Repère(s)
	development life cycle processes also provides a list of processes and activities for development and maintenance [...], and provides examples of mapping them to life cycle framework models. • Planning for software quality involves (1) defining, the required product in terms of its quality attributes and (2) planning the processes to achieve the required product.	
Le processus de développement du logiciel est divisé en sous processus ayant chacun leurs propres activités. Chaque activité a des entrées et des sorties qui sont identifiées selon le processus.	• In particular, the subtopic is concerned with how the activities of elicitation, analysis, specification, validation process to terminate. • [...], software design consist of two activities that fit between software requirements analysis and software coding and testing: i) software architectural design – [...]; ii) software detailed design [...] • This is because the construction process consumes the output of the	[CH2S3.1.1P16] [CH3S2P35] [CH4S2P53]

Tableau XXXI (suite)

Notion	Citation	Repère(s)
	Design process (KA3) and itself provides one of the inputs to the Testing process (KA5).	
Le logiciel peut aussi être produit en versions distinctes de façon à livrer le plus tôt possible une version répondant aux exigences prioritaires pour le bénéfice du client.	• Figure 1 A spiral model of the requirements engineering process • Software configuration items are placed under SCM control at different times; i.e. they are incorporated into a particular baseline at a particular point in software life cycle.	[CH2S3FIG1P16] [CH7S3.II.A.6P109]
Des processus de support soutiennent le processus de développement. Ces processus sont aussi divisés en activités.	• Maintenance activities are similar to those of software development. Maintainers perform analysis, design, coding, testing, and documenting. Maintainers must track requirements just as they do in development. • B. Process/Project Management Initiation and scope definition – [...] Planning – [...] Enactment – [...] Review and evaluation – [...] Closure – [...]	[CH6S3.2.2P91] [CH8S3.BP124]

Anglais

Software life cycle is problematic in and of itself. Like any other complex issue, the best way to look at it, is to decompose it in easier to control sub-parts (processes or deliveries). Software life cycle can be based on a life cycle framework model where processes are broken down into multiple activities producing outputs (software products). The selection of activities results from the adaptation of the framework model to the context of the project: the organization (e.g. size, maturity, business domain), the time and resources available (human, monetary, tools) and the constraints. Software requirements may also influence selection of the framework model. During project planning, the goal is to generate the required product within defined quality objectives. One can plan to meet multiple software requirements delivery deadlines; i.e. into a particular baseline at a specific point in time in the software life cycle. In the end, the software is delivered by the organization using the adapted development process.

Usually, the software life cycle will be divided in well-defined steps such as software requirements, software design, software construction, software testing and software maintenance. Each step to produce different types of outputs. Thus, software requirements process will produce documentation to enumerate requirements based on stakeholders' needs. Software engineers will go work through elicitation, analysis, specification and validation activities to document these needs. Software design process will utilize the outputs of the software requirements process to generate models (software architecture and software detailed design) of the targeted system. The construction process consumes the outputs of the design process and in turn provides the inputs for the software testing process.

Throughout the software life cycle, other activities will support development and maintenance functions. As is the case with the development process, the supporting process will be divided into activities such as initiating and scooping, planning, enacting,

reviewing, evaluating, and closing. Maintenance activities are similar to those of the software development process. Maintainers perform analysis, produce designs, code, tests and documentation. They must also track requirements as developers do. Modification requests are used as foundation for the maintenance process.

Français

Le cycle de vie du logiciel est un problème en soi. Comme tout autre problème complexe, la meilleure façon de le résoudre est de le décomposer en parties (processus ou livraisons) plus simples à contrôler. Le cycle de vie du logiciel peut être basé sur un modèle structuré de cycle de vie où les processus sont décomposés en multiples activités produisant des biens livrables (produits logiciels). La sélection des différentes activités est le résultat de l'adaptation du modèle structuré au contexte du projet : l'organisation (taille, maturité, domaine des affaires), l'échéancier et les ressources disponibles (humaines, monétaires, outils) et les contraintes. Les exigences logicielles peuvent aussi influencer la sélection du modèle structuré. Lors de la planification d'un projet, il y a un but de produire le logiciel souhaité, répondant aux objectifs de qualité définis. Plusieurs livraisons peuvent être planifiées pour répondre aux exigences logicielles à différents moments; c'est-à-dire, dans un référentiel particulier à un point particulier du cycle de vie du logiciel. À la fin, le logiciel est livré par l'organisation en utilisant un processus de développement adapté au contexte du projet.

Généralement, le cycle de vie du logiciel sera séparé en étapes définies comme les exigences logicielles, le design logiciel, la construction logicielle, les tests logiciels et la maintenance logicielle. Chacune d'elles produira différents types de biens livrables. Par exemple, le processus d'exigences logicielles produira la documentation énumérant les besoins des différentes parties intéressées. Les ingénieurs logiciels passeront au travers des activités de l'explicitation, de l'analyse, des spécifications et de la validation pour documenter ces besoins. Le processus de design logiciel utilisera les livrables du

processus des exigences logicielles pour générer des modèles (architecture logicielle et design logiciel détaillé) du système ciblé. Le processus de construction utilisera les livrables du processus de design logiciel et fournira à son tour les intrants du processus des tests logiciels.

Au cours du cycle de vie du logiciel, d'autres activités supporteront les activités de développement et de maintenance. Comme pour le processus du développement, les processus de support seront divisés en activités comme l'initiation, la planification, l'exécution, la révision et l'évaluation, et la fermeture. Les activités de maintenance sont similaires à celles du processus de développement logiciel. L'équipe de maintenance fait de l'analyse, du design, du code, des tests et de la documentation. Elle doit aussi faire le suivi des exigences comme le fait l'équipe de développement. Les demandes de changements servent comme points de départ du processus de maintenance.

4.2.11 Set quality objectives for each deliverable

Tableau XXXII

Notions du principe fondamental (K)

Notion	Citation	Repère(s)
Les objectifs de qualité définis pour chacun des livrables ont les exigences identifiées à leur source.	• Non-functional requirements are sometimes known as constraints or quality requirements. • In the planning process, the activities are designed to be fitted to the product and its purpose, including the quality attributes in	[CH2S2.1P10] [CH11S1P165] [CH11S2.1P166] [CH11S2.1P166]

Tableau XXXII (suite)

Notion	Citation	Repère(s)
	the requirements. • The software dictates the performance of the system, and is therefore important to the system quality. Much thought must therefore go into the value to place on each quality attribute desired and on the overall quality of the system. • The important point is that requirements define the required quality of the respective software,[…].	
Le contexte du projet (processus, produit, organisation, contraintes) influence les objectifs de qualité fixés pour chacun des livrables, et la planification des activités permettant d'atteindre les objectifs de qualité.	• […], requirements engineers are necessarily constrained by project management plans and must therefore take steps to ensure that the requirements' quality is as high as possible given the available resources. • Planning for software quality involves (1) defining, the required product in terms of its quality attributes and (2) planning the processes to achieve the required product.	[CH2S2.4P13] [CH11S2.2.1P170] [CH11S2.2.2P170]

Tableau XXXII (suite)

Notion	Citation	Repère(s)
	• The SQA plan defines the processes and procedures that will be used to ensure that the software developed for a specific product meets the requirements…	
L'atteinte des différents objectifs doit pouvoir être vérifiée. Il existe plusieurs facteurs de qualité et de mesures pour pouvoir les évaluer.	• A number of key issues must be dealt with when designing software systems. Some of these are really quality concerns that must be addressed by all systems, for example, performance. • Quality attributes: Various attributes are generally considered important for obtaining a design of good quality, e.g., various "ilities" […], various "nesses" […], […] • The test target and test objective together determine how the test set is identified;[…].	[CH3S3.IIP37] [CH3S3.IVP38] [CH5S2.1P70]

Anglais

The software dictates the performance of the overall system, and is therefore of prime importance for level of system quality. Much thought must therefore go into awarding the required value to each quality attribute as well as that of the quality of the system. A key point to consider during the software life cycle is that requirements define the necessary quality of the respective software – non-functional requirements are also known as "quality requirements". In the planning process, activities are designed to suit the product and its purpose, including requirements quality attributes.

Planning for software quality involves (1) defining the required product in terms of its quality attributes and (2) planning the processes to achieve the required product. The SQA plan defines the processes and procedures to be used to ensure that the software developed for a specific system meets the requirements and is of the highest possible quality and in keeping with project constraints. Therefore, quality targets must be clearly defined and understood. Nevertheless, requirements engineers are essentially constrained by project management plans and must therefore take steps to ensure that quality requirements are as high as can be achieved given the resources available. Thus, quality objectives will be also affected by project context.

Each software development activity output will be produced based on the identified quality requirements for the software and other quality attributes defined by best practices, standards or methods. For example, during software design activities, a number of key issues must be dealt with. Some of these are quality requirements, which must be addressed by all systems, such as performance. Other various attributes are generally considered important for obtaining a good quality design, e.g., various "ilities" (e.g., maintainability, portability, testability, traceability), various "nesses" (e.g., correctness, robustness), to match those quality requirements identified during software requirements activities. The test target and objectives both determine how the test set is identified, keeping in mind the specified quality objectives for the product to be tested.

Français

Le logiciel dicte la performance de l'ensemble du système, il est donc très important au niveau de la qualité du système. Il faut donc se concentrer sur la valeur désirée pour chacun des facteurs de qualité et sur la qualité visée pour l'ensemble du système. Le point important à considérer au cours du cycle de vie du logiciel est que les exigences définissent la qualité requise pour le logiciel – les exigences non fonctionnelles sont aussi appelées « exigences de qualité ». Au cours du processus de planification, des activités sont définies pour s'ajuster au produit et son but, incluant les attributs de qualité des exigences logicielles.

La planification de la qualité logicielle implique (1) la définition du produit requis en terme de ses attributs de qualité et (2) la planification des processus pour obtenir le produit requis. Le plan d'assurance qualité du logiciel définit les processus et les procédures qui seront utilisés pour s'assurer que le logiciel développé pour un système spécifique répond aux exigences et est de la plus haute qualité possible, selon les contraintes du projet. Il doit s'assurer que les objectifs de qualité sont clairement définis

et compris. Mais, les ingénieurs logiciels sont nécessairement contraints par les plans de gestion du projet et doivent alors prendre des moyens pour s'assurer que les exigences de qualité sont aussi élevées que possible étant donné les ressources disponibles. Les objectifs de qualité seront alors aussi affectés par le contexte du projet.

Chacun des biens livrables des activités de développement logiciel sera produit en se basant sur les exigences de qualité identifiées pour le logiciel et pour d'autres attributs de qualité définis par les meilleures pratiques, les standards ou les méthodes. Par exemple, au cours des activités de design logiciel, un certain nombre d'éléments clés doit être considéré. Certains de ceux-là sont des critères de qualité devant être adressés par tous les systèmes, comme la performance par exemple. Divers attributs sont généralement considérés comme importants pour obtenir un design de bonne qualité, comme différents « ilité » (ex. : maintenabilité, portabilité, testabilité, traçabilité), chacun s'ajustant aux exigences de qualité identifiées au cours des activités d'exigences logicielles. La cible des tests et les objectifs des tests déterminent comment sera identifié l'ensemble des cas de tests, toujours en relation avec les objectifs de qualité déterminés pour le produit à tester.

4.2.12 Since change is inherent to software, plan for it and manage it

Tableau XXXIII

Notions du principe fondamental (L)

Notion	Citation	Repère(s)
Il faut reconnaître l'inévitabilité du changement au	• Whatever the cause, it is important to recognize the inevitability of change and adopt measures to mitigate the	[CH2S2.4P13]

Tableau XXXIII (suite)

Notion	Citation	Repère(s)
cours du cycle de vie du logiciel.	effects of change.	
L'évaluation de la volatilité des items permet d'identifier ceux ayant une forte possibilité d'être modifiés au cours du cycle de vie du logiciel.	• Some requirements will change during the life-cycle of the software and even during the development process itself. It is useful if some estimate of the likelihood of a requirement changing can be made. • A first step in controlling change is to identify the software item to be controlled.	[CH2S3.3P19] [CH7S3.II.AP108]
Le logiciel doit être développé en fonction de la possibilité d'un changement (anticipation du changement) au cours de son cycle de vie.	• Flagging requirements that may be volatile can help the software engineer establishing a design that is more tolerant to change. • There is no such thing as an unchanging software construction. Any useful software construction will change in various ways over time, and the anticipation of change drives nearly every aspect of software construction.	[CH2S3.3P19] [CH4S3.1.2P58]
Il faut évaluer l'impact d'un changement sur le produit actuel.	• Tracing is crucial to requirements management because it allows, for example, the impact of any subsequent changes to the	[CH2S2.3P12] [CH2S3.6P23]

Tableau XXXIII (suite)

Notion	Citation	Repère(s)
	requirements to be assessed. • Requirements management is an activity that spans the whole software life cycle. It is fundamentally about change management and the maintenance of the requirements… Tracing is fundamental to performing impact analysis when requirements change.	
Après un changement, il faut valider l'impact réel du changement.	• […], regression testing is the "selective retesting of a system or component to verify that modifications have not caused unintended effects […]". • The objective is to modify existing software product while preserving its integrity.	[CH5S3.B2P76] [CH6S3.1.1P88]

Anglais

There is "no such animal" as unchanging software. Perhaps the most important point to consider is that a significant portion of the software will be subject to change. Some requirements will evolve during the maintenance phase of the software, while others will during the development process itself. It is useful to anticipate the likelihood of a requirement change. The first step in controlling change is to identify the software items needing to be controlled. The volatility/stability of each requirement should be evaluated during software requirements activities. Flagging of probable volatile requirements could allow the software engineer to establish a design that is more tolerant to change.

Regardless of its cause, one must recognize the inevitability of change and adopt measures to mitigate its effects. The software and any changes made to it must be controlled. Configuration control is concerned with managing changes during the software life cycle. It looks at how the interacting items will be identified and how each modified item will be managed and communicated in order to maintain overall system integrity. It covers the process of determining what changes to make, the authority level required for obtaining approval, support for their implementation, and the concept of formal deviation or waivers from project requirements. It is paramount that agreements among stakeholders are reached regarding the means by which scope and requirements are to be reviewed and revised (e.g. via agreed change management procedures).

Impact analysis identifies all systems and system products affected by a change request, and develops an estimate of the resources needed to effect the change. Furthermore, the risk of making the change is assessed. Maintainers must be vigilant concerning the "ripple effect" of any proposed change. Requirements management is an activity that spans the entire software life cycle. That is to say that it is concerned with change management and the maintenance of requirements. Tracing is crucial to requirements management because it allows the impact of any subsequent changes to the requirements or any other products to be assessed.

Good techniques and methods can curtail changes to the software one of which is isolation. By isolating software parts demonstrating high volatility, the software engineer can minimize the impact of changes when they arise. Any useful software construction will change in various ways over time, and anticipating change drives nearly every aspect of software construction. The objective is to be able to modify existing software product while preserving its integrity.

Français

Il est peu probable qu'un logiciel ne soit pas modifié au cours de son cycle de vie. Peut-être le point le plus important à considérer par rapport au logiciel, est qu'une portion significative de ce dernier changera. Certaines exigences changeront au cours de la maintenance du logiciel, d'autres changeront au cours du processus de développement même. Il est utile d'estimer la possibilité qu'un changement à une exigence puisse avoir lieu. La première étape du contrôle du changement est d'identifier les items à être contrôlés. Identifier les exigences qui pourraient être volatiles peut aider l'ingénieur logiciel à établir un design qui soit plus tolérant au changement.

Peu importe la cause, il est important de reconnaître l'inévitabilité du changement et d'adopter des mesures pour minimiser les effets du changement. Le logiciel et tous les

changements faits à ce dernier doivent être contrôlés. Le contrôle de la configuration s'occupe de gérer les changements au cours du cycle de vie du logiciel. Il considère comment les items interagissant seront identifiés et comment les changements aux items seront gérés et communiqués pour conserver l'intégrité à travers l'ensemble des systèmes. Il couvre le processus permettant de déterminer les changements à appliquer, l'autorité nécessaire pour approuver certains changements, le support pour l'implantation de ces changements et le concept de déviation formelle ou de dispense des exigences du projet. Il est vital qu'une entente entre les différentes parties intéressées soit obtenue au sujet des moyens par lesquels l'envergure et les exigences du logiciel seront vérifiées et révisées (accords sur les procédures de gestion des changements).

L'analyse d'impact identifie tous les systèmes et les produits du système affectés par une demande de changement et développe un estimé des ressources requises pour appliquer le changement. De plus, le risque d'effectuer le changement est déterminé. L'équipe de maintenance doit être impliquée pour déterminer les effets en cascade de tout changement proposé. La gestion des exigences est une activité qui couvre l'ensemble du cycle de vie du logiciel. C'est fondamentalement une question de gestion des changements et de suivi des exigences. La « traçabilité » est cruciale pour la gestion des exigences parce qu'elle permet d'évaluer l'impact des changements subséquents aux exigences ou à tout autre produit.

De bonnes techniques et méthodes peuvent minimiser les changements au logiciel. L'isolation en est une de ceux-là. En isolant les parties du logiciel ayant une volatilité plus élevée, l'ingénieur logiciel aide à minimiser l'impact des changements quand ils surviennent. Toute construction logicielle régulièrement utilisée changera de différentes façons à travers le temps et l'anticipation du changement dirige presque tous les aspects de la construction logicielle. L'objectif est d'être en mesure de modifier le produit logiciel tout en conservant son intégrité.

4.2.13 Since tradeoffs are inherent to software engineering, make them explicit and document them

Tableau XXXIV

Notions du principe fondamental (M)

Notion	Citation	Repère(s)
C'est l'ingénieur logiciel qui doit résoudre les conflits reliés aux exigences du produit au cours d'un projet. Sa connaissance du domaine des affaires lui permettra de diriger les négociations entre les différents intervenants.	• One of the fundamental tenets of good software engineering is that there is good communication between system users and system developers. [...] • The requirements engineer needs to acquire or to have available knowledge about the application domains. [...]	[CH2S1P9] [CH2S3.2.1P18]
La négociation des compromis constitue une activité importante et est influencée par le contexte du projet et la compréhension du	• Once identified, the system requirements should be validated by the stakeholders and tradeoffs negotiated before further resources are committed to the project. • [...], it may take considerable effort to reach consensus on requirements	[CH2S2.3P12] [CH2S2.3P12] [CH2S3.1.2P17]

Tableau XXXIV (suite)

Notion	Citation	Repère(s)
problème.	priorities because [...] • It will not be possible to perfectly satisfy the requirements of every stakeholder and the requirements engineer's job is to negotiate a compromise that is both acceptable to the principal stakeholders and within budgetary, technical, regulatory and other constraints.	
Les compromis doivent être documentés.	• Another name commonly used for this subtopic is 'conflict resolution'. It is concerned with resolving problems with requirements where conflicts occur; [...]. It is often important for contractual reasons that such decisions are traceable back to the customer.	[CH2S3.3.4P20]
Il y a d'autres types de compromis à faire au cours d'un projet de développement, en dehors des conflits entre les exigences.	• These models can be analyzed and evaluated to determine if they will allow the various requirements to be fulfilled. Various alternative solutions and tradeoffs can also be examined and evaluated. • Indeed, testing always implies a tradeoff between limited resources and schedules, and inherently	[CH3S2P35] [CH5S2P70]

Tableau XXXIV (suite)

Notion	Citation	Repère(s)
	unlimited test requirements.	

Anglais

One of the tenets of good software engineering is that there is excellent communication between system users and system developers. It is the requirements engineer (software analyst) – helped by the project manager when necessary – who is the conduit for this communication. He must mediate between the domain of the system users (and other stakeholders) and the technical world of the software engineer (software architect, developers and maintainers). This is why the requirements engineer needs to acquire or possess knowledge regarding the product's application domain. This enables him to assess the necessary tradeoffs between conflicting requirements.

Once they are identified, the stakeholders should validate the requirements and negotiate tradeoffs before further resources are committed to the project. It is not possible to entirely satisfy the requirements of every stakeholder and the requirements engineer's task is to negotiate a compromise that is both acceptable to the principal stakeholders and within budgetary, technical, regulatory and other constraints. It may require a considerable amount of effort to reach a consensus on requirements priorities since one stakeholder essential requirement may only have cosmetic value to another. Tradeoffs should be viewed as one mechanism for dealing with conflicting requirements. In reality, the existence of sufficient resources will allow some non-essential requirements to be satisfied, while a lack of resources may mean the exclusion of even strongly advocated requirements.

Conflicts can occur with requirements when two stakeholders require mutually incompatible features, or there is an imbalance between requirements and resources or between capabilities and constraints. It is important, for contractual, political or even possible system evolution reasons, that such decisions are traceable back to the customer or the stakeholder.

Development and maintenance processes should define procedures to document reasons for requirement's withdrawal or retirement. Requirements traceability will assist the software engineer to effect requirements retirement.

Even if many of the conflict resolution activities are part and parcel of the software requirements process, they must be applied, one way or another within other activities of the software development and software maintenance processes. For example, software design activities will produce models that can be analyzed and evaluated to determine if they will allow the various requirements to be fulfilled. Thus, various alternative solutions and tradeoffs can be examined and evaluated.

The production of test cases also implies a tradeoff between limited resources and schedules, and the inherently unlimited test cases, which could be applied to requirements.

Français

Une des notions fondamentales d'une bonne ingénierie logicielle est qu'il y ait une bonne communication entre les utilisateurs et les développeurs du système. C'est l'ingénieur des exigences (analyste logiciel) – pouvant être aidé par le gestionnaire du projet s'il y a lieu – qui dirige cette communication. Il doit servir d'intermédiaire entre le domaine des utilisateurs du système (et autres parties intéressées) et le monde technique de l'ingénieur logiciel (architecte logiciel, développeur et équipe de

maintenance). C'est la raison pour laquelle l'ingénieur des exigences a besoin d'acquérir ou d'avoir une bonne connaissance au sujet des domaines d'application du produit. Ceci lui permet d'évaluer les compromis qui seront nécessaires entre les exigences conflictuelles.

Une fois identifiée, les intéressés doivent valider les exigences et des compromis doivent être négociés avant que toute autre ressource soit réinvestie dans le projet. Il ne sera pas possible de satisfaire à la perfection les exigences de toutes les parties intéressées et c'est le travail de l'ingénieur des exigences de négocier un compromis qui est acceptable pour les principaux intéressés, et à l'intérieur des contraintes budgétaires, techniques, réglementaires ou autres. Des efforts considérables peuvent être exigés pour atteindre un consensus sur le niveau de priorité à donner aux exigences du fait qu'une exigence essentielle d'une partie intéressée peut n'avoir qu'une valeur esthétique pour un autre. Les compromis doivent être vus comme un mécanisme pour s'occuper des contraintes entre les exigences conflictuelles. En pratique, l'existence de ressources suffisantes permettra de satisfaire des exigences non essentielles, alors que des ressources insuffisantes pourrait nécessiter l'exclusion d'exigences fortement recommandées.

Les conflits peuvent survenir entre les exigences quand deux parties intéressées requièrent des fonctionnalités mutuellement incompatibles ou entre des exigences et des ressources ou entre des aptitudes et des contraintes. Il est important, pour des raisons contractuelles, politiques ou même pour des raisons d'évolutions possibles du système, que les compromis puissent être retracés jusqu'au client ou à l'intéressé.

Les processus de développement et de maintenance doivent définir les procédures de documentation des raisons pour le retrait ou l'abandon d'une exigence. La « traçabilité » des exigences aidera l'ingénieur logiciel au cours du processus de retrait d'une exigence.

Même si la majorité des activités de résolution de conflit font partie du processus des exigences logicielles, elles doivent être appliquées, d'une façon ou d'une autre à l'intérieur des autres activités des processus de développement et de maintenance. Par exemple, les activités de design logiciel produiront des modèles qui pourront être analysés et évalués pour déterminer s'ils permettent aux différentes exigences d'être rencontrées. De cette façon, plusieurs solutions alternatives ou de compromis pourront être examinés et évalués. La production des cas de tests nécessite aussi un compromis entre les ressources limitées et les échéanciers et l'ensemble infini des cas de tests pouvant être appliqués aux exigences.

4.2.14 To improve design, study previous solutions to similar problems

Tableau XXXV

Notions du principe fondamental (N)

Notion	Citation	Repère(s)
Des techniques permettent d'étudier et d'utiliser les solutions existantes à des problèmes similaires, comme le « reverse engineering ».	• Architectural design is a skill that is driven by many factors such as the recognition of reusable architectural "patterns" or the existence of off-the shelf components. • Toolkit languages are use to build applications out of toolkits (integrated sets of application-specific reusable parts), and are more complex than configuration	[CH2S2.3P12] [CH4S2.6P56] [CH10S3.EP159]

Tableau XXXV (suite)

Notion	Citation	Repère(s)
	languages. • Reverse engineering tools assist the process by working backwards from an existing product to create abstract artifacts such as design and specification descriptions, which then can be transformed to generate a new product from an old one.	
L'étude des données historiques permet aussi d'identifier la meilleure solution à appliquer au problème soulevé.	• Failure history of similar systems may also help in identifying which activities will be most useful in detecting faults and assessing quality.	[CH11S2.2.1P170]

Anglais

When selecting human resources for a software development project, managers start by identifying personal possessing a good expertise developing the same type of software and sound business domain knowledge. The same approach should be applied to the software. The software engineer should always endeavor to study the architecture of similar products when analyzing a new software design.

There are design techniques to study existing solutions. Architectural design is a skill that is driven by many factors such as the existence of off-the shelf components. Re-engineering is a technique whereby the software engineer examines and alters a subject system to reconstitute it in a new form, and the subsequent implementation of the new

form is applied. Reverse engineering tools assist the process of re-engineering by working backwards from an existing product to create abstract artifacts such as design and specification descriptions, which can then be transformed to generate a new product from an old one. Toolkit languages are another option that is available to developers to reuse solutions built to resolve specific software construction problems.

Reviewing available historical data compiled from software life cycle activities of similar existing softwares will also help the software engineer improve the design. For example, failure history of similar systems will be most useful in selecting which activities will be most effective in detecting defects and assessing quality.

Français

Lorsqu'il s'agit de sélectionner les ressources humaines pour un projet de développement logiciel, les gestionnaires identifient le personnel ayant une bonne expérience dans le développement de logiciels du même acabit et ayant une bonne compréhension du domaine des affaires. La même approche devrait être appliquée au logiciel. L'ingénieur logiciel devrait toujours considérer l'étude de l'architecture de produits similaires lors de l'analyse du design d'un nouveau logiciel.

Il existe des techniques de conception pour étudier des solutions existantes. La conception architecturale est une habileté qui est dirigée par plusieurs facteurs comme l'existence de composants de tierces parties. La « rétro-ingénierie » est une technique où l'ingénieur logiciel examine et modifie le système original pour le reconstituer dans une nouvelle forme et les implantations subséquentes de cette nouvelle forme. Des outils d'ingénierie inverse assistent le processus de « rétro-ingénierie » en permettant de créer des artéfacts sommaires comme la conception et les descriptions de spécification, qui peuvent être transformées pour générer un nouveau produit à partir de l'ancien. Les langages « outils » sont une autre option offerte au développeur pour réutiliser des solutions pour résoudre des problèmes spécifiques de construction logicielle.

La révision de données historiques disponibles, d'activités du cycle de vie du logiciel, provenant d'un logiciel similaire existant, aidera aussi l'ingénieur logiciel à améliorer son design. Par exemple, l'historique des anomalies de systèmes similaires sera des plus utiles pour détecter les anomalies et évaluer la qualité du nouveau logiciel.

4.2.15 Uncertainty is unavoidable in software engineering, identify and manage it

Tableau XXXVI

Notions du principe fondamental (O)

Notion	Citation	Repère(s)
La gestion du risque est à la base de l'identification et de la gestion de l'incertitude.	• Risk management – risk identification and analysis [...], critical risk assessment [...], risk mitigation and contingency planning [...] are all undertaken.	[CH8S3.B.2.6P125]

Tableau XXXVI (suite)

Notion	Citation	Repère(s)
Il existe plusieurs techniques pouvant être utilisées pour minimiser les risques associés au produit.	• [...], effective design techniques instead acknowledge risk, [...] • Structuring for validation means [...] • Selected: [...] How to identify the most suitable selection criterion under given conditions is a very complex problem; in practice risk analysis techniques and test engineering expertise are applied; • [...], testing must be driven based on risk, i.e., testing can also be seen as a risk management strategy.	[CH4S2.1P54] [CH4S3.1.3P59] [CH5S2P70] [CH5S3.A2P75]
Une analyse systématique de l'impact d'un changement proposé au produit, permet de minimiser les risques associés à ce changement.	• Impact analysis identifies all systems and system products affected by a change request and develops an estimate of the resources needed to accomplish the change [...]. Additionally, the risk of making the change is determined.	[CH6S3.2.2.1P91]
Un risque doit être régulièrement réévalué au cours d'un projet. Cet	• Management reviews determine adequacy of and monitor progress or inconsistencies against plans and schedules and requirements.	[CH11S2.3.1.1P172]

Tableau XXXVI (suite)

Notion	Citation	Repère(s)
ajustement du niveau du risque se fait à des points spécifiques au cours du projet et doit être planifié.	These reviews may be exercised on products such as audit reports, progress reports, V&V reports and plans of many types including risk management, software safety, and risk assessment, among others.	
Il y a toujours un risque à sélectionner un standard, une méthode ou à choisir le processus de développement.	• This is a complex issue, however, because the selection of an external standard may need to take account of such difficult-to-predict issues as the long-term economic viability of a particular software company or organization that promotes that standard.	[CH4S3.1.4P59]

Anglais

Uncertainty can be defined as a lack of certainty concerning someone or something. Uncertainty may range from being almost certain to an almost complete lack of conviction or knowledge specifically about an outcome or result [38]. It can also be a risk situation for which the probability of realization, the result and consequence are unknown and cannot be evaluated within reasonable limits. [39]

Software engineering must contend with uncertainty. Normally, the fulfilling of risk management activities allows the software engineer to minimize the effect of risk. Risk management includes risk identification and analysis, critical risk assessment, risk mitigation and contingency planning. All these activities are carried out within the various processes of the software life cycle.

Effective design techniques acknowledge risk, they work to reduce it and ensure that effective alternatives will be available should choices prove wrong. To prevent and tolerate defects as well as deal with exceptional conditions, error and exception handling are implemented within the software design. Structuring for validation is another technique for mitigating risk. It's a way of building software so that errors or omissions can be ferreted out more easily during unit testing and subsequent testing activities.

Testing is a primordial, mandatory part of software development whereby identifying the right test cases constitutes a form of risk management. How to identify the most suitable selection criterion is a very complex problem, in reality one needs to apply situation risk analysis techniques and test engineering expertise. The obvious reason is that complete testing is not feasible in real systems. Therefore, testing must be based on risk, i.e., testing can also be used as a risk management strategy.

Risk is also of prime consideration during software maintenance process. Impact analysis is necessary for risk abatement. It identifies all systems and system products affected by a change request, and develops an estimate of the resources needed to accomplish the change. Additionally, the risk of making the change is assessed.

Management reviews determine the relevance and monitor progress or inconsistencies against plans and schedules and requirements. These reviews may be carried out on products such as audit reports, progress reports and plans of many types including risk

management, software safety, and risk assessment, among others. Risk exposure and leverage are recalculated and decision trees; simulation and so on are running through again in the light of new data.

Finally, when selecting an external tool, standard, method or other product, the software engineer may need to take account of such difficult-to-predict issues as the long-term economic viability of a particular software company or organization that promotes the product.

Français

L'incertitude peut être définie comme un manque de certitude au sujet de quelqu'un ou de quelque chose. L'incertitude peut s'étendre du fait d'être tout près de la certitude jusqu'à un manque presque complet de conviction ou de connaissance par rapport au résultat [42]. Elle peut aussi être définie comme une situation de risque pour laquelle la probabilité de réalisation, le résultat et la conséquence sont inconnus et ne peuvent être raisonnablement estimés [43].

L'ingénierie logicielle doit négocier avec une bonne part d'incertitude. C'est généralement en réalisant les activités de gestion du risque que l'ingénieur logiciel minimisera l'effet du risque. La gestion du risque inclut l'identification et l'analyse du risque, l'évaluation des risques critiques, l'atténuation du risque et la planification de la contingence reliée au risque. Toutes ces activités seront réalisées à l'intérieur des différents processus du cycle de vie du logiciel.

Les techniques efficaces de conception reconnaissent le risque, travaillent à le réduire, et aident à s'assurer que des alternatives efficaces seront disponibles quand certains choix s'avéreront erronés. Pour prévenir et tolérer les fautes et traiter les conditions exceptionnelles, la gestion des erreurs et des exceptions est implantée dans le design

logiciel. Structurer le logiciel en fonction de la validation est une autre technique pour atténuer le risque. C'est une façon de construire le logiciel pour que les erreurs ou les omissions puissent être détectées plus facilement au cours des tests unitaires ou des activités subséquentes de test.

Les tests sont une partie importante et même obligatoire du développement logiciel où l'identification des cas de test constitue une forme de gestion du risque. La façon d'identifier l'ensemble des tests les plus souhaitables est un problème très complexe; en pratique les techniques d'analyse du risque et l'expérience des membres de l'équipe de tests sont utilisées. Une raison évidente est que l'exécution de tous les cas de tests possibles, est irréalisable pour les systèmes réels. Pour cette raison, les tests doivent être basés sur le risque et peuvent aussi être vus comme une stratégie de gestion du risque.

Le risque est aussi une considération majeure au cours du processus de maintenance logicielle. L'analyse d'impact est nécessaire pour la réduction du risque. Elle identifie tous les systèmes et les produits du système affectés par une demande de changement, et développe un estimé des ressources nécessaires pour réaliser le changement. De plus, le risque d'effectuer le changement est identifié.

Les révisions de gestion déterminent la pertinence et contrôlent le progrès ou les inconsistances des différents plans, des cédules et des exigences. Ces révisions peuvent être faites sur les produits, comme les rapports d'audit, les rapports de progression et les plans de plusieurs types incluant la gestion du risque, la sécurité du logiciel et l'évaluation du risque, entre autres. L'exposition au risque et son influence sont recalculées et les arbres de décision, simulations et autres sont ré-appliqués en fonction des nouvelles données.

Enfin, en sélectionnant un outil externe, un standard, une méthode ou un autre produit, l'ingénieur logiciel pourrait avoir besoin de considérer des problèmes difficiles à prévoir à long terme comme la viabilité économique d'une certaine compagnie produisant un logiciel particulier ou l'organisation faisant la promotion du produit.

4.3 Présentation de pistes d'amélioration

Cette section présente les pistes d'amélioration au Guide SWEBOK ou à la discipline du génie logiciel qui ont pu être identifiées au cours de l'analyse de correspondance avec les principes fondamentaux. Pour faciliter cette identification et présenter le raisonnement qui nous amène à identifier des pistes d'amélioration, nous utiliserons les tableaux produits à la section 4.1, et plus spécifiquement, la colonne du niveau de correspondance.

L'hypothèse à la base de cette section est que les principes fondamentaux de l'étude sont bons. Il faut aussi savoir que les pistes présentées constituent le point de vue de l'auteur suite à l'analyse du niveau de correspondance de ces principes fondamentaux par rapport aux différents domaines de connaissances. Il arrive que certaines références soient présentées pour appuyer ces pistes mais ce n'est pas dans la majorité des cas. Il faudrait une analyse plus approfondie pour confirmer les dires de l'auteur. Même si certaines pistes ciblent directement un domaine de connaissances, il est possible que la piste ne puisse être acceptée du fait que la discipline ne serait pas en mesure de la reconnaître comme généralement reconnue.

Pour chacun des principes fondamentaux à l'étude, un tableau sera produit pour présenter le niveau de correspondance maximum atteint et le nombre de citations reconnues dans chacun des domaines de connaissances. Les pistes seront présentées sous forme de texte, chacune étant numérotée pour en faciliter l'identification.

Un tableau final présentera une vue globale des niveaux de correspondance reconnus pour l'ensemble des principes fondamentaux par rapport aux domaines de connaissances du Guide SWEBOK. Celui-ci donnera une vision globale du niveau de correspondance par principe fondamental, et permettra de reconnaître les principes fondamentaux dont la couverture se situe à des niveaux plus élevés au sein de la taxonomie du Guide SWEBOK.

4.3.1 Apply and use quantitative measurements in decision-making

.

Tableau XXXVII

Analyse du niveau de correspondance de (A)

Domaine	**Plus haut niveau reconnu**	**Nombre de citations**
SR	TEXTE	1
SD	SUJET	1
SC	DEF	1
ST	SUJET	3
SM	N1	2
SCM	N1	4
SEM	SUJET	4
SEP	N1	2
SETM	N1	1
SQ	SUJET	4

En étudiant les résultats présentés au tableau XXXVII, nous pouvons remarquer que le niveau de correspondance atteint est assez élevé pour la plupart des domaines de connaissances (8/10 se trouvent dans la taxonomie), et que le principe fondamental a été reconnu dans chacun d'eux. Il a même été reconnu comme sujet principal (SUJET) dans le SD, ST, le SEM et le SQ. Les domaines de connaissances où le niveau de correspondance est faible sont le SR, et le SC.

Piste #1

Il y aurait assurément un besoin d'identifier ce principe fondamental à l'intérieur de la taxonomie du SR, étant donné l'importance d'énoncer des exigences quantifiables. Au cours des activités d'explicitation des exigences, des attributs de qualité sont identifiés par les principaux intéressés, avec leur cible quantitative à atteindre (exigences non fonctionnelles). La sélection des mesures pour vérifier l'atteinte des objectifs de qualité, devrait se retrouver dans la taxonomie du SR (niveau N+).

4.3.2 Build with and for reuse

Tableau XXXVIII

Analyse du niveau de correspondance de (B)

Domaine	Plus haut niveau reconnu	Nombre de Citations
SR	DEF	2
SD	SUJET	3
SC	N+	4
ST	N+	1
SM	N1	1
SCM	SUJET	1
SEM	N+	1
SEP	Aucune	-
SETM	DEF	1
SQ	N+	1

La production d'un logiciel, basée sur la réutilisation et en fonction de la réutilisation, n'a pas été reconnue à l'intérieur du domaine de connaissances du SEP. De plus, la correspondance a été reconnue à l'extérieur de la taxonomie du SR et du SETM.

Piste #2

La réutilisation est aussi bénéfique à la maintenance des systèmes. Une partie du travail de maintenance des systèmes peut être facilitée par la réutilisation d'une méthode efficace de gestion des erreurs et des exceptions, par exemple. La mise en place de standards et de procédures à l'intérieur des systèmes développés est aussi une sorte de réutilisation. Elle voit à définir la manière de produire ou de faire évoluer le logiciel, et l'applique par l'implantation de procédés clairs.

Pour suivre les traitements en production, l'équipe de maintenance exigera que le logiciel soit muni de certains indicateurs (journal de bord, courrier électronique, etc.) dans un format précis. L'équipe de développement mettra en place des composants réutilisables d'une application à l'autre, pour répondre aux exigences de la maintenance.

Les gabarits développés pour faciliter la documentation et offrir un cadre unique de présentation pour tous les intervenants sont aussi une forme de réutilisation. La réutilisation au niveau de la maintenance ne se limite donc pas à la ré-ingénierie.

4.3.3 Control complexity with multiple perspectives and multiple levels of abstraction

Tableau XXXIX

Analyse du niveau de correspondance de (C)

Domaine	Plus haut niveau reconnu	Nombre de citations
SR	N1	3
SD	SUJET	5
SC	SUJET	6
ST	SUJET	3
SM	N1	5
SCM	Aucune	-
SEM	N+	1
SEP	Aucune	-
SETM	Aucune	-
SQ	N+	1

Ce principe fondamental a été reconnu dans la taxonomie de la majorité des domaines de connaissances (7/10). Le niveau de correspondance est très élevé (SUJET et N1) dans les domaines décrivant les différentes activités du processus de développement logiciel (SR, SD, SC, ST et SM).

Piste #3

Aucune citation n'a pu être identifiée dans le SEP. Le processus d'ingénierie logiciel s'avère pourtant très complexe. Lorsqu'il y a un besoin d'en améliorer l'efficacité, cela doit se faire selon plusieurs perspectives. Une section du chapitre 9 devrait traiter de cette complexité à définir et à implanter un processus efficace répondant à la mission et aux objectifs de qualité de l'organisation. Le contexte dans lequel est réalisé un projet de développement de logiciel exige une bonne maîtrise de l'historique de projets de l'entreprise et des ressources mises à la disposition des gestionnaires de projets. En contrôlant la complexité du processus par la définition claire des activités, tout en permettant une adaptation au contexte du projet, l'ingénieur logiciel s'assure d'avoir les outils, méthodes, standards et techniques en main pour contrôler la complexité du produit désiré.

4.3.4 Define software artifacts rigorously

Tableau XXXX

Analyse du niveau de correspondance de (D)

Domaine	Plus haut niveau reconnu	Nombre de citations
SR	SUJET	4
SD	SUJET	3
SC	DEF	1
ST	N+	2
SM	N1	3
SCM	SUJET	4
SEM	N+	2
SEP	N1	1
SETM	SUJET	1
SQ	N1	3

Il a été possible de reconnaître une correspondance avec la définition rigoureuse des artéfacts dans l'ensemble des domaines de connaissances. Cette reconnaissance a eu lieu au niveau d'un sujet principal dans trois (3) des dix (10) domaines et dans la taxonomie de huit (8) des dix (10) domaines.

Piste #4

Pour le SC, il y a une faible reconnaissance du principe fondamental à l'intérieur des définitions du chapitre. Il y a pourtant plusieurs types d'artéfacts qui peuvent être produits par ce processus. En plus des produits exécutables, les librairies de codes et les outils permettant d'automatiser la production de codes en sont des exemples. Une section du domaine devrait traiter exclusivement des types d'artéfacts pouvant être produits par les activités du processus de SC.

Piste #5

Il y a aussi un vide au niveau de la documentation technique présentant la vue interne du logiciel : l'organisation du programme, les commentaires dans le code, la structure du code, etc. Les modèles du SD et la structure finale du produit ne sont pas présentés dans une forme précise de documentation. Peut-être est-ce un vide au niveau de la discipline et qu'il doit y avoir un travail à réaliser pour définir des standards à ce niveau. Ou bien, le guide n'a pu identifier cela dans la littérature du domaine de connaissances.

Piste #6

Il manque aussi d'uniformité au niveau de la correspondance reconnue du principe fondamental dans les différents domaines. Alors qu'il a pu être reconnu comme sujet principal (SUJET) dans le SR, le SCM et le SETM, on le reconnaît à des niveaux inférieurs de la taxonomie du ST, du SM et du SEM (N+), et en dehors de la taxonomie du SD et du SC (DEF). Il faudrait ajuster la taxonomie de l'ensemble des domaines de connaissances pour présenter les artéfacts à produire à un niveau uniforme, d'un domaine à l'autre.

Piste #7

Au niveau du SEP, même si une correspondance a pu être reconnue au niveau de la taxonomie du domaine de connaissances, cette correspondance n'était pas par rapport à la production d'un document contenant la définition des différentes activités des différents processus à utiliser au cours du cycle de vie du logiciel. Il est nécessaire de bien documenter les entrées et les sorties des différents processus et les ajustements possibles selon les contraintes du projet.

4.3.5 Establish a software process that provides flexibility

Tableau XXXXI

Analyse du niveau de correspondance de (E)

Domaine	Plus haut niveau reconnu	Nombre de citations
SR	N1	5
SD	DEF	1
SC	DEF	2
ST	N1	2
SM	Aucune	-
SCM	N1	2
SEM	N+	2
SEP	SUJET	1
SETM	Aucune	-
SQ	SUJET	1

Une correspondance par rapport à l'établissement d'un processus logiciel offrant de la flexibilité a été reconnue dans la taxonomie de six (6) des dix (10) domaines de connaissances. La correspondance a été reconnue au niveau d'un sujet principal du SEP et du SQ.

Piste #8

Pour le SD, c'est au niveau du SC où la meilleure correspondance a pu être reconnue. Selon l'envergure et la taille du produit à développer, les activités du SD et du SC pourraient être combinées [CH4S2.1P54]. Cette citation devrait aussi se retrouver au niveau du SD.

Piste #9

Pour le SM, il faudrait pouvoir reconnaître qu'il y a des différences entre le processus appliqué pour l'évolution, les corrections et les situations critiques de production reliées au logiciel. L'équipe de maintenance du produit doit aussi avoir un processus flexible en place, qui s'ajustera au contexte de la demande de changement. Il faut aussi comprendre que certaines demandes pourraient être assignées à l'équipe de développement, plutôt qu'à l'équipe de maintenance. Les analyses d'impact et de risque fournissent l'information nécessaire au gestionnaire pour définir le processus à appliquer, selon le contexte.

4.3.6 Implement a disciplined approach and improve it continuously

Tableau XXXXII

Analyse du niveau de correspondance de (F)

Domaine	**Plus haut niveau reconnu**	**Nombre de citations**
SR	N1	2
SD	Aucune	-
SC	DEF	2
ST	N1	2
SM	N+	1
SCM	N1	2
SEM	N1	4
SEP	SUJET	3
SETM	SUJET	2
SQ	SUJET	3

Les domaines de connaissances où l'on couvre les activités supportant les processus du cycle de développement logiciel (SEM, SEP, SETM et SQ), ont un niveau de correspondance élevé avec l'implantation d'une approche disciplinée devant être

améliorée constamment. En fait, à l'exception du SC et du SD, une correspondance a été reconnue à l'intérieur même de la taxonomie des domaines de connaissances. Le nombre de citations reconnues est aussi très bon dans la plupart des domaines; une correspondance avec le principe ayant été reconnue au moins deux (2) fois dans huit (8) chapitres sur dix (10).

Piste #10

Par contre, même si le niveau de correspondance est élevé avec le SETM, il serait sans doute pertinent d'ajouter un paragraphe précisant qu'un outil efficace est un outil qui améliore significativement la productivité et/ou la qualité [CH2S2.2P54]. Pour améliorer le processus en place, l'organisation investira dans l'évaluation de ces outils et déploiera ceux qui répondent le mieux au contexte général des projets. Par exemple, certains outils comme les compilateurs, les systèmes de contrôle des versions, les générateurs de code et les éditeurs spécialisés, permettent de supporter les activités de construction du logiciel et de systématiser le travail de production de code.

Piste #11

Au niveau du SC et du SD, il y aurait lieu de préciser que la mise en place de standards permet de systématiser le processus et qu'il est important de faire évoluer ces standards dans le temps pour s'adapter au contexte de l'organisation et des produits à développer.

Piste #12

Au niveau du SD, le choix de la notation utilisée pour produire les modèles des composants du logiciel est très important. L'organisation ne peut se permettre de changer ce choix d'un projet à l'autre. Chaque changement nécessite une formation et une période d'adaptation, autant pour les architectes devant l'appliquer, que pour les développeurs devant transformer les modèles en code.

Il y a donc des choix à faire qui doivent s'aligner avec les connaissances des ressources en place et la structure organisationnelle pour le développement (ex. : équipes distinctes de maintenance et de développement). La mise en place d'un standard sur la notation à utiliser pour définir les modèles de la conception du logiciel, au cours des activités de SD, constitue une approche disciplinée.

4.3.7 Invest in the understanding of the problem

Tableau XXXXIII

Analyse du niveau de correspondance de (G)

Domaine	Plus haut niveau reconnu	Nombre de citations
SR	DOM	6
SD	Aucune	-
SC	N+	1
ST	N+	1
SM	N1	5
SCM	N1	1
SEM	Aucune	-
SEP	Aucune	-
SETM	Aucune	-
SQ	N1	1

Une correspondance a été reconnue dans la taxonomie de six (6) des dix (10) domaines de connaissances. Il y a même un domaine de connaissances avec lequel nous avons identifié une correspondance complète (DOM). En effet, les activités de l'ingénierie des exigences sont là exclusivement pour la compréhension des besoins de tous les intéressés au produit à développer. Il n'y a que dans le SEM et le SEP où une amélioration devrait être apportée pour introduire l'investissement dans la compréhension du problème.

Piste #13

Au niveau du SEM, il est difficile pour les gestionnaires de bien suivre le déroulement d'un projet de développement logiciel si la direction du projet ne maîtrise pas les exigences du produit à développer. Au tout début du projet, un plan d'affaires doit être élaboré pour définir l'envergure du projet de développement, ses bénéfices, ses risques, etc. Par la suite, les activités d'explication des exigences et le plan du projet se basent sur ce plan d'affaires. Cela nécessite d'investir du temps pour bien comprendre les besoins des principaux intéressés et pour définir le plan identifiant les besoins d'affaires devant être comblés par le produit.

Piste #14

Au niveau du SEP, il faudrait voir la compréhension des exigences comme étant à la source de l'identification des activités qui devront avoir lieu au cours du processus. Pour chacune des exigences identifiées au cours des activités d'ingénierie des exigences, des attributs seront assignés à chacune d'elles. Ces attributs influenceront le processus à utiliser pour le développement du produit en définissant l'envergure, la criticité ou d'autres paramètres ayant une influence sur le contexte du projet. Il y a donc un lien majeur entre la compréhension du problème et le processus de développement qui doit être utilisé pour le produit.

4.3.8 Manage quality throughout the life cycle as formally as possible

Tableau XXXXIV

Analyse du niveau de correspondance de (H)

Domaine	Plus haut niveau reconnu	Nombre de citations
SR	N+	3
SD	SUJET	1
SC	SUJET	3
ST	SUJET	4
SM	N+	1
SCM	N1	1
SEM	N1	2
SEP	DEF	1
SETM	N1	1
SQ	DOM	2

Ce principe fondamental a pu être reconnu dans l'ensemble des domaines de connaissances du Guide SWEBOK. Il y a d'ailleurs un domaine de connaissances (SQ) qui traite exclusivement de la qualité du logiciel. La mise en place des activités de SQA et de V&V suffit à rendre formelle la gestion de la qualité.

Point #15

Un point qui nécessitera assurément que l'on y porte attention, est le fait que la correspondance avec ce principe fondamental n'ait pu être reconnue à l'intérieur même de la taxonomie du SEP. Pourtant, les processus sont ce qui permet à la gestion de contrôler la qualité des produits [CH8S2.4P122]. Les processus permettent de formaliser les activités à réaliser à chacune des étapes du cycle de vie du logiciel et à formaliser la façon de réaliser ces étapes.

4.3.9 Minimize software component interaction

Tableau XXXXV

Analyse du niveau de correspondance de (I)

Domaine	Plus haut niveau reconnu	Nombre de citations
SR	Aucune	-
SD	SUJET	3
SC	SUJET	2
ST	Aucune	-
SM	Aucune	-
SCM	Aucune	-
SEM	Aucune	-
SEP	Aucune	-
SETM	Aucune	-
SQ	N1	1

Minimiser l'interaction des composants du logiciel nécessite que les composants aient été identifiés. Ce principe fondamental est donc surtout lié à l'identification des composants à produire et à leur implantation dans le code. Il y avait donc une forte attente à reconnaître sa correspondance à un niveau élevé dans la taxonomie du SD et du SC, ce qui fut le cas. Il en est aussi question au niveau du SQ où des mesures peuvent être prises pour évaluer la cohésion et le couplage entre les composants.

Piste #16

Par contre, il devrait y avoir une correspondance avec ce principe fondamental dans le SM. L'étude et la compréhension d'un système existant sont facilitées par le respect de ce principe au cours du développement et de l'évolution du logiciel. En respectant ce principe fondamental, l'organisation facilite le travail devant être effectué lors des analyses d'impact [CH6S3.3.4P95] et minimise les impacts de ces changements.

4.3.10 Produce software in a stepwise fashion

Tableau XXXXVI

Analyse du niveau de correspondance de (J)

Domaine	Plus haut niveau reconnu	Nombre de citations
SR	N1	2
SD	SUJET	2
SC	DEF	2
ST	N1	1
SM	N1	4
SCM	N1	2
SEM	SUJET	1
SEP	N1	2
SETM	SUJET	1
SQ	N1	1

Le niveau de correspondance avec la production du logiciel par étapes « logiques » est très élevé. Il n'y a qu'à l'intérieur du SC où la reconnaissance du principe se retrouve à l'extérieur de la taxonomie. En fait, le découpage du guide en SR, SD, SC, ST et SM répond directement à ce principe.

Les chapitres du Guide SWEBOK décrivent avec assez de détails, les activités des différents processus suivis au cours du cycle de vie du logiciel. Par exemple, les activités du SR sont identifiées comme étant l'explicitation, l'analyse et la validation des exigences et que ces activités doivent être logiquement exécutées dans cet ordre.

Le SC a un niveau de correspondance très bas. Mais, il est précisé que les activités de production du code utilisent les modèles du SD, et que les livrables du SC servent par la suite d'intrants au ST. Il est donc difficile de présenter cela à un niveau élevé de la taxonomie.

Piste #17

L'étendue de la correspondance du principe fondamental par rapport à la vue de production de logiciels par versions est faible. Il n'y a que deux (2) correspondances dans l'ensemble du Guide qui ont été reconnues; une dans le SR et une autre dans le SCM. Il faudrait probablement ajouter une section dans le SEP par rapport à ce type de production du logiciel. Il existe des méthodologies sur le marché qui privilégient l'utilisation d'un processus en spiral (ex. : RUP) où le logiciel est produit en versions. Il pourrait aussi y avoir un ajout au niveau du SEM du fait que lors de la planification du projet, plusieurs livraisons peuvent être planifiées pour livrer les exigences logicielles à différents moments; c'est-à-dire, dans un référentiel particulier à un point particulier du cycle de vie du logiciel.

4.3.11 Set quality objectives for each deliverable product

Tableau XXXXVII

Analyse du niveau de correspondance de (K)

Domaine	Plus haut niveau reconnu	Nombre de citations
SR	N1	3
SD	SUJET	2
SC	DEF	2
ST	SUJET	2
SM	N+	2
SCM	Aucune	-
SEM	N1	1
SEP	N1	1
SETM	Aucune	-
SQ	SUJET	6

Étant donné qu'un des domaines de connaissances du Guide SWEBOK traite tout spécifiquement de la qualité du logiciel, nous nous attendions à ce que l'identification des objectifs de qualité, pour chacun des livrables, y soit fortement représentée. Le niveau de correspondance et le nombre de citations reconnus avec ce principe fondamental y sont effectivement très élevés. Pour ce qui est des autres domaines de connaissances, aucune correspondance avec le principe n'a pu être reconnue dans la taxonomie du SCM et du SETM, et la correspondance avec le SC se retrouve à l'extérieur de la taxonomie (DEF).

Piste #18

Pourtant, les produits réalisés par les activités du SC servent d'intrant au ST, et pour identifier les objectifs de qualité (ou les cas de tests), il faut d'abord avoir convenu du

niveau de qualité attendu des intrants. C'est au cours des activités d'explicitation des exigences que les objectifs de qualité sont identifiés pour l'ensemble des livrables à produire au cours du processus de développement. La majorité de ces objectifs s'appliquera au produit final : le logiciel. Il devrait donc y avoir une section propre à l'identification et à la mesure de la qualité à l'intérieur de la taxonomie du SC, pour conscientiser les développeurs sur la qualité de ce qu'ils produisent et sur la manière de planifier et de réaliser les activités permettant d'atteindre cette qualité.

4.3.12 Since change is inherent to software, plan for it and manage it

Tableau XXXXVIII

Analyse du niveau de correspondance de (L)

Domaine	Plus haut niveau reconnu	Nombre de citations
SR	SUJET	6
SD	N1	2
SC	SUJET	3
ST	N1	2
SM	N1	4
SCM	SUJET	5
SEM	N+	2
SEP	Aucune	-
SETM	N1	2
SQ	TEXTE	1

Ce principe fondamental a été reconnu dans huit (8) des dix (10) domaines de connaissances et dans la taxonomie de sept (7) d'entre eux. Par contre, il fut surprenant de ne pas le retrouver dans la taxonomie du SD et à un niveau si bas du SQ.

Piste #19

Même si une correspondance avec le principe fondamental a pu être reconnue dans le SD, cette correspondance était implicite. Il n'est pas précisé dans le domaine de connaissances que les différentes notions présentées, comme le couplage et la cohésion, la modularité ou autres, permettaient à l'architecte de baliser les sources probables de changement dans l'évolution du système. L'architecte doit être en mesure d'isoler les composants complexes et d'identifier les contraintes de la solution proposée. Le résultat de l'analyse des exigences (SR) lui fournit l'information nécessaire à sa prise de décision et il doit faire en sorte que l'analyse de la volatilité de chacune des exigences soit visible dans les modèles présentant la conception du produit.

Piste #20

La façon de faire évoluer un système tout en conservant sa qualité – ou en l'améliorant – est une constituante importante des activités de SM. La « maintenabilité » du logiciel doit être évaluée lors de l'application d'un changement pour s'assurer que le produit peut être maintenu sans affecter les coûts futurs de maintenance. Il doit donc être précisé, au niveau du SM, que les changements effectués au logiciel au cours de son évolution ne doivent pas diminuer le niveau de qualité du produit. C'est en planifiant les activités de SQA et V&V à travers le processus de maintenance que la qualité du logiciel peut être maintenue.

4.3.13 Since tradeoffs are inherent to software engineering make them explicit and document them

Tableau XXXXIX

Analyse du niveau de correspondance de (M)

Domaine	Plus haut niveau reconnu	Nombre de citations
SR	SUJET	7
SD	DEF	1
SC	Aucune	-
ST	DEF	1
SM	Aucune	-
SCM	Aucune	-
SEM	N+	1
SEP	Aucune	-
SETM	Aucune	-
SQ	SUJET	1

Le niveau de correspondance avec la négociation au cours des activités d'ingénierie du logiciel, est très bas dans le guide SWEBOK. Il n'y a que trois (3) des dix (10) domaines de connaissances où la correspondance avec ce principe fondamental a pu être reconnue à l'intérieur même de la taxonomie.

Le SR est le domaine où la négociation des exigences est la plus couverte. Le nombre de citations où elle a été reconnue est très élevé (7). C'est ce qui était attendu, car c'est au cours des activités de ce processus que l'ingénieur logiciel doit négocier la plupart des compromis avec le client. C'est en effet au cours des activités d'explicitation et d'analyse des exigences que sont identifiés les conflits potentiels entre les exigences, et que le niveau d'importance de chacune des exigences est fixé.

Piste #21

Le SM est un autre domaine où la négociation des besoins et des priorités est courante en industrie. Pourtant, le principe fondamental n'a pu être reconnu à l'intérieur du domaine. Les résultats, de l'analyse de l'impact des changements demandés et les besoins de corriger les anomalies ayant différents niveaux d'importance, nécessitent que l'équipe de maintenance ait à négocier des priorités avec les différents intervenants et parties intéressées. Cette négociation doit se faire en tenant compte des ressources disponibles pour effectuer les changements requis. Il est même fréquent de devoir définir si la demande est un changement ou une anomalie. La « traçabilité » des exigences permet de faire cette distinction.

4.3.14 To improve design, study previous solution to similar problems

Tableau L

Analyse du niveau de correspondance (N)

Domaine	Plus haut niveau reconnu	Nombre de citations
SR	DEF	1
SD	SUJET	1
SC	DEF	1
ST	Aucune	-
SM	N1	1
SCM	Aucune	-
SEM	Aucune	-
SEP	Aucune	-
SETM	N1	1
SQ	N1	1

L'amélioration du design par l'étude de solutions à un problème similaire, est un principe fondamental qui cible tout particulièrement le SD. Pourtant, la correspondance

avec le principe n'a été reconnue qu'à une seule reprise dans le chapitre. La citation reconnue expliquait les besoins de modéliser l'application en identifiant les possibilités de réutiliser des composants existants.

Piste #22

Nous croyons qu'il faudrait ajouter intégralement le libellé du principe fondamental au niveau du « Software Design Basic Concepts ». La description produite à la section 4.2.14 pourrait servir de contenu à cette nouvelle section du domaine de connaissances.

Piste #23

Un autre domaine de connaissances où une correspondance avec ce principe fondamental aurait dû être reconnue, est le SR. La base de l'architecture du logiciel à produire y est définie [CH2S2.3P12]. L'analyse de la faisabilité de certaines exigences demande qu'une étude de solutions existantes à des problèmes similaires, soit effectuée.

4.3.15 Uncertainty is unavoidable in software engineering, identify and manage it

Tableau LI

Analyse du niveau de correspondance de (O)

Domaine	Plus haut niveau reconnu	Nombre de citations
SR	N1	1
SD	SUJET	1
SC	N1	4
ST	N+	3
SM	N+	3
SCM	N1	1
SEM	N+	2
SEP	N+	1
SETM	N+	1
SQ	SUJET	3

Le principe fondamental traitant de l'identification et de la gestion de l'incertitude, a été reconnu au niveau de la taxonomie de tous les domaines de connaissances. Pour reconnaître une correspondance avec ce principe, nous avons constaté que l'identification et la gestion de l'incertitude faisaient partie intégrante de l'identification et de la gestion des risques. Nous avons donc identifié les citations où il y avait une correspondance avec le risque, plutôt que de limiter cette dernière au seul concept d'incertitude.

Piste #24

Même si une correspondance a été reconnue au niveau du SEP, le principe fondamental aurait dû être identifié plus clairement. Pour mettre en place une infrastructure efficace du processus [CH9S3.2P140], il faut que les risques liés à cette infrastructure soient bien identifiés, pour le contexte dans lequel il sera implanté. Par la suite, tout changement au

processus doit passer par une analyse d'impact et une identification et une évaluation du risque, pour pouvoir planifier les activités de contrôle du risque associé au changement.

Piste #25

Au niveau du domaine traitant de la qualité du logiciel (SQ), il faudrait pouvoir retrouver les activités de mesure du risque à l'intérieur de la taxonomie. L'identification et l'analyse du risque fait partie de la taxonomie de la planification du projet [CH8S3.B.2.6P125], mais il n'y a rien sur la mesure du risque qui a pu être reconnu dans la taxonomie d'un des domaines du guide. Ces sujets devraient faire partie du SQ; il n'y a qu'un texte qui en fait mention [CH11S2.3.1.1P172].

4.3.16 Évaluation globale de la correspondance

Cette dernière section des pistes d'amélioration, présente la compilation des plus hauts niveaux de correspondance reconnus. Ces résultats permettront de présenter aux auteurs du Guide SWEBOK les faiblesses du guide par rapport aux principes fondamentaux suggérés ou à la façon de réaliser l'analyse de correspondance.

Tableau LII

Plus hauts niveaux reconnus[1] de la correspondance

		Domaines de connaissances du Guide SWEBOK										Plus Haut Niveau des Plus Niveaux Reconnus
		SR	**SD**	**SC**	**ST**	**SM**	**SCM**	**SEM**	**SEP**	**SETM**	**SQ**	
Principes fondamentaux de Bourque et Al.	(A)	T	S	DE	S	N1	N1	S	N1	N1	S	SUJET
	(B)	DE	S	N+	N+	N1	S	N+	-	DE	N+	SUJET
	(C)	N1	S	S	S	N1	-	N+	-	-	N+	SUJET
	(D)	S	S	DE	N+	N1	S	N+	N1	S	N1	SUJET
	(E)	N1	DE	DE	N1	-	N1	N+	S	-	S	SUJET
	(F)	N1	-	DE	N1	N+	N1	N1	S	S	S	SUJET
	(G)	DO	-	N+	N+	N1	N1	-	-	-	N1	DOM
	(H)	N+	S	S	S	N+	N1	N1	DE	N1	DO	DOM
	(I)	-	S	S	-	-	-	-	-	-	N1	SUJET
	(J)	N1	S	DE	N1	N1	N1	S	N1	S	N1	SUJET
	(K)	N1	S	DE	S	N+	-	N1	N1	-	S	SUJET
	(L)	S	N1	S	N1	N1	S	N+	-	N1	T	SUJET
	(M)	S	DE	-	DE	-	-	N+	-	-	S	SUJET
	(N)	DE	S	DE	-	N1	-	-	-	N1	N1	SUJET
	(O)	N1	S	N1	N+	N+	N1	N+	N+	N+	S	SUJET

[1] DE : DEF, DO : DOM, I : INTRO, S : SUJET, T : TEXTE, - : Aucune

Les résultats présentés permettent de voir que tous les principes fondamentaux ont pu être reconnus à l'intérieur de la taxonomie d'au moins un domaine de connaissances du Guide SWEBOK. Une correspondance a été reconnue dans chacun des domaines de connaissances pour les principes (A), (D), (H), (J) et (O). On peut aussi remarquer que le principe fondamental (O) est le seul où une correspondance a pu être reconnue à l'intérieur de la taxonomie de tous les domaines de connaissances, en considérant que les correspondances au niveau des définitions ou des textes ne constituent pas une reconnaissance à l'intérieur de la taxonomie. À l'inverse, les principes (I) et (M) ont été reconnus à peu d'endroits dans les domaines de connaissances.

Piste #26

Par contre, les résultats des niveaux de correspondance obtenus sont influencés par le fait que la numérotation du découpage de la taxonomie n'est pas standardisée d'un domaine de connaissances à l'autre. Par exemple, les chapitres du SR, du SCM et du SEM ont tous une numérotation différente pour leur taxonomie respective. De plus, le domaine du SQ ne contient aucune section pour les définitions contrairement à la plupart des domaines de connaissances. Il faudrait que la numérotation de la taxonomie du guide soit standardisée, d'un domaine de connaissances à l'autre, pour que ces résultats puissent être interprétés adéquatement. Il en est de même pour les figures présentant la taxonomie de haut niveau des différents domaines de connaissances. La figure 15 présentant le SC, ne correspond pas vraiment à l'image de la taxonomie du domaine. La taxonomie est présentée selon deux vues dans ce domaine de connaissances et il a été difficile de bien représenter l'étendue de la correspondance des principes fondamentaux dans la figure (section 4.1.16).

4.4 Sommaire

Le tableau LIII présente le sommaire de la phase d'exécution par rapport au cadre de Basili.

Tableau LIII

Exécution du projet

Exécution
Analyse de la correspondance
Au cours de l'analyse de la correspondance entre les principes fondamentaux suggérés par Bourque et al. [5], et le contenu du Guide SWEBOK [6], 15 tableaux de correspondances ont été produits, pour présenter les 259 citations de correspondance.
Production d'une description d'accompagnement par principe fondamental
Pour chacun des 15 principes fondamentaux suggérés par Bourque et al. [5], une description d'accompagnement en anglais et en français a été produite à l'aide des 15 tableaux de présentation des citations sélectionnées. Il y a aussi quinze (15) tableaux qui ont été produits pour présenter les notions de la discipline devant être couvertes par les différents principes fondamentaux.
Présentation de pistes d'amélioration
Au cours de l'analyse du niveau de correspondance, il a été possible d'identifier 26 pistes d'amélioration.

CHAPITRE 5

INTERPRÉTATION

La dernière phase du cadre de la recherche consiste en l'interprétation des résultats. Nous allons d'abord présenter le contexte dans lequel l'étude a été produite et son influence sur l'interprétation des résultats obtenus. Nous mettrons à nouveau l'emphase sur les motivations à la base de l'étude et les objectifs fixés découlant de ces motivations. Par la suite, il y aura une discussion sur l'extrapolation de ces résultats pour identifier les possibilités de réutilisation de la méthode d'analyse et pour l'application des changements suggérés au Guide SWEBOK ou aux principes fondamentaux de l'étude. Enfin, nous présenterons les travaux éventuels pouvant d'écouler des résultats de l'étude.

5.1 Contexte d'interprétation

L'étude était motivée par l'amélioration de la base de la discipline pour qu'elle soit plus facilement reconnue comme discipline du génie. Elle a permis de définir une méthode d'analyse pour reconnaître un principe fondamental dans l'infrastructure de la discipline, de créer une description d'accompagnement pour les principes fondamentaux de l'étude, et d'identifier des pistes d'amélioration au Guide SWEBOK ou à la discipline. La liste de principes fondamentaux à la base de notre étude avait été identifiée par Bourque et al. [5]. Elle faisait partie d'une liste plus importante identifiée par Séguin [7] au cours de son étude de différents ouvrages de spécialistes en génie logiciel, ayant comme but de produire une liste reconnue de principes fondamentaux du génie logiciel.

Avant de débuter la phase d'exécution du projet, il a fallu définir la méthode d'analyse qui permettrait de reconnaître les correspondances entre les principes fondamentaux à l'étude et les domaines de connaissances du Guide SWEBOK. La méthode privilégiée exigeait que chacun des quinze (15) principes soit reconnu partiellement ou entièrement à travers les dix (10) chapitres couvrant les domaines de connaissances du Guide SWEBOK. Lorsqu'une correspondance était reconnue – établie explicitement ou interprétée – le niveau de la correspondance à l'intérieur du domaine était identifié, à l'intérieur (DOM, SUJET, N1, N+) ou à l'extérieur (introduction, définition ou texte) de la taxonomie du domaine. La cotation du niveau de correspondance de la méthode d'analyse est dépendante de l'objet comparé, c'est-à-dire que le découpage de la taxonomie du document influence les niveaux de correspondance choisis. Elle nécessite aussi que la subdivision de la taxonomie de l'objet comparé soit uniforme d'un chapitre à l'autre, ce qui n'était pas nécessairement le cas d'un domaine de connaissances à l'autre dans le Guide SWEBOK.

Cette méthode d'analyse de correspondance est subjective et exige une bonne connaissance de chacun des domaines de connaissances et une certaine expérience en industrie pour réussir à interpréter les correspondances implicites. L'interprétation du découpage des principes dans l'introduction de chacune des parties de l'analyse était aussi subjective. Il est donc possible que certaines correspondances identifiées soient rejetées par un spécialiste en raison des différences de connaissances et d'expérience avec l'auteur. Aussi, il est possible que certaines correspondances n'aient pu être reconnues. En effet, il est peu fréquent de trouver en industrie des individus maîtrisant l'ensemble des connaissances présentées dans le Guide SWEBOK et ayant une expérience concrète pour chacune d'elles. Par contre, ce sont les correspondances les plus évidentes qui ont été identifiées.

Si l'identification des correspondances reconnues était influencée par les connaissances et l'expérience en industrie de l'auteur, l'identification du niveau de correspondance d'une citation était plus simple à déterminer avec la classification définie pour l'étude. Les résultats ont permis de produire les ébauches de description des principes à l'étude et d'identifier certaines pistes d'amélioration au Guide SWEBOK par rapport à ces principes. Mais, les correspondances reconnues et leur niveau de correspondance pourraient être différents si la même étude était réalisée par d'autres spécialistes. Les descriptions résultantes pourraient alors être différentes.

Il faudra voir si Séguin pourra appliquer cette méthode telle quelle ou si des ajustements y seront nécessaires pour ses travaux de recherche. Le travail permettant de reconnaître une correspondance est ardu et nécessite beaucoup d'efforts. Par contre, les résultats obtenus dans cette étude sont concluants et ont permis d'atteindre les objectifs fixés au début du projet :

a. Établir le niveau de correspondance entre les principes fondamentaux et les domaines de connaissances du Guide SWEBOK;
b. Produire une version initiale des descriptions d'accompagnement pour chacun des principes fondamentaux à l'étude; et
c. Identifier des pistes d'amélioration au Guide SWEBOK et à la discipline.

En plus du niveau de correspondance, l'étude a aussi permis d'identifier l'étendue de cette correspondance. Cette variable nous permet de reconnaître si un principe influence ou non le contenu du domaine de connaissances. Un principe qui a été reconnu à plusieurs reprises à travers les différentes sections de la taxonomie d'un domaine de connaissances, a une influence certaine sur le contenu du domaine ou du moins, en est un concept important. Par exemple, tout le domaine du SR repose sur le principe fondamental d'investissement dans la compréhension du problème (G), alors que le domaine du SQ repose sur le principe fondamental de la gestion de la qualité de façon formelle tout au long du cycle de vie (H).

5.2 Extrapolation

Premièrement, il faut comprendre que la démarche de notre méthode d'analyse de la correspondance s'extrapole bien, mais pas les résultats détaillés qu'elle a produits. Ainsi, l'avantage de la méthode d'analyse utilisée au cours de l'étude, est qu'elle permet de définir l'échelle de cotation de la correspondance reconnue selon les documents comparés et que le résultat de l'application de cette cotation peut lui-même être analysé pour identifier les faiblesses des objets comparés. Elle peut aussi être utilisée dans une direction comme dans l'autre. Par contre, la subjectivité de la méthode d'analyse minimise l'extrapolation possible des résultats. Cette méthode est dépendante des connaissances et de l'expérience de l'auteur par rapport à l'objet comparé. Elle exige aussi beaucoup d'efforts et il est difficile de maintenir un niveau d'efficacité uniforme, sur l'ensemble des sujets couverts par l'objet comparé, si l'analyste ne les maîtrise pas tous.

Ainsi, Séguin, pour sa recherche, pourrait utiliser la méthode de correspondance de l'étude pour définir le niveau de correspondance du sous-ensemble de principes fondamentaux qu'il identifiera. Cette analyse de correspondance serait réalisée contre le

corpus de connaissances, les normes IEEE et le curriculum du génie logiciel, tel que cela a été défini en [3]. Par contre, l'utilisation de nos résultats serait limitée par rapport à l'envergure de l'analyse de la correspondance qu'il désire réaliser pour sa recherche. En effet, nos résultats ne touchent qu'à une faible partie du travail de correspondance que Séguin désire réaliser. Lorsqu'il aura complété cette analyse, il aura à faire valider les résultats par des spécialistes de chacun des domaines de connaissances du corpus de connaissances, des différentes normes IEEE et du curriculum du génie logiciel, comme le suggère le cadre de Basili [8].

Par contre, il est difficile de définir une méthode permettant d'identifier un principe fondamental car ce sont les principes fondamentaux qui sont à la base du corpus de connaissances, des normes, des méthodes, des techniques et du curriculum du génie logiciel. Les principes fondamentaux sont plutôt reconnus par les intervenants du domaine. La méthode de correspondance peut être utilisée pour voir si cette base du domaine est bien appliquée et si le principe fondamental est complet par rapport à ce qui a été reconnu comme connaissance, norme, méthode ou technique par l'industrie.

En second lieu, les descriptions produites au cours de notre étude limiteront les contextes d'interprétation des principes fondamentaux suggérés par Bourque et al. [5] dans les étapes subséquentes amenant à leur reconnaissance. Mais, il faut que ces descriptions soient révisées par des experts de la discipline avant de pouvoir servir à faire reconnaître ces principes. Cela aussi introduit une limitation importante aux résultats de l'étude et conséquemment à leur extrapolation à d'autres contextes. Une ou plusieurs sessions de travail pourraient éventuellement améliorer les descriptions d'accompagnement. Cette étape du cadre de Basili ne faisait pas partie de notre étude car il était déjà prévu qu'un atelier sur les principes fondamentaux aurait lieu en utilisant les descriptions d'accompagnement comme intrants.

Enfin, une limitation importante à considérer pour l'extrapolation des résultats est que plusieurs des intervenants ayant travaillé sur les deux principaux documents de notre étude sont les mêmes (Bourque, Abran, Moore, Dupuis et Tripp). De plus, c'est Bourque qui a dirigé l'auteur de cette étude. Il est donc possible que certains biais se soient glissés dans les résultats obtenus. Par contre, ces deux documents ont fait l'objet de plusieurs rondes de révision par plusieurs spécialistes de la discipline, ce qui minimise ces possibilités de biais.

5.3 Travaux subséquents

Une suite logique à cette étude serait de compléter la correspondance entre les principes et l'ensemble des références des différents domaines de connaissances du Guide SWEBOK. Cela pourrait permettre de confirmer les pistes d'amélioration de la section 4.3 en identifiant une ou des références permettant de justifier ces pistes par rapport aux connaissances généralement reconnues du génie logiciel. Le même travail de correspondance pourrait aussi être refait par des spécialistes de chacun des domaines de connaissances, pour valider le travail d'analyse de la correspondance (section 4.1) réalisé au cours de l'étude.

En second lieu, un atelier basé sur l'étude des principes fondamentaux suggérés par Bourque et al. [5], en utilisant les descriptions produites à la section 4.2, confirmerait le potentiel de ces principes – ou d'une partie de la liste – à être reconnus par la discipline. Cet atelier permettrait probablement d'ajuster certains des libellés et d'améliorer les descriptions d'accompagnement.

Il faudrait aussi reproduire l'étude dans l'autre sens, du Guide SWEBOK vers les principes fondamentaux. C'est ce qui a dû être fait, en partie, pour produire les ébauches de description des principes à l'étude à l'aide des citations de correspondance. Le fait de réaliser l'étude dans un sens seulement – des principes vers les domaines – ne permet pas de confirmer si la liste suggérée par Bourque et al. [5] est complète.

En effet, même si ce n'était pas l'un des objectifs de l'étude, il a été possible de déceler certaines lacunes au niveau des principes à l'étude. Les pistes d'amélioration au Guide SWEBOK et à la discipline prenaient pour acquis que la liste des principes fondamentaux à l'étude était reconnue alors qu'en fait, elle ne l'est pas. Des améliorations doivent assurément être apportées à cette liste et par ricochet, aux descriptions d'accompagnement produites. Par exemple, le libellé des principes où il est question de « gestion » devrait contenir les termes suivants :

a. Identification (Identify),
b. Planification (Plan),
c. Suivi ou gestion (Manage), et
d. Révision (Review).

Concernant le fait que cette liste pourrait être incomplète, Séguin [7] identifie une classification possible des principes fondamentaux selon le processus, le produit et les individus (Process-Product-People). Aucun des principes à l'étude ne peut être catégorisé au niveau des individus alors que le Guide SWEBOK identifie à plusieurs endroits les besoins de planifier en fonction des ressources disponibles, autant humaines que matérielles, et l'importance de l'investissement dans les ressources humaines. Les projets sont influencés par la disponibilité, la formation, la motivation et le développement de carrière du personnel impliqué dans le développement et la

maintenance d'un logiciel. Les citations suivantes présentent certaines pistes permettant de produire un libellé pour un principe fondamental sur la gestion des ressources humaines en génie logiciel :

a. [CH8S3.C.3.3P128]

« [...] *the primary resource that needs to be managed in software engineering is personnel.* »

b. [CH9S3.2P140]

« *At the initiation of process engineering, it is necessary to have an appropriate infrastructure in place. This includes having the resources (competent staff, tools and funding), as well as the assignment of responsibilities.* »

5.4 Récapitulation

Dans cette section, le tableau LIV présente la récapitulation de la phase d'interprétation des résultats de l'étude.

Tableau LIV

Interprétation, récapitulatif

***IV.* Interprétation**		
Contexte d'interprétation	**Extrapolation**	**Travaux subséquents**
Objectif 1 Le niveau de correspondance entre les principes fondamentaux suggérés et les domaines de connaissances a	L'analyse de correspondance est subjective. Elle dépend des connaissances et de l'expérience de l'auteur dans le domaine de connaissances	L'analyse du niveau de correspondance devrait être étendue à l'ensemble des références du corpus

Tableau LIV (suite)

Contexte d'interprétation	Extrapolation	Travaux subséquents
été évalué. Ce niveau permet de vérifier si un principe fondamental est bien couvert dans le contenu des différents domaines de connaissances. Il identifie aussi les faiblesses du Guide SWEBOK par rapport à un principe fondamental. Cet objectif a été atteint.	comparé. Un autre spécialiste pourrait obtenir une liste différente de citations de correspondance. La taxonomie du Guide SWEBOK n'est pas uniforme d'un chapitre à l'autre, ce qui peut causer des distorsions par rapport au niveau de correspondance identifié dans certains domaines de connaissances.	de connaissances identifié dans le Guide SWEBOK. L'analyse de correspondance devrait être réalisée dans l'autre sens. Il est en effet possible que la liste suggérée par Bourque et al. [5] ait des lacunes et qu'elle pourrait être incomplète.
Objectif 2 En utilisant le résultat de l'analyse de correspondance, il a été possible de produire des descriptions d'accompagnement pour chacun des principes fondamentaux. L'objectif était de produire des ébauches et des travaux devront être effectués pour en raffiner le	Les descriptions produites sont directement associées aux résultats de l'analyse de correspondance. Elles sont donc dépendantes des connaissances et de l'expérience de l'analyste, et pourrait être incomplètes. De plus, elles n'ont pas fait l'objet d'une ronde de révision par des spécialistes des principes	Un atelier de spécialistes en génie logiciel ayant en entrée la liste des principes fondamentaux de Bourque et al. [5] et les descriptions d'accompagnement, permettrait de solidifier les libellés

Tableau LIV (suite)

Contexte d'interprétation	Extrapolation	Travaux subséquents
contenu. Les descriptions d'accompagnement permettront de limiter les interprétations possibles des principes fondamentaux. Cet objectif a été atteint.	fondamentaux du génie logiciel.	de la liste et de confirmer ou d'infirmer certaines notions présentées.
Objectif 3 Suite à l'analyse de correspondance, des pistes d'amélioration au Guide	Les suggestions d'amélioration ont été produites en considérant que la liste des principes fondamentaux de	Les pistes d'amélioration seront remises à l'équipe du Guide SWEBOK
SWEBOK ou à la discipline ont pu être identifiées. Ces pistes ont pour but de mieux reconnaître les principes fondamentaux à l'intérieur même des domaines de connaissances présentées dans le Guide SWEBOK.	l'étude était reconnue, alors qu'il n'en est rien. Certaines de ces suggestions pourraient donc ne pas être appropriées et devront être confirmées par l'étude plus approfondie du corpus de connaissances. Elles étaient aussi dépendantes des connaissances et de l'expérience de l'auteur par rapport à chacun des domaines de connaissances. Il faudrait que chacune des	pour étude.

Tableau LIV (suite)

Contexte d'interprétation	Extrapolation	Travaux subséquents
	suggestions soit appuyée par une référence reconnue par des spécialistes du génie logiciel.	

CONCLUSION

Cette étude a été réalisée pour faire suite aux travaux de Bourque et al. [5] qui avaient pour but de produire une liste de principes fondamentaux qui serait reconnue par la discipline. Elle constitue une étape essentielle dans la reconnaissance de ces principes en identifiant des éléments de justification à partir des connaissances généralement reconnues du génie logiciel. Elle a aussi l'avantage d'avoir identifié des changements qui pourront être appliqués au Guide SWEBOK [6], advenant que certains de ces principes étaient reconnus comme fondamentaux.

L'étude a un lien important avec le travail de recherche de Séguin [7]. Ce projet a un but similaire à celui de Bourque et al. [5], mais en se basant sur la littérature traitant des principes fondamentaux en génie logiciel. Dans son projet de recherche, Séguin veut réaliser une analyse de correspondance similaire à celle qui a été utilisée dans notre étude.

L'auteur a pu remarquer, au cours de l'analyse de la correspondance, que la liste des principes fondamentaux suggérés par Bourque et al. [5] permettait de répondre à des préoccupations énoncées dans le Guide SWEBOK. Certaines de ces préoccupations sont présentées dans le SEM [CH8S2.2P121]. L'auteur présente ici ces citations qui proviennent de cette section du Guide SWEBOK. Cette identification ne faisait pas partie de l'étude, c'est pour cette raison qu'elle est présentée en conclusion. L'auteur a associé les principes qui permettent de minimiser les impacts par rapport à chacune des situations identifiées :

a. *the perception of clients is such that there is a lack of appreciation for the complexity inherent in software engineering, particularly in relation to the impact of changing requirements* :

- Control complexity with multiple perspectives and multiple levels of abstraction (C),
- Since change is inherent to software, plan for it and manage it (L),
- Since tradeoffs are inherent to software engineering, make them explicit and document them (M),
- Uncertainty is unavoidable in software engineering, identify and manage it (O)

b. *it is almost inevitable that the software engineering process itself will generate the need for new or changed client requirements*:
   - Establish a software process that provides flexibility (E),
   - Produce software in a stepwise fashion (J),
   - Since change is inherent to software, plan for it and manage it (L)

c. *software is often built in an iterative process rather than a concrete sequence of closed tasks* :
   - Implement a disciplined approach and improve it continuously (F),
   - Since change is inherent to software, plan for it and manage it (L),
   - Uncertainty is unavoidable in software engineering. Identify and manage it (O)

d. *software engineering necessarily incorporates aspects of creativity and discipline – maintaining an appropriate balance between the two is often difficult* :
   - Establish a software process that provides flexibility (E)
   - Manage quality throughout the life cycle as formally as possible (H),

e. *unlike many other disciplines, we are largely lacking an underlying theory (e.g. engineering is founded on the principles of physics and mathematics)* :
   - Control complexity with multiple perspectives and multiple levels of abstraction (C),
   - Minimize software component interaction (I),
   - To improve design, study previous solutions to similar problems (N)

f. *software engineers create intangible products [...]* :
   - Define software artifacts rigorously (D),
   - Invest in the understanding of the problem (G),
   - Manage quality throughout the life cycle as formally as possible (H),
   - Set quality objectives for each deliverable (K),
   - To improve design, study previous solutions to similar problems (N),
   - Uncertainty is unavoidable in software engineering. Identify and manage it (O)

g. *software engineers create intangible products that cannot easily be tested in the same sense that a physical product can* :
   - Control complexity with multiple perspectives and multiple levels of abstraction (C),
   - Define software artifacts rigorously (D),
   - Invest in the understanding of the problem (G)

h. *we are faced with an extremely rapid rate of change in the underlying technology* :
   - Establish a software process that provides flexibility (E),
   - Minimize software component interaction (I),
   - Since change is inherent to software, plan for it and manage it (L)

En respectant les principes fondamentaux présentés dans l'étude, l'organisation s'assure de minimiser l'impact des diverses préoccupations identifiées dans cette partie du Guide. Ces préoccupations sont des situations courantes en industrie.

Un autre élément intéressant qui a été remarqué par l'auteur au cours de l'étude, est la façon de percevoir le contenu d'un document au cours de sa lecture, selon l'objectif fixé avant sa (lecture) réalisation. Le Guide SWEBOK a été lu à plusieurs reprises mais à chaque fois avec une vue différente, selon le principe fondamental qui était étudié. Cette façon de lire le Guide SWEBOK a permis d'en percevoir toute la profondeur, ainsi que ses avantages et ses inconvénients.

L'auteur a pu aussi constater que la partie la plus complexe d'un travail de cette envergure est d'en définir la portée et de clarifier cette portée. Il faut être rigoureux pour que le document résultant de l'étude, puisse être compris et accepté par les spécialistes de la discipline. Il est souvent difficile de mettre en place cette rigueur et de faire en sort qu'elle soit constante dans l'ensemble du document. C'est une technique qui s'apprend par la pratique.

Le travail d'un chercheur est bien différent de celui d'un praticien. Mais, avec l'existence d'Internet, il devient plus simple à l'un et à l'autre d'échanger des résultats ou des préoccupations par rapport à la discipline. Cette banque de données universelle permet d'accélérer la vitesse à laquelle la discipline évolue; ce qui est un bien pour l'ensemble de la communauté.

BIBLIOGRAPHIE

[1] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, New York, Std 610.12-1990 (Revision and redesignation of IEEE Std 792-1983), 1990.

[2] Gabrini, P. (2003). *Pourquoi le génie logiciel est-il d'actualité?*, Avril 2003, pp. 1-4, [En ligne]. http://www.sciences.uqam.ca/pdf/gabrini.pdf (Consulté le 20 mars 2004).

[3] Knuth, D.E. (1968, 1969, 1973). *The Art of Computer Programming*, Vol. 1, 2, 3. Addison-Wesley.

[4] Article de l'institut des ingénieurs

[5] Bourque, P., Dupuis, R., Abran, A., Moore, J., Tripp, L., Wolff, S. (2002). *Fundamental Principles of Software Engineering – A Journey*, Journal of Systems and Software

[6] Abran et al. (2001). *Guide to the Software Engineering Body of Knowledge : Trial Version (Trial Version 1.00 – May 2001)*. A project of the Software Engineering Coordinating Committee (Joint IEEE Computer Society – ACM committee). Executive Editors : Alain Abran, Université du Québec à Montréal, James W. Moore, The MITRE Corp.; Editors : Pierre Bourque, Université du Québec à Montréal, Robert Dupuis, Université du Québec à Montréal; Chair of the Software Engineering Coordinating Committee : Leonard L. Tripp, IEEE Computer Society.

[7] Séguin, N. (2003). *L'établissement d'un premier corpus structuré et documenté de principes fondamentaux en génie logiciel*, DGA1020 – Proposition de recherche, École de Technologie Supérieure, Juin 2003.

[8] Basili, V. et R., Selby R.W., Hutchens, D.H. (1986). *Experimentation in Software Engineering*, IEEE Transactions on Software Engineering, Vol. SE.12, no 7.

[9] Bourque, P., Côté, V. (1991). *An Experiment in Software Sizing with Structured Analysis Metrics*, in Journal of Systems and Software, vol. 15, pp. 159-172.

[10] Brouillette, M. (2002). *Comparative Analysis of the Guide to the Software Engineering Body of Knowledge and the Rational Unified Process®*, Projet de Maîtrise en genie logiciel, École de Technologie Supérieure, Septembre 2002.

[11] Belkebir, Y. (2003). *Analyse et amelioration des definitions de rôles du processus d'ingénierie logicielle du centre de compétence en génie logiciel de Bombardier Transport Montréal*, Projet de Maîtrise en génie logiciel, École de Technologie Supérieure, Mai 2003.

[12] Abran, A., Laframboise, L., Bourque, P. (2003). *Systems Management*, Novembre 2003.

[13] *Le Petit Larousse Grand Format 2001*, Larouse / HER 2000

[14] Moore, J.W. (1998). *Software Engineering Standards – A User's Road Map*, IEEE Computer Society Press, Los Alamitos, California.

[15] Royce, W. (1970). *Managing the development of large Software Systems*, Reprinted in 9th International Conference on Software Engineering, IEEE Computer Society Press, pp. 328-338

[16] Ghezzi, C., Jazayeri, M., Mandrioli, D. (2003). Fundamentals of Sofware Engineering, (2e éd.), New-Jersey : Prentice Hall.

[17] Ross, D.T., Goodenough, J.B., Irvine, C.A. (1975). *Software engineering: Process, principles, and goals*. COMPUTER 8, 5 (May 1975), pp. 17-27

[18] Mills, H.D. (1980). *The management of software engineering : Part I : Principles of software engineering. IBM Systems Journal*. Vol 19, No 4, pp. 414-420.

[19] Lehman, M. (1980). *On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle*. Journal of Systems and Software, Vol.1, No 3, pp. 213-221, Juillet 1980.

[20] Boehm, B.W. (1983). *Seven Basic Principles of Software Engineering*. The Journal of Systems and Software, Vol.3, no 1, pp. 366-371, Mai 1983.

[21] Booch, G., Bryan, D. (1994). *Software Engineering with Ada, 3e éd.*, California : Benjamin/Cummings Publishing.

[22] Davis, A.M. (1995). *201 Principles of Software Development*, New-York : MgGraw-Hill.

[23] Davis, A.M. (1994). "*Fifteen Principles of Software Engineering*". IEEE Software, pp. 94-101, Novembre 1994.

[24] Bushcmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. (1996). *Pattern-Oriented Software Architecture*, England : Wily.

[25] Wiegers, K.E. (1996). *Creating a Software Engineering Culture*, New-York : Dorset House Publishing.

[26] Wasserman, A.I. (1996). *Toward a Discipline of Software Engineering*, IEEE Software, Novembre 1996, pp. 23-31.

[27] Maibaum, T.S.E. (2000). *Mathematical foundation of software engineering*, Proceedings of the Conference on the Futur of Software Engineering, ACM, pp. 161-172.

[28] Vincenti, W.G. (1990). *What Engineers Know and How They Know It*, The Johns Hopkins University Press.

[29] Taylor, P. (2001). *Interpreting Mayall's 'Principles in Design'*, Proceedings Software Engineering Conference, pp. 297-305.

[30] Mayall W. (1979). *Principles in Design*, London: Design Council.

[31] Meyer, B. (2001). *Software Engineering in the Academy*, IEEE Computer, pp. 28-36, Mai 2001.

[32] SWEBOK, Institute of Electrical and Electronics Engineers Inc., *Project Overview*, Copyright (c) 2001, [En ligne]. http://www.swebok.org/overview/ (Consulté le 1er mars 2004).

[33] *A Guide to the Project Management Body of Knowledge (PMBOK® Guide) - 2000 Edition*, Project Management Institute, Inc. (PMI), 2000.

[34] *IEEE guide - Adoption of PMI standard - A Guide to the Project Management Body of Knowledge*, IEEE Std 1490-1998, Adaptation of PMI Guide to PMBOK, The Institute of Electrical and Electronics Engineers, Inc., New York, 1er mars 1999.

[35] *ISESS'97*, Third International Symposium and Forum on Software Engineering Standards, Walnut Creek, CA, IEEE Computer Society, June 1997.

[36] Construx Software Builders, Inc., *Getting Started With CxOne, Top Level Overview*, CxOne 2.1 (11/05/02), p.1, [En ligne]. http://www.construx.com/docs/open/CxOneOverview.pdf (Consulté le 1er mars 2004).

[37] Thompson, J.B., Hardy, C.J. (2002). *Use and Evaluation of SWEBOK by Postgraduate Students*, Proceedings of the 15th Conference on Software Engineering Education and Training (CSEET'02), IEEE, School of Computing Engineering and Technology, University of Sunderland, Sunderland, UK, [En ligne]. http://www.cs.wm.edu/~coppit/csci690/papers/15150066.pdf (Consulté le 1er mars 2004).

[38] Wikipedia, The Free Encyclopedia . *SWEBOK*, [En ligne]. http://en.wikipedia.org/wiki/SWEBOK (Consulté le 1er mars 2004).

[39] Kaner, C. (2003). *SWEBOK Problems, Part 2*, CEM KANER'S BLOG, 27juin 2003, [En ligne]. http://blackbox.cs.fit.edu/blog/kaner/archives/000095.html, (Consulté le 1er mars 2004).

[40] White, J., Simons, B. (2002), *ACM's position on the licensing of software engineers*, Communications of the ACM, Volume 45, Number 11 (2002), Page 91.

[41] ElsevierComputerScience, *Journal of Systems and Software*, Copyright 2004 Elsevier B.V., [En ligne]. http://www.elsevier.com/wps/find/journaldescription.cws_home/505732/description#description (Consulté le 1er mars 2004).

[42] Merriam-Webster Inc. Company Information. *Merriam-Webster Online Dictionary*, [En ligne]. http://www.m-w.com (Consulté le 25 janvier 2004).

[43] Office québécois de la langue française, Gouvernement du Québec. *Le Grand dictionnaire terminologique*, [En ligne]. http://w3.granddictionnaire.com (Consulté le 24 janvier 2004).